№10 Издание основано в 1995 г. inf.1september.ru
УЧЕБНО-МЕТОДИЧЕСКИЙ ЖУРНАЛ ДЛЯ УЧИТЕЛЕЙ ИНФОРМАТИКИ

ИНФОРМАТИКА

Электронная версия журнала
дополнительные материалы
в Личном кабинете
на сайте
www.1september.ru

издательский дом 1september.ru
Первое сентября
октябрь 2014
ИНФОРМАТИКА Подписка на сайте www.1september.ru или по каталогу «Почта России»: 79066 — бумажная версия, 12684 — CD-версия

НА ОБЛОЖКЕ

Учителя информатики со стажем наверняка помнят нешуточные битвы сторонников “пользовательского” и “алгоритмического” курсов. Вторые шли на первых, держа наперевес заостренный тезис “Информатика – это не Microsoft Office”. Первые отбивались щитами с поурочным планированием, на которых было написано “8-й класс: Microsoft Word”, “9-й класс: Microsoft Excel”, “10-й класс: Microsoft PowerPoint”, “11-й класс: Microsoft Access”. Эх, были времена. С этими приятными воспоминаниями мы и отмечаем четвертьвековой юбилей главного офисного пакета всех времен и стран.

В НОМЕРЕ

В ЛИЧНОМ КАБИНЕТЕ

Облачные технологии от Издательского дома “Первое сентября”

Уважаемые подписчики бумажной версии журнала!

Дополнительные материалы к номеру и электронная версия журнала находятся в вашем Личном кабинете на сайте www.1september.ru.

Для доступа к материалам воспользуйтесь, пожалуйста, кодом доступа, вложенным в №7–8/2014.

Срок действия кода: с 1 июля по 31 декабря 2014 года.

Для активации кода:

- зайдите на сайт www.1september.ru;
- откройте Личный кабинет (создайте, если у вас его еще нет);
- введите код доступа и выберите свое издание.

Справки: podpiska@1september.ru или через службу поддержки на портале “Первого сентября”.

ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ

Презентации и исходные файлы к статьям номера

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ
по каталогу “Почта России”: 79066 — бумажная версия, 12684 — электронная версия

http://inf.1september.ru

Учебно-методический журнал для учителей информатики
Основан в 1995 г.
Выходит один раз в месяц

РЕДАКЦИЯ:
гл. редактор С.Л. Островский
редакторы
Е.В. Андреева,
Д.М. Златопольский
(редактор вкладки “В мир информатики”)
Дизайн макета И.Е. Лукьянов
верстка Н.И. Пронская
корректор Е.Л. Володина
секретарь Н.П. Медведева
Фото: фотобанк Shutterstock
Журнал распространяется по подписке
Цена свободная
Тираж 27 000 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

ИЗДАТЕЛЬСКИЙ ДОМ “ПЕРВОЕ СЕНТЯБРЯ”
Главный редактор:
Артем Соловейчик
(генеральный директор)
Коммерческая деятельность:
Константин Шмарковский
(финансовый директор)
Развитие, IT и координация проектов:
Сергей Островский
(исполнительный директор)
Реклама, конференции и техническое обеспечение Издательского дома:
Павел Кузнецов
Производство:
Станислав Савельев
Административно-хозяйственное обеспечение:
Андрей Ушков
Педагогический университет:
Валерия Арсланьян (ректор)

ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА “ПЕРВОЕ СЕНТЯБРЯ”
Английский язык – А.Громушкина
Библиотека в школе – О.Громова
Биология – Н.Иванова
География – и.о. А.Митрофанов
Дошкольное образование – Д.Тюттерин
Здоровье детей – Н.Сёмина
Информатика – С.Островский
Искусство – О.Волкова
История – А.Савельев
Классное руководство и воспитание школьников – М.Битянова
Литература – С.Волков
Математика – Л.Рослова
Начальная школа – М.Соловейчик
Немецкий язык – М.Бузоева
ОБЖ – А.Митрофанов
Русский язык – Л.Гончар
Спорт в школе – О.Леонтьева
Технология – А.Митрофанов
Управление школой – Е.Рачевский
Физика – Н.Козлова
Французский язык – Г.Чесновицкая
Химия – О.Блохина
Школа для родителей – Л.Печатникова
Школьный психолог – М.Чибисова

УЧРЕДИТЕЛЬ:
ООО “ЧИСТЫЕ ПРУДЫ”
Зарегистрировано ПИ № ФС77-44341 от 22.03.2011
в Министерстве РФ по делам печати
Подписано в печать:
по графику 13.08.2014,
фактически 13.08.2014
Заказ №
Отпечатано в ОАО “Первая Образцовая типография”
Филиал “Чеховский Печатный Двор”
ул. Полиграфистов, д. 1,
Московская область,
г. Чехов, 142300
Сайт: www.chpd.ru
E-mail: sales@chpk.ru
Факс: 8 (495) 988-63-76

АДРЕС ИЗДАТЕЛЯ:
ул. Киевская, д. 24,
Москва, 121165
Тел./факс: (499) 249-31-38

Отдел рекламы:
(499) 249-98-70
http://1september.ru

ИЗДАТЕЛЬСКАЯ ПОДПИСКА:
Телефон: (499) 249-47-58
E-mail: podpiska@1september.ru

Четверть века в офисе

Осенью 1989 года начались активные продажи самой первой версии пакета программ Microsoft Office. Пакета, который, даже несмотря на появление и все более широкое распространение свободных (а значит, бесплатных) его аналогов типа OpenOffice.org, был и до сих пор остается самым популярным программным обеспечением на любых компьютерах — не только офисных (для которых он, собственно, и предназначался изначально), но и домашних.

"Предтечами" офисного пакета были отдельные, разрозненные программы. В 1983 году был создан первый текстовый редактор для ОС MS-DOS — Multi-Tool Word. А еще раньше, 11 октября 1979 года, фирма Apple подарила пользователям одноименных компьютеров первый редактор электронных таблиц VisiCalc.

Таким образом, первооснова для создания пакета Microsoft Office уже была. Главное же, что сделали программисты компании Microsoft, — это создание комплекса программ, нацеленных на решение различных задач, стоящих перед обычным пользователем (работа с текстом, с числовой информацией, создание презентаций и т.д.) и способных взаимодействовать между собой с возможностью использовать объекты данных, созданные в одной программе пакета, для подготовки документа в другой программе.

Это и определило такую успешную дальнейшую судьбу Microsoft Office. Уже в первой версии 1989 года в комплекте пакета были текстовый редактор Microsoft Word, приложение для работы с таблицами Microsoft Excel и редактор презентаций Microsoft PowerPoint, а также почтовый сервис Microsoft Mail. Позже, уже в 1994 году в версии MS Office 4, к ним добавилась система управления базами данных Access, а в версии Office 97 появился свой графический редактор — Photo Editor и настольная издательская система Microsoft Publisher. В той же версии 97 пакет Microsoft Office был переориентирован на web-технологии (реально, впрочем, функция сохранения документов в формате HTML была замечена еще в версии 95), а начиная с версии 2000 в его комплекте появился визуальный редактор web-страниц и сайтов FrontPage. А в одной из самых современных версий — Microsoft Office 365, появившейся 28 июня 2011 года, применены уже новейшие "облачные" технологии, когда программное обеспечение фактически представляет собой услуги, предоставляемые через Web и доступные (как и обрабатываемые с их помощью данные) в любом месте и в любое время, когда и где только у пользователя есть возможность выхода в сеть Интернет.

Что касается дальнейшего развития пакета Office, пока Microsoft не спешит раскрыть свои планы. Но можно ожидать, что будущие версии Microsoft Office продолжат ориентацию на "облачные" технологии. В числе прочего это позволит предоставить возможности работы с мощными приложениями для обработки данных пользователям любых компьютеров, в том числе сравнительно слабых планшетных. И "первая ласточка" уже есть — в марте 2014 года на базе Office 365 создана версия Microsoft Office for iPad для планшетов Apple iPad (ОС iOS). Что будет дальше? Как говорится, "поживем — увидим"…

Дмитрий Усенков

Азбука 3: роботландский зоопарк исполнителей

А.А. Дуванов,
г. Переславль-Залесский,
kurs@robotland.pereslavl.ru
Н.Д. Шумилина,
г. Тверь,
nshumilina@yandex.ru

Когда ребенок лепит куличики из песка, он делает это для собственного удовольствия, но больше в подарок папе или маме, ожидая в ответ искренних восторгов и, может быть, даже слез радости!

Авторам “Азбуки Роботландии” — курса информатики для начальной школы — ровно также не терпится поделиться с читателями “Информатики” очередным куличиком под номером 3, ибо радость от того, что получается, затмевает их природную скромность!

Посмотреть демоверсии учебника и методички можно в электронных материалах к этому номеру, а также со страницы сайта *www.robotlandia.ru/abc.htm*.

Кратко об “Азбуке Роботландии”

Об этом проекте читатели “Информатики”, конечно, наслышаны. Поэтому совсем кратко о курсе в целом, и быстрее к третьей части, над которой мы сейчас работаем!

“Азбука Роботландии” — четырехгодичный курс информатики для начальной школы, отвечающий, с одной стороны, современному уровню развития информатики (как по интерфейсу самого УМК, так и по его содержанию), а с другой, работающий на опережение за счет опоры на фундаментальные основы информатики и развитие алгоритмического мышления учащихся.

В “Азбуке” запланировано четыре ключевых раздела, соответствующих четырем годам обучения:

- 1-й год. **Компьютер** (разработано)
- 2-й год. **Информация** (разработано)
- 3-й год. **Алгоритмы** (разрабатывается)
- 4-й год. **Интернет**

Таким образом, “Азбука Роботландии” позиционируется как курс, закладывающий основы компьютерной, информационной, алгоритмической и коммуникационной грамотности младших школьников.

Мы хотим научить детей конкретным приемам работы с информацией, но самое главное, мы хотим научить детей мыслить алгоритмически, ведь основа информатики — это теория и практика *составления алгоритмов* для обработки информации.

Развитие алгоритмического мышления — наша основная цель, но мы не хотим подменять другие формы мышления алгоритмическим. Мы хотим сформировать алгоритмическое мышление *наряду с другими*.

Алгоритмическое мышление (как мы это понимаем) — способность облечь абстрактную идею в последовательность конкретных шагов, необходимых для ее воплощения на практике.

Алгоритмическое мышление формируется вместе с абстрактным и логическим. Кроме того, мы заботимся о формировании правильного отношения к инструменту, поиску оптимальных алгоритмов работы с ним. Такие рефлекторные мысленные установки образуют то, что, по нашему мнению, составляет понятие информационной компетентности — навык выбора инструмента, адекватного решаемой задаче, и эффективное его использование.

Наконец, “Азбука Роботландии” закладывает в начальной школе реальную основу базовой (по школьной программе) информатики по всем ключевым ее темам. В старших классах изучение информатики станет более осмысленным.

Раз, два и три: “Алгоритмы”

Первая часть курса — “Компьютер” — знакомит учеников с базовым инструментом современной информатики.

Во **второй части** — “Информация” — закладываются общие представления об информации и информационных процессах, подробно рассматриваются **списки**, **таблицы**, **деревья**, осваивается **многострочное редактирование** плоского текста на базе учебного редактора РМ-1 с блоком контроля выполнения заданий и в стандартном (для операционной системы) редакторе плоского текста (Блокнот для ОС Windows).

Термин “*плоский текст*” (англ. *plain text*) относится к такой форме электронного письма, в которой присутствуют только символы самого текста, символы табуляции, символы конца строк и нет разметки, отвечающей за структуру текста (заголовки, абзацы, таблицы, цитаты, списки…) и его визуальное представление (цвет, начертание, размер, шрифт, выравнивание…). Плоский текст принимают поля ввода форм (строчные и многострочные), редакторы плоского текста используют для записи кода на языках программирования, в том числе для создания веб-страниц. Программа Блокнот из набора программ ОС Windows является примером редактора плоского текста. Заметим, что автоматическая подсветка ключевых слов в некоторых “плоских” редакторах выполняется программным путем без использования разметки в самом тексте.

В **части три** в доступной для детей форме излагаются ***основы алгоритмизации и программирования***. Дети будут создавать и выполнять алгоритмы и программы (в том числе ветвящиеся, циклические и рекурсивные). Кроме того, дополнительный слой посвящен ***конструированию рисунка*** в растровом графическом редакторе (на базе бесплатного продукта Paint.NET, *paintnet.ru*). Краткое содержание Азбуки 3 выглядит так:

Тема 1 (уроки 01–08) Исполнители

- Урок 1. Как познакомиться с исполнителем?
- Урок 2. Рисуем на компьютере. Выделение, перенос, копирование
- Урок 3. ***Роботландский зоопарк***
- Урок 4. Рисуем на компьютере. Преобразования рисунка 1
- Урок 5. ***Исполнитель Машинист***
- Урок 6. Рисуем на компьютере. Преобразования рисунка 2
- Урок 7. Исполнители Автомат и Плюсик
- Урок 8. Исполнители. Повторение

Тема 2 (уроки 09–16) Кукарача. Часть 1

Тема 3 (уроки 17–24) Кукарача. Часть 2

Тема 4 (уроки 25–32) Scratch

Тема 5 (уроки 33–34) Алгоритмы. Повторение

Изложение начинается традиционно по-роботландски: вводится понятие ***исполнителя*** (*исполнителем* называется человек, коллектив, животное или техническое устройство, которые понимают и умеют исполнять задаваемые им команды), система команд исполнителя (***СКИ***) и схема знакомства с формальным исполнителем.

Познакомиться с исполнителем — значит научиться им пользоваться. А для этого нужно:

- знать среду исполнителя (обстановку, в которой работает исполнитель)
- и знать его систему команд, то есть СКИ.

При анализе СКИ, кроме алгоритма исполнения *каждой команды*, рассматриваем ситуации, приводящие к сообщению ***“Не могу”*** (семантические ошибки), учитывая, что ***“Не понимаю”*** (синтаксическая ошибка) все исполнители “говорят” в одном и том же случае — когда им задают команду, не входящую в СКИ.

В этой публикации мы проиллюстрируем авторские подходы в теме “Исполнители” на примере двух уроков учебника (уроки 3 и 5).

Роботландский зоопарк (урок 3)

Поход в зоопарк

Однажды Кукарача пригласил своих студентов в Роботландский зоопарк.

— Это современный парк! — пояснил Кукарача. — Все техническое хозяйство в нем управляется при помощи робота.

— А какие задачи решает этот робот? — поинтересовался Лисенок.

— Задач много, сегодня мы познакомимся с самой интересной: задачей Волка, Цапли и Пингвина! Послушайте их историю.

Товарищи *Волк*, *Цапля* и *Пингвин* работают в Роботландском зоопарке. Они принимают посетителей, рассказывают о том, где и как живут их сородичи.

Работают товарищи 23 дня каждый месяц в двухэтажном офисе в собственных передвижных комнатах (товарищи не любят называть их клетками).

Каждый день комнаты должны располагаться по-разному, в соответствии с установленным графиком.

Робот, выполняя команды, может загружать клетки-комнаты в лифт и перемещать комнаты с одного этажа на другой.

Роботом управляет администратор зоопарка, его обязанность — расположить комнаты в нужном порядке, то есть решить задачу Волка, Цапли и Пингвина для каждого из 23 рабочих дней.

Знакомимся с исполнителем

Давайте знакомиться с роботом-исполнителем!

Помните, *что надо знать об исполнителе*, чтобы успешно с ним работать?

Среда исполнителя

Робот (сотрудники называют его **Зоопарк**) управляется пользователем при помощи компьютера. На мониторе мы видим клетки-комнаты с животными, лифт и пульт управления.

Прежде всего нужно выбрать один из 23 рабочих дней (*выбор задачи*), и на табло слева высветится схема расположения клеток в этот день (*что надо получить*).

Затем приступаем к работе с роботом. Пока задача не будет решена, работаем так:

1. Составляем команду (щелчками по ее частям).
2. Запускаем команду (кнопкой **Выполнить**).

Кнопка **Исх** приводит среду в исходное состояние.

Система команд исполнителя

Какие команды входят в СКИ

Команда состоит из четырех частей. Первые три части указывают буквами, какие клетки перемещаются. Возможные пробелы при этом в расчет не берутся. Четвертая часть — стрелка — указывает направление перемещения: вверх или вниз.

Примеры команд	Что означает
П ▢ ▢ ↓	Переместить клетку пингвина вниз.
Ц П ▢ ↓	Переместить клетки цапли и пингвина вниз.
П В Ц ↑	Переместить клетки пингвина, волка и цапли вверх.

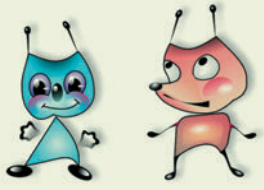
Возникает вопрос: сколько всего команд содержится в СКИ исполнителя **Зоопарк**?

Ответ на этот сложный вопрос попросим дать известного роботландского математика Плюсика.

— Вопрос не очень сложный! Число команд в СКИ легко подсчитать.

Обозначим через Б любую из трех возможных букв: В, Ц, П. Через ↕ будем обозначать любую из стрелок: вверх или вниз (см. табл. на с. 7).

— Получается в СКИ исполнителя **Зоопарк**: 6 + 12 + 12 = **30** команд.



Трям и Прям грустно вздохнули. Они не поняли заумных объяснений Плюсика. Кукарача решил подбодрить их.

Вид команды	Пояснение	Сколько таких команд
Б ▲▼	В команде только одна буква (или В, или Ц, или П). Получается три варианта со стрелкой вверх и три варианта со стрелкой вниз.	6
Б Б ▲▼	Две буквы. Первая — любая из трех, вторая — любая из двух (одна занята первой буквой). Получается 3 × 2 = 6 вариантов, и каждый из них может быть со стрелкой вверх или со стрелкой вниз.	12
Б Б Б ▲▼	Три буквы. Первая — любая из трех, вторая — любая из двух, для третьей остается только одно значение. Получается 3 × 2 × 1 = 6 вариантов, и каждый из них может быть со стрелкой вверх или со стрелкой вниз.	12

— Не расстраивайтесь, если пока не все поняли. Но будет полезно, если вы выпишете все однобуквенные команды (их 6), все двухбуквенные (их 12) и все трехбуквенные (их тоже 12). И тогда вы сами увидите, что в СКИ исполнителя 30 команд.

Как команды передаются исполнителю

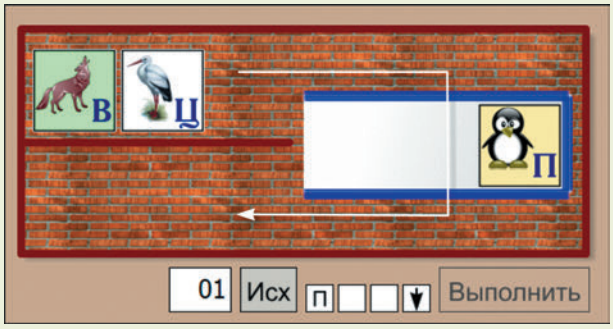
Команда для исполнителя формируется при помощи мышиных щелчков по составляющим ее частям:

— А почему кнопки **Исх** и **Выполнить** такие бледные?

— Приглушенность элемента означает — он неактивен! В самом деле, команда еще не набрана, выполнять нечего, поэтому кнопка **Выполнить** пока не работает. Кнопка **Исх** тоже приглушена, ведь в среде еще нет изменений, и возвращать в исходное состояние ее не надо.

Выберем букву, обозначающую клетку, — кнопка **Выполнить** становится активной:

Нажимаем **Выполнить**, команда передается роботу, и он приступает к работе:



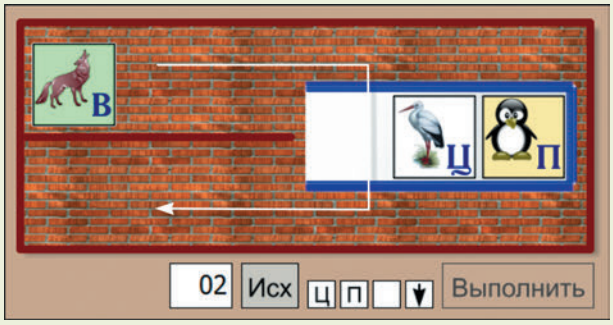
Во время работы кнопка **Выполнить** снова неактивна, как неактивен и список выбора задачи, зато кнопка **Исх** приходит в рабочее состояние. Щелчок по ней вернет среду в исходное состояние.

Как исполнитель выполняет команды

— Не буду отдельно рассказывать, как работает каждая из тридцати команд, назову лишь общие правила, по которым работает исполнитель.

Исполнитель выполняет команду так.

1. Определяет (по стрелке в команде), откуда и куда нужно перемещать клетки.
2. Смотрит, можно ли выполнить такое перемещение.
3. Если перемещение возможно, выполняет его, предварительно перемещая пустой лифт (если это необходимо) на нужный этаж.



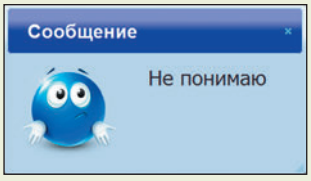
Заметим, что в роботландском зоопарке исполнитель передвигает реальные клетки, а на экране компьютера мы наблюдаем модель его работы.

Сообщения исполнителя

Если пользователь ошибается, исполнитель реагирует на ошибку — выдает сообщение. Сообщение выдается и тогда, когда задача решена успешно.

Не понимаю

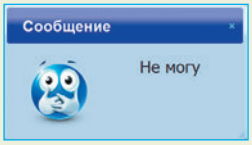
Такое сообщение исполнитель выдает, когда набранная команда не входит в его СКИ.



Пример команды, не входящей в СКИ:

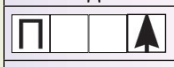
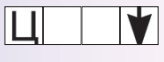
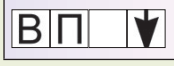
Не могу

Так всегда реагируют роботы, когда не могут по каким-либо причинам выполнить команду из своей СКИ.



Пусть, например, среда находится в начальном состоянии:

Следующие команды вызовут появление сообщения **`Не могу`**:

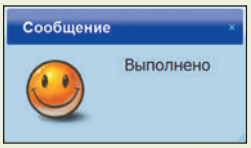
Команда	Причина сообщения **`Не могу`**
	Внизу нет клетки Пингвина.
	Клетку Цапли невозможно загрузить в лифт — мешает клетка Пингвина.
	Клетку Волка невозможно загрузить в лифт — мешает клетка Цапли.

Видим, что в лифт можно перемещать только последние клетки и только в том порядке, в котором они стояли на этаже.

Соответственно из лифта на этаж клетки передвигаются в том же порядке, в котором стояли в лифте.

Выполнено

Такое сообщение исполнитель выдает, если комнаты выстроились на этажах в правильном порядке (в соответствии с условием задачи):



Решаем задачи

Начинаем с простого

Задачи 1–3

— Ну вот, мы основательно познакомились с исполнителем и теперь готовы решать его задачи!

Первые три очень просты, решаются одной командой.

Задача	Дано	Надо	Решение
01	**`вцп`** `---`	**`вц`** `---` **`п`**	1. П↓
02	**`вцп`** `---`	**`в`** `---` **`цп`**	1. ЦП↓
03	**`вцп`** `---`	`---` **`вцп`**	1. ВЦП↓

Попробуйте самостоятельно решить эти задачи в среде исполнителя.

Продолжаем решать задачи

Задачи 4 и 5

Посмотрите на задачу 4. Одной командой ее не решить:

Задача	Дано	Надо	Решение
04	**`вцп`** `---`	**`вп`** `---` **`ц`**	?

Нужно переместить клетку Цапли вниз, но команда Ц↓ вызовет сообщение **`Не могу`**.

Чтобы клетка Цапли оказалась внизу, придется переместить ее вместе с клеткой Пингвина. А потом вернуть клетку Пингвина на второй этаж:

Задача	Дано	Надо	Решение
04	**`вцп`** `---`	**`вп`** `---` **`ц`**	1. ЦП↓
			2. П↑

Решите задачу 4 и следующую задачу 5 в среде **`Зоопарка`** самостоятельно.

Ищем общие подходы

В исходном положении клетки всегда на втором этаже. Значит, первой командой нужно что-то переместить вниз.

Что же перемещать?

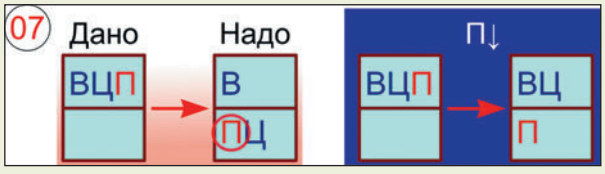
Рассмотрим два случая, ориентируясь на конечный результат решения задачи:

- На первом этаже должны быть клетки.
- На первом этаже не должно быть клеток.

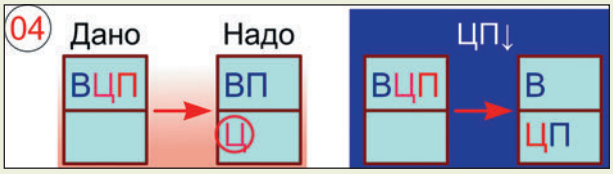
На первом этаже должны быть клетки

Смотрим, с какой клетки начинается результат на первом этаже, и перемещаем вниз группу клеток, начиная с нее. Тогда эта клетка окажется внизу на своем месте.

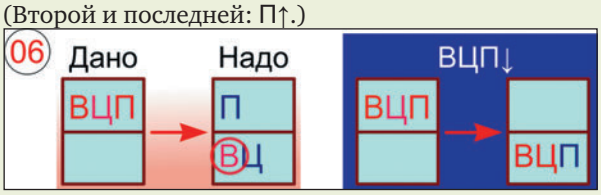
Проиллюстрирую это рассуждение на примерах.



В задаче 7 нижняя цепочка комнат должна начинаться с П. Значит, первой будет команда П↓. (Второй и последней: Ц↓.)



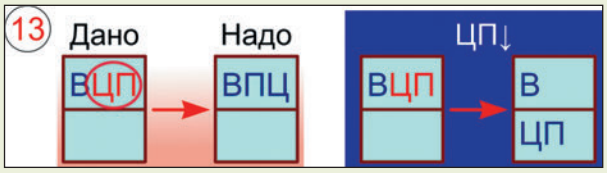
В задаче 4 нижняя цепочка комнат должна начинаться с Ц. Значит, первой будет команда ЦП↓. (Второй и последней: П↑.)



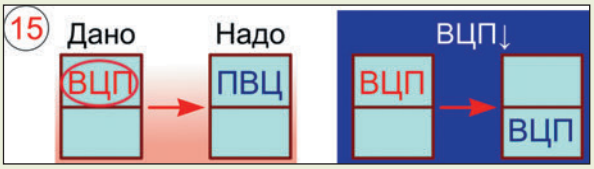
В задаче 6 нижняя цепочка комнат должна начинаться с В. Значит, первой будет команда ВЦП↓. (Второй и последней: П↑.)

На первом этаже не должно быть клеток

Смотрим **Дано** слева направо и пропускаем клетки, стоящие на своих местах. Первую неверно стоящую клетку вместе с последующими перемещаем вниз.



В задаче 13 первая неверно стоящая клетка — Ц. Все клетки, начиная с нее, перемещаем вниз командой 1. ЦП↓. Завершаем решение двумя командами: 2. П↑ 3. Ц↑



В задаче 15 цепочка неверных клеток начинается с самой первой клетки В. Значит, все клетки перемещаем вниз командой 1. ВЦП↓. Завершаем решение двумя командами: 2. П↑ 3. ВЦ↑

Заметим, что, сравнивая **Дано** с **Надо** для первого хода, достаточно проверить только одну первую клетку слева.

Почему достаточно проверить только первую клетку? Если первая клетка стоит неверно, то неважно, как стоят остальные — перемещать надо все. Если первая стоит верно, то вторая не может стоять верно, иначе и третья стояла бы на своем месте и никаких перестановок делать вообще не пришлось бы.

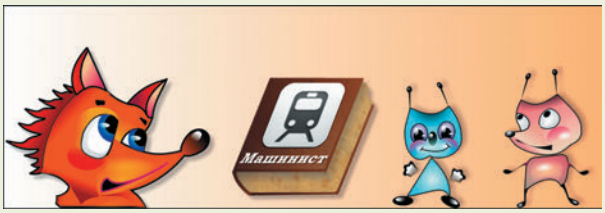
Если первая клетка на своем месте, вниз перемещаем группу из двух последующих клеток (см. задачу 13). Если первая клетка не на своем месте, вниз перемещаем все три клетки (см. задачу 15).

Решите задачи исполнителя согласно заданию учителя.

Исполнитель Машинист (урок 5)

Лисенок читает о паровозах

Когда Трям и Прям пришли к Лисенку, то увидели, что тот увлеченно читает книгу с интригующим названием “Машинист”.



— Из этой книги, — рассказал Лисенок, — я узнал, что первый американский паровоз, построенный в 1830 году, назывался “Мальчик-с-пальчик” — такой он был маленький!

YANGCHAO / Shutterstock.com

Инженер Питер Купер, который построил этот паровоз, устроил рекламную гонку “Пар против лошади”.

И что вы думаете? Лошадь обогнала этот паровоз!

Jim Pruitt / Shutterstock.com

В России первый паровоз (или, как его тогда называли, сухопутный пароход) был построен в 1834 году Ефимом и Мироном (отец и сын) Черепановыми.

Паровоз возил поезд весом более трех тонн со скоростью 16 км/ч.

На очередном занятии в Роботландии Лисенок показал свою книгу Кукараче.

— Замечательная книга, — оценил Кукарача, — история возникновения железных дорог в ней рассказана интересно. Но главная ее часть — описание исполнителя **Машинист**, и сегодня мы будем с ним знакомиться.

Машинист локомотива

Кукарача повел студентов на роботландскую железнодорожную станцию.

Paolo Bona / Shutterstock.com

— *Машинистом* называют человека, который управляет локомотивом, — пояснил Кукарача. — Локомотив цепляют к поезду, и он тянет за собой вагоны по рельсам.

Слово *локомотив* — “вешалка”. Этим словом обозначают и паровоз, и тепловоз, и электровоз.

Вспомним, что в Роботландии *“вешалкой”* называют слово, которым можно заменять другие слова.

Например, слово *дерево* — “вешалка”! Им можно заменить, или, как любят говорить робята, на него можно “повесить” ель, сосну, березу, тополь, дуб, ветлу — любое *конкретное* дерево.

Сортировочная станция

Роботландский **Машинист** — это робот. Он работает на сортировочной станции.

Сортировочная станция — это специальное место, в котором из вагонов собирают поезда, *сортируют* вагоны: вот и получается — “сортировочная станция”.

На картинке выше показана одна из таких станций.

Пульт управления

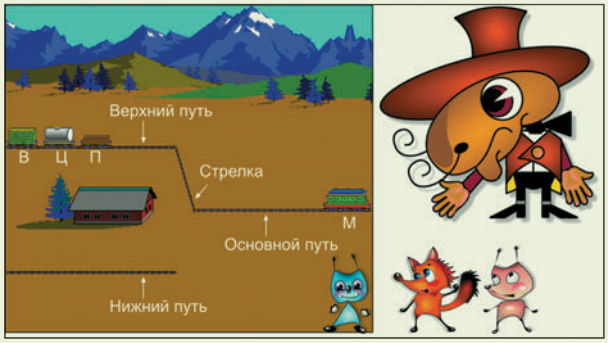
На роботландской сортировочной станции локомотив работает без человека.

Локомотивом управляет робот, которого мы и называем **Машинистом**.

Наш робот встроен в локомотив, связь с ним выполняется по радио с пульта управления.

Знакомимся с исполнителем Машинист

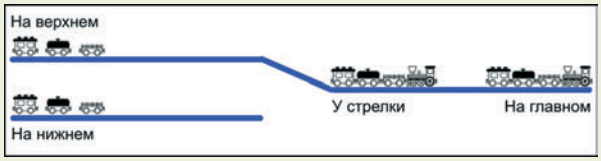
Среда исполнителя



Среда **Машиниста** включает:

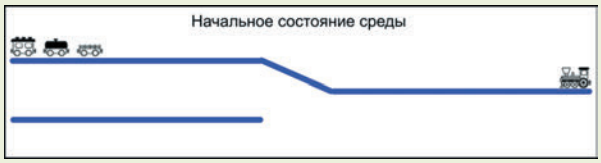
- Три железнодорожных пути (верхний, нижний и основной).
- Стрелку, переключающую основной путь на верхний или нижний.
- Три вагона: крытый вагон (В), цистерна (Ц) и платформа (П).
- Локомотив с **Машинистом** (М).

Локомотив может перемещать вагоны по всем путям, но останавливаться состав может только в четырех местах, обозначенных на рисунке:

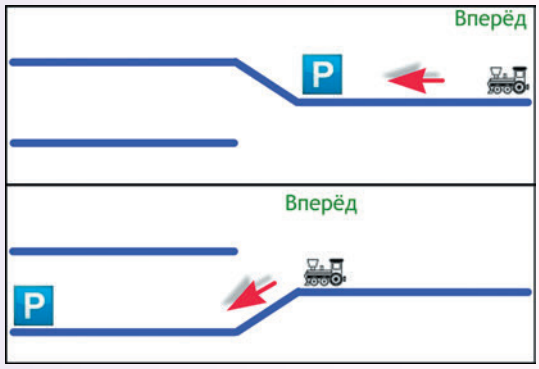
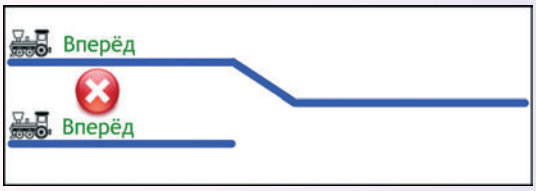
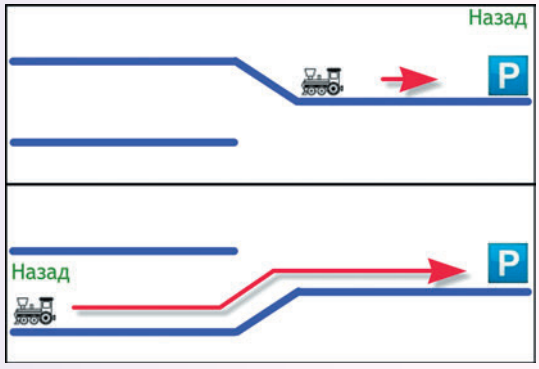
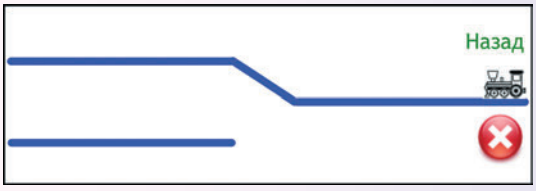
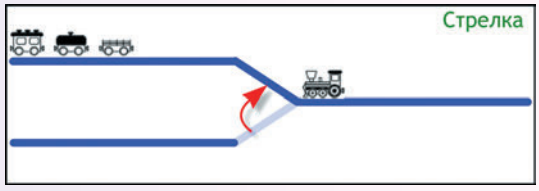
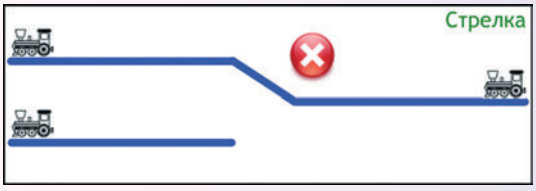
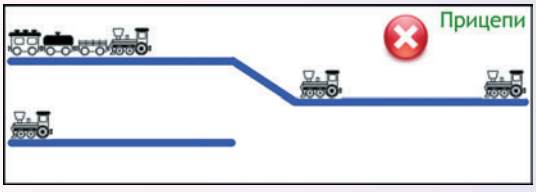
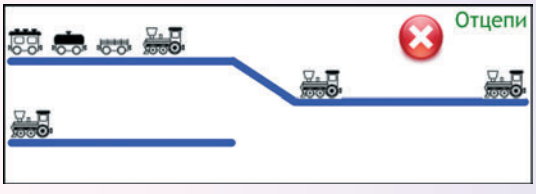


И только в тупиках на верхнем и нижнем путях локомотив может отцеплять и прицеплять вагоны и, следовательно, оставлять их.

В начальный момент вагоны в порядке ВЦП расположены в тупике на верхнем пути, локомотив — в начале главного пути, стрелка переключена на верхний путь:



Система команд исполнителя

Команда	Как выполняется	Когда Не могу
Вперед	 Если локомотив (один или с вагонами) стоит в начале основного пути, он идет до стрелки. Если стоит у стрелки, идет в тупик по той ветке, путь на которую открыт стрелкой.	 Локомотив (один или с вагонами) в тупиках на верхней или нижней ветке.
Назад	 Локомотив (один или с вагонами) идет в начало основного пути.	 Локомотив расположен в начале основного пути.
Стрелка	 Стрелка переключает основной путь на нижний, если связь была с верхним, и наоборот.	 Локомотив не у стрелки.
Прицепи	 К локомотиву (или к составу, который везет локомотив) прицепляется один вагон.	 Локомотив не у вагонов, прицеплены все вагоны, локомотив в начале главного пути или у стрелки.
Отцепи	 От локомотива (или от состава, который везет локомотив) отцепляется один последний вагон.	 Локомотив не у вагонов, отцеплены все вагоны, локомотив в начале главного пути или у стрелки.

Как команды передаются исполнителю

— Исполнители могут работать в двух режимах:

- *командном* и
- *программном*.

В *командном* режиме исполнитель получает команду, выполняет ее и ждет следующую. В таком режиме работает исполнитель **Зоопарк**.

Исполнитель **Машинист** работает в *программном режиме*. Он получает не одну команду, а *программу* — список команд, затем приступает к выполнению команд, входящих в этот список.

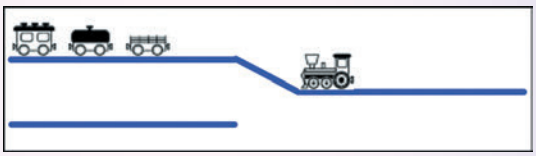
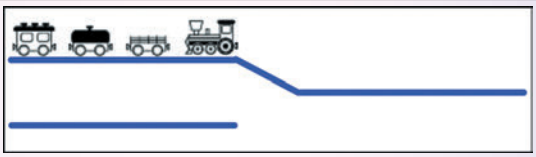
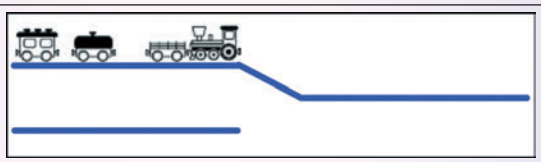
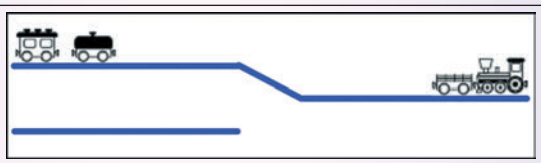
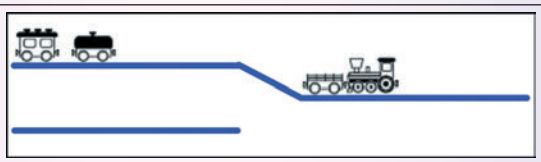
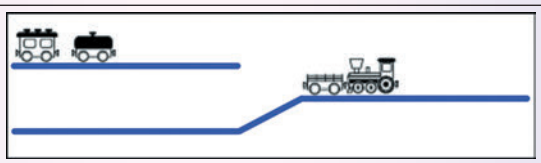
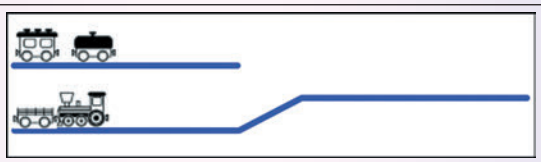
Первые опыты. Зачем нужны программы?

Первую задачу **Машиниста** решим, моделируя командный режим.

Задача 01. Переставить платформу на нижний путь.

Условие задачи кратко будем обозначать схемой (положение стрелки в конечном состоянии неважно):

Решение

№	Команда	Результат выполнения
01	**Вперед**	
02	**Вперед**	
03	**Прицепи**	
04	**Назад**	
05	**Вперед**	
06	**Стрелка**	
07	**Вперед**	

08	**Отцепи**	
09	**Назад**	

Таким образом, решением задачи 1 для **Машиниста** будет следующая программа:

1. **Вперед**
2. **Вперед**
3. **Прицепи**
4. **Назад**
5. **Вперед**
6. **Стрелка**
7. **Вперед**
8. **Отцепи**
9. **Назад**

— Как вы думаете, в каком режиме легче работать с исполнителем, в командном или программном? — спросил своих учеников Кукарача.

— В командном режиме работать легче, — ответил Лисенок. Трям и Прям поддержали его. — Мы видим, как меняется среда после каждой команды, и это помогает выбрать следующую команду.

В программном режиме нужно мысленно следить за изменением среды после выполнения каждой команды, а это так трудно! Зачем он только нужен, этот программный режим?

— Программный режим существенно упрощает работу *оператора* — человека, который сидит за пультом управления сортировочной станции, — пояснил Кукарача.

В командном режиме оператор работал бы так: посылал бы исполнителю команду, ожидал бы ее выполнения, затем посылал бы новую команду. И так много раз, пока задача не будет решена.

В программном режиме оператор посылает исполнителю не отдельные команды, а заранее созданную программу, и исполнитель выполняет ее самостоятельно. Оператор в это время может спокойно попить чаю с сухариками!

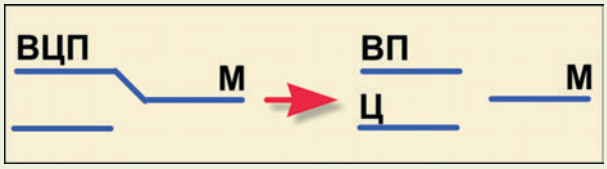
На самом деле оператор работает на сортировочной станции не с одним, а с несколькими исполнителями, и чаевничать ему некогда.

— Получается, оператору легко, а программисту — трудно!

— У каждого своя работа, и в каждой работе бывает и легко, и трудно, и интересно! Мы будем учиться программировать.

Продолжаем программировать

Задача 04



Решение

— Сначала, конечно, запишем две команды **Вперед**, а что делать дальше, непонятно…

— Давайте попробуем думать не командами **Машиниста**, а перестановками вагонов. Составим не программу для **Машиниста**, а алгоритм для самих себя.

— Внизу должна стоять цистерна, значит, сначала нужно переместить вниз цистерну и платформу (одну цистерну мы взять не можем), а затем вернуть платформу наверх. В алгоритме получается два шага:

1. Переставь ЦП на нижнюю ветку.
2. Переставь П на верхнюю ветку.

Когда алгоритм написан, команды для каждого шага написать, конечно, легко!

— Я бы переписал алгоритм короче:

1. ЦП↓
2. П↑

Маленький Трям делает открытие

Маленький Трям, большой любитель конфет “Коровка”, шалун и непоседа, сегодня стал героем дня!

— Ребята, — заявил он с большим подъемом, — задачи **Маши-**

ниста поразительно похожи на задачи роботландского **Зоопарка**!

В зоопарке перемещают клетки В, Ц, П, а на сортировочной станции — вагоны В, Ц, П. Разница только в том, что в зоопарке работает лифт, а на станции — локомотив!

Задачи формулируются одинаково, и правила перемещения одни и те же: нельзя, например, в исходном состоянии вытащить один объект Ц, можно только вместе с объектом П. При этом не важно, что это за объекты, клетки или вагоны!

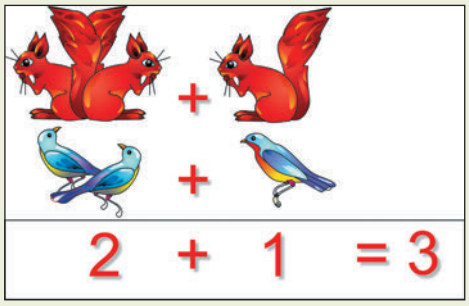
Кукарача пришел в неописуемый восторг от открытия Тряма! Он так обрадовался и возгордился Трямом, что чуть не потерял свою шляпу.

— Смотрите, какой важный вывод сделал Трям, — пояснил свою радость Кукарача. — Тряма не смутили внешние отличия сред **Зоопарка** и **Машиниста**.

Двухэтажное здание, клетки и лифт в зоопарке. Железнодорожные ветки, вагоны и локомотив на станции. Все такое разное!

Но Трям увидел в разных средах одинаковые условия задач. А это означает, что и решения будут одинаковые!

Так, математику все равно, что считать, белок или птичек. За разными внешностями он видит одну и ту же математическую сущность!



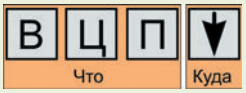
Машинист+

Как только Кукарача произнес слово “математик”, неизвестно откуда, будто из воздуха, возник роботландский Плюсик и присоединился к компании.

А Кукарача углубился в историю Роботландии.

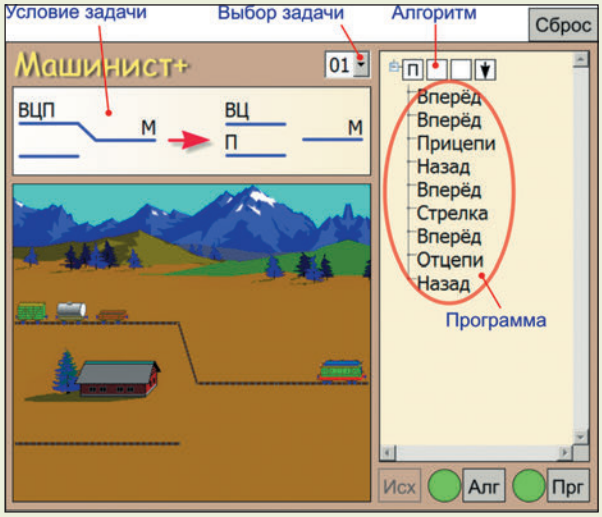
— Изначально исполнитель **Машинист**, — рассказывал Кукарача, — работал только в командном режиме. Потом исполнителя переделали, и теперь он выполняет программы.

Кроме того, для удобства программистов был создан еще один исполнитель — его назвали **Алгоритмик**, — он умеет выполнять *алгоритмы* **Машиниста**. СКИ **Алгоритмика** совпадает со СКИ исполнителя **Зоопарк**. То есть команды **Алгоритмика** имеют вид:



Обновленный исполнитель стали называть **Машинист+**. Фактически это два исполнителя. Один (**Алгоритмик**) выполняет алгоритмы на экране, другой — **Машинист** — программы на сортировочной станции (на экране мы тоже видим его работу).

Вот как выглядит среда нового исполнителя:



Справа от изображения среды исполнителя располагается редактор, в котором можно собирать алгоритм. Каждый шаг алгоритма — вершина дерева, потомки вершины — команды программы, которая выполняет этот шаг алгоритма.

Работают с исполнителем так. Составляют алгоритм и запускают его кнопкой **Алг** на панели управления. Если алгоритм работает правильно, ***для каждого шага задают команды***, выполняющие этот шаг. Программу запускают кнопкой **Прг**.

Вы увидите, как удобно работать с обновленным **Машинистом**! Краткое, но полное описание интерфейса среды приводится в практикуме.

Слово берет Плюсик

— Прошлый раз мы подсчитали число команд в СКИ исполнителя **Зоопарк**. Как вы помните, таких команд оказалось 30.

Понятно, что в СКИ исполнителя, выполняющего алгоритм перестановки вагонов, команд тоже 30.

Сегодня мне хочется разобрать вопрос, связанный с количеством возможных задач этих исполнителей.

Вы помните, что в среде **Зоопарка** запланировано 23 задачи. Ровно столько задач предусмотрено и в среде исполнителя **Машинист+**.

Почему именно столько? Можно ли предложить новые задачи?

Попробуем разобраться.

Давайте будем записывать состояние среды тремя буквами и одной косой чертой. Буквы будут обозначать вагоны (или клетки), а косая черта — отделять верхний путь от нижнего (второй этаж от первого).

В принятых обозначениях запись ПЦ/В, например, будет обозначать платформу и цистерну на верхнем пути, а крытый вагон — на нижнем (клетки Пингвина и Цапли на втором этаже, а клетку Волка — на первом).

Количество задач (множество **Надо**) определяется числом возможных состояний среды, уменьшенным на 1 (так как одно состояние соответствует **Дано**).

Состояние среды задается набором из трех букв и положением косой черты.

Подсчитаем число возможных перестановок из трех букв.

На первом месте мы можем поставить любую из трех букв.

На втором — любую из двух оставшихся.

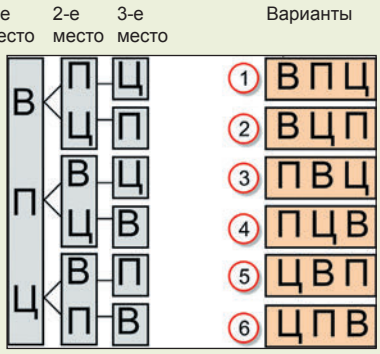
А букву для третьего места выбирать не приходится — она остается одна, когда две первые выбраны.

Видим, что для каждого из трех вариантов на первом месте получается два варианта на втором месте и один вариант на третьем.

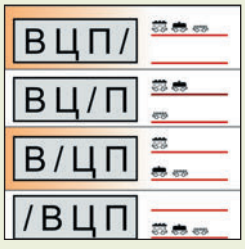
Всего: $3 \times 2 \times 1 = 6$ вариантов.

Таким образом, число перестановок из трех букв равно 6.

Вот все эти шесть вариантов:



Для каждого из шести вариантов существуют четыре варианта расположения косой черты. Ниже это продемонстрировано для варианта ВЦП:



Таким образом, получается $6 \times 4 = 24$ возможных состояний среды. Одно состояние ВЦП/ зарезервировано для **Дано**, а оставшиеся 23 варианта образуют множество вариантов **Надо**.

— Получается, и в среде **Зоопарка**, и в среде **Машиниста+** можно сформулировать только 23 задачи!

— Именно так! И нет смысла искать новые задачи — их просто нет! Мы доказали это математически!

Решаем задачи

— После задач **Зоопарка** задачи **Машиниста+** мои студенты решат играючи, — заявил Кукарача. — Ведь главное — составить правильный алгоритм, а расписать команды для локомотива — это *тривиально*! (Кукарача употребил слово, подслушанное у Плюсика! Если кто-то не знает, что оно означает, посмотрите в Интернете.)

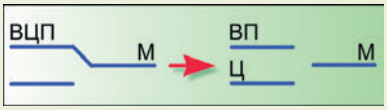
— Садимся за пульт управления и приступаем к решению!

Зоопарк и Машинист (поясняющий методический комментарий)

Исполнитель **Машинист** традиционен для начального курса информатики, а исполнитель **Зоопарк** — новинка, специально придуманная для роботландской “Азбуки”.

В **Машинисте** для “Азбуки” была реализована старая идея, возникшая еще в 2006 году. Суть ее в создании нового исполнителя **Машинист+**. Работа с ним предполагалась в два этапа: на первом — составляется алгоритм перестановки вагонов, на втором — программа для робота **Машинист**. Эта идея была доработана до конкретной реализации, о которой читатель может сложить собственное мнение по тексту этой статьи и после работы с исполнителем, который можно найти в электронных материалах к этому номеру. В этом послесловии мы поясним методические посылы, на которых базируются реализации исполнителей **Зоопарк** и **Машинист+** (**Алгоритмик** и **Машинист**).

В качестве иллюстрирующего примера возьмем задачу 04 из среды исполнителя **Машинист+**. Условие задачи отображается в виде схемы, на которой показано начальное состояние среды и конечное, в которую ее надо перевести построенной программой:



Программа, решающая задачу, содержит 15 команд из СКИ **Машиниста**. Эти команды описывают “технические” действия робота: движения вперед, назад, перевод стрелки, сцепку вагонов… Программирование **Машиниста** напоминает программирование компьютера в машинных кодах. В этих мелких подробностях общая идея решения практически исчезает, программисту приходится удерживать ее в голове, соотнося ее с каждой записываемой командой, а это непросто, особенно программисту маленькому и начинающему. Более того, это и неправильно с методологической точки зрения: прежде чем записывать какие-то

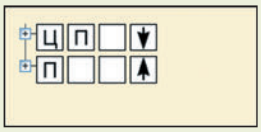
команды **Машиниста**, необходимо не просто “поймать за хвост” идею решения, но и зафиксировать ее, записать в виде формальных шагов алгоритма. Когда алгоритм построен, можно приступать к реализации каждого его шага в виде “машинного” кода.

Алгоритм решения задачи 04 содержит два шага:

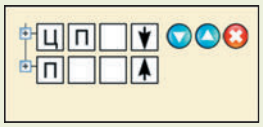
1. Переместить ЦП на нижний путь.
2. Переместить П на верхний путь.

В алгоритме не говорится о том, как будет проходить перемещение вагонов, мы не отвлекаемся на технические детали, а сосредотачиваемся на логической (алгоритмической) сути решения.

В среде исполнителя **Машинист+** этот алгоритм собирается из кубиков, содержимое которых меняется щелчками на них:



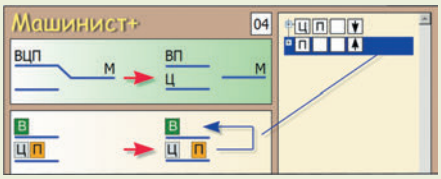
Когда курсор появляется над шагом алгоритма, справа возникают три интерфейсные кнопки, позволяющие вставить новый шаг ниже или выше текущего или удалить текущий шаг алгоритма:



Алгоритм записан. Он содержит только два шага. Два понятных шага! Что же дальше? Алгоритм можно выполнить! Пульт управления выглядит так:

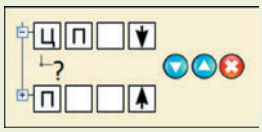
Информатор рядом с кнопкой **Алг** горит зеленым. Это означает, что алгоритм записан правильно.

Нажимаем кнопку **Алг** и наблюдаем выполнение алгоритма, но не в среде **Машиниста**, а в среде надстроенного над ним исполнителя **Алгоритмик** (выполняемый шаг алгоритма подсвечивается):

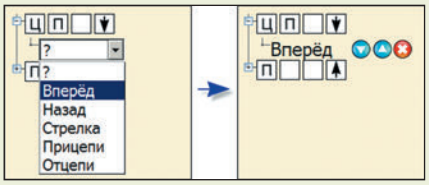


Заметим, что на пульте управления рядом с кнопкой запуска программы **Прг** информатор светится синим. Это означает, что программа для **Машиниста** еще не написана. Как же ее написать?

Видим, что слева каждый шаг алгоритма снабжен маркером со знаком “+” подобно закрытой папке в компьютерном дереве папок. Щелкнем по этому маркеру, “папка” предсказуемо откроется:



Эта “папка” предназначена для кодов, реализующих открытый шаг алгоритма. По маркерам справа видим, что коды можно добавлять и удалять, а щелчок по знаку вопроса позволяет выбирать команду из выпадающего списка:



Теперь нам надо писать не 15 команд, а 8 для первого шага алгоритма и 7 для второго. Кроме того, мы записываем команды, реализующие конкретные перестановки вагонов (ЦП↓ для первого шага и П↑ для второго), что превращает программирование из замысловатых действий почти в рутину (творчество заканчивается составлением алгоритма).

Видим, что задача существенно упрощается, тем более что в любой момент мы можем запустить недописанный код и увидеть его выполнение в среде исполнителя:

Подсветка в коде соответствует выполняемой команде, и она остается, если выполнение заканчивается “аварийно”.

Таким образом, среда нового исполнителя **Машинист+** позволяет проектировать решение задач методом сверху вниз: от логики перестановки вагонов к техническим командам реализации. Учебная среда устроена таким образом, что на каждом этапе работы она самостоятельно контролирует действия ученика и реагирует при помощи системы информаторов (“светофоров”). Информаторы отмечают ошибки синтаксиса, ошибки времени выполнения и качество построенного решения (желтый цвет означает верное, но длинное решение).

А зачем нужен исполнитель **Зоопарк**?

Как мы понимаем (и как неожиданно отмечает в учебнике молчун Трям, любитель конфет “Коровка”), исполнитель **Зоопарк** тождественен исполнителю **Алгоритмик**. Само по себе это очень важное наблюдение, оно наводит на мысль, что, несмотря на внешние различия явлений, сути могут совпадать (и мяч, и камень, и кусок дерева ведут себя одинаково в поле тяготения Земли).

Исполнитель **Зоопарк** позволяет предварительно поработать с алгоритмами **Машиниста** в **командном режиме**, который, как справедливо отмечает в учебнике Лисенок, проще программного — ведь мы видим, как меняется среда после каждой команды, и это помогает выбирать следующую команду. То есть **Зоопарк** был придуман для пропедевтики **Машиниста**. После **Зоопарка** программирование **Машиниста** идет гораздо успешнее. Можно, конечно, было предусмотреть и в **Машинисте** командный режим (наряду с программным), но, согласитесь, с **Зоопарком** получилось назидательно и гораздо веселее!

фестиваль
УЧИТЕЛЬСКАЯ
КНИГА

фестиваль/ярмарка

«УЧИТЕЛЬСКАЯ КНИГА»

- выставка-продажа книг ведущих фирм и издательств
- семинары и встречи с методистами и авторами учебников
- более 1000 наименований книг для учителей

5 ноября

Предметы гуманитарного цикла

| Русский язык | Литература | История
| География | МХК | Музыка
| Изобразительное искусство
| Библиотека в школе

6 ноября

Предметы естественно-научного цикла

| Математика | Информатика
| Физика | Биология | Химия

7 ноября

Начальная школа
Дошкольное образование
Воспитание

| Психология | Физкультура
| Здоровье детей

8 ноября

Иностранные языки

| Английский язык | Немецкий язык
| Французский язык

Регистрация на фестиваль начнется 10 октября 2014 года на сайте

http://bookfair.1september.ru

Выставка-продажа книг работает с 9.30 до 13.30

Все мероприятия фестиваля пройдут с 10.00 до 14.45
в Московском педагогическом государственном университете
по адресу: ул. Малая Пироговская, дом 1, стр. 1
(в 5 минутах ходьбы от станции метро «Фрунзенская»)

вход свободный

Язык Python: избранные алгоритмы*

Обход деревьев

К.Ю. Поляков,
д. т. н.,
Санкт-Петербург

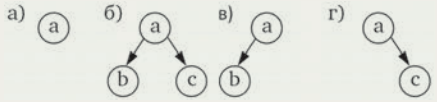
Дерево — одна из самых популярных структур данных в теории алгоритмов. Деревья используются не только для представления иерархически связанных данных, но и как способ хранения информации, обеспечивающий быстрый поиск.

Дерево состоит из узлов и соединяющих их ребер. Каждый узел связан с некоторыми данными, в наших примерах это будет символьная метка. В языке Python нет специальной встроенной структуры данных, которая бы в точности соответствовала дереву, но можно использовать для этой цели вложенные списки.

Для простоты будем рассматривать только *двоичные* деревья (в которых каждый узел имеет не более двух потомков), поскольку они используются чаще всего. Для хранения данных о двоичном дереве в виде списка можно использовать рекурсивное определение:

- пустое дерево — это двоичное дерево;
- двоичное дерево — это узел и два связанных с ним непересекающихся двоичных поддерева.

Получается, что двоичное дерево можно задать списком из трех элементов. Первый элемент — это данные узла (метка), второй — “левый сын”, а третий элемент — “правый сын”, причем каждое из поддеревьев может быть пустым. Например, дерево, состоящее из одного узла с меткой “a” (рисунок *а*):



можно представить в виде `["a",[],[]]`. Если оба поддерева пустые, не будем их записывать вообще, то есть запись сократится до `["a"]`. Дерево с тремя узлами (на рисунке *б*) можно представить как `["a",["b","c"]]`, а деревья на рисунках *в* и *г* могут быть

* Окончание. Начало см. в № 9/2014.

заданы как `["a",["b"]]` и `["a",[[],"c"]]` соответственно. Заметим, что в последнем случае пустой список `[]` на месте левого сына обязателен, потому что иначе узел с меткой “c” будет рассматриваться как левый сын, а не правый (во многих задачах это существенно).

Рассмотрим задачу обхода узлов двоичного дерева. Простейший вариант — это обход в порядке КЛП (“**К**орень — **Л**евое поддерево — **П**равое поддерево”).

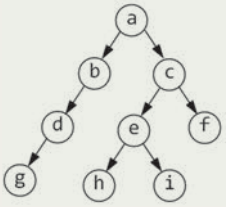
Пусть двоичное дерево задано в виде списка `T`. Как мы договорились, первый элемент этого списка, `T[0]`, — это данные узла (метка), `T[1]` — левое поддерево и `T[2]` — правое поддерево. Для каждого узла нужно выполнить следующие действия:

- обработать (посетить) узел (мы будем просто выводить на экран метку узла);
- обойти все поддеревья, сначала левое, потом правое.

Алгоритм обхода получается рекурсивным:

```
def DFS ( T ):
  if not T: return
  print(T[0], end=" ")
  for sub in T[1:]:
    DFS ( sub )
```

Если функции передано пустое дерево, происходит выход (окончание рекурсии). Если дерево непустое, на экран выводится метка узла и затем в цикле выполняется обход всех поддеревьев через рекурсивные вызовы.



Название функции DFS — это сокращение от английского термина ***d**epth-**f**irst **s**earch* — поиск в глубину. Действительно, при таком обходе мы сначала проходим в глубину дерева по самой левой ветке до ее последнего элемента (листа), а потом возвращаемся назад. Для дерева, показанного на рисунке, результат работы функции DFS — это последовательность

a b d g c e h i f

Существуют и другие варианты обхода дерева, наиболее известный из них называется “поиск в ширину” (англ. *breadth-first search*). Его суть в том, что сначала обрабатывается корень, затем — его “сыновья”, затем — “сыновья сыновей” и т.д. Такой обход удобно организовать с помощью очереди:

```
def BFS ( T ):
  if not T: return
  Q = [T]
  while Q:
    node = Q.pop(0)
    print(node[0], end=" ")
    for sub in node[1:]):
      if sub: Q.append ( sub )
```

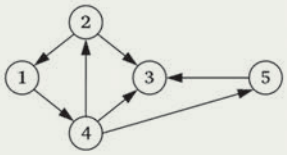
Сначала в очередь записывается корень дерева. Затем в цикле, пока очередь не пуста, берем один элемент с начала очереди, посещаем этот узел (выводим его метку на экран) и записываем в очередь ссылки на все его непустые поддеревья. При этом сначала из очереди будет извлечено левое поддерево, а потом — правое. Обход в ширину дерева, показанного на рисунке, дает результат

a b c d e f g h i

Количество путей в графе

В ряде задач графы удобно описывать с помощью списков смежности, которые определяют для каждого узла множество вершин, в которые можно из него перейти. Такой подход легко реализуется в Python, где есть встроенный тип данных “список”.

Рассмотрим ориентированный граф, связывающий пять вершин:



Для него списки смежности (с учетом направлений ребер) выглядят так:

вершина 1: (4)
вершина 2: (1,3)
вершина 3: ()
вершина 4: (2,3, 5)
вершина 5: (3)

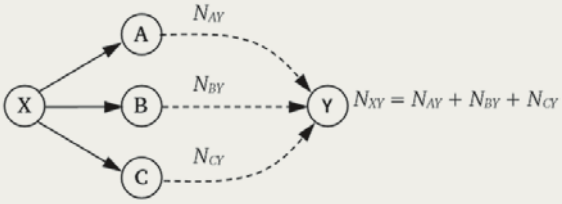
Представим этот граф в виде вложенного списка:

```
Graph = [ [],       # фиктивный элемент
          [4],      # список смежности
                    # для вершины 1
          [1,3],    # для вершины 2
          [],       # для вершины 3
          [2,3,5],  # для вершины 4
          [3] ]     # для вершины 5
```

Для того чтобы использовать нумерацию вершин с единицы, мы добавили фиктивный элемент — пустой список смежности для несуществующей вершины 0.

Используя такое представление, построим функцию **`pathCount`**, которая находит количество путей из одной вершины графа в другую.

Сначала предположим, что в графе нет циклов (замкнутых фрагментов путей). Пусть из исходной вершины `X` можно перейти только в вершины `A`, `B` и `C`. Тогда количество путей N_{XY} из вершины `X` в некоторую вершину `Y` равно сумме количеств путей из вершин `A`, `B` и `C` в вершину `Y`.



Если в графе есть циклы, мы будем искать только те пути, которые не содержат циклов. Для этого при переборе вариантов оставшейся части пути на каждом шаге нужно учитывать только те вершины, которые еще не посещались. Чтобы запомнить эту информацию, мы будем использовать список посещенных вершин.

Таким образом, в функцию **`pathCount`** нужно передать следующие данные (аргументы):

• описание графа в виде списков смежности для каждой вершины, **graph**;

• номера начальной и конечной вершин, **vStart** и **vEnd**;

• список посещенных вершин, **visited**.

Тогда основной цикл функции **pathCount** приобретает вид

```
count = 0
for v in graph[vStart]:
  if not v in visited:
    count += pathCount ( graph, v,
                         vEnd, visited )
```

Функция **pathCount** получилась *рекурсивной*, то есть она вызывает сама себя (в цикле). Поэтому нужно определить условие окончания рекурсии: если начальная и конечная вершины совпадают, то существует только один (пустой) путь. В этом случае можно сразу выйти из функции с помощью оператора **return**:

```
if vStart == vEnd: return 1
```

Приведем полный текст функции

```
def pathCount ( graph, vStart,
                vEnd, visited ):
  if vStart == vEnd: return 1
  visited.append ( vStart )
  count = 0
  for v in graph[vStart]:
    if not v in visited:
      count += pathCount ( graph, v, vEnd,
                           visited )
  visited.pop()
  return count
```

и основную программу, которая находит количество путей из вершины 1 в вершину 3:

```
Graph = [[], [4], [1,3], [], [2,3,5], [3]]
print ( pathCount (Graph, 1, 3, []) )
```

Отметим, что функция pathCount не находит сами маршруты, а только определяет их количество.

Алгоритм Дейкстры

В 1960 году нидерландский ученый Эдсгер Дейкстра предложил алгоритм, позволяющий найти кратчайшие расстояния от одной вершины графа до всех остальных и соответствующие им маршруты. Предполагается, что длины всех ребер (расстояния между вершинами) положительные.

Зададим граф весовой матрицей, в которой элемент с индексами (i,j) хранит вес ребра из вершины i в вершину j (в качестве веса может выступать, например, расстояние между городами или стоимость проезда). Пусть весовая матрица графа называется W и начальная вершина имеет индекс 0.

Алгоритм использует дополнительные массивы: в одном (назовем его R) хранятся кратчайшие (на данный момент) расстояния от исходной вершины до каждой из вершин графа, а во втором (массив P) — предыдущие вершины для каждого оптимального маршрута.

В алгоритме Дейкстры постепенно строится множество вершин, оптимальный путь в которые из начальной вершины уже определен. Сначала это множество содержит только начальную вершину. Очередной шаг алгоритма Дейкстры может быть сформулирован следующим образом:

• выбрать из всех еще не рассмотренных вершин такую вершину v, для которой текущее кратчайшее расстояние R[v] от начальной вершины минимально;

• перебрать в цикле все остальные вершины; для каждой вершины с номером j попытаться улучшить текущую длину маршрута, разрешив проезд через вершину v:

```
R[j] = min(R[j], R[v] + W[v][j])
```

где W[v][j] — длина ребра между вершинами с индексами v и j.

После того как все вершины рассмотрены, в массиве R остаются кратчайшие расстояния из начальной вершины во все остальные.

Сначала записываем в массив R веса всех ребер от исходной вершины (с номером 0) до всех остальных вершин, это будет копия верхней строки матрицы W:

```
R = W[0][:]
```

и строим начальный массив P:

```
P = [-1] + [0] * (N - 1)
```

Условное значение “–1” в элементе P[0] говорит о том, что это начальная вершина. В остальные элементы массива записывается номер начальной вершины (здесь — 0).

Сначала в список rest номеров не рассмотренных вершин войдут все вершины, кроме начальной:

```
rest = list(range(1, N))
```

Чтобы обработать все оставшиеся вершины, нужен цикл, который выполняется N-1 раз:

```
for i in range(N - 1):
  ...
```

В цикле находим вершину v с кратчайшим текущим расстоянием от исходной вершины и удаляем ее из списка невыбранных вершин:

```
T = [(R[j],j) for j in rest]          # 1
minDist, v = min(T)                   # 2
rest.remove(v)                        # 3
```

В строке 1 с помощью генератора списка строится список пар (кортежей) “расстояние – номер вершины” для всех невыбранных вершин. В строке 2 выбирается вершина с минимальным текущим расстоянием (поиск минимума в списке кортежей по умолчанию выполняется по первому элементу кортежа). Затем, в строке 3, эта вершина удаляется из списка не рассмотренных.

Теперь нужно перебрать все оставшиеся вершины и для каждой попытаться улучшить кратчайший путь:

```
for j in rest:
  if R[v] + W[v][j] < R[j]:
    R[j] = R[v] + W[v][j]
    P[j] = v
```

В последней строке в массиве P запоминается новая предыдущая вершина в оптимальном маршруте из исходной вершины в вершину j.

Приведем функцию целиком:

```
def dijkstra ( W ):
  R = W[0][:]
  P = [-1] + [0] * (N - 1)
  rest = list(rest(1, N))
  for i in range(N - 1):
```

```
      T = [(R[j],j) for j in rest]
      minDist, v = min(T)
      rest.remove(v)
      for j in rest:
        if R[v] + W[v][j] < R[j]:
          R[j] = R[v] + W[v][j]
          P[j] = v
  return R, P
```

Теперь для любой вершины `j` длина оптимального маршрута находится в элементе массива `R[j]`. Остается определить сам маршрут. Все данные, необходимые для этого, есть в массиве `P`. Действительно, для любой вершины `j` в элементе массива `P[j]` можно сразу найти номер предыдущей вершины в оптимальном маршруте. Поэтому "раскручивать" маршрут нужно с конца, с той вершины, которая нас интересует:

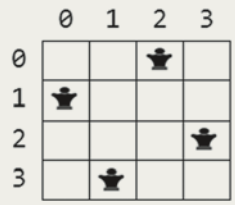
```
i = j
while i >= 0:
  print (i)
  i = P[i]
```

Как только мы пришли в начальную вершину, для которой `P[i] = -1`, цикл заканчивается.

Задача о расстановке ферзей

Одна из популярных задач, с которыми знакомы все программисты, — задача о расстановке восьми ферзей на шахматной доске так, чтоб они не били друг друга. Обычно эта задача рассматривается при изучении темы "перебор с возвратом назад". Мы приведем ее решение на языке Python, использующее вложенные списки.

Будем рассматривать задачу в общем виде, для `N` ферзей на доске размером `N` × `N`. Поскольку все ферзи должны стоять на разных строках, по одному в строке, достаточно хранить только номера столбцов, в которых стоят ферзи. То есть каждое решение можно представить как массив (список) из `N` элементов. Тогда все множество решений — это список, состоящий из списков-решений, например, для `N=4` список всех решений выглядит так:




```
[ [1, 3, 0, 2], [2, 0, 3, 1] ]
```

Будем строить множество решений последовательно. Сначала построим все варианты размещения ферзя в первой строке (очевидно, что их будет `N`), не учитывая остальных ферзей. Затем пытаемся ставить второго ферзя, оставляя только подходящие варианты, при которых первый и второй ферзи не бьют друг друга, и т.д. Покажем это на примере доски размером 4 × 4. На первом шаге получаем все варианты расстановки одного ферзя в строке 0:

```
[ [0], [1], [2], [3] ]
```

Далее ставим второго ферзя: если первый стоит в столбце 0, то второй может стоять только в столбцах 2 или 3, и т.д. Получаем список допустимых расстановок ферзей в первых двух строках:

```
[ [0,2], [0,3], [1,3], [2,0], [3,0], [3,1] ]
```

Для некоторых из этих случаев, например для варианта `[0,2]`, поставить третьего ферзя уже некуда (все клетки третьей строки оказались под боем первых двух ферзей), поэтому на следующем шаге число вариантов сокращается:

```
[ [0,3,1], [1,3,0], [2,0,3], [3,0,2] ]
```

На четвертом шаге остается только два варианта:

```
[ [1, 3, 0, 2], [2, 0, 3, 1] ]
```

Теперь составим функцию на языке Python. Эта функция будет содержать внутреннюю функцию `placeQueens`, которая расставляет заданное количество ферзей (от 1 до `N`):

```
def queens ( N ):
  def placeQueens ( k ):
    ...
  return placeQueens ( N )
```

Функция `placeQueens` будет рекурсивной. В первой строке обработаем условие окончания рекурсии: при `k = 0` возвращается список `[[]]`, содержащий единственный пустой элемент:

```
if k == 0: return [[]]
```

В противном случае создаем новый список для накопления решений:

```
R = []
```

и в цикле перебираем все варианты расстановки `k - 1` ферзей, добавляя все допустимые варианты для `k` ферзей в массив `R`:

```
for prev in placeQueens(k - 1):    # 1
  for col in range(N):             # 2
    if safe(col, prev):            # 3
      R.append ( [col] + prev )    # 4
```

В строке 1 функция вызывает себя рекурсивно, и все полученные варианты перебираются в цикле.

Строка 2 тоже содержит цикл, но уже по *N* возможным позициям расположения *k*-го ферзя. Если очередное расположение безопасно, то есть нового ферзя не бьют остальные (функция `safe` в строке 3 вернула значение `True`), новое частичное решение добавляется в массив `R` (строка 4).

Чтобы было удобнее проверять допустимость позиции, мы будем добавлять столбец, в котором стоит новый ферзь, в начало массива-решения, а не в конец, то есть новое решение строится как `[col] + prev`. Таким образом, функция, которая находит все допустимые расположения ферзей на доске размером `N` × `N`, принимает вид:

```
def queens ( N ):
  def placeQueens ( k ):
    if k == 0: return [[]]
    R = []
    for prev in placeQueens(k - 1):
      for col in range(N):
        if safe(col, prev):
          R.append ([col] + prev)
    return R
  return placeQueens ( N )
```

Остается написать функцию `safe`, которая возвращает значение `True`, если позиция допустима, и `False`, если недопустима. Она может выглядеть, например, так:

```
def safe(col, prev):
  for d, c in enumerate(prev):
    if c == col or \
```

```
            abs(c - col) == abs(d + 1):
            return False
    return True
```

Функция принимает два параметра: столбец `col`, в котором мы хотим разместить нового ферзя, и список `prev`, в котором хранятся все (допустимые!) расстановки предыдущих ферзей. Поэтому остается только проверить, не бьют ли нового ферзя предыдущие. Цикл

```
for d, c in enumerate(prev):
  ...
```

перебирает не только сами элементы из массива `prev` (столбцы, в которых расположены предыдущие ферзи), но и номера этих элементов в массиве. Номер попадает в переменную `d`, а столбец очередного ферзя — в переменную `c`. Новый ферзь бьет текущего, если

1) они стоят в одном и том же столбце, то есть `c==col`, или

2) они стоят на одной диагонали, то есть модуль разности номеров столбцов `c - col` равен модулю разности номеров строк.

Поскольку мы добавляем позицию нового ферзя в начало списка-решения, значение `d` на единицу меньше, чем разница номеров строк нового и текущего ферзей, поэтому второе условие недопустимости записывается как `abs(c - col) == abs(d + 1)`. Если новый ферзь бьет хотя бы одного из предыдущих, функция возвращает `False`, а если цикл завершен удачно, то результат функции равен `True`.

Вычисление арифметических выражений

Построим функцию, которая вычисляет арифметическое выражение, записанное в символьной строке. Для простоты будем считать, что в выражении используются только

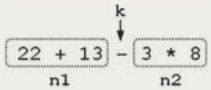
- целые числа и
- знаки арифметических действий `+-*/`.

Предположим также, что выражение не содержит ошибок и посторонних символов.

Последовательность выполнения операций при вычислении выражения определяется их *приоритетом* (старшинством). Например, умножение и деление выполняются раньше, чем сложение и вычитание. Отсюда следует, что последней выполняется крайняя справа операция с наименьшим приоритетом. Таким образом, можно сформулировать следующий алгоритм вычисления арифметического выражения, записанного в символьной строке **`s`**:

1. Найти в строке **`s`** последнюю операцию с наименьшим приоритетом (и сохранить номер этого символа в переменной **`k`**).

2. Используя дважды этот же алгоритм, вычислить выражения слева и справа от символа с номером **`k`** и записать результаты вычисления в переменные **`n1`** и **`n2`**.



1. Выполнить операцию, символ которой записан в **`s[k]`**, с переменными **`n1`** и **`n2`**.

Обратите внимание, что на шаге 2 этого алгоритма нужно решить ту же самую задачу для левой и правой частей исходного выражения, то есть функция будет *рекурсивной*.

Оформим алгоритм вычисления в виде функции, которая принимает символьную строку и возвращает целое число — результат вычисления выражения, записанного в этой строке. Алгоритм ее работы на псевдокоде:

```
k = номер символа, соответствующего
    последней операции
if k < 0: # нет знака операции
  перевести всю строку в число
else:
  n1 = результат вычисления левой части
  n2 = результат вычисления правой части
  применить найденную операцию к n1 и n2
```

Для того чтобы найти последнюю выполняемую операцию, будем использовать функцию **`lastOp`**, которую мы напишем далее. Если эта функция вернула "–1", то операция не найдена, то есть вся переданная ей строка — это число (предполагается, что данные корректны).

На языке Python функция запишется так:

```
def calc ( s ):
  k = lastOp ( s )
  if k < 0:
    return int(s)
  else:
    n1 = calc( s[:k] )
    n2 = calc( s[k + 1:] )
    if   s[k] == "+": return n1 + n2
    elif s[k] == "-": return n1 - n2
    elif s[k] == "*": return n1 * n2
    else: return n1 // n2
```

При вычислении левой и правой частей выражения используются срезы строки `s[:k]` (от начала строки до символа с индексом `k`, не включая его) и `s[k + 1:]` (начиная с символа с индексом `k + 1` до конца строки).

Теперь напишем функцию `lastOp`, которая ищет последнюю операцию с наименьшим приоритетом. Предварительно введем функцию, возвращающую приоритет операции (переданного ей символа):

```
def priority(op):
  if op in "+-": return 1
  if op in "*/": return 2
  return 100
```

Сложение и вычитание имеют приоритет 1, умножение и деление — более высокий приоритет 2, а все остальные символы (не операции) — приоритет 100 (условное значение). Тогда функция `lastOp` может выглядеть так:

```
def lastOp(s):
  minPrt = 50
  k = -1
  for i, c in enumerate(s):
    if priority(c) <= minPrt:
      minPrt = priority(c)
      k = i
  return k
```

Обратите внимание, что в условном операторе указано нестрогое неравенство (<=), чтобы най-

ти именно *последнюю* операцию с наименьшим приоритетом. Начальное значение переменной `minPrt` можно выбрать любое между наибольшим приоритетом операций (2) и условным кодом неоперации (100). Тогда, если найдена любая операция, то условный оператор срабатывает, а если в строке нет операций, то условие всегда ложно, и в переменной **`k`** остается начальное значение "–1".

Функция `enumerate` в заголовке цикла возвращает список пар "номер – символ" для всех символов строки `s`. На каждом шаге цикла номер попадает в переменную `i`, а сам символ — в переменную `c`.

Приведем пример вызова функции `calc`:

```
n = calc("12+2*23-34/3-5")
```

Правильный результат в данном случае — 42.

Числа Фибоначчи

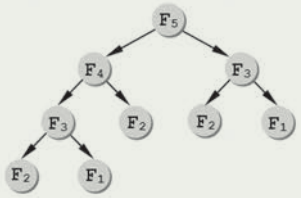
Во многих практических задачах встречается последовательность чисел Фибоначчи, которая задается рекуррентной формулой:

$F_1 = F_2 = 1; F_n = F_{n-1} + F_{n-2}$ при $n > 2$.

Для вычисления значения F_n "в лоб" можно использовать рекурсивную функцию:

```
def fibRec ( n ):
  if n < 3: return 1
  return fibRec(n - 1) + fibRec(n - 2)
```

Каждое из этих чисел связано с предыдущими, и вычисление F_5 приводит к рекурсивным вызовам, которые показаны на рисунке ниже. Таким образом, мы два раза вычислили F_3, три раза F_2 и два раза F_1. Рекурсивное решение получилось очень простое, но оно не оптимально по быстродействию: компьютер выполняет лишнюю работу, повторно вычисляя уже найденные ранее значения.



Какой же выход? Напрашивается такое решение — для того чтобы быстрее найти F_n, будем сохранять все уже вычисленные числа Фибоначчи. В языке Python для этого можно использовать словарь (коллекцию пар "ключ – значение"), в котором ключом будет номер числа, а значением — соответствующее число Фибоначчи. Фактически мы будем использовать *кэширование* — запоминать полученные ранее результаты в словаре.

Функция, основанная на этой идее, приведена ниже:

```
def fibCache ( n ):
  cache = {1: 1, 2: 1}              # 1
  def fib ( k ):                    # 2
    f = cache.get(k)                # 3
    if f is not None: return f      # 4
    f = fib(k - 1) + fib(k - 2)     # 5
    cache[k] = f                    # 6
    return f                        # 7
  return fib(n)                     # 8
```

В самом начале работы функции (строка 1) создается словарь-кэш, содержащий два элемента: пары 1 → 1 и 2 → 1, которые задают начальные значения последовательности Фибоначчи. В строках 2–7 записана внутренняя функция `fib`, после нее следует единственный оператор `return fib(n)`, вызывающий эту функцию (строка 8).

Функция `fib` вычисляет очередное число Фибоначчи по рекуррентной формуле (строка 5), но перед этим она "заглядывает" в словарь `cache`, проверяя, не было ли нужное значение вычислено раньше. Метод `get` ищет ключ в словаре (строка 3) и, если не находит, возвращает специальное "пустое" значение `None`. Если ключ найден (метод `get` вернул непустое значение), то результат функции берется прямо из словаря-кэша, без вычислений (строка 4).

Если ключ не найден в кэше, то значение нужно вычислить по общей формуле (строка 5) и записать в словарь (строка 6).

Поскольку ключ в этой задаче — натуральное число, вместо словаря можно использовать массив (список), который назовем `f`. Здесь удобно применить нумерацию с единицы, так что элемент массива `f[0]` использоваться не будет. Поэтому мы построим список длиной **`n + 1`**. Эта функция сначала заполняет массив, а потом возвращает последнее вычисленное значение, F_n:

```
def fibDyn ( n ):
  f = [1] * (n + 1)
  for i in range(3, n + 1):
    f[i] = f[i - 1] + f[i - 2]
  return f[n]
```

Можно сделать еще один шаг в сторону оптимизации программы. Заметим, что очередное значение элемента массива зависит только от двух (а не от всех!) предыдущих. В итоге нас интересует только одно последнее значение F_n, поэтому массив можно вообще не заводить, а вместо этого обойтись двумя переменными, которые хранят два предыдущих числа Фибоначчи:

```
def fibIter ( n ):
  f1 = f2 = 1
  for i in range(3, n + 1):
    f2, f1 = f1 + f2, f2
  return f2
```

Эта функция работает намного быстрее всех предыдущих. В таблице сравнивается время вычисления F_{35} всеми рассмотренными методами. Время, полученное при использовании функции `fibIter`, принято за единицу.

Функция	fibRec	fibCache	fibDyn	fibIter
Время	926482	8,0	2,3	1

Эти данные показывают, что кэширование (запоминание предыдущих результатов в словаре) позволяет принципиально улучшить решение в сравнении с "лобовой" рекурсией. Используя массив (и тем более отказавшись от массива!), можно ускорить вычисления еще в несколько раз.

Задача о рюкзаке

Задача о рюкзаке (англ. *knapsack problem*) — одна из самых известных трудных задач оптимизации, решаемых перебором вариантов. Дан набор из N предметов, известны их веса W_i ($i = 1, \ldots N$) и

стоимости V_i ($i = 1, \ldots N$). Требуется выбрать из этого набора предметы наибольшей общей стоимости, общий вес которых не превышает C.

Сначала составим простую (но неэффективную!) рекурсивную функцию, которая решает эту задачу. Будем добавлять предметы в набор по одному, находя на каждом шаге наилучшее решение для оставшегося "свободного" веса *rest*. Количество используемых на данном шаге предметов обозначим через k ($k = 0, 1, \ldots, N$).

Понятно, что при $k = 0$ (не осталось ни одного предмета) или при $rest = 0$ (совсем нет места) максимальная стоимость равна нулю, это и есть условие окончания рекурсии. Можно написать начало функции так:

```
def maxValue(k, rest):
  if k == 0 or rest == 0: return 0
  ...
```

Здесь многоточие обозначает строки, которые мы далее добавим.

Теперь предположим, что на очередном шаге мы добавляем в рассматриваемый набор предмет с номером k. У нас есть выбор — брать его или не брать. Если не берем, то лучшее решение остается то же самое, что и для `k - 1` предметов, которые мы уже рассмотрели ранее:

```
val0 = maxValue(k - 1, rest)
```

Взять предмет можно только тогда, когда его вес не больше, чем свободное место в рюкзаке, то есть $W_k <= rest$. Тогда лучший набор имеет стоимость V_k плюс лучшая стоимость для оставшейся части рюкзака ($rest — W_k$) при использовании всех $k - 1$ предыдущих предметов:

```
val1 = V[k - 1] + \
maxValue(k - 1, rest - W[k - 1])
```

Так как нумерация элементов списка в Python начинается с нуля, индексы элементов, в которых хранятся V_k и W_k, равны `k - 1`.

Результат функции `maxValue` — это максимальное значение из `val0` (если не берем новый предмет) и `val1` (если берем):

```
def maxValue(k, rest):
  if k == 0 or rest == 0: return 0
  val0 = maxValue(k - 1, rest)
  val1 = V[k - 1] + \
         maxValue(k - 1, rest - W[k - 1])\
         if W[k - 1] <= rest else 0
  return max(val0, val1)
```

Здесь при вычислении `val1` использован так называемый "тернарный оператор" языка Python. Эта запись равносильна следующему условному оператору в привычной форме:

```
if W[k - 1] <= rest:
      val1 = V[k - 1] + \
      maxValue(k - 1, rest - W[k - 1])
else: val1 = 0
```

Теперь остается "встроить" внутреннюю функцию `maxValue` в функцию `knapsack`, которая принимает массивы `W` и `V`, а также предельный вес `C`:

```
def knapsack(W, V, C):
  def maxValue(k, rest):
    if k == 0 or rest == 0: return 0
    i = k - 1
    val0 = maxValue(k - 1, rest)
    val1 = V[i] + \
           maxValue(k - 1, rest - W[i]) \
           if W[i] <= rest else 0
    return max(val0, val1)
  return maxValue(len(W), C)
```

Конечно, эта функция при больших значениях N работает очень медленно, потому что много раз вычисляет одни и те же значения во время рекурсивных вызовов. Для того чтобы ускорить ее, можно применить кэширование: сохранять все ранее вычисленные значения в словаре, как мы это делали при вычислении чисел Фибоначчи. Но в данном случае в качестве ключа будет выступать пара (k, *rest*), которую можно объединить в *кортеж* — неизменяемую коллекцию. Тогда запись вычисленного значения `result` в словарь будет выглядеть так:

```
cache[(k, rest)] = result
```

а проверка наличия данных в кэше — так:

```
if (k, rest) in cache:
  return cache[(k, rest)]
```

В остальном функция не изменится:

```
def knapsackCache(W, V, C):
  cache = {}
  def maxValue(k, rest):
    if (k, rest) in cache:
      return cache[(k, rest)]
    if k == 0 or rest == 0: return 0
    i = k - 1
    val0 = maxValue(k - 1, rest)
    val1 = V[i] + \
           maxValue(k - 1, rest - W[i]) \
           if W[i] <= rest else 0
    result = max(val0, val1)
    cache[(k, rest)] = result
    return result
  return maxValue(len(W), C)
```

Теперь построим еще один вариант решения, использующий динамическое программирование, то есть последовательное вычисление всех решений задач меньшей размерности и сохранение их в массиве (списке). Здесь не будет внутренней функции, но будет матрица размером $(N + 1) \times (C + 1)$ для хранения всех промежуточных решений.

К сожалению, в отличие от Паскаля и C, в языке Python нет простого способа выделить память под матрицу. Матрицу можно представить как список списков (каждый внутренний список — строка матрицы) и создать так:

```
maxValue = []
for i in range(N + 1):
  maxValue.append ( [0] * (C + 1) )
```

или в краткой форме:

```
maxValue = [[0] * (C + 1)
   for i in range(N + 1)]
```

Заполняем матрицу в двойном цикле (по строкам), вычисляя значение `maxValue[k][rest]` для каждого варианта:

```
for k in range(1,N + 1):
  i = k - 1
  for rest in range(1,C + 1):
    val0 = maxValue[k - 1][rest]
    val1 = V[i] + \
           maxValue[k - 1][rest - W[i]] \
```

```
                if W[i] <= rest else 0
            maxValue[k][rest] = max(val0, val1)
```

Решение исходной задачи — это значение `maxValue[N][C]`.

Использование динамического программирования позволяет не только вычислить наибольшую стоимость предметов, которые можно унести в рюкзаке, но и найти сам набор взятых предметов. Все необходимые данные находятся в матрице `maxValue`. Сначала рассмотрим идею на примере. Пусть

```
W = [1, 5, 3, 4]
V = [15, 10, 9, 5]
C = 8
```

Для этого случая матрица `maxValue` принимает вид, как показано в табл. ниже.

Начинаем с правого нижнего угла, там находится число 29 — максимальная стоимость для исходной задачи. Для того чтобы определить, взят ли предмет с `k = 4`, нужно проверить ячейку над текущей в том же столбце (она выделена желтым фоном, там находится число 24). Таким образом, при добавлении нового предмета решение изменилось, поэтому предмет № 4 взят в набор. Его вес равен 4, поэтому остается использовать вес 4 на три оставшихся предмета.

Смотрим в ячейку (3, 4), там число 24, а выше него — другое число, 15. Поэтому предмет № 3 тоже взят в набор.

Оставшийся вес — 1, значение в ячейке (2, 1) равно значению в ячейке (1, 1), поэтому второй предмет не взят, остаемся в том же столбце.

Наконец, значения ячеек (1, 1) и (0, 1) разные, поэтому первый предмет использован. Таким образом, оптимальный набор составляют предметы с номерами 1, 3 и 4.

Теперь запишем этот алгоритм на языке Python, построив битовый массив `take`, где 1 означает, что соответствующий предмет взят, а 0 — что не взят.

```
take = [0]*N
rest = C
for k in range(N,0,-1):
  if maxValue[k][rest] !=
     maxValue[k - 1][rest]:
    take[k - 1] = 1
    rest -= W[k - 1]
```

В итоге функция будет возвращать не только оптимальную стоимость взятых предметов, но и этот битовый массив:

```
def knapsackDyn(W, V, C):
  N = len(W)
  maxValue = [[0]*(C + 1)
  for i in range(N + 1)]
  for k in range(1,N + 1):
    i = k - 1
    for rest in range(1,C + 1):
      val0 = maxValue[k - 1][rest]
      val1 = V[i] + \
      maxValue[k - 1][rest - W[i]] \
             if W[i] <= rest else 0
      maxValue[k][rest] = max(val0, val1)
  take = [0] * N
  rest = C
  for k in range(N,0,-1):
    if maxValue[k][rest] !=
       maxValue[k - 1][rest]:
      take[k - 1] = 1
      rest -= W[k - 1]
  return maxValue[N][C], take
```

Напоследок сравним время решения задачи о рюкзаке для $N = 22$ [7] при использовании показанных выше функций:

Функция	knapsack	knapsackCache	knapsackDyn
Время	364	1,2	1

Как и ранее, лучшее время принято за единицу.

Заключение

В статье рассмотрена реализация некоторых алгоритмов, рассмотренных в учебнике [3–4], на языке Python. В сравнении с языками Паскаль и С, которые традиционно используются в школах России для обучения программированию, Python позволяет строить короткие и понятные решения, использующие встроенные возможности языка.

Автор благодарит В.М. Гуровица за обсуждение материалов статьи и полезные замечания.

Литература

1. *Сукин И.А.* Python, проглатывающий слона // Информатика, № 2, 2012, с. 22–42.

2. *Поляков К.Ю.* Язык Python глазами учителя // Информатика, № 9, 2014, с. 4–17.

3. *Поляков К.Ю., Еремин Е.А.* Информатика. 10-й класс. Углубленный уровень. В двух частях. М.: Бином, 2013.

4. *Поляков К.Ю., Еремин Е.А.* Информатика. 11-й класс. Углубленный уровень. В двух частях. М.: Бином, 2013.

5. Язык Python [Электронный ресурс] URL: *http://kpolyakov.spb.ru/school/probook/python.htm* (дата обращения 18.05.2014).

6. *Hetland M.L.* Python Algorithms: Mastering Basic Algorithms in the Python Language, Apress, 2010.

7. Knapsack problem/0–1 [Электронный ресурс] URL: *http://rosettacode.org/wiki/Knapsack_problem/0-1* (дата обращения 18.05.2014).

	rest = 0	rest = 1	rest = 2	rest = 3	rest = 4	rest = 5	rest = 6	rest = 7	rest = 8
k = 0	0	0	0	0	0	0	0	0	0
k = 1	0	15	15	15	15	15	15	15	15
k = 2	0	15	15	15	15	15	25	25	25

Архитектура и программное обеспечение компьютеров

И.А. Калинин,
Н.Н. Самылкина

Темы “Архитектура компьютера” и “Программное обеспечение компьютеров” в углубленном курсе информатики[1] должны получить серьезное развитие, так как являются основой для многих направлений отрасли информационных технологий. В предлагаемом материале авторы раскрывают современные подходы к реализации Фоннеймановской и Гарвардской архитектур в их сравнении и практическом использовании. Достаточно полно систематизирована тема основных классов программного обеспечения. Рассмотрены популярные линии операционных систем, их состав и функционирование, а также обобщены виды прикладного ПО (без введения лишних классификаций), рассказывается о специализации компьютеров и связанных с этим профессиональных требованиях к некоторым специальностям ИТ-отрасли.

Архитектура компьютеров

Традиционно под словом “архитектура” (лат. *architectura* — строительство, зодчество, от др.-греч. ἀρχι — старший, главный и τέκτων — строитель, плотник) понимают в первую очередь всевозможные задачи, связанные с организацией жилого пространства вокруг человека, общий вид здания и т.д.

Применительно к другим областям деятельности, в частности к компьютерам, под ***архитектурой*** понимают основные принципы организации объекта, его составные части и общую систему их взаимодействия. Под архитектурой понимают и общие положения создания и работы компьютеров (то есть принципы организации аппаратуры), и базовую систему команд (фак-

[1] Информатика. Углубленный уровень: учебник для 10-го класса / *Калинин И.А., Самылкина Н.Н.*

тически — восприятие аппаратуры с точки зрения программы).

Мы начнем с описания архитектуры собственно компьютера — основных его блоков, принципов их взаимодействия и конструктивных решений.

Ранее мы подробно разобрали логические устройства, из которых могут быть созданы отдельные компоненты компьютера. Неясным остается, каким образом будет осуществляться их взаимодействие, т.е. как будут обеспечены “стрелочки” в общей схеме устройства компьютера.

Все преобразования сигналов (вычисления) происходят пошагово. Шаги задаются специальным устройством — тактовым генератором.

Вам уже известны практически все простейшие блоки, из которых строится современный компьютер: это сумматоры — как основной элемент модуля обработки, регистры — как устройство запоминания, шина — как основной элемент связи и т.д. Тем не менее очевидно, что связать эти элементы между собой можно очень по-разному.

В современных компьютерах чаще всего взаимодействие осуществляется с помощью **шины** — устройства, обеспечивающего прямой обмен между двумя компонентами компьютера. Это не означает, что таких устройств должно быть ровно два, это означает, что два устройства могут через шину обмениваться данными.

С технической точки зрения шина — это в основном набор проводников, обеспечивающих обмен сигналами между отдельными устройствами, и микросхемы, которые обеспечивают управление работой этого набора проводников.

В зависимости от способа передачи данных шины делят на *параллельные* (когда порция данных передается как единый набор сигналов по нескольким проводникам) и *последовательные* — когда данные передаются последовательно, изменением отдельного сигнала на проводнике.

От общего подхода к реализации шины зависит, какие компьютеры можно собрать. Именно шина в значительной степени определяет архитектуру всего компьютера. Одним из наиболее важных конструктивных решений стало повсеместное использование *открытых архитектур*. Эти архитектуры предусматривают возможность взаимодействия с произвольным, а не заранее известным устройством.

В применении открытых архитектур очень важную роль играют стандарты — то есть те самые общие правила и требования к устройствам. В стандартах оговариваются наборы команд, способ соединения, физические размеры, уровень и способ электропитания устройств и т.д. Стандарты, как правило, разрабатываются и принимаются коллегиально — группами разработчиков из заинтересованных организаций. И очень часто это приводит к тому, что разрабатывается несколько конкурирующих стандартов, из которых потом и “выживает” наиболее подходящий всем остальным.

Одно из самых распространенных архитектурных решений — ***шинно-модульный принцип***. Основу такой архитектуры составляет ***системная шина*** — основной канал взаимодействия процессора с различными устройствами. Первоначально все устройства подключались к общей системной шине через устройство взаимодействия, контролировавшее их работу, — ***контроллер***. Для организации

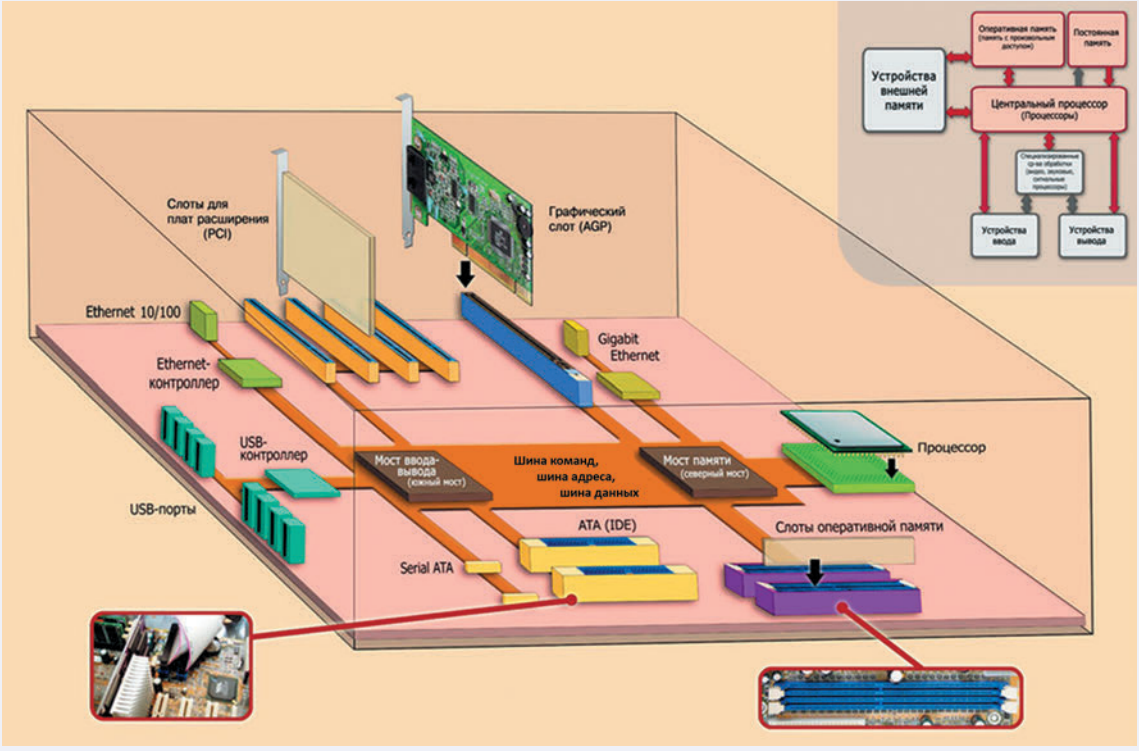


Рис. 1

взаимодействия в системной шине предусматривается три основных набора проводников: ***шина команд***, ***шина адреса*** и ***шина данных***.

Например, чтобы проиллюстрировать принцип действия системной шины, приведем упрощенный список действий по получению данных из оперативной памяти:

1. Процессор формирует адрес памяти и задает место получения данных — регистр процессора.
2. Процессор устанавливает на шине адреса адрес памяти.
3. Процессор устанавливает команду чтения ячейки памяти с записью в заданный регистр на шине команд (реализуется как сигнал на специальной линии).
4. Процессор устанавливает на шине команд флаг готовности.
5. На следующем такте контроллер памяти "обнаруживает" на шине команд свою команду. Он получает данные из памяти и устанавливает их на шине данных.
6. Контроллер памяти устанавливает на шине команд флаг готовности.
7. На следующем такте процессор "фиксирует" данные.

То есть шина данных применяется для передачи произвольных данных, шина адреса — для указания места их размещения, а шина команд — для указания и инициации действия.

Стоит заметить, что шина команд фактически является набором независимых проводников, каждый из которых задает команду или выполняет вспомогательную функцию — передает тактовую частоту, контролирует готовность, обновляет память и т.д.

Схема передачи данных из регистра/в регистр применяется и для взаимодействия с устройствами ввода-вывода — для них выделяется часть адресного пространства оперативной памяти, запись и получение информации из которых фактически и являются взаимодействием с устройством.

Из описанного алгоритма также видно, что одним из основных факторов, определяющих скорость обмена данными, является ***тактовая частота***.

В современных цифровых устройствах наиболее широко применяются две архитектуры, имеющие множество реализаций.

1. ***Фоннеймановская архитектура.*** Именно эта архитектура используется в подавляющем большинстве привычных компьютеров, начиная с персональных.

2. ***Гарвардская архитектура.*** Эта архитектура чаще всего используется там, где мы компьютеров не замечаем, например, в микрокомпьютерах для носимых или встроенных устройств, в станках, в системах управления, в автоматизированных станциях наблюдения.

Обе эти архитектуры были разработаны во время конкурса на разработку системы вычислений для военно-морской артиллерии в 30–40-х годах XX века. За счет простоты и универсальности в конкурсе победила Фоннеймановская архитектура.

Фоннеймановская архитектура

Эта архитектура была предложена американским математиком и инженером Джоном фон Нейманом и его соавторами[2] в 1946 году как основной принцип организации работы компьютеров.

В основе данной архитектуры лежат следующие принципы:

1. Для представления (кодирования) данных и команд используется двоичная система счисления.
2. И данные и программы хранятся в едином устройстве (памяти) с произвольным доступом.
3. Память представлена в виде ячеек фиксированного размера, каждая ячейка имеет адрес — т.е. номер, определяющий ее положение.
4. Все команды выполняются последовательно, т.е. следующая команда выполняется после завершения предыдущей.

В компьютере, реализующем эту схему, как правило, используются:

1. Арифметико-логическое устройство, выполняющее вычисления.
2. Устройство управления — извлекающее команды из памяти.
3. Устройства ввода и вывода, подающие данные.
4. Память — устройство, хранящее данные и команды и обеспечивающее к ним доступ во время работы.

Архитектура фон Неймана очень широко распространена, поскольку позволяет сократить и количество устройств, и, что бывает очень важно, количество вводов и выводов данных в арифметико-логическое устройство процессора.

Сама архитектура фон Неймана никак не оговаривает способ соединения и взаимодействия устройств. Чаще всего реализуется такая общая схема, как на *рис.* 2 на с. 29.

Схема иллюстрирует только состав и не демонстрирует способы соединения устройств.

Арифметико-логическое устройство и устройство управления объединяются в общий конструктивный модуль, который называется ***процессором***.

Для компьютеров с Фоннеймановской архитектурой было разработано множество разных реализаций. Из них мы рассмотрим те, которые стали промышленными стандартами, применяемыми во всей отрасли и составляющими основу большинства современных компьютеров.

Одна из первых промышленно применявшихся шин — ISA была организована так, как показано на *рис.* 3 (см. с. 29).

К сожалению, такая простая схема имеет очень серьезное ограничение — медленные устройства.

[2] Артур Бёркс (англ.), Герман Голдстайн (англ.) и Джон фон Нейман, "Предварительное рассмотрение логического конструирования электронного вычислительного устройства".

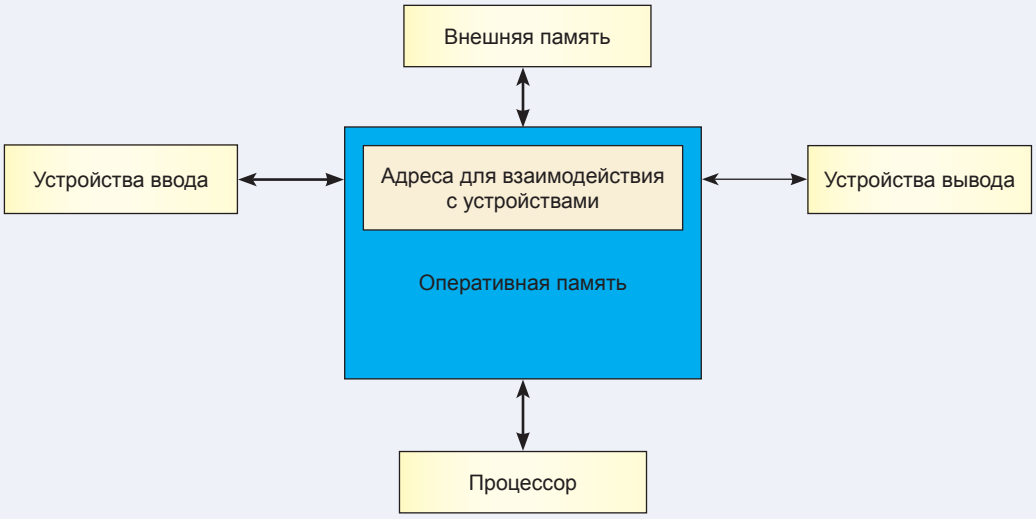


Рис. 2

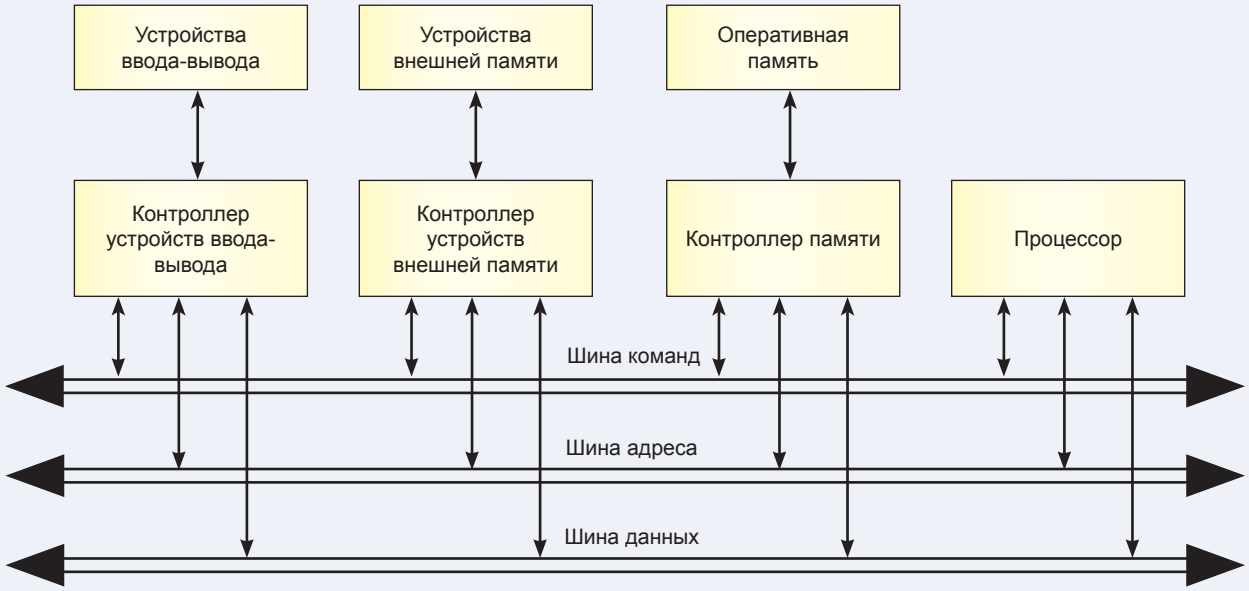


Рис. 3

Если к шине подключено медленное устройство (например, плата расширения), то устройство быстрое (например, память) вынуждено работать с его скоростью. Разработать же устройство-контроллер, работающее с высокой частотой (особенно там, где она для собственно работы не нужна), трудно, а использование ранее разработанных устройств сильно затрудняется.

Тем не менее существуют быстрые устройства, которые должны обрабатывать большие объемы данных, например, графический адаптер или высокоскоростные сетевые адаптеры.

В рамках разработки новой архитектуры PCI[3] с самого начала было введено разделение: взаимодействие организуется с помощью отдельных микросхем — мостов. Микросхемы-мосты обеспечивают взаимодействие устройств с разной скоростью, дополнительно используя делители частоты шины.

Этот подход получил название локальной шины. Схема взаимодействия в нем показана на *рис*. 4 на с. 30.

Можно вернуться к первой иллюстрации, на которой изображено то же самое, но более наглядно.

Вот принципиальная схема одной из современных реализаций такого подхода (*рис*. 5).

Набор микросхем предназначен для поддержки высокопроизводительных процессоров со встроенным контроллером памяти и, возможно, графическим ядром.

Первый основной блок Intel Core Processors имеет прямые соединения с модулями памяти и высокоскоростным слотом — для отдельной графической платы. Взаимодействие с другими устройствами осуществляется по линиям DMI и специальной линии передачи графического изображения.

[3] Были, конечно, разработаны и другие шины, организованные по-другому (например, VESA), но именно эти проблемы в итоге заставили от них отказаться. PCI стала промышленным стандартом.

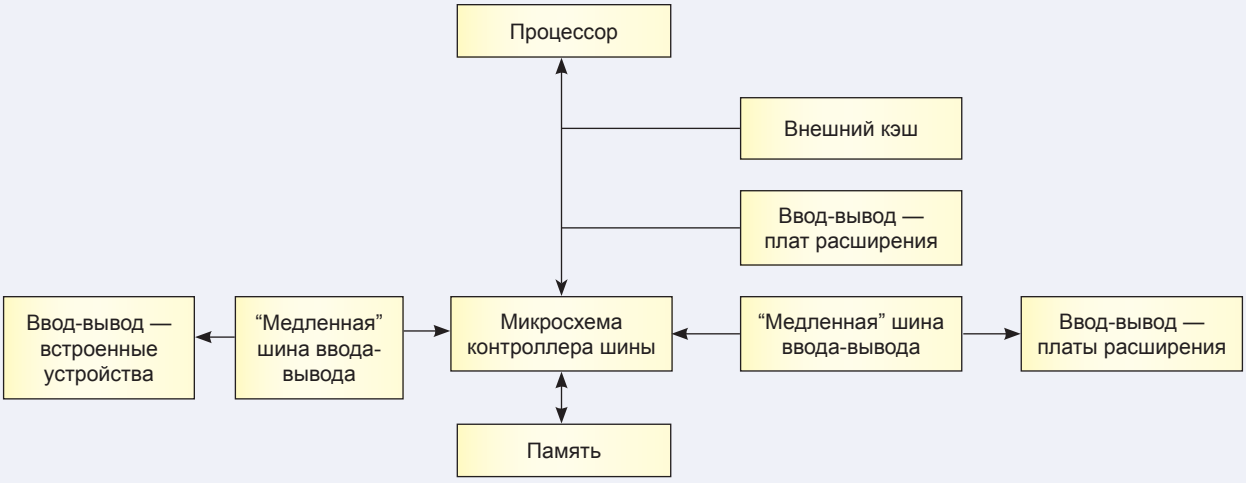


Рис. 4

Обмен с основным концентратором ввода-вывода (микросхемой чипсета) ведется через контроллер ввода-вывода, второй основной блок. К нему отдельными шинами разных стандартов подсоединяются порты USB, встроенные сетевые порты, порты для плат PCI Express, SATA — для жестких дисков.

Чипсет полностью реализует архитектуру PCI Express, с центральным коммутатором.

Наиболее быстрые устройства (процессор, память, графический адаптер) соединяются с процессором практически напрямую, контроллер памяти реализован прямо в структуре процессора. Более медленные устройства (например, контроллеры дисков) подключены к другой микросхеме — общему концентратору ввода-вывода, бывшему “южному” мосту.

Вторым фактором, определяющим быстродействие, будет разрядность шины, а фактически разрядность регистров процессора, которые используются для обмена данными. Сама шина будет иметь значительно больше проводников — и для команд, и для адресов. Современные компьютеры используют два основных размера — 32 или 64 бита.

На практике эти факторы не единственные, многое будет зависеть и от того, как работают сами устройства.

Для сокращения времени обработки прямо на кристалле процессора предусматривается несколько отдельных высокоскоростных разделов памяти, так называемой “кэш-памяти”. Ее использование позволяет заранее считывать большие блоки и этим сократить временные затраты на получение данных. Фактическая производительность процессора зависит, в частности, и от алгоритмов предсказания того, какая часть программы или данных потребуется дальше.

Еще один важный фактор — это взаимодействие с устройствами внешней памяти. Первоначальный вариант предусматривал использование процессора. Но устройства внешней памяти по сравнению с ним очень медленные, а их операции почти одинаковые. Поэтому шина PCI предусматривает возможность прямого взаимодействия контроллера, например, жесткого диска и памяти — режим DMA (***d****irect* ***m****emory* ***a****ccess* — прямой доступ к памяти). Для этого необходимо специализированное устройство — контроллер прямого доступа к памяти, реализованное отдельно или в составе одной из микросхем.

В приведенных архитектурах каждое подключаемое устройство требует для своей работы:

— *область памяти* для обмена данными;

— *прерывание* (возможно, несколько) — линию шины команд, сообщающую об изменении состояния устройства;

— *набор портов* — средство передачи команд к устройству.

В современных реализациях шин предусмотрены средства управления для корректного распределения ресурсов между устройствами.

С ростом скорости работы процессоров, объема и скорости доступной оперативной памяти, увеличением количества требуемых устройств и скорости их работы шина PCI перестала удовлетворять современным требованиям.

В первую очередь это проявилось на взаимодействии с графическими адаптерами,

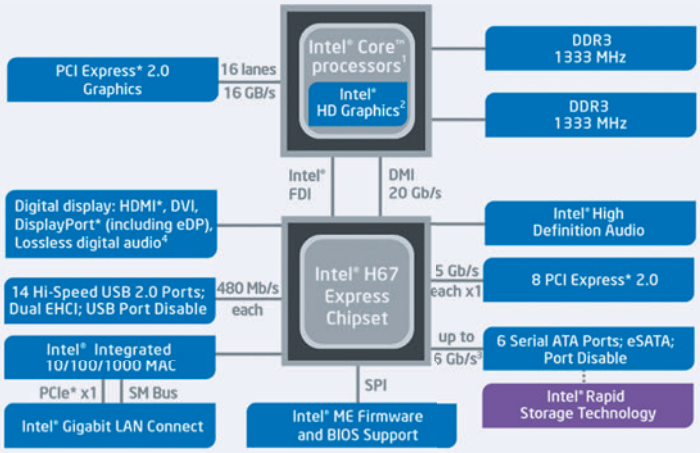


Рис. 5. Схема чипсета Intel H67 Express

там, где поток данных был самым большим. Был разработан новый стандарт специализированной графической шины AGP.

Со временем такая необходимость возникла и для ряда других устройств, в частности, для эффективного подключения нескольких процессоров и скоростных устройств внешней памяти.

С ростом скорости работы шин выявилось существенное ограничение — использование параллельного принципа требовало все больших усилий по обеспечению синхронности сигнала на параллельных линиях.

С определенного момента стало значительно выгоднее использовать последовательную передачу данных по фактически независимой линии.

Поэтому альянсом производителей была разработана промышленная архитектура третьего поколения (3Gio), получившая название PCI Express.

В этой архитектуре предусмотрено использование скоростных последовательных каналов связи, причем устройство может использовать несколько линий при необходимости.

Пропускная способность одной линии в текущей версии стандарта PCI Express 2.0 составляет от 500 мегабайт до 1 гигабайта в секунду. Устройство обязано поддерживать минимум одну линию, но может быть рассчитано на 2, 4 и более (до 32) линии.

Архитектура PCI Express позволяет соединять одним каналом только два устройства, то есть организована по принципу “звезды”. Схема может быть реализована очень разными способами, например, как на ранее приведенном рисунке (*рис.* 5). Или вариант для мощного сервера:

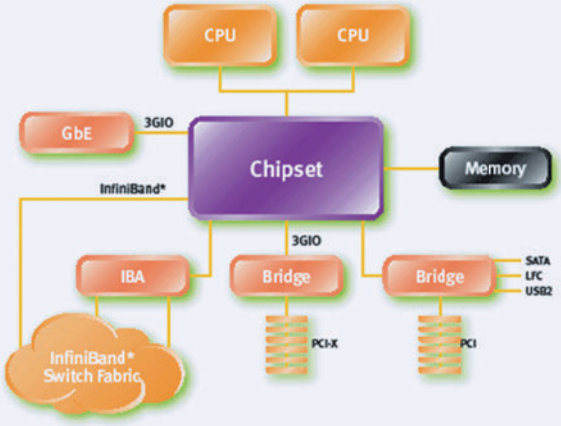


Рис. 6. PCI Express в производительном сервере

В производительном многопроцессорном сервере архитектура PCI Express позволяет организовать быстрый обмен информацией через центральный концентратор, обозначенный как Chipset.

К нему отдельными шинами подключаются память (memory), процессоры (CPU), разные внешние устройства и, через мосты (bridge), устройства взаимодействия с периферийным оборудованием — устройствами внешней памяти, универсальной шиной, дополнительными платами старого стандарта.

Такая схема позволяет обеспечить максимально быстрый обмен данными, в котором ни одно устройство не препятствует другим. Можно наращивать мощности сервера или проводить ремонт, заменяя блоки, без отключения.

Архитектура PCI Express также позволяет:

1. “Горячую” замену устройств, т.е. замену любого устройства без выключения компьютера при соблюдении процедуры.
2. Управление питанием.
3. Контроль четности передаваемых данных для борьбы с ошибками передачи.
4. Гарантированную полосу пропускания, т.е. конкуренция устройств не приводит к падению скорости.

PCI Express также позволяет подключать внешние устройства (например, в ноутбуках) с помощью слотов Mini PCI Express.

Эта архитектура применяется не только в компьютерах, но и, например, в высокопроизводительных сетевых устройствах.

Шина третьего поколения обладает большим резервом расширения. Ее работа построена по уровневой схеме, т.е. позволяет использовать разные физические линии связи и таким способом наращивать производительность без замены устройств, позволяет строить взаимодействие с другими шинами с помощью специального слоя организации взаимодействия и т.д.

Распространение общеизвестных и открытых промышленных стандартов стало одним из существенных факторов, определивших широчайшее распространение компьютеров, удешевивших разработку и совершенствование устройств, резко повысивших надежность и предсказуемость их работы.

Гарвардская архитектура

Универсальные промышленные модульные архитектуры выгодны и эффективны в тех случаях, когда есть техническая возможность (и необходимость) обеспечивать работу и взаимодействие отдельных частей — модулей. В реальном же мире часто возникают ситуации, когда вычислительная техника очень узко специализирована и выполняет строго заданные функции в рамках одного устройства. В таком случае гораздо важнее обеспечить высокую надежность, низкое энергопотребление и маленькие размеры.

В таких устройствах часто применяется подход, получивший название ***Гарвардской архитектуры***.

В основе Гарвардской архитектуры лежит идея хранения данных и программ в физически разных устройствах. В первых попытках реализации этой архитектуры в качестве устройства хранения программ использовалась перфолента, в современных — специализированная отдельная память.

До 70-х годов XX века эта архитектура практически не использовалась. Для ее реализации требовалось создавать АЛУ с двойным комплектом контактов, что делало их слишком сложными.

Для уменьшения общего количества контактов была разработана расширенная Гарвардская архитектура, в которой используются общая шина данных и команд, но две шины адреса.

В современных системах эта архитектура реализуется на едином кристалле-микросхеме, что дела-

ет их очень компактными и быстрыми. Такое техническое исполнение называется ***однокристальной ЭВМ***, или ***микропроцессором***.

В микропроцессорах на одном физическом кристалле реализовано несколько разных устройств, в частности:

1. Собственно процессор, т.е. исполнитель команд.
2. Память программ — энергонезависимый модуль, в который можно с помощью специальной процедуры (прошивки) записать программу. Как правило, количество циклов записи невелико. Объем памяти программ в современных микропроцессорах колеблется от 1 до 256 Кб.
3. Память данных, ОЗУ. Это фактически оперативная память, предназначенная для хранения данных во время обработки. Ее объем может быть от 0 до 8 Кб.
4. Энергонезависимая память данных, позволяющая сохранить некоторый объем данных при отсутствии питания. Объем ее колеблется от 0 до 4 Кб.
5. Порты ввода-вывода — устройства взаимодействия с внешними устройствами. Как правило, реализуют доступ к “ножкам” микросхемы.
6. Встроенные таймеры.
7. Генератор сигнала с широтно-импульсной модуляцией.
8. Аналого-цифровой преобразователь — устройство, преобразующее аналоговый сигнал в цифровой код.
9. Последовательный интерфейс.
10. Нетрудно заметить, что памяти в микропроцессорах мало. Поэтому их программирование чаще всего ведется самым экономичным способом — на языке Ассемблера. От эффективности программы напрямую зависит эффективность и стоимость решения задачи.

Физически такая микросхема имеет типовой корпус с заданным числом контактов-ножек. С их помощью микропроцессор взаимодействует с внешними устройствами, среди которых могут быть различные датчики, индикаторы (например, цифровой индикатор), шаговые электродвигатели, выключатели, устройства для работы с локальной сетью и шиной USB.

Для типовых микропроцессоров разработано много дополнительных типовых устройств, что позволяет быстро создавать с их помощью самые разные управляемые устройства: от станков до электронных часов и плееров.

Взаимодействие с внешними устройствами

Рассмотренные нами устройства — процессор, оперативная память, шины — относят к *внутренним устройствам* компьютера. Без этих устройств говорить о существовании компьютера нельзя. Тем не менее для организации реальной работы перечисленных устройств явно недостаточно — необходимы дополнительные устройства, осуществляющие ввод, вывод и хранение информации при отключении питания. Такие устройства получили название *внешних устройств*.

Работа с внешними устройствами требует взаимодействия, т.е. обмена сигналами, между внешними и внутренними устройствами. Осуществляется такой обмен не напрямую, а с помощью устройств-посредников. Основные два вида таких устройств — это ***адаптеры*** (устройства, преобразующие сигналы) и ***контроллеры*** (устройства, выполняющие на основе сигналов логические преобразования и управляющие подключенными к ним устройствами). В современных условиях четкую границу между этими видами устройств не всегда можно провести точно.

Изначально при разработке таких устройств предусматривались разные решения для подключения разных устройств. Но со временем был выработан целый ряд стандартов, которые позволили подключать разные по назначению устройства с помощью общих стандартных шин.

Самое распространенное решение такого рода — ***универсальная последовательная шина (Universal Serial Bus, USB)***.

Шина USB предусматривает не только передачу цифровых данных, кодированных по схеме NRZI, но и питание подключенных устройств.

Каждое подключенное к этой шине устройство получает уникальный номер, и одновременно с компьютером может работать до 256 устройств. Такое количество разъемов установить невозможно, поэтому шина также предусматривает возможность разветвления канала — в этом случае его пропускная способность и мощность электропитания делится на несколько устройств.

С учетом того, что цифровая передача данных — универсальный способ обмена, а скорость современных реализаций USB достаточно высока, то с ее помощью подключают: клавиатуры, мыши, планшеты, устройства хранения данных, видеокамеры, звуковые устройства и даже мониторы.

Универсальная шина — удобный способ подключения устройств, но в ряде случаев использование его нецелесообразно. Использование общего контроллера с ограниченной пропускной способностью стало бы узким местом в тех случаях, когда нужен высокоскоростной обмен данными, учитывающий специфику устройств. Например, шина USB не может обеспечить достаточной скорости при работе с основными жесткими дисками.

Для работы с современными жесткими дисками применяются более скоростные последовательные шины, ориентированные именно на устройства внешней памяти (жесткие диски, оптические дисководы и т.д.), — Serial ATA, Serial Attached Scsi. Эти шины обеспечивают высокую скорость и надежность обмена данными, значительно эффективнее обрабатывают параллельные запросы.

Системное программное обеспечение

Используя компьютеры в повседневной работе, мы опираемся на огромное количество самых разных программ, которые и составляют ***программное обеспечение (ПО)***.

Прежде чем рассматривать специализированные средства решения различных задач, мы рассмотрим общую структуру программного обеспечения в связке с архитектурой.

Рис. 7. Структурно-уровневая схема

Операционные системы

Операционная система — основной компонент программного обеспечения современных компьютеров. Из-за высоких требований к основе именно операционные системы считаются самыми сложными и объемными продуктами с точки зрения программного кода. Именно это программное обеспечение в значительной степени и определяет возможности компьютера.

Прежде чем дать определение и описать структуру этого вида программного обеспечения, мы вкратце опишем историю его появления и развития.

Изначально (у компьютеров первого поколения) такого вида программ не было. Поскольку с формальной точки зрения операционная система занимает ресурсы компьютера для целей, не оправданных непосредственными задачами, то в очень ограниченном пространстве создавать ОС возможности не было.

Каждая программа представляла собой замкнутый программный код, в котором нужно было решить все элементы задачи — включая ввод данных, расчеты, вывод, хранение и т.д. Переключение с одной программы на другую, а точнее, загрузка и запуск новой программы, выполнялось оператором.

С распространением компьютеров второго поколения, увеличением объема доступных ресурсов и ростом количества разрабатываемых программ появилась необходимость, во-первых, автоматизировать переключение и загрузку (это существенно уменьшало простой), а во-вторых, уменьшить время разработки программ за счет стандартизации некоторых функций: например, обращения к внешним устройствам и т.п.

Таким образом, возник отдельный класс программного обеспечения, решающий не непосредственно вычислительные, а сервисные, обслуживающие задачи. В первую очередь он включал в себя загрузчик программ и набор функций, выполнявших ввод-вывод на периферийные устройства.

С появлением третьего поколения ЭВМ его задачи усложнились. Он стал не просто загружать программы и переключать систему на исполнение других программ, но и обеспечивать взаимодействие с пользователем при работе в разных программах, предоставлять возможности по организации хранения данных и обеспечивать доступ к стандартным функциям во время работы программы. Кроме этого, появились средства, обеспечивающие исполнение нескольких программ одновременно. Например, ожидая ввода данных в одной задаче, система выполняла расчеты в другой.

Таким образом, складывалась программная система, обеспечивающая основные типовые операции по управлению компьютером.

С этого момента появилась возможность говорить о полнофункциональных ***операционных системах (ОС)***. Таких комплексах программ, которые управляют работой программного обеспечения[4], предоставляют программам интерфейс (средства взаимодействия) с различными устройствами, упрощают операции ввода-вывода и т.д.

Управление работой программного обеспечения включает в себя организацию эффективного распределения имеющихся ресурсов (процессорного времени, памяти, устройств хранения), управление

[4] ОС управляет и своей собственной работой.

вычислительными процессами, защиту программ и их данных во время работы.

Интерфейс, предоставляемый операционными системами, позволяет программам взаимодействовать с аппаратурой, между собой и с пользователями, не имея реализации всех этих функций в своем составе.

Современные операционные системы классифицируют по многим признакам, из которых мы выделим несколько. Во-первых, это классификация по назначению:

1. ОС общего назначения (универсальные). Это операционные системы для широко распространенных аппаратных платформ, не ограничивающие спектр решаемых задач, допускающие расширение и подготовку новых программ. В первую очередь это операционные системы для обычных компьютеров.

2. Узкоспециализированные ОС. Это системы, предназначенные для узкоспециализированных устройств (например, коммуникационного оборудования, устройств управления технологическими линиями и т.д.), решающие узкие классы задач, но зато оптимизированные именно для этого оборудования. Многих возможностей у них нет, но то, что есть, реализовано предельно эффективно.

Во-вторых, по способу управления задачами:

1. Однозадачные ОС. Эти ОС (чаще всего — узкоспециализированные) выполняют задачи последовательно и не позволяют запустить больше одной пользовательской задачи. Широко применяются в промышленности.

2. Многозадачные ОС. В этих ОС время процессора(ов) автоматически делится между несколькими программами, создавая иллюзию одновременного выполнения. Это требует дополнительных действий и ресурсов, поэтому формально эти ОС менее надежны — но в подавляющем большинстве случаев разница невелика.

3. Операционные системы реального времени. Бывают как одно-, так и многозадачными, но имеют очень существенную особенность — позволяют работать с точным распределением времени, синхронизируя выполнение с внешними процессами, то есть реакцию в заданный и предсказуемый срок. Такая возможность очень важна для управления производственными системами, но реализовать ее трудно.

Мы будем рассматривать те операционные системы, с которыми вы в явном виде сталкиваетесь чаще всего, т.е. многозадачные универсальные операционные системы, развертываемые на обычных персональных компьютерах.

Состав операционных систем

Запуск файлов операционной системы выполняется с помощью компактной программы — ***загрузчика ОС***. Его задача — получить управление от аппаратных программ и выполнить правильную загрузку основной части ОС. Для этого загрузчик располагается в строго определенном месте диска и имеет минимальные средства для доступа к файлам ОС.

Основной частью современной операционной системы является сравнительно небольшой по объему набор файлов, составляющих ***ядро системы***. Например, для популярной ОС Windows XP это файлы: NTLDR, boot.ini, bootsect.dos, ntdetect.com, NTBOOTDD.SYS, hal.dll, system.dll, ntoskrnl.exe. Все остальное — “просто” дополнения.

Ядро системы выполняет управление процессами и распределение ресурсов. Второе название этого компонента — супервизор.

Супервизор запускается как основная программа и выполняет координацию всей деятельности и ОС, и других программ. Программный код, выполняемый на уровне ядра, имеет наивысший приоритет. В современных ОС ядро имеет право выполнить любые действия.

Ядро содержит основные системные функции, общую модель представления различных видов оборудования.

От скорости, надежности и универсальности ядра системы напрямую зависит работа всех программ.

Существенная часть задач ОС — управление аппаратным обеспечением. Для этого в состав ОС входят специальные программы — ***драйверы устройств***. Именно драйверы обеспечивают взаимодействие с устройством всех остальных программ. Применение драйверов означает, что прикладные программы не должны учитывать особенности работы конкретного устройства — это задача драйвера. Чаще всего драйверы поставляют производители оборудования. Тем не менее в составе современных ОС обычно есть большая библиотека драйверов, в том числе стандартных — разработанных производителем ОС и/или тщательно протестированных.

Современные ОС обеспечивают ***работу с файлами***.

Как вы знаете, ***файл*** — именованная область внешней памяти. Поэтому фактически эта часть операционной системы обеспечивает доступ программ к внешней памяти, выступая в качестве посредника.

В результате понятия “файл” и “файловая система” можно определить двумя способами: как объекты во внешней памяти, которыми управляет часть операционной системы, и как представление этих объектов для внешних программ. В современных ОС эти два понятия не совпадают.

Файл для компонента ОС — именованная часть внешней памяти, а файловая система — в первую очередь способ организации работы с этими частями.

Файл для программы, использующей ОС, — это именованный поток данных, с которым она работает через посредника. Файловая система для такой программы — это представление, полученное от посредника.

С помощью понятия файла ОС обеспечивают работу программ с некоторыми устройствами ввода-вывода (например, с принтерами и клавиатурой), доступ к внешним данным (например, через сеть), передают данные между программами и т.д.

Современные ОС обеспечивают прозрачную для приложений работу с файлами на разных носителях с самыми разными файловыми системами. Наиболее распространенным способом организации такой работы является ***иерархическая файловая система***.

Иерархическая файловая система предусматривает использование ***каталогов,*** т.е. списков файлов (на внешнем носителе это специальные файлы, в рамках

ОС — возможно, результат поиска и т.п.). Элементом такого списка может быть и другой каталог, поэтому создается иерархия файлов. Во избежание логических нарушений каталог не может содержать ссылки сам на себя (и через вложенные каталоги тоже). Полное имя файла в этом случае включает в себя не только название самого файла, но и каталогов, в которых он содержится.

Правила именования файлов и каталогов зависят и от базовой ОС, и от используемой на носителе файловой системы.

Такой подход к хранению позволяет организовать работу с сотнями тысяч файлов самого разного назначения.

Еще один практически обязательный компонент современных ОС — это ***средства работы с локальными и глобальными сетями***. Как и в случае с файлами, ОС выступает как посредник, предоставляя программам стандартный интерфейс для работы с сетями. Архитектура большинства современных ОС предусматривает возможность использования разных сетевых протоколов. ОС, имеющие в своем составе такой компонент, называются *сетевыми операционными системами*.

Современные ОС, предназначенные для работы с пользователями, содержат также ***средства организации пользовательского интерфейса***. Самый распространенный тип современного пользовательского интерфейса — оконно-графический. Реализуется он в разных формах: иногда — как обязательная часть ОС (например, в ОС Windows 7), иногда — как дополнительная система (комплекс X Window для Unix-систем с оконными менеджерами).

Во всех случаях наличие таких возможностей означает, что программы используют стандартные элементы при взаимодействии с пользователями и между собой. За счет этого пользователи быстро осваивают новые программы и легко ориентируются в уже известных.

Следующая необходимая подсистема современных операционных систем — это ***средства защиты.*** Средства организации защиты и операционной системы, и пользователей и их данных — это не только отдельные компоненты в составе ОС, но и существенная часть ее остальных рабочих функций. Средства обеспечения информационной безопасности — не отдельный компонент, вызываемый время от времени, а постоянно действующая сложная процедура, интегрированная в логику работы всех базовых компонентов ОС. Задача средств защиты — обеспечить надежную и бесперебойную работу пользователей, сохранность данных, соблюдение общей политики информационной безопасности.

Дополнительное системное программное обеспечение

Перечисленные компоненты — не все, что обычно входит в поставку операционной системы. Для работы с операционной системой используют множество дополнительных программ, без которых использование ОС пользователем, ее настройка и техническое обслуживание были бы затруднительны.

Среди таких дополнительных программ нужно отметить:

- ***Файловые менеджеры***. Это программы, позволяющие пользователю управлять файловой системой компьютера. Файловые менеджеры позволяют искать, просматривать, копировать и удалять файлы, выполнять групповые операции и т.п.
- ***Архиваторы***. Архиваторы — программы, предназначенные для организации компактного хранения файлов.
- ***Средства технического обслуживания и диагностики.*** В состав таких средств входят программы диагностики аппаратного обеспечения, проверки и исправления ошибок файловых систем и т.д.
- ***Дополнительные компоненты и библиотеки.*** Не все необходимые функции реализуются в стандартной поставке ОС. Некоторые из таких возможностей (например, связанные с играми) предоставляются дополнительными библиотеками функций, которые работают практически на уровне ядра операционной системы.

Помимо этих программ, современные ОС поставляются с набором специализированного программного обеспечения, решающего типовые задачи пользователя: набор текстов, просмотр и редактирование иллюстраций, средства работы с сетью.

Популярные линии операционных систем

Несмотря на то что операционных систем (как общего, так и специального назначения) разработано много, популярны и широко распространены только некоторые из них.

ОС Unix

Исторически первой универсальной и широко распространенной ОС была операционная система Unix, первоначально разработанная компанией AT&T в подразделении Bell Labs.

Основной целью разработки этой операционной системы было создание переносимой ОС. Такой, которую можно было бы без массовых переделок переносить на вновь разрабатываемые компьютеры.

В ее основе лежит несколько принципов, которые широко используются в различных ОС до сих пор. Некоторые из этих идей предлагались и ранее, но популярными стали именно в Unix:

1. Реализация значительной части функций в виде небольших программ-утилит.
2. Создание конвейеров из утилит для решения более крупных задач.
3. Применение файлов-потоков для взаимодействия с устройствами ввода-вывода и между программами.
4. Иерархическая файловая система с неограниченным уровнем вложенности.

Операционная система Unix была в основном разработана на языке С, а не на ассемблере, как это делалось раньше. Это позволило переносить ее на новые компьютеры со значительно меньшими затратами, активно развивать и дорабатывать. Именно на основе этих систем были позднее впервые реализованы средства сетевого взаимодействия на базе стека TCP/IP.

На базе этих идей и разработок появилось множество типовых операционных систем. Некоторые из них являются свободно распространяемыми.

Чтобы связать эти системы между собой, было разработано несколько стандартов, из которых первым общепринятым стал стандарт POSIX (***P**ortable **O**perating **S**ystem **I**nterface for UNI**X***). Оговаривая обязательные программные интерфейсы, этот стандарт позволяет обеспечить переносимость многих программ (если они написаны на языке C).

Unix-системы, в особенности на базе Linux, популярны для реализации серверных приложений, а также для коммуникационных и встроенных систем. С появлением большого количества доступных приложений и развитием пользовательского интерфейса эти системы стали внедряться и как ОС для персональных компьютеров.

Одним из существенных достоинств этих ОС является их модульный характер, то есть для них могут быть разработаны модули, поддерживающие самые разные устройства и файловые системы. Поскольку ядро многих систем этой линии распространяется в виде С-файлов, то оно дорабатывается и пересобирается как программа для конкретных применений.

Долгий период развития привел к появлению большого количества хорошо отлаженного и документированного программного кода, который при необходимости используется в самых разных областях.

К сожалению, оборотной стороной этих достоинств являются недостатки: многие решения в Unix-системах поддерживаются для обеспечения совместимости, добиться устойчивой работы нестандартных конфигураций и ПО довольно трудно, а настройка и поддержание такой системы в рабочем состоянии требует значительных усилий и квалификации.

ОС Windows

В отличие от операционных систем линии Unix линия Windows изначально разрабатывалась не как отдельная операционная система, а как графическая оболочка для приложений под операционную систему MS-DOS.

Разработка общих принципов построения интерфейса велась фирмами Microsoft и IBM, но позже между ними возникли разногласия и разработку продолжила Microsoft[5]. Со временем количество функций, которые самостоятельно реализовывала оболочка, стало настолько велико, а MS-DOS стала ощутимо недостаточной для работы системы. Поэтому в 1995 году была реализована полнофункциональная операционная система для персональных компьютеров — Windows 95.

Параллельно с Windows 3.1 была разработана линия операционных систем с таким же интерфейсом, но принципиально другой основой — Windows NT. В основу этой линии операционных систем были заложены несколько принципов, отличающих ее от Unix:

1. Система поставлялась в виде готовых программных модулей и не предусматривала модификации и изучения пользователями.
2. Поддержка различных устройств была вынесена в отдельно устанавливаемые программы-драйверы (в Unix многие драйверы интегрируются в ядро).
3. В ядре было выделено микроядро — особо защищенный модуль системы, обеспечивающий надежность работы.
4. В ОС была интегрирована сложная и гибкая система безопасности, на основе понятия учетных записей.
5. Изначально была предусмотрена организация корпоративной сети с единой системой управления.

Сейчас эти две линии операционных систем для компьютеров общего назначения слились в одну общую. В отличие от Unix свободных вариантов ОС Windows не имеют и являются коммерческим продуктом.

Операционные системы линии Windows используются более чем на 90% всех персональных компьютеров. Стоит отметить, что в результате конкуренции эти линии Windows и Unix многое заимствуют друг у друга (и в строении, и в функциях, и в интерфейсе), предлагают пользователям различные решения, разрабатываются и средства интеграции их в общих сетях. В конечном итоге наличие нескольких линий идет потребителям только на пользу, хоть и требует от них хорошей общей подготовки.

Прикладное программное обеспечение

Прикладным программным обеспечением называют все программы и комплексы программ, предназначенные для решения задач пользователей.

Это самый большой и разнообразный класс программного обеспечения. Перечислить все возможные виды таких систем невозможно, но некоторые мы можем назвать:

1. Офисные пакеты. Под этим названием обычно понимают наборы программных продуктов, применяемых для организации делопроизводства: текстовый процессор, электронная таблица, система подготовки презентаций, программы для работы с электронной почтой. Часто такие пакеты интегрируются в более сложные комплексы, включающие систему цифровой подписи, бухгалтерский учет и другие элементы.
2. Комплексы для издательской деятельности. Включают в себя средства подготовки макетов печатных изданий, обработки и создания иллюстраций, средства работы со шрифтами.
3. Комплексы автоматизированного проектирования. Комплексы программ, предназначенных для создания самых различных проектов, например, архитектурных и машиностроительных. Они включают в себя средства подготовки чертежей, трехмерных моделей, различных расчетов, проектирования коммуникаций, сложные системы оптимизации, стандарты.
4. Системы управления ресурсами. Предназначены для автоматизации управления организациями и включают в себя системы управления складами, закупками, ведение документооборота, прогнозирование затрат и потребностей.

[5] IBM долгое время разрабатывала собственный вариант OS/2, но в итоге от него отказалась.

5. Системы разработки программного обеспечения. Эти системы сложно отнести к какому-то одному классу программ, поскольку они используются и для модификации операционных систем. Они включают в себя средства проектирования программ, подготовки программного кода, проверку и трансляцию в машинный код, средства поиска и устранения ошибок, системы организации групповой работы над программным кодом.

Многие из перечисленных систем опираются на общие типовые компоненты, реализующие поддержку работы большого количества людей с данными или программами. Наиболее часто используемый компонент такого рода — системы управления базами данных, которые мы рассмотрим далее в данном курсе вместе с информационными системами.

Специализация компьютеров и задачи управления

Эффективная реализация тех или иных задач требует не только специализированного программного обеспечения, но и предъявляет определенные требования к применяемой аппаратуре. При этом исходят из того, что необходимые задачи должны быть решены без привлечения чрезмерно дорогостоящей аппаратуры, возможности которой все равно не будут востребованны.

Чаще всего рассматривают следующие варианты:

1. Решение задач делопроизводства. Основным требованием к таким компьютерам является высокая надежность. Им не требуются высокая производительность, большие объемы оперативной памяти, качественный вывод и ввод звука, но требуются достаточно качественное (хотя и не так быстро обновляемое) изображение на мониторе, высокая надежность хранения данных, хорошая сетевая поддержка. Такие машины, как правило, комплектуются принтерами, обеспечивающими высокое качество монохромного отпечатка при его минимальной стоимости, — лазерными и светодиодными.

2. Домашние компьютеры. Основные задачи таких компьютеров связаны с подключением довольно большого количества разнообразной периферии, с качественным воспроизведением звука, с необходимостью запуска графических программ (игровых), требующих применения графических акселераторов. Мониторы таких компьютеров должны работать быстро и соответствовать санитарным нормам, но могут неточно отражать оттенки. Домашним ПК не обязательно иметь высоконадежные крупные системы хранения данных.

3. Компьютеры для обработки графической информации. Такие машины, в зависимости от характера графики, должны иметь большие профессиональные мониторы, профессиональные графические ускорители, мощные и точные системы печати. Для решения таких задач существует большое количество специализированных периферийных устройств: различные принтеры, графические планшеты для художников и инженеров, системы цветовой калибровки. Практически всем таким компьютерам требуются мощные процессоры и большие объемы оперативной памяти. Сходные требования, с поправкой на специфику, предъявляются и к компьютерам для обработки звука.

4. Серверные комплексы. Это компьютеры, предназначенные не для непосредственного “контакта” с пользователями, а для обслуживания большого количества запросов от клиентских рабочих станций, выполнения большого количества вычислений, хранения массивов данных общего доступа. В серверных комплексах особое внимание уделяют надежности работы всей системы, оперативной памяти, высокой производительности, емкости, надежности и скорости дисковых систем. Зачастую в серверных системах параллельно работают несколько процессоров. Серверные комплексы комплектуются также дополнительным оборудованием резервного копирования информации, специальными системами электропитания и охлаждения. В то же время, такие системы вообще могут не иметь ни мониторов, ни клавиатур, ни звуковой системы, поскольку они практически никогда не используются.

Развертывание, поддержка работы самих сетей и связанного с ними программного обеспечения, модернизация сетей — сложные задачи, требующие большого количества специальных знаний и навыков.

Для управления комплексом программных и аппаратных средств, которым является даже сравнительно небольшая современная сеть, требуется выполнять целый ряд специфических операций, которые не нужны в случае “одиночной” рабочей станции.

Общее название для этого комплекса задач — системное администрирование.

К задачам системного администрирования относят:

1. Прокладку и поддержку работы каналов связи, настройку оборудования.

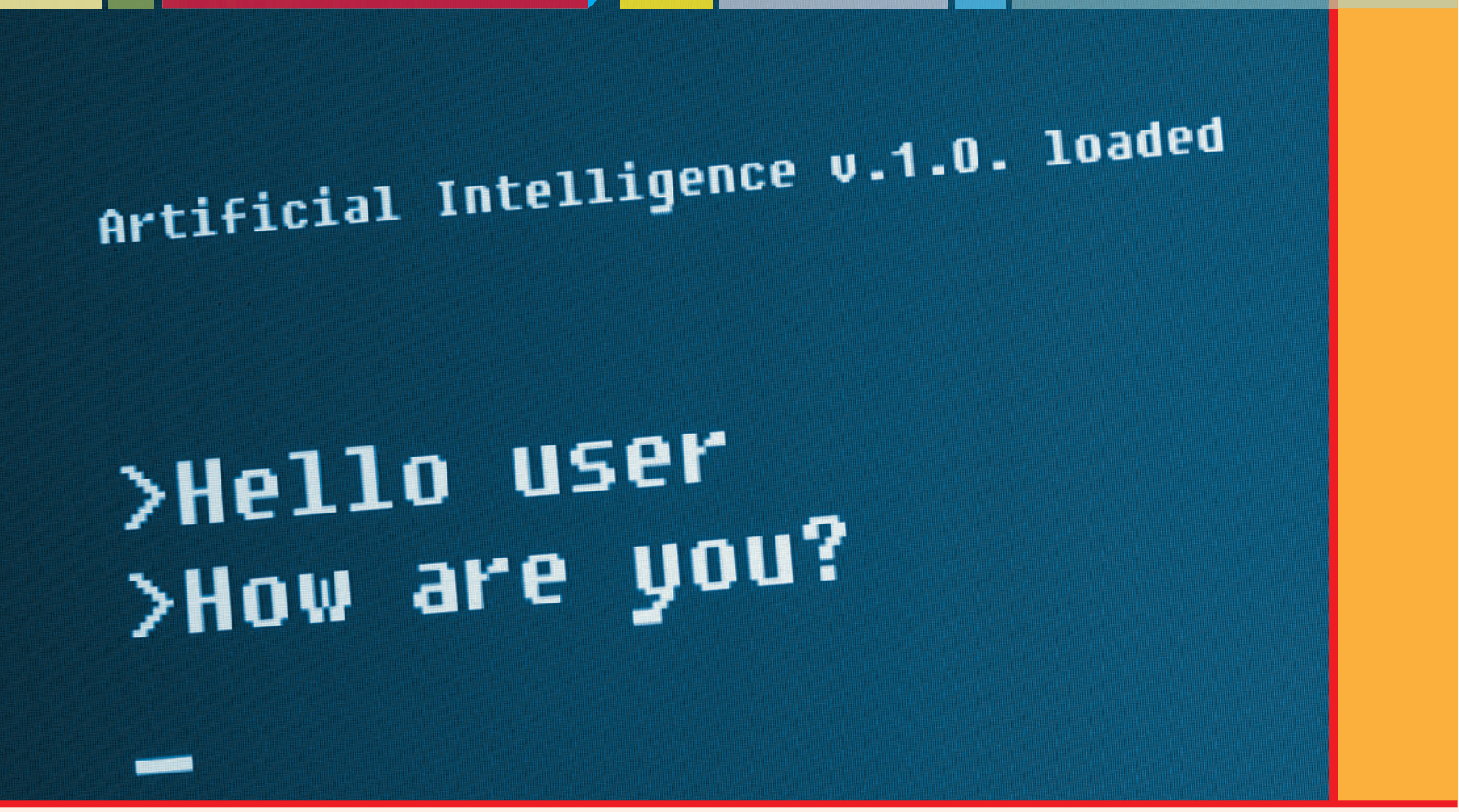
2. Установку, настройку и обновление программного обеспечения рабочих станций и машин с серверным обеспечением, управление полномочиями пользователей, контроль за работой сети в целом.

3. Администрирование прикладных серверных комплексов — обеспечение работы баз данных, web-серверов в организации, разграничение доступа к ним, технические задачи по обеспечению их работы.

4. Обеспечение безопасности информации в сети. Сюда входят планирование защитных мероприятий, выявление и устранение уязвимых мест, борьба со злоумышленниками, постоянное совершенствование систем защиты, контроль за деятельностью пользователей, организация надежной и безопасной работы с различными службами в сетях.

С формальной точки зрения к системному администрированию не относятся задачи технической поддержки (консультирования) пользователей, но в небольших организациях эти задачи тоже делегируют системному администратору.

Следует понимать, что, несмотря на все разнообразие решаемых задач, многие из принципов и средств для их решения — общие, применяемые во всех случаях.



Интеллектуальные алгоритмы и искусственный интеллект

И.А. Калинин,
Н.Н. Самылкина

Так называется глава из нового учебника информатики для 11-го класса. Наличие достаточного количества часов и уровня подготовки позволяет дать школьникам представление о некоторых результатах этого направления развития IT-отрасли, об их возможностях и ограничениях, точнее описать глобальные задачи, решаемые специалистами не один десяток лет. Не менее существенно и то, что обсуждение этого направления позволяет показать границу между автоматизируемыми (пусть и трудоемкими) процессами и неавтоматизируемыми, показать результаты, полученные при решении подобных задач. Авторы представляют содержательный материал по теме.

Интеллект и его моделирование

С самого начала автоматизации обработки информации с помощью различных технических систем ученые и разработчики столкнулись с целым рядом проблем, которые порождаются очевидным противоречием: для человека *информацией* являются годные к дальнейшему использованию сведения, причем очень разные по своей природе и дальнейшему применению. Для технической системы *информация* — это всего лишь содержание некоторого сигнала.

Методы и алгоритмы, о которых мы обычно говорим, выполняют цифровую обработку информации строго формально. Это означает, что обычная

программная система может правильно действовать только в строго определенных условиях, выполняя заранее определенную последовательность действий. Такие системы не могут решать задачи, в которых количество рассматриваемых вариантов слишком велико, возникают непредусмотренные ситуации, входные данные неполны и т.д.

Человек, ставящий и решающий задачу с помощью такой системы, вынужден "подстраиваться" под схему обработки. Это требует от него хорошего понимания методов и средств обработки, а особенно их ограничений. Между тем человека интересуют не методы, а конечный результат, который он оценивает исходя из своего определения информации — некоторых сведений, имеющих для него значение.

Ключевым отличием "человеческого" и "машинного" определений информации является трудноопределимая категория интеллекта (от лат. *intellectus* — понимание, познание) — способности развиваться, получать новые сведения самыми разными способами и применять их для своей деятельности.

Научное направление, исследующее "понимание" и связанные с ним задачи, получило название *искусственный интеллект*. Это название в русском языке — результат не совсем удачного перевода англоязычного термина *Artificial Intelligence*, который следовало перевести как искусственное понимание[1].

Ключевым отличием таких систем от обычных программных решений является то, что они решают задачи, не имеющие четкого общего алгоритма решения (например, принятия решения в меняющейся обстановке) или ранее не встречавшиеся.

Поначалу казалось, что задача создания искусственного интеллекта — чисто техническая. Поскольку вычислительные машины быстро и безошибочно выполняли операции, ранее свойственные лишь людям (причем людям образованным), то и автоматизация мышления казалась задачей вполне разрешимой.

Разработка "машинного разума" (аппаратного и/или программного) позволила бы автоматизировать огромное количество рутинных операций в производстве, научной деятельности, быту, стала бы крупным шагом на пути понимания сущности разума как такового и т.д. — перечислять перспективы можно долго.

Задача формулируется так: создать воспроизводимый искусственный объект (программный и/или аппаратный), который будет в аналогичных условиях устойчиво воспроизводить полезные результаты применения интеллекта человека.

Работы в этой области начались практически одновременно с появлением первых ЭВМ. Несмотря на то что "машинный разум" до сих пор не создан, во время исследований было получено много важных результатов.

В настоящее время к задачам искусственного интеллекта относят:

1. Задачу представления знаний, т.е. описание "понимания" человеком некоторой предметной области для последующего использования этого описания в автоматизированной обработке.

2. Задачу машинного обучения, т.е. выделения и записи общих закономерностей из наблюдений, организацию этих наблюдений. Люди учатся практически всю жизнь и обладают массой навыков, позволяющих им выделять общие закономерности.

3. Средства машинного зрения, т.е. установления соответствия между некоторым изображением и записью об объекте в базе данных. Человек (да и вообще живые существа) выполняет такое распознавание практически мгновенно. Технические системы — только в строго заданных узких ситуациях.

4. Задачу обработки текстов на естественных языках, т.е. возможность извлекать сведения и воспринимать команды на естественном человеческом языке.

5. Задачу автоматического формирования логических выводов.

6. Задачу управления средством робототехники, т.е. манипуляции объектами в реальном пространстве, перемещений в пространстве.

Интеллект в общем смысле чрезвычайно трудно определить, поэтому для систем такого рода используется не точное определение, а описательный критерий. Критерием удачи в разработке искусственного интеллекта чаще всего называют тест Тьюринга. Алан Тьюринг предложил считать искусственно созданный объект обладающим интеллектом, если его поведение нельзя отличить от поведения безусловно интеллектуального существа — человека.

Компьютер может считаться интеллектуальным, если ответит на письменные вопросы так, что человек-экспериментатор не сможет отличить его ответы от ответов человека. Такой способ определения интеллекта не является строгим. Например, один из критиков Тьюринга, Джон Серль, предложил мысленный эксперимент "Китайская комната".

Суть этого эксперимента такова: представим себе комнату с двумя окнами, в которой сидит человек, знающий только русский язык. У человека есть несколько специальных справочников, в которых написаны инструкции, и бумага для записей. Через первое окно человек получает записки с иероглифами, которые он, конечно, не понимает. Но, руководствуясь инструкциями по выделению символов и их преобразованию, он формирует новую записку с иероглифами и отдает ее во второе окно. Если у человека, отправляющего записки и

[1] Слово "интеллект" переводится на английский чаще как *brain* или *intellect*, а слово "intelligence" относится именно к рассуждениям и логическим построениям.

получающего ответы, возникнет впечатление, что человек в комнате знает китайский, то набор справочников-инструкций будет играть роль искусственного интеллекта.

Возникает вопрос — человек прошел тест Тьюринга, ничего не понимая (и не совершая интеллектуальных действий)? Или тест прошла система, состоящая из комнаты, инструкций, бумаги и исполнителя? Существует ли на самом деле смысл, или это было только манипулирование правилами записи, т.е. синтаксисом? Как подобная конструкция будет соотноситься с понятием “сознание”?

Технически попытки создать систему, проходящую тест Тьюринга, чаще всего реализуют в виде программы, отвечающей на вопросы, вводимые с клавиатуры. Таких симуляторов довольно много, один из самых знаменитых — самообучающаяся система “Элиза”. Несмотря на то что поведение таких систем программируют довольно изощренно, перепутать их с человеком все-таки нельзя.

Несмотря на все перечисленное, именно прохождение теста Тьюринга является самым распространенным критерием оценки искусственного интеллекта. Возможные решения (точнее, поскольку полных решений нет, их отдельные элементы) иногда делят на две группы.

Во-первых, *слабый искусственный интеллект* (решающий только узкие, поставленные человеком задачи). Во-вторых, *сильный искусственный интеллект*, который сам может ставить задачи, искать пути их решения, планировать свою деятельность, осознавать ее результаты и окружающий мир в целом, общаться на естественном языке и объединять все эти возможности для достижения общей цели.

Очевидно, что в принципе подобный комплекс может существовать — поскольку вполне реально существует мышление человека. Но удастся ли реализовать такие возможности в виде стабильно работающей и воспроизводимой технической системы, пока совершенно неясно.

Тем не менее в рамках разработки искусственного интеллекта было достигнуто большое количество практических результатов. Для их обозначения вводят термин ***интеллектуальные системы*** — системы программ и данных, основанные на *знаниях*.

На чем основывают такие системы? Есть несколько подходов, некоторые из которых мы перечислим.

Алгебра логики

Попытки создания максимально алгоритмизированной системы рассуждений предпринимались задолго до появления механических средств обработки информации. В первую очередь они были связаны с созданием системы записи рассуждений в отдельной научной дисциплине, получившей название ***логика***.

В середине XIX века ирландский математик и логик Джордж Буль разработал математический аппарат для решения логических задач, который получил название *алгебры логики*.

С помощью этого аппарата математики пытались создать универсальный способ проверки и вывода математических утверждений, т.е. фактически систему рассуждений, гарантированно приводящую к верному результату. В результате этих работ были получены уже упоминавшиеся нами формальные определения алгоритма (машины Тьюринга), понятия вычислимости и многие другие.

Алгебра логики была одним из основных отправных пунктов не только при создании компьютеров, но и при создании первых программ, относящихся к искусственному интеллекту, и до сих пор остается одной из основ создания и описания такого рода систем.

Механизм записи и преобразования рассуждений с помощью алгебры логики потребует от нас отдельного описания всех предикатов, известных нам относительно какой-то конкретной предметной области. Возникает несколько существенных вопросов. Всегда и во всех ли предметных областях можно описать все в виде предикатов? Как должны быть записаны предикаты, чтобы можно было не просто записывать, но и создавать и проверять рассуждения?

Попытки создать системы автоматических рассуждений, выделить и эффективно описать предикаты, автоматизировать процессы поиска доказательств привели к появлению и развитию многих направлений в развитии интеллектуальных систем.

Сейчас такие системы занимаются, например, проверкой или даже выводом математических теорем — но, конечно, создать математическую теорию “с нуля” никто таким образом даже не пытается.

Несмотря на то что такая упрощенная конструкция, как выяснилось, не может служить основой для универсальных и широкоприменяемых систем, алгебра логики — одна из очень существенных частей современных интеллектуальных систем.

Знания и их представление

Знание, как и информация, одно из понятий, которое всем знакомо. Но строго определить его очень трудно. Для изучаемой нами области основным отличием знаний является то, что они — ключевой компонент принятия некоторого решения. Определим их так: ***знаниями*** *называется информация, используемая для формирования решения некоторой задачи в заранее определенной предметной области.*

Интеллектуальная система, хотя бы в узкой области соответствующая критерию Тьюринга, моделирует деятельность человека — в первую очередь принимая решения. Таким образом, нам необходимо описать (представить) знания так, чтобы можно

было использовать их с помощью некоторого алгоритма. От того, насколько удачно будут описаны знания, напрямую зависят и возможности системы.

Для общего обозначения "технического" представления знаний используют термин "база знаний", а для дисциплины, занятой вопросами разработки таких баз, — термин "инженерия знаний".

В первую очередь для создания базы знаний нужно решить, как эти знания будут представлены, чтобы они могли быть эффективно использованы и полностью описывали предметную область.

Описывая знания (как информационные объекты, предназначенные для автоматизированной и автоматической обработки), их делят на три категории:

1. ***Факты*** — достоверные суждения о предметной области, описывающие какие-то ее аспекты (входящие в нее объекты, их свойства, связи и т.п.).

2. ***Процедуры*** — доступные алгоритмы обработки фактов для самых разных целей.

3. ***Эвристические правила*** — общие для предметной области закономерности, позволяющие принять решение. Ключевым отличием эвристики от алгоритма является то, что она позволяет сделать некоторый промежуточный вывод, но не гарантирует, что будет найдено итоговое решение. Эвристика позволяет искать решение по аналогии, используя общий подход, когда конкретного пути нет.

Эти категории должны (в рамках выделенной нами предметной области) описать все используемые "человеческие" средства поиска решения: понятия, теории (как наборы правил и понятий), способы рассуждений (т.е. способы построения новых понятий и правил) и т.д.

С помощью указанных категорий всю предметную область (возможно, более полно и точно) описывают как отдельный "мир", "пространство знаний". Такое полное описание предметной области как части реального мира, структуру, описывающую значение компонентов системы, называют ***онтологией***[2]. Онтология должна полностью описать всю область и позволить автоматизировать поиск решения.

Различают онтологии общие, описывающие реальный мир целиком (в частности, и ту часть, которая нужна для решения задачи), и специальные, описывающие только необходимое.

Рассмотрим пример такого "мира": генеалогию. Определим простейшие факты (пол и прямые родственные отношения) и набор правил, например:

(пол(A) = женский) и (A — родитель B) $\Rightarrow$
A — мать B;
(A — мать B) и ((B — родитель C) $\Rightarrow$
A — бабушка C).

Система должна выявить родственную связь (даже дальнюю) и правильно назвать ее.

[2] В философии онтология — это наука о мире в целом.

Правила могут быть и более сложными, позволяющими выявлять и достраивать недостающие и не указанные прямо факты. Например, из того факта, что человека зовут "Иван Степанович Смирнов", практически достоверно следует, что это мужчина, у него был отец по имени Степан, и скорее всего фамилия отца — Смирнов. Из записи имени и фамилии в русском языке можно точно определить пол и некоторые другие сведения[3].

Классификаций онтологий существует довольно много. Одна из них — классификация по степени формализации и способу представления знаний:

1. **Словари.** Словарь — способ придать словам точное значение, определение. В приведенном примере мы можем в форме словаря задать названия родственных отношений. Словари могут быть однозначными (словари идентификаторов) и словарями определений (яркий пример — толковый словарь).

2. **Тезаурусы.** Тезаурус — специальный вид словаря, устанавливающий *отношения* между словами, например, синонимии (сходного значения), вхождения и т.п.

3. **Классификации** (таксономии). Классификация — способ разбить все множество объектов на вложенные друг в друга классы[4]. При этом вложенный класс обладает свойствами класса-родителя и своими собственными. Классификации важны еще и потому, что зачастую наши рассуждения — это и есть предположения о принадлежности предмета к классу.

4. **Семантические сети.** "Мир" описывается как система связанных объектов. В нашем случае такими связями будут отношения родства.

5. **Логические модели.** Здесь мы описываем знания в виде некоторых логических формул, задающих разрешенные преобразования. Так часто описывают математические теории.

6. **Фреймы.** Фреймы — структура, позволяющая описать не только взаимодействие объектов, но и объекты, состоящие из разных частей. При этом совершенствуя описание части, мы совершенствуем и все объекты, в которые она входит.

Развитые и удачные онтологии — необходимый компонент самых разных интеллектуальных систем: поиска в большом объеме текста на естественных языках, перевода, поиска правил и закономерностей и т.д. Фактически онтологии — способ передачи *смысла*.

Первые три из перечисленных видов онтологий были созданы и активно используются для описания и передачи самых разных знаний и без применения вычислительной техники — в учебной литературе, научных трудах и т.п. В частности, класси-

[3] С привлечением средств извлечения фактов, описанных в разделе "Обработка текстовой информации", созданные продуктивные правила могут стать чрезвычайно мощным статистическим инструментом.

[4] В частности, популярный вид классификатора — всевозможные рубрикаторы.

фикациями и словарями мы активно пользуемся и в этом учебнике.

Моделей представления знаний (т.е. подходов к их описанию) также существует несколько. Одна из самых популярных — ***продукционная модель*** представления знаний. В этой модели знания представляются в виде правил (продукций) такого вида:

Если (условие) то (действие).

Набор таких правил, включающий в себя факты (правила, не имеющие условий), и будет представлять описание какой-то области в виде заранее принятых решений.

Модель названа продукционной, поскольку она продуцирует, порождает решения.

Вот пример такого правила, записанного на языке описания экспертных систем CLIPS. Оно позволяет определить, кому из имеющегося списка людей человек является дядей:

```
(defrule isUncle
 (isParent (parentMember ?c) (childMember ?a)) ; Если c — родитель a
 (link (linkName brother)(sideA ?b) (sideB ?c)) ; Если между b и c есть связь “брат”
 =>
 (assert (link (linkName uncle)(sideA ?b)(sideB ?a))); Создать связь “дядя” между b и a
 (printout t ?b “uncle” ?a crlf); Вывести сообщение об этом
)
```

Эта модель представления знаний — самая популярная, но имеет ряд ограничений:

- при описании большой и сложной области возникают противоречия между правилами, которые не всегда удается разрешить;
- на больших базах знаний подход оказывается неэффективным (например, появляется слишком много равновероятных решений);
- такие наборы правил трудно применять в ситуациях, которые не предусмотрены разработчиком системы. Например, такая система в реальном мире никогда не “догадается” о том, что вода может перелиться через край сосуда.

Приведенный нами пример системы определения родства в такой модели будет состоять из базовых утверждений (фактов) и правил, порождающих новые утверждения. Фактически именно этим подходом мы воспользовались при описании задачи.

Второй подход — построение ***логических моделей***.

Логическая модель подразумевает, что знания описываются опять-таки в виде набора фактов и связывающих их логических соотношений, записанных в виде формул.

Машина обработки может порождать новые факты и формулы, используя метод резолюций: когда берется итоговое утверждение и доказывается методом “от противного”.

Достоинство логической модели в том, что найденное с ее помощью решение — гарантированно правильное. Недостаток — поиск решения может быть серьезно затруднен отсутствием нужных утверждений-преобразований.

Пример такой экспертной системы — система для исследования математических теорий AM (Автоматический математик). Это разработка доктора Д.Лената (Стэнфордский университет) для исследовательских и учебных целей. В систему AM первоначально было заложено около 100 правил вывода и более 200 эвристических алгоритмов обучения, позволяющих строить произвольные математические теории и представления. Сначала результаты работы системы были весьма многообещающими. Она могла сформулировать понятия натурального ряда и простых чисел. Дальнейшее развитие системы замедлилось, и было отмечено, что система не может породить принципиально новое решение, но может обнаруживать и доказывать некоторые теоремы в рамках существующих теорий.

От описанных нами ранее систем на основе математической логики эта система отличалась в первую очередь самим механизмом выводов — и пополнения базы знаний. Для поиска использовались не только строго формальные рассуждения, но и *эвристические* правила, то есть правила, позволяющие предположить, а не гарантировать результат.

Третья модель представления знаний — ***семантическая сеть*** — структура данных, в которой отдельные понятия-элементы связываются между собой связями, отражающими их взаимоотношения.

Например, в приводившемся примере экспертной системы по определению родства такой связью будет связь “является родителем” и “пол”. Семантической сеть называется потому, что отражает смысл отношений между объектами.

Семантическая сеть также имеет свои недостатки. В частности, если есть две сети, описывающие пересекающиеся области, нам будет очень трудно их объединить или хотя бы определить, для чего сеть предназначалась.

Четвертый вид — ***фреймовые*** модели.

Во фреймовой модели каждый объект — “рамка”, объединяющая “слоты” — места соединения фреймов. Фреймы могут порождаться от других фреймов, наследуя их слоты. В нашем примере исходным слотом был бы объект “человек”, имеющий слоты “пол”, “мать” и “отец”.

Фреймовая модель позволяет описать сложный “мир”, но трудно разработать эффективную машину обработки данных, т.е. принятия решений.

Экспертные системы

Одна из ключевых возможностей человеческого мышления применительно к любой сложной системе — способность, опираясь на знание общих закономерностей и представление об аналогичных системах, быстро предсказать ее поведение или выявить его причины (например, поломку), предло-

жить более удачную конструкцию, выполняющую аналогичные функции, и т.п.

Человек, способный на такую деятельность в некоторой предметной области, традиционно называется экспертом. Эксперт высокого класса — редкий специалист, которого практически невозможно просто "выучить", поскольку результативность его работы нельзя объяснить только большим количеством запомненных им фактов.

Автоматизация такой деятельности требует глубокого понимания процесса поиска и принятия решения и выполняется чаще всего в виде специального программного комплекса — *экспертной системы*.

Обычно экспертная система состоит из следующих частей:

1. *Конкретные факты*. Это, как мы уже говорили, достоверные утверждения — в данном случае о текущей ситуации. В используемом нами примере это конкретные факты о людях и их связях.

2. *База знаний*, чаще всего реализованная в виде правил. Это знания об общих фактах, возможных вариантах развития событий, связях, названиях и т.д. Вполне возможно, что для решения конкретной задачи большая часть этих знаний не потребуется. Важно понимать, что знаний может оказаться *недостаточно* для решения задачи, т.е. в отличие от алгоритма никакой гарантии решения дать нельзя.

3. *Машина логического вывода*. Этот механизм, используя имеющиеся факты и знания, должен выработать некоторое решение — ответ на вопрос.

В основном машины логического вывода бывают:

• *прямого вывода.* Эти системы получают на входе все имеющиеся факты для рассуждений и применяют к ним правила, порождая все новые и новые факты, пока не дойдут до ответа. Такие системы часто применяются в диагностике;

• *обратного вывода.* В таких системах исходно берется готовый вывод — гипотеза. Дальше с помощью правил отыскиваются порождающие ее факты и сверяются с имеющимися. Цель таких систем — выявление возможных причин.

Экспертные системы и языки их представления не могут опираться на обычный процедурный подход — слишком много вариантов придется рассматривать, а в некоторых случаях прямая алгоритмическая обработка (точнее, заранее предписанная последовательность действий) невозможна. Экспертные системы опираются на декларативный подход, при котором мы не указываем процедуру обработки, а описываем желательный результат.

Простейшие экспертные системы основываются на пропозициональной логике — способе рассуждений, опирающемся на привычную для нас двоичную логику. Чтобы записывать факты и правила, в таких системах используется также теория множеств с операциями-кванторами "Все" и "Некоторые".

Такие системы трудно применять в ситуациях слабоструктурированных задач, в которых приходится иметь дело с утверждениями вроде "в основном", "иногда", "редко" или "почти всегда". Для таких случаев в современных системах реализуется ***нечеткая логика*** — логика, в которой указывается степень истинности утверждения.

Немаловажно и то, что в реальном мире редко удается точно указать на единственную причину какого-то явления. Имеющиеся факты скорее всего будут давать несколько разных вариантов решения, но позволят приписать им разные вероятности. В таких системах реализуются правила, влияющие на вероятность тех или иных гипотез.

Еще одна важная проблема, которую приходится решать в современных экспертных системах, — это проблема неполноты данных. В идеальном мире можно было бы принимать решение только после того, как получены все необходимые для этого данные, но в реальном мире это возможно далеко не всегда. Поэтому часто реальные экспертные системы учитывают необходимость работы в условиях неполных данных. Один из механизмов работы в таких условиях — это сравнение ситуации с некоторым шаблоном.

Главной частью, от которой зависит работа экспертной системы, является база знаний. Именно от ее полноты и соответствия задаче будет зависеть, справится ли экспертная система со своей задачей или нет.

Создание таких баз знаний или возможности их создания — одно из самых современных направлений развития информатики. Наличие универсальной базы знаний о мире позволило бы решить массу интеллектуальных задач, в первую очередь — задач поиска, перевода, анализа закономерностей, выявления противоречий и т.д.

Приведенный нами пример — это узкие, специальные онтологии, которые описывают небольшой, замкнутый "мир". Для работы в условиях реального мира этого явно недостаточно, поэтому существует несколько проектов создания общих, всеобъемлющих онтологий. Например, несколько общих онтологий и способов их описания создается для направления Semantic Web — интеллектуальных систем обработки данных в сети. В качестве примеров можно привести:

• Дублинское ядро: базовый набор элементов для описания метаданных, т.е. "данных, описывающих данные", в частности, их связи в виде URL-ссылок. Вот текущее содержание ядра:

- Title — название;
- Creator — создатель;
- Subject — тема;
- Description — описание;
- Publisher — издатель;
- Contributor — внесший вклад;
- Date — дата;
- Type — тип;
- Format — формат документа;
- Identifier — идентификатор;

- Source — источник;
- Language — язык;
- Relation — отношения;
- Coverage — покрытие;
- Rights — авторские права.

• Проект CYC (читается как “ЦИК”, от слова “энциклопедия”), описывающий правила здравого смысла. В настоящий момент в базе более трех миллионов таких правил и фактов[5].

• Например: *(#$genls #$Tree–ThePlant #$Plant)* — это правило “все деревья являются растениями”.

По замыслу разработчиков наличие таких связей позволит выполнять в сети интеллектуальный поиск на языке близком к естественному (например, агрегировать данные из разных источников), обнаруживать отдаленные связи (т.е., даже не имея достаточного количества фактов, отвечать, например, на вопрос о вкусе салата из кумкватов или на вопрос, могут ли расти карандаши) и т.д.

Самообучающиеся технические системы

Одно из важнейших преимуществ человеческого интеллекта — возможность обучения и самообучения. Человек учится на протяжении всей жизни, получая знания направленно или в результате анализа текущей деятельности. Люди разработали целый ряд систем и подходов изучения окружающего мира.

Создание алгоритмов, реализующих в технических системах хотя бы часть этих возможностей, — чрезвычайно важная и сложная задача.

Приведем еще несколько алгоритмов, которые позволяют пополнять и формировать базы знаний.

Алгоритм APRIORI предназначен для выявления ассоциативных правил в больших объемах информации. Фактически этот алгоритм позволяет обнаруживать закономерности, шаблоны. Он был разработан для анализа данных о покупках.

В основе этого алгоритма лежит очевидный факт: если набор из трех покупок встречается пять раз, то набор из двух любых покупок этого набора встречается не менее пяти раз.

Количество упоминаний товара (или набора товаров) называется в алгоритме *поддержкой*.

Значит, мы можем искать часто встречающиеся типовые наборы автоматически (лат. *a priori* — до опыта), отбрасывая те покупки, которые встречаются редко. Будем для определенности считать, что “редко” — это меньше трех раз.

На псевдокоде алгоритм можно записать примерно так:

```
List = getItemsWithPrefix(null,3); Получить все
элементы с поддержкой не менее 3
Do
ResultList = null
for (p in List); Перебрать все элементы списка
tempList = getItemsWithPrefix(List[p],3); Все
элементы с префиксом
if not (tempList isNull)
   flag = true
      for (i in tempList)
         resultList = resultList + tempList[i]
   else
       resultList = resultList + List[p]; Нет
более длинных, но терять "серию" нельзя
  if not (resultList is null); Удалось ли найти
более длинные покупки?
      list = ResultList
   while not (resultList is null);
   for (I in List)
      Show(List[i])
```

Этот вариант алгоритма поиска покупок не оптимальный (например, каждый раз проверяются уже “заблокированные” покупки), но основную идею иллюстрирует.

Следует отметить, что найденные подобным образом ассоциативные правила необходимо оптимизировать — иначе их будет слишком много и они станут противоречить друг другу.

Полученный набор правил можно дополнительно обработать для выявления общих характеристик. Например, правило может быть выражено фактом “Все живые деревья растут” или подмеченной закономерностью[6].

Аналогично этому можно построить алгоритмы выделения последовательных шаблонов — не просто встречающихся наборов, а именно последовательностей действий клиента, типового набора покупок клиентов разных категорий[7] или, например, для предсказания сезонного спроса или типовой последовательности покупок.

Мы уже рассматривали некоторые алгоритмы, которые выполняют интеллектуальные действия, например, поиск закономерностей и зависимостей, классификацию и т.д.

Простейший обучающийся алгоритм формирует так называемое “дерево решений”. Дерево решений — способ представления цепочки принятия решения в виде шагов, на каждом из которых мы выбираем из фиксированного количества вариантов один. Реализуется такая структура с помощью структуры данных “дерево”.

Достоинством деревьев решений является их интуитивная ясность и возможность подробно объяснить процесс принятия решения.

---

[5] Существует ее доступная версия: *http://www.opencyc.org*.

[6] Программа “Эвриско”, написанная Дугласом Ленатом для подбора оптимального состава космического флота в коллективной игре, проверяя разные наборы кораблей, выявила закономерность “Крайние случаи часто эффективны”. Применив ее, программа выяснила, что флот с крохотным невооруженным “кораблем” гарантировал ничью — в такой корабль по формальным признакам оказалось невозможно “попасть”. Флот выиграл чемпионат “всухую”.

[7] По удивительному совпадению все крупные торговые сети активно собирают данные о своих клиентах и стараются постоянных клиентов идентифицировать, например, выдавая им дисконтные карты в обмен на заполненные анкеты.

Дерево можно строить на основании ответов пользователя, а при наличии данных о решениях и параметров — построить его автоматически. Одним из алгоритмов, который строит такое дерево, является **алгоритм CART** (***C**lassification **a**nd **R**egression **T**ree* — дерево классификаций и регрессий).

Алгоритм строит бинарные деревья решений, которые содержат только два потомка в каждом узле. В процессе работы происходит рекурсивное разбиение примеров обучающего множества на подмножества, записи в которых имеют одинаковые значения целевой переменной.

Выстраивая дерево, алгоритм CART проводит для каждого узла полный перебор всех атрибутов, на основе которых может быть построено разбиение, и выбирает тот, который максимизирует значение показателя

$$Q(s|t) = 2P_L P_R \sum_{j=1}^{N} (P(j|t_L) - P(j|t_R)),$$

где s — идентификатор разбиения, t — идентификатор узла, t_L и t_R — соответственно левый и правый потомки узла t, P_L и P_R — отношение числа примеров в левом и правом потомках к их общему числу в обучающем множестве, $P(j \mid t_L)$ и $P(j \mid t_R)$ — отношение числа примеров класса j в левом и правом потомках к их общему числу вариантов в каждой ветви, N — общее количество примеров.

Если пытаться перевести эту формулировку с математического на "человеческий" язык, речь пойдет о том, что, имея множество объектов, мы будем подбирать критерий деления так, чтобы на каждом этапе множество разделилось на две примерно равные части. Повторяя эту операцию рекурсивно, мы построим дерево разбиения — и сможем принимать решение для тех объектов, которые в учебный набор не входили.

Например, подобные деревья используются для принятия решения о выдаче кредита в банках.

Деревья анализа вариантов

В целом ряде случаев дерево возможных решений может быть построено заранее. Особенно часто такая задача возникает в играх, т.е. в процедурах со строго определенными правилами. Яркий пример такой игры — шахматы.

В виде таких игр, часто со сложными правилами и большим количеством участников, моделируются экономические процессы, боевые столкновения и т.д. Изучением их занимается специальный раздел прикладной математики — *исследование операций*. Существенная часть этого раздела, оказавшая огромное влияние на современную экономическую и научную деятельность, так и называется — *теория игр*.

Самый наглядный и частый способ описания пошаговой игры — это построение и изучение всех возможных вариантов действий участников. Перебор всех вариантов часто показывают в виде ***дерева решений***.

Дерево решений (т.е. полная запись всех возможных вариантов ходов) в большинстве случаев чрезвычайно велико. Выписать его целиком и решать задачу полным анализом всех возможных вариантов — невозможно. Требуется метод, который оптимизирует организацию перебора, заранее отбрасывая неудачные варианты, хотя бы частично.

Один из методов, применяемых для этих целей, — метод альфа-бета отсечения (он же — метод ветвей и границ).

Метод построен на моделировании игры двух участников, которые принимают решения по очереди и имеют противоположные цели. Каждый шаг игры (ход) состоит из двух полуходов.

Игроков часто называют Max (ходит первым) и Min (ходит вторым). Выигрыш первого игрока, максимально возможный с его точки зрения, — минимальный для второго игрока.

В каждой позиции (полуходе), обозначенной p, для игрока есть несколько разрешенных ходов, обозначим это количество d. Если полуход — терминальный (т.е. игра заканчивается), то выигрыш игрока Max можно обозначить как $F(p)$ — функция выигрыша в текущей позиции. Для игрока Min результат тогда окажется $-F(p)$, поскольку он выигрыш теряет.

Потенциальный выигрыш игрока Max на полуходе p можно определить функцией:

$$F(p) = \begin{cases} F(p); d = 0; \\ \max(F(p_1), F(p_2) \ldots F(p_d)); d > 0. \end{cases}$$

Для игрока Min аналогичная функция на этом полуходе $G(p) = -F(p)$.

Перебрать все ходы невозможно, поэтому мы будем стараться отсекать те ходы, результат которых нам уже известен (например, позиция уже анализировалась раньше и мы знаем наилучший доступный результат). Тогда очевидно, что первый же найденный результат с таким же или лучшим значением — искомый. Для улучшения вводят еще и нижнюю границу, чтобы не анализировать гарантированно "плохие" результаты.

На псевдокоде это можно описать так:

```
Function AlphaBeta(p,alpha,beta)
   PositionList = GetList(p); Получить список
всех доступных ходов на текущем полуходе
    If Length(PositionList) = 0 then
       Return f(p)
    Else
       m := alpha ; Минимальный результат
       for i := 1 to Length(PositionList)
          t := – AlphaBeta(PositionList[i],
               -beta, -m); Следующий полуход –
               обмен местами
          if t > m then
             m := t;
          if m >= beta then
             return m;
```

Анализ дерева начнется с вызова AlphaBeta(1, –4, +4), постепенно улучшая оценку.

Основная идея этого алгоритма — отбрасывать варианты, которые заведомо хуже уже найденного. Эффективность алгоритма зависит от расположения элементов, но в среднем позволяет увеличить глубину просмотра (при таком же времени) в два раза.

Еще одна возможность, присутствующая в этом алгоритме, — не выписывать заранее всего дерева в целом, ограничиваясь только узлами, необходимыми для обработки (в приведенном псевдокоде эта возможность задана функцией GetList). В целом ряде игр дерево вообще не может быть выписано из-за своего размера.

Эта стратегия улучшает перебор, но все равно требует анализа очень большого количества перебираемых вариантов. Поэтому в реальных задачах ограничивают глубину перебора (например, все варианты с глубиной больше 6 объявляют терминальными), а для вычисления функции F при этом используют оценочный подход (подбирают оценку, стремясь сделать ее наиболее точной и минимизировать последствия ошибок).

Шахматная программа Deep Fritz, выигравшая у чемпиона мира по шахматам[8], использовала анализ на глубину до 20 ходов, а для оценки — специально разработанную функцию оценки, базу данных партий и заранее проанализированных позиций.

Существуют игры, в которых такие алгоритмы применить не удается, и программы “играют” только на уровне среднего любителя — например, игра “Го”[9].

Нейронные сети и генетические алгоритмы

Все рассмотренные нами алгоритмы и подходы опирались на анализ и использование готовых данных, т.е. на принятие решений на основе информации, которая специально обработана и представлена в соответствующей модели. Фактически рассмотренный нами подход — это *моделирование процесса мышления*.

Но в реальном мире в готовом виде знания не встречаются — нужно получить и обработать информацию, чтобы их сформировать. Задача организации восприятия и анализа информации из окружающего мира — одна из самых сложных и общего решения пока не имеет.

Наибольшее внимание уделяется задаче ***оптического распознавания***, т.е. задаче сопоставления с достаточным уровнем достоверности изображению набора записей из некоторой базы данных (фактически нужно перечислить все, что находится на некоторой картинке). Еще один вариант такой задачи — оценка степени визуальной схожести двух и более объектов.

Для некоторых узких классов таких задач решения существуют, например, для задачи оптического распознавания символов, лиц, снятых в определенных условиях, задачи распознавания речи (которая решается для очень ограниченного набора слов любого человека или чуть более широкого набора конкретного человека, т.е. после обучения).

Но, например, не существует системы искусственного интеллекта, способной по торчащему из-под кресла хвосту предположить там наличие кошки.

Чтобы решать подобные задачи, применяют как уже приведенные нами подходы, так и подходы, разработанные в рамках второго направления развития систем искусственного интеллекта — *моделирования работы мозга*.

В рамках этого подхода наилучшие результаты получены с использованием *нейронных сетей*. Нейронная сеть (точнее — искусственная нейронная сеть) — это набор программных объектов, каждый из которых имеет входы (переменные, значения которых устанавливаются извне), выходы (переменные, значения которых объект формирует сам) и некоторую функцию внутри (процедуру формирования из входных значений выходных). Выходные значения одного нейрона можно подать на вход другого — и тогда они будут связаны. Такой метод работы очень похож на электрохимические процессы в головном мозге.

Построив нейронную сеть, проводят процедуру ее обучения — подают данные на вход и сравнивают результаты на выходе. Совпадающие результаты подкрепляются, различающиеся — штрафуются. Для нейронной сети это означает изменение параметров преобразования в нейронах (например, ослабление влияния некоторых параметров на результат).

Обученная сеть может находить решения с достоверностью 80–90%, а иногда и выше, в задачах, в которых аналитические методы не дают уверенного результата. Нейронные сети распознают зрительные образы, предсказывают колебания курсов валют и акций и т.д.

Но результат работы нейронной сети (даже небольшой) чрезвычайно трудно объяснить — потому что ее работа фактически моделирует биологические и электрохимические процессы в мозге[10], а не процесс логического мышления. Это означает, что знания, представленные в виде нейронной сети, очень трудно перевести в вербальную описательную форму.

Еще один подход, позволяющий найти решение в не совсем ясной ситуации, — *генетические алгоритмы*.

[8] Декабрь 2006 года, 6 партий с Владимиром Крамником, счет 4–2.

[9] Известный школьный вариант этой игры — “Точки”.

[10] Принятие решений муравьем обеспечивает надглоточное нервное кольцо, в котором всего четыре нейрона. Из курса биологии вам известно, какие возможности это дает. Причем поведению большого сообщества муравьев вообще не удается дать достоверное и полное объяснение.

Этот подход применяется для поиска оптимального решения, когда ясны параметры и критерий сравнения решений, но не понятно, как среди всего множества возможных решений найти самое лучшее.

Генетические алгоритмы моделируют процесс эволюции в биологии, когда в составе популяции живых организмов выживают наиболее приспособленные, становясь лучше в результате конкуренции и естественного отбора. Поэтому и терминология в этой группе методов очень похожа на биологическую.

Для нахождения оптимума каждое возможное решение (например, возможный вид формулы), описанное как точка в пространстве решений, называют *особью*. У особи есть *генотип*, например, формула, состоящая из частей — *хромосом*. При поиске решения алгоритм моделирует эволюцию в составе *популяции* — набора особей. Популяция возможных решений (неизвестно — оптимальных или нет) сравнивается по *фенотипу* — по проявлениям оцениваемых параметров.

Вероятность того, что решение продолжит развиваться, пропорциональна его оптимальности, вычисленной с помощью функции оценки.

Для порождения следующего "поколения" решений их скрещивают — организуют обмен хромосомами и вносят мутации, т.е. некоторые решения немного изменяют в случайном порядке.

После нескольких итераций (например, нескольких сотен поколений) в популяции неудачные решения "вымрут", зато появятся или "выживут" эффективные. Возможно, не абсолютно, но достаточно эффективные для заявленных целей.

Базы знаний в различной форме используются в большом количестве разных систем. Чаще всего они применяются в *системах поддержки принятия решений*, т.е. в информационных системах, которые используются для организации управления.

Даже если само решение принимает человек, то система, выявляющая и представляющая знания, способна значительно ускорить этот процесс, спрогнозировать результаты того или иного решения, представить в компактной форме обоснование решения, уменьшить вероятность ошибочного решения.

Развивая интеллектуальные системы, ученые и разработчики стремятся решить целый класс общих задач, например:

1. Перевод. Решение задачи адекватного перевода текста с одного естественного языка на другой требует представления о предметной области (например, из-за неоднозначности слов в художественных текстах нужно не просто выбрать нужное, но и сохранить эту многозначность, не добавив к ней лишнего). Ключевым понятием в переводах является понятие *смысла*, т.е. того представления, которое текст формирует.

2. Автоматическое планирование, задача которого — составление эффективного плана действий, распределения времени и ресурсов для всех участников. Актуальное направление — для военных целей.

3. Автоматическое управление. Эти средства предназначены для непосредственного управления поведением агентов в изменяющихся условиях. Например, это системы автоматического управления автомобилями, поездами, судами и т.д.

4. Общение на естественном языке и интеллектуальный поиск.

5. Автоматическая классификация.

Стоит заметить, что в этом научном направлении значительную роль играют философские вопросы, поскольку оно фактически пытается ответить на вопрос о сути человеческого мышления, его природе и возможностях. Получение устойчиво работающего "сильного" искусственного интеллекта, помимо полного изменения производства и науки, оказало бы огромное влияние практически на все сферы деятельности человечества и изменило бы самые основные его представления о самом себе.

Использованные источники информации

1. *Джарратано Дж., Райли Г.* Экспертные системы, принципы разработки и программирования. / Пер. с англ. М.: ИД "Вильямс", 2007.

2. *Рассел С., Норвиг П.* Искусственный интеллект: современный подход. М.: ИД "Вильямс", 2007.

3. *Лапшин В.А.* Онтологии в компьютерных системах. М: Научный мир, 2010.

4. *Сегаран Т.* Программируем коллективный разум. М.: Символ-плюс, 2008.

МИР ИНТЕРНЕТА

Как устроен и работает Интернет

В.А. Юнев,
эксперт по стратегическим технологиям, Microsoft, Москва

Сейчас трудно найти человека, который не знал бы, что такое Интернет. Но очень мало кому известно, что послужило прототипом глобальной паутины. В конце 60-х годов прошлого века была создана сеть под названием ARPANET. Именно с нее берет начало современный Интернет.

История

История Интернета начинается с тех времен, когда компьютеры были огромными, они занимали целые этажи, и их было немного. В те времена только-только начала возникать потребность в обмене информацией между компьютерами, разнесенными физически, то есть находящимися порой в разных городах.

Задача, которая ставилась перед создателями сети, состояла в том, чтобы избавиться от технических ограничений, существующих в то время. В конце 60-х годов к центральному компьютеру подключались, работая за удаленным от него терминалом, для чего требовалось прямое физическое подключение между рабочим местом и компьютером. Очевидным минусом этой схемы является высокая уязвимость, когда потенциальный противник мог нарушить коммуникации, что полностью лишало связи с компьютерами, которые в то время считались стратегическими ресурсами.

В ответ на потребности своего времени начались исследования по созданию распределенных сетей между удаленными физически компьютерами на основе пакетной коммутации. Пакетная коммутация и соответствующие протоколы передачи данных предлагают отделение физической реализации от логических концепций сети. Доставка информации больше не зависит от типа содержимого. Пакеты с данными имеют полную информацию об источнике и точке назначения, что позволяет доставить пакет информации точно по адресу с помощью промежуточных узлов, даже не имея представления о количестве и физической реализации таких узлов.

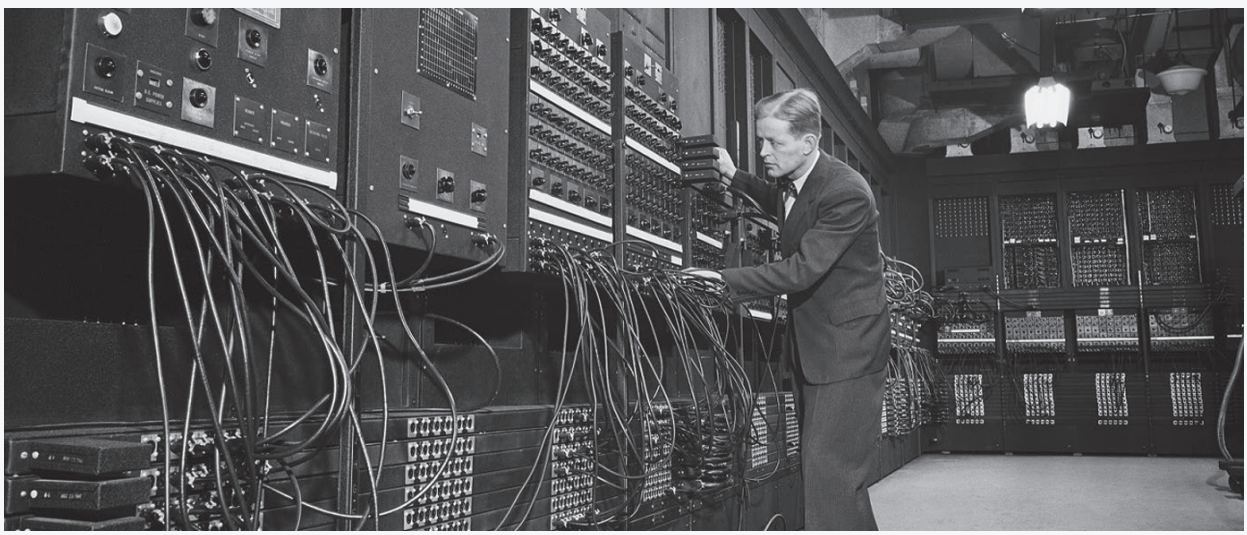


Рис. 1. Один из первых компьютеров в сети ARPANET

После череды разных экспериментов в конце 60-х годов XX столетия организация под названием DARPA (Агентство передовых оборонных исследовательских проектов США) начала исследования по созданию компьютерной сети и стала делать первые шаги, которые привели в будущем к созданию того, что сегодня мы называем Интернетом.

Сеть, созданная организацией DARPA, получила название ARPANET и, как многие высокотехнологичные продукты своего времени, предназначалась для военных целей. Первоначально сеть объединяла исследовательские институты, по сути одни из тех немногих мест, где находились компьютеры тех лет. Сеть ARPANET стала быстро расти, к ней присоединялось все больше компьютеров и организаций, все больше ученых стали пользоваться сетью для своих научных проектов.

Большим шагом вперед в развитии сетей стала маленькая программа mail, написанная в Массачусетсском технологическом институте (MIT) в 1965 году, которая позволила пользователям обмениваться сообщениями. "Почтовыми сообщениями" — как говорим мы сегодня.

Следующим шагом в развитии ARPANET стало объединение в сети компьютеров с европейского континента. Первые организации из Великобритании получили доступ к сети по трансатлантическому телефонному кабелю в начале 70-х годов XX века.

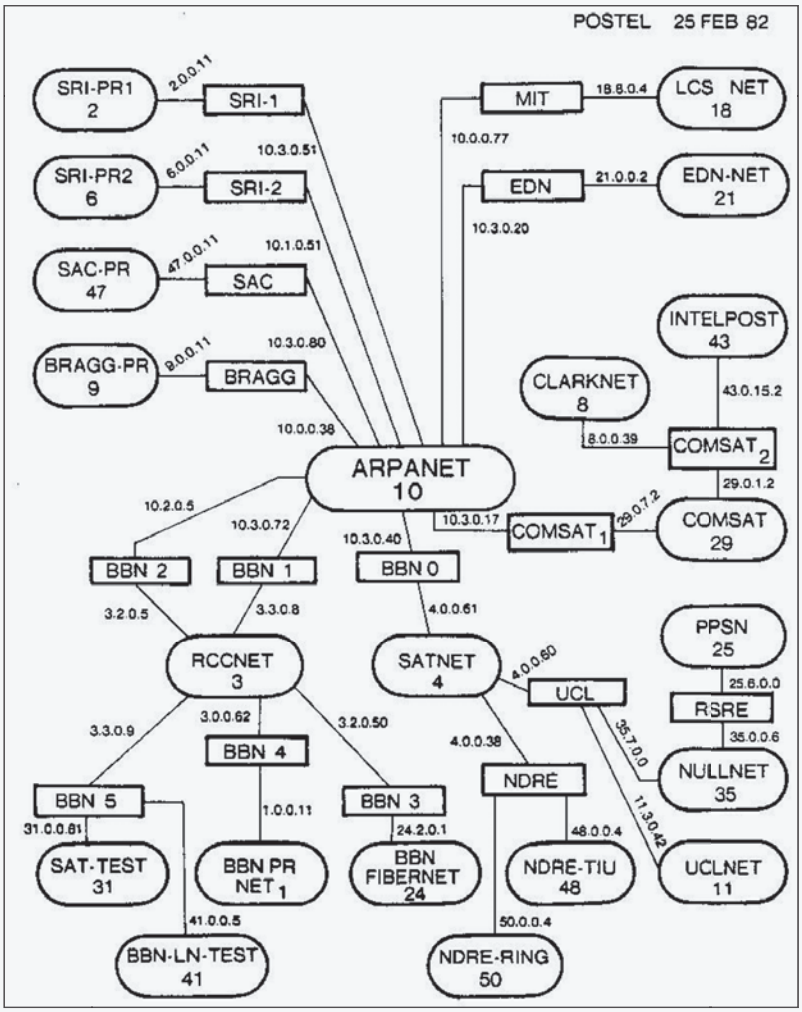


Рис. 3. Карта сети ARPANET 1982 года. Источник — *Wikipedia.com*

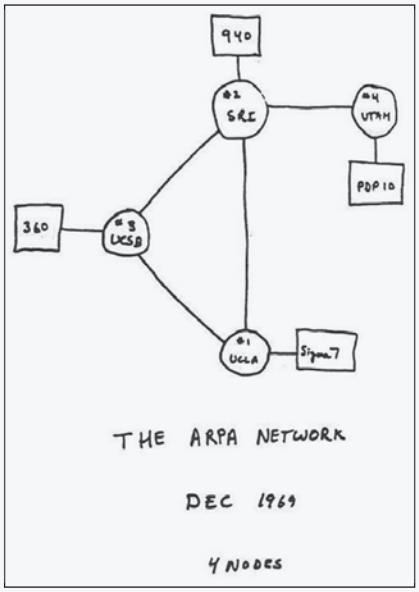


Рис. 2. Карта сети ARPANET в декабре 1969 года

DNS и адреса в сети

В начале 80-х годов для определения компьютера в сети ARPANET ему присваивался определенный номер (или адрес), состоящий из четырех шестнадцатеричных чисел. Этот адрес уникально идентифицировал компьютер в сети, что позволяло компьютерам обмениваться пакетами данных между собой.

Например, на *рис.* 3 наборы цифр вида 2.0.0.11 или 18.8.0.4 являются одними из первых адресов компьютеров, распределенных в сети ARPANET. MIT = 10.0.0.77 — такая адресация и форма записи адреса компьютера в сети характерна для протокола обмена данными под названием TCP/IP, который был адаптирован в сети ARPANET в начале 80-х годов. Нужно заметить, что и в наше время TCP/IP является основой современного Интернета и обмена данными по сети.

Однако понятно, что человеку сложно оперировать адресами в числовом представлении. Вероятно, в самом начале пользователи сети могли удержать в памяти соответствие компьютера института MIT адресу 10.0.0.77. Но со временем, с ростом ARPANET и увеличением количества ком-

пьютеров в сети, подобные вещи запомнить стало невозможно.

Так как человеку проще запоминать названия, то всем компьютерам в сети можно присвоить уникальные имена (настоящие или кодовые), а потом по дружелюбному к пользователю имени определять адрес компьютера в сети для обмена информацией. Однако была проблема в том, как организовать такое соответствие имен в сети.

Первоначально для решения проблемы было предложено использовать единый файл HOSTS TXT, в котором вручную были прописаны все соответствия имен компьютеров их адресам. Такой файл лежал в строго определенном месте на определенном компьютере в сети (института Stanford Research Institute). Каждый из компьютеров ARPANET обращался к этому единому файлу, и таким образом все компьютеры знали друг о друге.

Но сеть росла слишком быстро, и один файл уже не мог решать проблему адресации в сети, поиска адреса компьютера по его имени. Когда проблема стала очевидной, была разработана новая система адресации в сети, которая получила наименование “система доменных имен”, или DNS.

Систему DNS изобрели и разработали ее первые реализации в начале 80-х годов. Идея системы DNS заключается в том, чтобы предложить организацию сети компьютеров как дерева доменов. В таком дереве группа компьютеров в сети обращается к одному общему компьютеру, который является сервером DNS. В свою очередь, этот сервер и подобные ему серверы обращаются к еще более вышестоящему серверу DNS. Таким образом, образуется дерево, в вершине которого находится домен верхнего уровня (top lever domain).

Организация сети компьютеров в виде дерева с серверами DNS позволяет каждому компьютеру иметь доступ к таблице соответствий имен компьютеров их адресам. Такую таблицу хранит сервер DNS, работа которого состоит в том, чтобы по запросу имени от клиента выдать точный адрес требуемого компьютера.

Так как DNS-сервера сами объединены в дерево, то это позволяет им обмениваться адресами с вышестоящим сервером DNS, получать и отправлять изменения в структуре сети.

Древовидная структура доменной сети позволяет за короткое время распространять изменения в адресации сети. Если где-то на участке сети появился новый компьютер, либо он был удален, либо у него был обновлен адрес, то по связям в дереве эти изменения поднимаются вверх по дереву к домену верхнего уровня и распространяются от него по всем остальным лежащим ниже доменам.

Для организации сети с системой DNS были предложены специальные имена так называемых “корневых доменов”, от которых должны происходить все остальные имена в компьютерной сети. Ими стали знакомые нам сегодня имена: *.com*, *.net*, *.org*, *.gov*, *.edu* и другие. Эти имена — не просто набор букв. Они имеют определенное значение и предназначены для разделения компьютеров в глобальной сети по назначению. Например, *com* — от английского *commercial* — означает коммерческую направленность компьютеров (их владельцев) в этом дереве доменов. *Org* — дерево доменов организаций, как правило, некоммерческих, *mil* — домены для компьютеров военных, *edu* — образовательные домены и т.д.

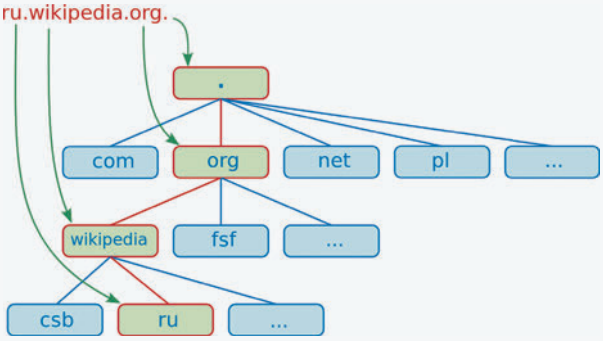


Рис. 4. Организация DNS

Если мы взглянем на схему на *рис.* 5, то сможем представить себе, что у домена *com*, кроме example, могут быть и другие домены.

То же касается и самого домена *example* — кроме mail, он может содержать массу других лежащих ниже доменов. Более того, сам домен mail может иметь домены еще более низкого уровня и так далее. На практике редко когда встречаются домены 4-го уровня, но система DNS не исключает их.

Важной особенностью организации глобальной сети является то, что корневые домены, как основу сети, контролируют специальные организации. Эти организации решают, когда следует создать новый корневой домен. Так, были созданы домены, относящиеся к государствам. Их отличительная особенность — всего два символа. Например, *ru* — домен России, *ua* — Украины, *uk* — Великобритании, *us* — США. В настоящее время с развитием Интернета контролирующими организациями стали регистрироваться локализованные имена доменов, которые состоят из букв алфавитов, отличных от латиницы. Так, у России появился домен *.рф* (Российская Федерация).

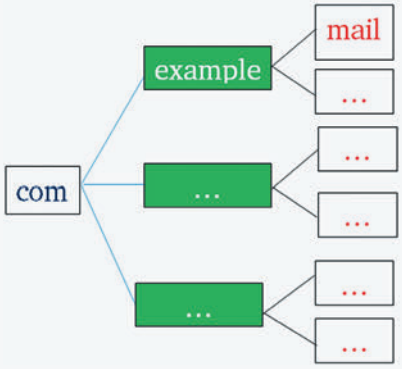


Рис. 5. Схема *mail.example.com*

Ниже по дереву расположились домены, которые можно зарегистрировать самому, например, *example*. Но эту регистрацию оформляют специальные организации-регистраторы. Вы обращаетесь в такую организацию и оплачиваете услуги компании по регистрации домена и внесению его в DNS глобальной сети. После регистрации имени вашего компьютера он становится доступен с любой точки мира в глобальной сети.

Домены, которые лежат еще ниже, создаются и управляются уже собственниками компьютеров. Вы или ваша организация можете организовать внутри своей сети доменную структуру любой сложности. Она будет доступна внутри вашей сети и снаружи в глобальной сети через ваши собственные серверы DNS, которые вам потребуется запустить в сети и соответственно настроить. Примером таких доменов является домен *mail* на схеме *рис*. 5. Домен с таким именем часто заводят для доступа к электронной почте компании изнутри и снаружи сети предприятия.

Домены и электронная почта

У серверов DNS есть важное понятие “записи ресурсов”. Кроме простого определения адреса компьютера по его имени, серверы DNS по записям ресурсов способны выполнять и другие операции. Например, сообщать об адресе почтового сервера по указанному имени. Дело в том, что электронная почта, знакомая и привычная сегодня, основана на DNS. Рассмотрим пример: *user@example.com.*

Здесь, используя наши знания про организацию DNS, мы можем определить, что речь идет о компьютере с именем *example* в дереве доменов *com*. Вы, вероятно, уже догадались, что *user* в данном случае — это имя пользователя, одно из многих, которое относится к компьютеру example. Вероятно, *user@example.com* — это почтовый ящик сотрудника компании, которая владеет доменом *example*.

Однако, как работает электронная почта? На самом деле сервер DNS содержит специальную запись, которая определенно сообщает адрес почтового сервера (компьютера, который занимается организацией получения и отправки писем), сопоставленного домену *example.com*. Очень часто почтовый сервер и сервер с веб-сайтом расположены на различных компьютерах и, соответственно, имеют различные адреса в сети.

Приведем пример:

example.com	A 192.0.2.21
mail.example	A 192.0.2.23
example.com	MX mail.example.com

Представлены примеры записей DNS-сервера: первая указывает соответствие имени компьютера его адресу. Вторая запись указывает второе соответствие: компьютер с именем *mail.example.com* сопоставляется с адресом 192.0.2.23. Последняя в списке MX-запись указывает на имя почтового сервера данного доменного имени. Имя почтового сервера должно быть заранее определено в записях DNS-сервера (вторая запись). Обратите внимание, что адреса серверов — разные.

Теперь, когда кому-то нужно отправить почту адресату *user@example.com*, почтовый клиент начинает поиск соответствующей MX-записи *example.com* на DNS-сервере. Запись обнаруживается (третья строка последней таблицы), она соответствует имени *mail.example.com*, осталось найти адрес этого компьютера. Такой адрес находится, это 192.0.2.23. После обнаружения определенного адреса почтового сервера почтовый клиент связывается с сервером по определенным почтовым протоколам и пересылает почтовое сообщение для пользователя *user*.

Конец истории ARPANET

Рассказ об истории ARPANET подходит к концу. В середине 80-х годов у этой сети появляется “конкурент” — сеть NSFNet, созданная по тем же принципам, но организацией Национальный научный фонд США. А к началу 90-х годов с появлением первых технологий Всемирной паутины сети ARPANET и NSFNet потеряли свое влияние на глобальную сеть. Сеть ARPANET прекратила свое существование, а NSFNet отделилась от всемирной сети, которая уже обрела свое имя — Интернет.

Какие же технологии послужили основой Интернета? Это протоколы обмена данными HTTP, язык разметки HTML и браузеры — программы, которые могут отображать информационное содержимое компьютеров в сети, которые начали называть серверами или сайтами.

ПОИСК ИНФОРМАЦИИ

Четыре (с лишним ☺) вопроса

Ответы на заданные вопросы найдите в Интернете или по другим источникам информации.

1. В старину китайские монахи носили с собой “оберегающий посох” из сандалового дерева. К нему были подвешены бронзовые кольца, чей звон “заглушал все житейские звуки для того, кто его несет, предупреждая небольших зверей, что они могут быть раздавлены”. Кто был его первым обладателем?

2. Кто из героев американского писателя Марка Твена едва не стал принцем, о чем мечтал с детства?

3. Какое искусство помогло героине детективного романа “Чаша Бланшара” раскрыть преступление?

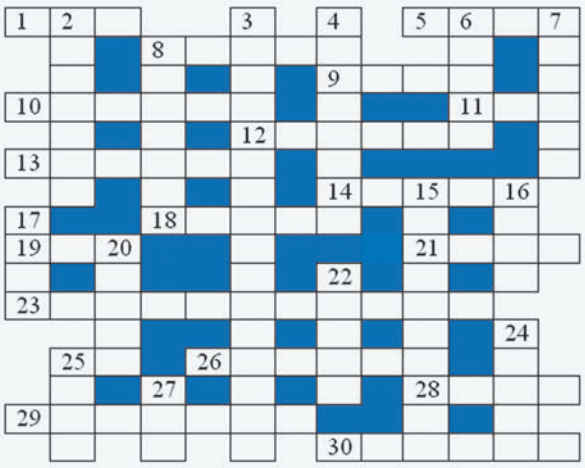
4. Какой ученый был удостоен того, что благодарные соотечественники поместили на банкноте сразу два его портрета? Каким было достоинство этой банкноты? Чем прославился этот ученый?

Отмеривание кваса

У одного человека была емкость с 20 л кваса. Сосед просил налить ему 5 л, а сам пришел с ведрами на 7 и 13 л. “Нет проблем!” — сказал хозяин. Как он поступил?

Кроссворд

Решите, пожалуйста, кроссворд.



По горизонтали

1. Характеристика файла или переменной величины.

5. Синоним слова “дорожка” (как части магнитного диска), происходящий от английского аналога этого слова.

8.

9.

10. Форма взаимодействия пользователя с компьютером.

11. … координат.

12. Устранение неисправности компьютера.

13. Наука о законах и формах мышления.

14. Последовательность символов, предназначенная для чтения человеком.

18. Порядковое число.

19. То, без чего не может работать исправный и включенный в сеть настольный компьютер.

21. Устройство для соединения двух участков локальной сети, а также транспортное сооружение.

23. Создание второго экземпляра файла, символа, фрагмента текста.

25. Шестая нота.

26. Политическая организация, а также совокупность товаров.

28. Знак препинания.

29. Утверждение, используемое в условных операторах.

30. Веб-ресурс, предоставляющий множество сервисов и услуг: поиск в Интернете, новости и др.

По вертикали

2. Устройство для вывода информации в персональном компьютере.

3. Процесс составления программы.

4. Аргумент или результат процедуры.

6. Управляемая по программе и имитирующая действия человека машина.

7. Тип топологии локальной компьютерной сети.

8. 1 000 000.

15. Разновидность транслятора, которая “переводит” в машинные коды всю программу целиком, а затем исполняет их.

16. Проверка работоспособности компьютера.

17. Структура данных, в которых применен принцип “последним пришел — первым вышел”.

20. Второй экземпляр (файла и др.).

22. Совокупность точек графического изображения.

24. Внешнее очертание, наружный вид предмета, а также объект базы данных.

25. Часть рабочей книги программы Microsoft Excel или подобной.

27. Единица измерения скорости передачи данных.

Двенадцать спичек

Из 12 спичек нужно на столе составить фигуру, в которой было бы три одинаковых четырехугольника и два одинаковых треугольника. Как это сделать?

Как выйти из комнаты?

Граф Кристомонте находился в комнате, в которой было четыре двери и маленькое окошко. Три из четырех дверей “мнимые” — то есть за ними была глухая стена и только одна вела наружу. У него был ключ, который подходил к замкам всех дверей, но он не знал, какая из четырех дверей ведет на улицу. Если бы граф попытался открыть одну из “мнимых” дверей, то все замки остальных дверей автоматически закрылись бы навсегда. Еще в комнате было темно, только одна свеча давала немного света. Кристомонте смог за одну попытку найти ту дверь, что вела наружу. А вы смогли бы? Как?

Переставить четыре шашки

Вы, конечно, знаете игру “Уголки”, в которой на шахматной доске нужно переставить шашки? “Ходить” можно только по вертикали и горизонтали на соседние свободные клетки или “прыгая” через другую шашку на свободную клетку. А как за наименьшее число ходов переставить по правилам уголков четыре шашки из левого нижнего угла доски — полей $a1$, $a2$, $b1$, $b2$ в правый верхний?

Числовой ребус “ТРОПКА — ДОРОЖКА”

Решите, пожалуйста, числовой ребус:
ТРОПКА + ТРОПКА + ТРОПКА = ДОРОЖКА
Как принято в таких головоломках, в нем одинаковыми буквами зашифрованы одинаковые цифры, разными буквами — разные цифры.

Четыре числовых ребуса в троичной системе

В приведенных ниже ребусах зашифрованы числа, записанные в троичной системе счисления. Одинаковым буквам соответствуют одинаковые цифры. Звездочкой (“*”) может быть любая цифра.

1.

		M
+	**N**	**N**
	*	*

2.

	*	*
+	*	*
	*	0

3.

		B
+	**A**	*
	A	1

4.

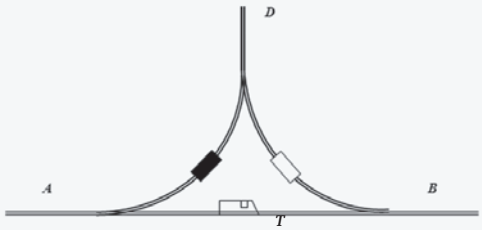
	*	**C**
+	*	*
	D	**E**

Ответы присылайте в редакцию (можно решать не все ребусы).

Маневровый треугольник

Основной железнодорожный путь AB (см. рисунок) и две небольшие маневровые ветки AD и BD образуют маневровый треугольник. Машинисту тепловоза T необходимо поменять местами белый и черный вагоны и вернуться на свое место. Как ему сделать это, если в тупичке D за стрелкой размещается только один вагон или тепловоз?

Если каждое сцепление и расцепление тепловоза с вагонами считать за один ход, то машинист тепловоза решил задачу за десять ходов. Но вы сможете сделать это за шесть ходов. Попробуйте!



Ответ, пожалуйста, оформите в виде текста и/или схем.

ЗАДАЧНИК

> Задача, которую вы решаете, может быть очень скромной, но если она бросает вызов вашей любознательности и если вы решите ее собственными силами, то вы сможете испытать ведущее к открытию напряжение ума и насладиться радостью победы.
>
> *Джордж Пойа*

Ответы, решения, разъяснения к заданиям, опубликованным в разделе “В мир информатики” ранее

Задача “Две шкатулки”

Напомним условие: “Имеются две шкатулки, в одной из которых лежит ключ. На первой шкатулке написано: “Ключ находится во второй шкатулке”. На второй написано: “Одно из двух утверждений, записанных на этих шкатулках, истинно, а второе — ложно”. В какой из шкатулок находится ключ?”

Решение

Допустим, что утверждение на второй шкатулке является истинным, тогда утверждение на первой шкатулке ложно, то есть на самом деле ключ находится в первой шкатулке.

Если же утверждение на второй шкатулке ложное, то ложными являются оба утверждения (истинными они быть не могут по второму допущению). И в этом случае следует, что ключ находится в первой шкатулке.

Ответ: ключ находится в первой шкатулке.

Правильные ответы представили:

— Абрамов Алексей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Герасимова Наталья и Костина Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа,

Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Крысанов Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Лазаренко Нина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Лежнева Александра, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Матвеева Виктория, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Новикова Анна и Потапова Алевтина, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Задача "Котлеты"

Напомним, что требовалось определить минимальное время в секундах, за которое можно пожарить семь котлет, если на сковороде умещается шесть котлет, и с каждой стороны котлету нужно жарить пять минут.

Решение

Если бы каждую котлету жарили по одной отдельно ☺, то на обжаривание семи котлет с двух сторон по пять минут на каждой стороне потребовалось бы 7 × 2 × 5 = 70 минут = 4200 секунд. Но так как одновременно на сковороде может находиться шесть котлет, то она должна "работать" в шесть раз меньше, то есть 700 секунд.

Далее рассуждения такие. Так как 700 секунд больше, чем время жарки одной котлеты — 5 + 5 = 10 минут (600 секунд), то 100 секунд каждая котлета на сковороде должна "отсутствовать" ☺, а оставшиеся 600 секунд должна обжариваться с двух сторон — по 300 секунд на каждой. Один из многих вариантов размещения котлет в ходе приготовления показан на схеме ниже.

На схеме котлеты условно обозначены в виде того или иного цвета. Число означает номер стороны котлеты, которая жарится в этот период времени (семь периодов по десять секунд каждый).

Итак, ответ: 700 секунд.

Ответы прислали:

— Андреев Алексей, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Коптева Алена и Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Красненков Александр, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Костина Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Лазаренко Нина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Мищенко Маргарита, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Новиков Сергей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Новиченко Владислав, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Стороженко Степан, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Токмурзин Данил, Свердловская обл., Красноуфимский р-н, Тавринская средняя школа, учитель **Ярцев В.А.**

В большинстве ответов указывается время поджаривания котлет — 900 сек. Оптимальный (описанный) вариант нашли только Екатерина Одинцова из школы № 124 г. Челябинска и Сергей Новиков из средней школы села Сердар. Редакция решила наградить Екатерину и Сергея дипломами. Поздравляем!

Задача "Очередь в музей"

Напомним условие: "Митя, Сережа, Толя, Юра и Костя пришли в музей до открытия и встали в очередь. Если бы Митя встал посередине очереди, то он стоял бы между Сережей и Костей, причем Сережа стоял бы впереди Мити, а если бы Митя встал в конец очереди, то рядом с ним стоял бы Юра, но Митя встал впереди своих товарищей. В каком порядке ребята встали в очередь?".

Ответ

Последовательность, начиная с начала очереди, — Митя, Толя, Сережа, Костя, Юра.

Правильный ответ представили:

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения,

Время приготовления, секунды

На сковороде находятся	1–100	101–200	201–300	301–400	401–500	501–600	601–700
	1	1	1	2	2	2	2
	1	1	1	1	2	2	2
	1	1	1	1	2	2	2
	1	1	1	1	2	2	2
	1	1	1	2	2	2	2
	1	1	1	2	2	2	2
	Нет 1-й (белой)	Нет 2-й (серой)	Нет 3-й (синей)	Нет 4-й (зеленой)	Нет 5-й (желтой)	Нет 6-й (розовой)	Нет 7-й (фиолетовой)

Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Донникова Анна, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Иванов Николай, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Игнатьева Екатерина и Мезина Ольга, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Кириллова Л.Н.**;

— Кренгель Евгений и Харламов Виталий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Крысанов Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Мусатов Тимофей, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

Задача "Сколько лет человеку?"

Напомним условие: "Предложите кому-нибудь умножить число его лет на 2 и к произведению прибавить 4, затем полученную сумму умножить на 5, к этому произведению прибавить 12 и полученную сумму умножить на 10. После объявления результата предложенных арифметических действий вы можете объявить число лет. Как это сделать?"

Ответ

Надо от объявленного числа отнять 320, затем полученный результат разделить на 100.

Пример. Пусть возраст некоторого человека равен 37 годам. Умножая это число на 2 и прибавляя 4, получим 78. Умножая это число на 5, имеем 390; прибавив 12, находим 402; умножая на 10, получаем 4020. Вычитая из этого числа 320, имеем 3700. Если это число разделить на 100, то получим число 37.

Обоснование. Пусть число лет некоторого человека равно m. После выполнения всех предложенных арифметических действий получится число:

$$((2m + 4) \times 5 + 12) \times 10.$$

Это число можно записать в виде $100m + 320$. Если от него отнять 320, то получится $100m$. Разделив последнее число на 100, будем знать искомое число лет m.

Правильные ответы прислали:

— Андреев Алексей, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа, Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Клипперт Илона и Коптева Алена, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Лазаренко Нина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Лошак Антон и Турков Андрей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Мухуров Евгений, Смоленская обл., г. Демидов, школа № 1, учитель **Кордина Н.Е.**;

— Стороженко Степан, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Харламов Виталий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**

Задача "Куклы"

Напомним условие: "У одной девочки-инопланетянки было 21 разных кукол. Она отдала подруге 12 кукол, после чего у нее осталось 3 куклы. Как такое могло быть?"

Решение

Ясно, что речь идет о расчетах в недесятичной системе счисления. Но в какой именно? Для ответа удобно записать информацию из условия так:

$$\begin{array}{cc} & 1\ \ 2 \\ + & \ \ \ 3 \\ \hline 2 & 1 \end{array}$$

Сумма цифр 2 и 3 оканчивается на цифру 1 только в четверичной системе.

Ответ. Все числа в условии приведены в четверичной системе счисления.

Правильные ответы представили:

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Донникова Анна, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Евграфов Алексей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Иванов Николай, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Коробов Сергей, Марков Алексей и Яснов Федор, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Мищенко Маргарита, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Нуретдинова Лилия, средняя школа села Сулеево им. Р.Г. Галеева, Республика Татарстан, Альметьевский р-н, учитель **Валиева Д.И.**;

— Стороженко Степан, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Таран Алина, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**

Можно также представить числа 21 и 12 в развернутой форме ($21_p = 2p + 1$, $12_p = p + 2$), а затем провести их сложение или вычитание, что даст уравнение относительно основания p использованной системы счисления. Но понятно, что этот вариант более трудоемкий.

Задача "Зашифрованный вопрос"

Напомним, что предлагалось расшифровать текст

5343934*150413*6*8156215044414*305041080,

зашифрованный с помощью приведенной ниже таблицы, следующим образом: каждой цифре сопоставляется одна из букв, расположенных под ней, а знаку "*" — пробел или одна из букв *ю* или *я*. В разных местах зашифрованного текста одна и та же цифра может соответствовать разным буквам.

0	1	2	3	4	5	6	7	8	9	*
А	Г	Ж	Й	М	П	Т	Х	Ш	Ы	Ю
Б	Д	З	К	Н	Р	У	Ц	Щ	Ь	Я
В	Е	И	Л	О	С	Ф	Ч	Ъ	Э	

Решение

Первое слово может оканчиваться на одну из трех букв: *м*, *н* или *о*. Так как в задании речь идет о вопросе, то подходит только слово *сколько*.

Третье слово — однобуквенное (*т*, *у* или *ф*), то есть союз или предлог. Среди таких слов подходит только предлог *у*.

Рассмотрение и анализ возможных вариантов других слов приводит к такому тексту: "Сколько граней у шестигранного карандаша?".

Правильный ответ прислали:

— Андреев Алексей, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Донникова Анна, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Живило Андрей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа, Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Клипперт Илона, Коптева Алена и Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Мазанова Екатерина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Марун Виталий, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Мусатов Максим, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**

Учитывая сложность задачи, редакция решила наградить всех перечисленных читателей дипломами. Поздравляем!

Кроссворд (мартовский выпуск)

Ответы

По горизонтали: 1. Трассировка. 6. Три. 8. Фон. 9. Корпус. 10. Азия. 13. Паскаль. 14. Сеть. 16. Счетчик. 17. Остаток. 19. Принтер. 24. Число. 25. Дек. 26. Евклид. 28. Воскресенье. 29. Ось.

По вертикали: 1. Тип. 2. Сноска. 3. Рок. 4. Вирус. 5. Акустика. 6. Триггер. 7. Запись. 11. Палитра. 12. Кластер. 15. Шаг. 18. Индекс. 20. Цикл. 21. Код. 22. Киев. 23. Плюс. 26. Ель. 27. Икс.

Ответы прислали:

— Антипов Анатолий, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Бородюк Анна и Василенко Татьяна, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Булатова Анна и Калугин Сергей, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**;

— Васина Светлана и Хомутова Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Дегтярь Анатолий и Новиченко Владимир, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Землянская Надежда, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Иванова Виолетта и Левченко Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Игнатьева Екатерина и Мезина Ольга, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Кириллова Л.Н.**;

— Лёвина Татьяна и Цыплаков Евгений, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Лошак Антон и Турков Андрей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Машонина Ирина, г. Фрязино Московской обл., школа № 4, учитель **Сенюта Е.И.**;

— Морозов Дмитрий, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Нуретдинова Лилия, средняя школа села Сулеево им. Р.Г. Галеева, Республика Татарстан, Альметьевский р-н, учитель **Валиева Д.И.**

За активное участие в решении кроссвордов, опубликованных в прошлом учебном году, дипломами будут награждены следующие читатели:

— Байбуза Дарья, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**;

— Гололобов Дмитрий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Землянская Надежда, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Новикова Анна и Потапова Алевтина, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Хомутова Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**

Они правильно решили не менее трех кроссвордов (всего было опубликовано 8).

За активное участие в выполнении заданий рубрики “Поиск информации”, предложенных в прошлом учебном году, дипломами будут награждены следующие читатели:

— Бородюк Анна и Василенко Татьяна, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Вышегородских Анастасия, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Дибров Сергей, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Заева Кристина, Республика Башкортостан, г. Уфа, школа № 54 (Центр дистанционного обучения), учитель **Искандарова А.Р.**;

— Кротов Олег, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Лоскутова Светлана и Хомутова Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Михайлова Светлана, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Воеводина Р.В.**;

— Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Соловьева Марина, Вадьковская средняя школа, Брянская обл., Погарский р-н, учитель **Цыганкова И.Ю.**

Поздравляем всех награжденных!

Задача “Разлив бензина”

Напомним, что следовало описать алгоритм решения задачи заполнения двух ведер объемом по 7 литров шестью литрами бензина в каждое ведро, если имеется бочка с 18 литрами бензина и черпак объемом 4 литра.

Решение

Удобно начать анализ, так сказать, с “конца”. Ясно, что две последние операции при решении задачи такие, как в табл. 1.

Из таблицы видно, что сначала надо заполнить шестью литрами бензина одно из ведер (в указанном случае — первое ведро). Это можно сделать так, как показано в табл. 2.

Итак, первая частная задача решена. Осталось из ситуации, представленной в строке 13 последней таблицы, не используя уже заполненное ведро, получить ситуацию, показанную в 1-й строке табл. 1. Соответствующие действия приведены в табл. 3.

Таблица 1

Действие	Бочка	Ведро 1	Ведро 2	Черпак
	10	2	6	0
Наполнить черпак из бочки	6	2	6	4
Перелить из черпака в ведро 1	6	6	6	0

Таблица 2

	Действие	Бочка	Ведро 1	Ведро 2	Черпак
Исходное состояние		18	0	0	0
1	Наполнить черпак из бочки	14	0	0	4
2	Перелить из черпака в ведро 1	14	4	0	0
3	Наполнить черпак из бочки	10	4	0	4
4	Перелить из черпака в ведро 2	10	4	4	0
5	Наполнить черпак из бочки	6	4	4	4
6	Отлить из черпака в ведро 2	6	4	7	1
7	Перелить из черпака в ведро 1	6	5	7	0
8	Наполнить черпак из бочки	2	5	7	4
9	Перелить из черпака в ведро 1	2	7	7	2
10	Вылить из ведра 2 в бочку	9	7	0	2
11	Перелить из черпака в ведро 2	9	7	2	0
12	Наполнить черпак из бочки	5	7	2	4
13	Перелить из черпака в ведро 2	5	7	6	0

Таблица 3

	Действие	Бочка	Ведро 1	Ведро 2	Черпак
…					
14	Наполнить черпак из бочки	1	7	6	4
15	Вылить из ведра 1 в бочку	8	0	6	4
16	Перелить из черпака в ведро 2	8	4	6	0
17	Наполнить черпак из бочки	4	4	6	4
18	Перелить из черпака в ведро 1	4	7	6	1
19	Вылить из ведра 1 в бочку	11	0	6	1
20	Перелить из черпака в ведро 1	11	1	6	0
21	Наполнить черпак из бочки	7	1	6	4
22	Перелить из черпака в ведро 1	7	5	6	0
23	Наполнить черпак из бочки	3	5	6	4
24	Перелить из черпака в ведро 1	3	7	6	2
25	Вылить из ведра 1 в бочку	10	0	6	2
26	Перелить из черпака в ведро 1	10	2	6	0

Общее число операций по решению задачи — 28.

Правильный ответ прислали:

— Антипов Анатолий, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Баженов Михаил, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Герасимова Наталья и Костина Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа, Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Исакова Алина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Коптева Алена, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Красненков Александр, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Федосеева Анастасия, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Яковлева Александра, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**

Учитывая сложность задачи, редакция решила наградить всех перечисленных читателей дипломами. Молодцы!

Головоломка “Все цифры в квадратиках”

Напомним, что требовалось вписать в квадратики цифры от 1 до 9 так, чтобы все эти девять цифр были использованы и выполнялись указанные равенства.

□□ : □ = □ − □ = □ + □ = □ × □

5 6 : 8 = 9 − 2 = 3 + 4 = 7 × 1

Решение

Правильные ответы представили:

— Бальцеров Семен и Медведев Владислав, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Барановская Татьяна и Жукова Ирина, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гаязова Илюза и Нуретдинов Айдар, средняя школа села Сулеево им. Р.Г. Галеева, Республика Татарстан, Альметьевский р-н, учитель **Валиева Д.И.**;

— Глушаков Андрей, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Довгань Алексей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа, Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Круглякова Мария и Яснова Дарья, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Лёвина Татьяна и Цыплаков Евгений, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Михайлов Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Головоломка “Как получить верное равенство?”

Напомним, что требовалось определить, как в написанном на доске неверном равенстве 5+5+5+5=555, проведя всего один прямолинейный отрезок, получить верное равенство.

Ответ

Для этого нужно отрезок провести так, чтобы один из знаков “+” превратился в цифру 4:

В одном из писем было предложено такое решение:

5+5+5+5≠555

Указанная запись является верной, но не является равенством.

Правильный ответ прислали:

— Барановская Татьяна и Жукова Ирина, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гаязова Илюза и Нуретдинов Айдар, средняя школа села Сулеево им. Р.Г. Галеева, Республика Татарстан, Альметьевский р-н, учитель **Валиева Д.И.**;

— Глушаков Андрей, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Довгань Алексей и Чурикова Елизавета, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Исакова Алина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Круглякова Мария и Яснова Дарья, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Михайлов Иван и Перевалова Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Задание “Четыре вопроса” (рубрика “Поиск информации”)

Ответы

1. Фраза “Даже когда меня разобьет паралич, я буду критиковать чужую походку” принадлежит французскому классику Жюлю Ренару.

2. Знаменитая статуя Венеры Милосской из парижского Лувра выполнена из мрамора и изображает богиню любви Афродиту.

3. К созданию одного из семи чудес света — Храму Артемиды Эфесской “приложили руку” архитекторы Пеоний и Деметрий. (“Приложили руку” — потому что они уже заканчивали постройку.)

4. Перед регистрацией брака молодая пара должна оплатить пошлину крысиными хвостами на индонезийском острове Ява.

Ответы представили:

— Аксёненко Ирина и Чумаков Илья, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Григоренко Дмитрий и Есипова Мария, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Демьянова Елена, Костюнин Александр и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Дружинина Анастасия, Невтыра Анжелика и Субботина Галина, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Евграфова Ксения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Иванова Виолетта и Левченко Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Иванова Ксения, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Игнатьева Екатерина и Мезина Ольга, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Кириллова Л.Н.**;

— Клипперт Илона, Коптева Алена и Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Пашнова Елена, Свердловская обл., г. Ревда, школа № 10, учитель **Игошева А.А.**;

— Перевалова Ника, средняя школа города Пионерский, Калининградская обл., учитель **Багрова О.А.**;

— Подольская Арина, г. Рязань, школа № 44, учитель **Марцинкевич Е.Е.**;

— Потапова Алевтина, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Решение головоломок “судоку”, опубликованных в мартовском выпуске “В мир информатики”, представили:

— Абрамова Татьяна, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Айзетуллова Ландыш, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Кириллова Л.Н.**;

— Булатова Анна и Калугин Сергей, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**;

— Евтюхин Максим, г. Рязань, школа № 44, учитель **Марцинкевич Е.Е.**;

— Есипова Мария, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа, Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Клипперт Илона и Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Коростелев Иннокентий и Ланской Дмитрий, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Курилович Михаил, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Краснова Диана, Свердловская обл., г. Ревда, школа № 10, учитель **Игошева А.А.**;

— Машонина Ирина, г. Фрязино Московской обл., школа № 4, учитель **Сенюта Е.И.**;

— Мусатов Тимофей, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Цыплаков Евгений, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

Ответы на вопросы второй части викторины «Да будет “свет”!» прислали:

— Григоренко Дмитрий и Есипова Мария, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Евграфова Ксения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Демьянова Елена и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Иванова Виолетта и Левченко Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Иванова Ксения, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Игнатьева Екатерина и Мезина Ольга, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Кириллова Л.Н.**;

— Потапова Алевтина, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Сенотрусова Алина и Степанова Полина, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Чумаков Илья, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**

Ответы к задаче “Найти четырехзначное число” (мартовский выпуск) представили:

— Абрамов Алексей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**

Алексей и Тарас получили решение, разработав компьютерную программу.

Пять задач на системы счисления

Напомним условия задач и сразу же будем приводить их решение.

1. Выпишите числа 2^{n-1}, $2^n + 1$, $2^n - 3$, где $n > 10$, в порядке возрастания количества единиц в их двоичной записи.

Решение

Для решения задач такого типа (которые, кстати, встречаются в ЕГЭ по информатике) следует прежде всего знать двоичный вид чисел 2^n. Из анализа таблицы

k	1	2	3	…
2^k	2	4	8	
Двоичная запись числа 2^k	10	100	1000	
Количество единиц	1	1	1	

— следует, что двоичная запись числа 2^k представляет $(k + 1)$-значное число, состоящее из одной единицы и k нулей.

Тогда в двоичной записи числа $2^k + 1$ будут две единицы и $(k - 1)$ нулей.

Вычитание единицы из двоичной записи числа 2^n даст число из n единиц, а вычитание тройки — число, в котором будет один ноль и $(n - 1)$ единиц.

Итак, можем составить такую таблицу:

Двоичная запись числа 2^n	Количество единиц	Количество нулей
2^n	1	n
2^{n-1}	1	$n-1$
2^n+1	2	$n-1$
2^n-1	N	0
2^n-3	$(n-1)$	1

Из этой таблицы и можно получить искомую последовательность чисел.

2. Каждую строку черно-белого рисунка размером $n \times n$ пикселей закодировали следующим образом: пиксель черного цвета обозначили единицей, белого — нулем и соответствующее число представили в восьмеричной системе счисления. Весь рисунок при этом кодируется последовательностью восьмеричных чисел: 234, 242, 242, 202, 204, 210, 0, 210.

Что изображено на рисунке?

Аналогично можно этот рисунок закодировать в шестнадцатеричной системе счисления. Перекодируйте рисунок соответствующим образом.

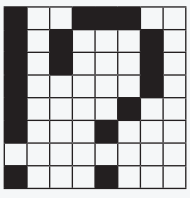
Решение

Чтобы получить рисунок, следует знать двоичное представление числа, которым кодируется каждая строка. Поэтому переведем заданные восьмеричные числа в двоичную систему счисления и представим результат как 8-разрядное двоичное число, дописывая в начале соответствующее количество нулей. Это можно сделать без промежуточного перевода чисел восьмеричных чисел в десятичную систему и последующего последовательного деления на 2.

Для перевода целого восьмеричного числа в двоичную систему счисления необходимо каждую цифру заданного числа, рассматривая ее как десятичную, представить в виде трехразрядного двоичного числа (может быть — с начальным нулем, с начальными нулями или состоящим из трех нулей):

$234_8 = 10011100_2$, $242_8 = 10100010_2$,
$202_8 = 10000010_2$, $204_8 = 10000100_2$,
$210_8 = 10001000_2$.

Соответствующее изображение (в увеличенном масштабе):



Для кодирования рисунка в шестнадцатеричной системе счисления удобно использовать полученные двоичные числа. Здесь также можно обойтись без промежуточного перевода чисел двоичных чисел в десятичную систему и последующего деления на 16.

Для перевода целого двоичного числа в шестнадцатеричную систему счисления необходимо:

1) начиная с правых разрядов, разбить цифры заданного числа на тетрады — группы из четырех цифр (в крайней слева группе может оказаться одна, две или три цифры);

2) число, образованное цифрами в каждой группе, перевести в десятичную систему. Так как при этом не могут получиться числа, большие 15, то их можно рассматривать как цифры шестнадцатеричной системы счисления (при наличии чисел, больших 9, их следует записать с использованием обозначений для соответствующих цифр — A, B, …, F);

3) из полученных на предыдущем этапе цифр сформировать единое число, которое и будет являться искомым результатом.

В нашем случае:

$10011100_2 = 9C_{16}$, $1010\ 0010_2 = A2_{16}$,
$1000\ 0010_2 = 82_{16}$, $1000\ 0100_2 = 84_{16}$,
$10001000_2 = 88_{16}$.

3. Два некоторых десятичных числа X и Y перевели в системы счисления с основаниями 16 и 8. Часть символов при записи была утеряна. Два из четырех полученных чисел имеют вид (позиции утерянных символов обозначены символом “*”):

$F*_{16}$ и $33*_8$.

Можно ли сделать вывод о том, какое из чисел X или Y больше, или о равенстве этих чисел?

Решение

Прежде всего, сравнив заданные числа, можно установить, что это представления разных чисел (первое — не менее 240, второе — не более 223).

Запишем каждое из приведенных в условии чисел в двоичной системе счисления, заменяя неизвестные цифры тремя или четырьмя символами “*”:

Основание								
16	↓	F				*		
2		1	1	1	1			
2	↑	1	1	0	1	1		
8		3		3		*		

Сравнение строк с основанием 2 показывает, что число $F*_{16}$ больше, чем $33*_8$. Но какое из них соответствует числу X, а какое — Y, определить невозможно.

4. Некоторое десятичное число X перевели в системы счисления с основаниями 16, 8, 4. Часть символов при записи была утеряна. Позиции утерянных символов обозначены символом “*”:

$X = *7*_{16} = 5*6_8 = ***1*_4$

Восстановите все числа и определите число X.

Решение

Запишем известную информацию о записи чисел в виде таблицы, выделив, начиная справа, пары, тройки (триады) и группы по 4 (тетрады) цифры:

№ пп	Основание	Запись числа											
1	8				5			*			6		
2	2												
3	4			*						1		*	
4	16		*			7				*			

Обратим внимание на расположение строк с тем или иным основанием системы счисления.

Сравнение данных в строках 3 и 4 позволяет получить ряд новых цифр — они оформлены полужирным начертанием и синим цветом, а данных в

строках 1 и 2 — новые цифры, выделенные зеленым цветом:

№ пп	Основание	Запись числа
1	8	5 * 6
2	2	1 0 1 1 1 0
3	4	* 1 3 1 *
4	16	* 7 *

После этого можно также получить ряд новых цифр:

№ пп	Основание	Запись числа
1	8	5 * 6
2	2	1 0 1 1 1 0 1 1 0
3	4	1 1 3 1 2
4	16	1 7 *

Синим цветом выделены цифры, полученные при сравнении строк 2 и 3, красным — при сравнении строк 3 и 4.

Итак, ответ:

№ пп	Основание	Запись числа
1	8	5 6 6
2	2	1 0 1 1 1 0 1 1 0
3	4	1 1 3 1 2
4	16	1 7 6

$X = 374$.

5. Некоторое десятичное число X перевели в системы счисления с основаниями 16 и 8. Часть символов при записи утеряна (позиции утерянных символов обозначены символом “*”):

$$3*9_{16} \text{ и } 1**_{8}.$$

Можно ли однозначно определить значение числа X? Если нет, то укажите все возможные значения числа X.

Решение

Рассуждения, аналогичные проведенным при выполнении предыдущего задания, позволяют получить таблицу:

№ пп	Основание	Запись числа
1	16	2 * 9
2	2	1 0 0 1 1 0 0 1
3	8	1 * 1
		X

В строке 2 таблицы неизвестны две цифры, то есть однозначно определить значение числа X невозможно. Эти цифры могут быть в четырех вариантах (00, 01, 10 и 11). Соответствующие четыре возможные значения числа X найдите самостоятельно.

Пять задач на системы счисления. Часть 2

Е.А. Мирончик,
учитель информатики лицея № 111,
г. Новокузнецк Кемеровской обл.

1. Выпишите числа 2^{2n}, $2^n + 3$, $2^n - 3$, где $n > 10$, в порядке возрастания количества единиц в их двоичной записи. Ответ обоснуйте.

2. Каждую строку черно-белого рисунка размером $n \times n$ пикселей закодировали следующим образом: пиксель черного цвета обозначили единицей, белого — нулем и соответствующее число представили в шестнадцатеричной системе счисления. Весь рисунок при этом кодируется последовательностью восьмеричных чисел: 82, 144, 148, 90, 22, 45, 85, 102.

Что изображено на рисунке?

Аналогично можно этот рисунок закодировать в восьмеричной системе счисления. Перекодируйте рисунок соответствующим образом.

3. Два некоторых десятичных числа X и Y перевели в системы счисления с основаниями 16 и 8. Часть символов при записи была утеряна. Два из четырех полученных чисел имеют вид (позиции утерянных символов обозначены символом “*”):

$$18*_{16} \text{ и } 72*_{8}.$$

Можно ли сделать вывод о том, какое из чисел X или Y больше, или о равенстве этих чисел?

4. Некоторое десятичное число X перевели в системы счисления с основаниями 16, 8, 2. Часть символов при записи была утеряна. Позиции утерянных символов обозначены символом “*”:

$$X = 10******_{2} = *4*_{8} = *2_{16}.$$

Восстановите все числа и определите число X.

5. Некоторое десятичное число X перевели в системы счисления с основаниями 16 и 8. Часть символов при записи утеряна (позиции утерянных символов обозначены символом “*”):

$$*E_{16} \text{ и } 2*6_{8}.$$

Можно ли однозначно определить значение числа X? Если нет, то укажите все возможные значения числа X.

Семь кошельков

Как разложить по семи кошелькам 127 рублевых монет, чтобы любую сумму от 1 до 127 рублей можно было бы выдать, не открывая кошельков (то есть вместе с кошельками)?

У кого какая профессия?

Корнилов, Джаганян, Марченко и Семеновский — жители одного города. Их профессии: парикмахер, врач, инженер и милиционер.

Известно, что:

1) Корнилов и Джаганян — соседи и всегда на работу ездят вместе;

2) Джаганян старше Марченко;

3) Корнилов регулярно обыгрывает Семеновского в шахматы;

4) парикмахер на работу всегда ходит пешком;

5) милиционер живет рядом с врачом;

6) инженер и милиционер встречались только один раз — когда второй оштрафовал первого за нарушение правил дорожного движения;

7) милиционер старше врача и инженера.

Определите профессию каждого.

Три одноклассника

Три одноклассника — Саша, Дима и Женя — занимаются в различных спортивных секциях: бокс, лыжи, плавание. Каким видом спорта занимается каждый мальчик, если известно, что Саша плаванием не увлекается, Дима в лыжную секцию никогда не ходил, Женя является победителем соревнований по лыжам?

Задача предназначена для учащихся начальной школы.

Большая скорость

Один инопланетянин написал, что за четыре часа, двигаясь со скоростью 400 км/ч, он проехал 3100 км. Как такое могло быть?

ВНИМАНИЕ! КОНКУРС!

Итоги конкурса № 110

Напомним, что требовалось перечислить ошибки, имеющиеся на приведенном в условии конкурса изображении.

Участниками конкурса являлись:

— Аваков Арсен, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Гаязова Фатима, Михайлов Иван и Хорькова Анна, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Герасимова Наталья и Костина Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Жаткина Екатерина, средняя школа села Черемушка, Красноярский край, Каратузский р-н, учитель **Колодюк В.Н.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа, Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Землянская Надежда, Мусатов Тимофей, Пашкова Юлия, Ревков Александр и Шадрина Алина, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Карданова Амина, Картлыкова Мариям и Натбиева Эльмира, Кабардино-Балкарская Республика, г. Нальчик, школа № 32, учитель **Пышная Е.А.**;

— Клипперт Илона и Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Кротов Олег, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Крысанов Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Курганский Давид, Починок Илья и Таран Дарья, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Милютина Софья, средняя школа деревни Павлищево, Московская обл., Можайский р-н, учитель **Иванова Л.Н.**;

— Прудникова Дарья, г. Кострома, гимназия № 33, учитель **Исакова Е.В.**;

— Семенов Александр, Свердловская обл., Красноуфимский р-н, Тавринская средняя школа, учитель **Ярцев В.А.**;

— Соловьёв Иван, Костромская обл., Буйский р-н, г.п.п. Чистые Боры, школа № 1, учитель **Васнина О.В.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Яковлева Александра, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Яковлева Татьяна, средняя школа села Новое Барятино, Республика Башкортостан, Стерлитамакский р-н, учитель **Евдокимова Н.Л.**

Так как на приведенном изображении ошибок много, то с целью экономии места ☺ редакция решила не перечислять их, а всех участников конкурса, правильно указавших эти ошибки, наградить дипломами. Поздравляем!

Конкурс № 111 “Переправы”

Напомним, что задания данного конкурса представляют собой задачи на переправы. Конкурс будет проводиться в несколько туров, а его итоги будут подводиться с учетом всех туров в целом.

Тур 2

Три жулика, каждый с двумя чемоданами, находятся на одном берегу реки, через которую они хотят переправиться. Есть трехместная лодка, каждое место в ней может быть занято либо человеком, либо чемоданом. Никто из жуликов не доверит свой чемодан спутникам в свое отсутствие, но готов оставить чемоданы на безлюдном берегу. Смогут ли они переправиться?

Ответ с обоснованием отправьте в редакцию до 1 ноября по адресу: 121165, Москва, ул. Киевская, д. 24, “Первое сентября”, “Информатика” или по электронной почте: *vmi@1september.ru*. Пожалуйста, четко укажите в ответах свои фамилию и имя, населенный пункт, номер и адрес школы, фамилию, имя и отчество учителя информатики.

журнал

Информатика – Первое сентября

1-е полугодие 2015 года

ПОДПИСКА

на сайте www.1september.ru и в почтовых отделениях РФ

Индекс	Название издания	Периодичн. в полугодие	1 месяц		6 месяцев	
			Каталожная цена (руб.)	Подписная цена (руб.)	Каталожная цена (руб.)	Подписная цена (руб.)
Название блока в разделе «Журналы»	**ПЕРВОЕ СЕНТЯБРЯ. ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА** (499)249-31-38					
79066	**Информатика – Первое сентября. Бумажная версия** С электронными приложениями и презентациями. *В июне не выходит.* *Подписка на июнь не принимается* *(-) 160 г 64 стр.*	5	340.00		1700.00	
12684	**Информатика – Первое сентября. Электронная версия на CD (полная копия бумажной версии)** *В июне не выходит.* *Подписка на июнь не принимается* (-) *75* г	5	118.80		594.00	
сайт 1september.ru	**Информатика – Первое сентября. Электронная версия**	5	–		–	300.00

Подписку принимают во всех отделениях связи Российской Федерации, включая Крым и Севастополь, а также на сайте www.1september.ru

При оформлении подписки на сайте оплата производится по квитанции в отделении банка или электронными платежами on-line

VISA MasterCard QIWI WebMoney

Общероссийский проект

Школа цифрового века

Издательский дом «ПЕРВОЕ СЕНТЯБРЯ» • Издательство «ПРОСВЕЩЕНИЕ»

Каждый педагогический работник образовательной организации, вошедшей в проект «Школа цифрового века», получает доступ ко всем материалам проекта по принципу «все включено» (без дополнительной платы)

МАТЕРИАЛЫ ПРОЕКТА

- **23 предметно-методических журнала** по всем предметам и направлениям школьной жизни плюс журнал для родителей
- **Модульные дистанционные курсы*** из циклов «Навыки профессиональной и личной эффективности педагога» и «Инклюзивный подход в образовании»
- **Дистанционные 36-часовые курсы**** повышения квалификации с выдачей удостоверения установленного образца
- **Методические брошюры** по всем школьным предметам

Стоимость участия образовательной организации в проекте – 6 тысяч рублей за весь учебный год. Стоимость участия не зависит от количества педагогических работников в образовательной организации

Участие образовательной организации и педагогических работников в проекте удостоверяется соответствующими документами. Для дошкольных организаций предусмотрен свой набор удостоверяющих документов

Срок действия проекта в 2014/15 учебном году: с 1 августа 2014 года по 30 июня 2015 года

Подробности и прием заявок
от образовательных организаций
на сайте

digital.1september.ru

---

* В течение указанного срока предоставляются без ограничения количества курсов.

** Предоставляется по одному курсу для одного педагогического работника в течение одного учебного года (выбор конкретного курса – на усмотрение педагога).