МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

Национальный технический университет
„Харьковский политехнический институт”

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ по курсу

“Основы дискретной математики”

для студентов специальности:

7.080404 «Интеллектуальные системы принятия решения»

Утверждено
редакционно-издательским
советом университета,
протокол № 3 от 21.12.07

Харьков НТУ “ХПИ” 2008

Методические указания к выполнению лабораторных работ по курсу «Основы дискретной математики» для студентов специальности: 7.080404 «Интеллектуальные системы принятия решений» / Сост. Н.В. Савченко. – Харьков: НТУ “ХПИ”, 2008. – 152 с.

Составитель Н.В. Савченко

Рецензент О.В. Серая

Кафедра системы информации

Не пройдешь через дело – не станешь умнее.
Китайская мудрость **[1].**

Общие указания

Целью лабораторного практикума является:

- закрепление теоретических знаний по дисциплине «Основы дискретной математики»;
- приобретению практических навыков по моделированию объектов и операций над ними, изучаемых в курсе;
- знакомство с технологиями сетевого программирования на стороне клиента.

Предлагаемые в данном пособии лабораторные работы, используются при проведении занятий со студентами 2-го курса кафедры «Системы информации» факультета «Компьютерных информационных технологий» НТУ «ХПИ» в ходе изучения курса «Основы дискретной математики». В течение 4-х недель проводится одна лабораторная работа. Девятая учебная неделя используется для проведения модульного контроля. На первом лабораторном занятии студенты регистрируются в сети Интернет на сайте http://dl.kpi.kharkov.ua/techn/nvs3/, размещенном на портале НТУ «ХПИ».

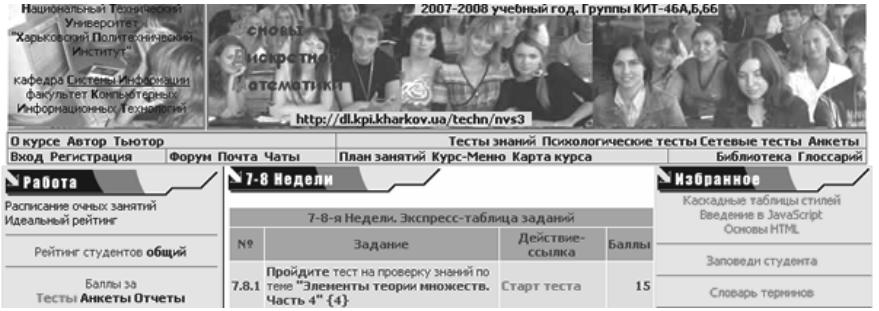


Рис. 1. Фрагмент стартовой страницы сайта курса

Данный сайт создан с использованием виртуальной учебной среды «Веб-класс ХПИ» и выступает в качестве электронной поддержки обычных занятий. Он содержит минимально необходимый теоретический материал, тесты, рейтинг, подсистемы онлайн коммуникации. В ходе лабораторной работы студент должен сдать соответствующий тест и написать компьютерную программу. За каждое выполненное задание студенту начисляется определенное количество баллов. На стартовой странице курса студент может просмотреть свой текущий суммарный рейтинг.

Особенности прохождения тестов. Для тестирования знаний используется подсистема Х-тесты виртуальной среды. На текущий момент база состоит из 600 карточек-вопросов, в среднем по 75 карточек на тест. При каждом тестировании студентам предлагается по 15 карточек. Максимально возможная оценка за выполнение теста – «15 баллов».

Особенности написания компьютерной программы. При выполнении этого задания студент должен использовать си-подобный язык программирования JavaScript. Интерфейс программ создается с использованием языка HTML. Работа над таким мини-проектов включает в обязательном порядке создание блок-схемы, составления электронного отчета. Окончательные результаты представляются студентом преподавателю в напечатанном виде. Максимально возможная оценка за выполнение этого пункта задания – «35 баллов».

При подготовке заданий была использована литература [2-12]. В качестве книг для начального ознакомления с языком разметки гипертекста HTML и системой программирования JavaScript можно порекомендовать книги [3, 4]. Для успешного выполнения настоящих лабораторных работ студентам полезно в первую очередь познакомится с книгами [2, 4, 5, 9, 10].

Для подготовки к лабораторным работам следует пользоваться приведенными ниже литературными источниками. К выполнению каждой работы допускаются студенты, выдержавшие собеседование с преподавателем по теоретическим и практическим вопросам, относящимся к тематике работы.

Часть материала: книги в формате djvu, бесплатное программное обеспечение, электронный курс «Основы HTML. Введение в JavaScript. Каскадные таблицы стилей» собраны на лазерном диске. Копия этого диска распространяется бесплатно. Каждый студент может получить эту информацию, обратившись в Проблемную лабораторию дистанционного обучения НТУ «ХПИ», которая находится в вечернем корпусе (за электрокорпусом, 2-й этаж, левая сторона, комната 28, тел. (057) 70-76-382, веб-сайт http://dl.kpi.kharkov.ua/techn/nvs3/).

1. ЛАБОРАТОРНАЯ РАБОТА № 1.

МОДЕЛИРОВАНИЕ ОПЕРАЦИИ «ПЕРЕСЕЧЕНИЕ» ДЛЯ ДВУХ ЧИСЛОВЫХ МНОЖЕСТВ

1.1. Цель работы

Изучить способы численного моделирования операции пересечения для множеств и составить компьютерную программу для выполнения пересечения двух конечных множеств.

1.2. Краткие теоретические сведения [2, 5, 9–11]

Под **множеством** M понимается совокупность некоторых объектов, которые будут называться элементами множества. Элементы множества сами могут являться множествами. Множество M может быть **бесконечным** или **конечным**, т.е. состоять из конечного числа элементов. Множество можно задать перечислением принадлежащих ему элементов или указанием свойств, которым элементы множества должны удовлетворять. Например, множество M арабских цифр можно задать двояко: перечислением $M=\{0,1,2,3,4,5,6,7,8,9\}$ или посредством свойств $M=\{x \mid x$ – арабская цифра$\}$.

Множество A называется **подмножеством** множества B (обозначается $A \subseteq B$), если все элементы множества A принадлежат B, т.е.

$A \subseteq B \Leftrightarrow \forall x\ (x \in A \Rightarrow x \in B)$.

Множества называются **равными**, если они состоят из одних и тех же элементов, т.е. если $A \subseteq B$ и $B \subseteq A$.

Существует множество, не содержащее ни одного элемента, которое называется **пустым** и обозначается через $\varnothing$. Ясно, что $\varnothing \subseteq A$ для любого множества A.

Совокупность всех подмножеств множества A называется его **булеаном** или **множеством-степенью** и обозначается через P(A) или 2^A.

Множество, содержащее все элементы, находящиеся в рассмотрении, называется **универсальным** или **универсумом** и обозначается через U.

Если множества A, B∈ P(U), то **пересечение** A∩B множеств A и B определяется равенством A∩B = {x | x ∈ A и x ∈ B }.

1.3. Порядок выполнения работы

Составить компьютерную программу для выполнения пересечения двух конечных множеств. Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- помеченных полей ввода для двух множеств;
- поля вывода результатов;
- управляющей кнопки для выполнения операции сравнения множеств.

На рис. 1.1 приведен возможный вид интерфейса.

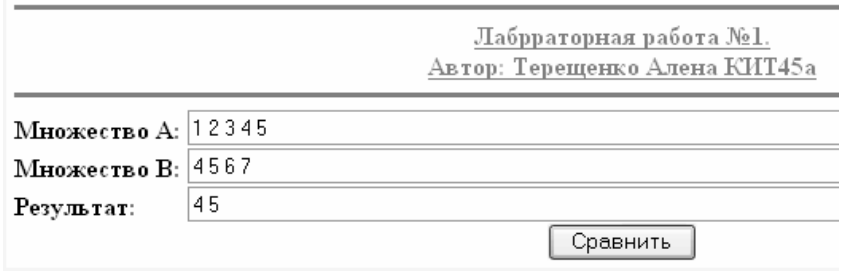


Рис. 1.1

Данный интерфейс может быть реализован такой последовательностью тегов (каждый тег состоит из имени, за которым может следовать список необязательных атрибутов, все они находятся внутри угловых скобок “<“ и “>“ [3,4]):

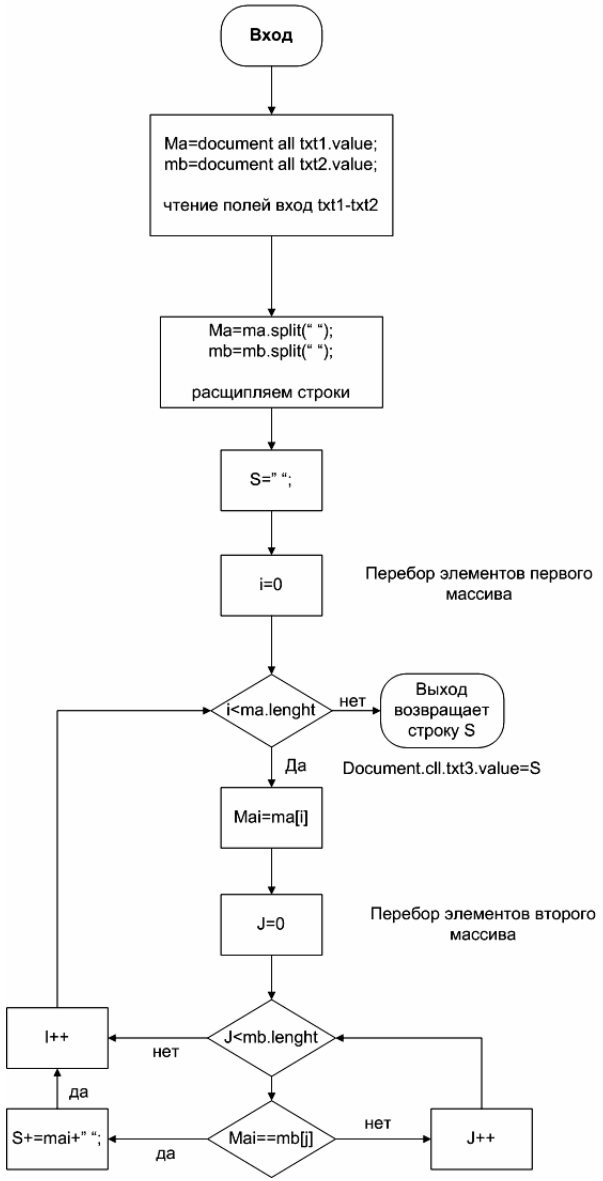
```
<hr color=gold size=3>
<center> <b> <u> <font color=blue size+1>
Лабораторная работа №1.<br> Автор: Терещенко Алена КИТ45а
</font> </u> </b> </center>
<hr color=gold size=3> <b>Множество A:</b>
<input type='text' size=66 name='text1' value='1 2 3 4 5'>
<b> <br>Множество B:</b>
<input type='text' size=66 name='text2' value='4 5 6 7'>
<b> <br>Результат:</b>
<input type='text' size=66 name='text3' value=''> <br>
<center>
<input type='button' name='but1' value='Сравнить' onclick='f()'>
</center>
```


Для правильного функционирования данного интерфейса необходимо написать функцию **f**, реализующую алгоритм сравнения двух множеств [4].

Возможное решение может иметь вид:

```
<script language="JavaScript">
function f()
{ma=document.all.text1.value; // Извлекаем строку из поля ввода
mb=document.all.text2.value;
ma=ma.split(' '); // Расщепляем строку в массив
mb=mb.split(' ');
s=''; // Строка результата пустая
for(i=0;i< ma.length;i++) // Перебираем элементы 1-го массива
{mai=ma[i];
for(j=0;j< mb.length;j++)  // Перебираем элементы 2-го массива
if(mai==mb[j]) // Сравниваем элементы разных массивов
{s+=mai+' '; // Найден одинаковый элемент
break; // Дальше сравнивать не стоит
}}
document.all.text3.value=s; // Строку результата переносим в поле
вывода результата}
</script>
```

Блок-схема данного алгоритма может иметь вид, как показано на рис. 1.2.



Чертеж выполнила студентка Ковальчук Наталья, группа КИТ-66.

Рис. 1.2

При создании блок-схемы алгоритма [8] необходимо придерживаться соответствия между действием алгоритма и его геометрическим представлением (ГОСТ 19.701-90, см. табл.1.1).

Таблица 1.1

Название блока	Обозначение	Название блока
Терминатор	Действие	Начало, завершение программы или подпрограммы
Процесс	Действие	Обработка данных (вычисления, пересылки и т. п.)
Данные	Данные	Операции ввода-вывода
Решение	Условие	Ветвления, выбор, итерационные и поисковые циклы
Подготовка	Действия	Счетные циклы
Граница цикла	Начало Конец	Любые циклы
Предопределенный процесс	Имя	Вызов процедур
Соединитель	Имя	Маркировка разрывов линий
Комментарий	Комментарий	Пояснения к операциям

С помощью текстового редактора Блокнот создать файл **prog1.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование

программы на контрольных примерах. **Блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

1.4. Содержание отчета

Печатный отчет должен соответствовать требованиям по оформлению документов такого типа и содержать следующую информацию:

1. Титульный лист.
2. Название лабораторной работы.
3. Тема лабораторной работы.
4. Краткое описание метода выполнения.
5. Блок-схема алгоритма решения поставленной задачи.
6. Краткое объяснение интерфейса программы.
7. Листинг кода программы.
8. Результаты контрольных расчетов.
9. Краткие выводы.

1.5. Контрольные вопросы

1. Может ли множество содержать одинаковые элементы?
2. Что называется мощностью множества?
3. Перечислите основные теги, используемые для формирования интерфейсной части программы.
4. С помощью какого атрибута определяется обработка события: нажатие управляющей кнопки «Сравнить»?
5. Между какими тегами помещается текст функции вашей программы?
6. Может ли программа обрабатывать нечисловые множества?
7. Найдите на блок-схеме, как изображается оператор break. Для чего он используется в программе.
8. Кратко объясните назначение метода split.
9. Каким образом осуществляется в программе доступ к свойствам объектов, определяемых в интерфейсной части?

10. Назовите расширение, которое использует программа Microsoft Visio, при стандартном сохранении рабочего документа.
11. Что вы знаете о спиральной или итерационной схеме разработки программного обеспечения?
12. Каким ГОСТом необходимо руководствоваться при составлении схем алгоритмов?

2. ЛАБОРАТОРНАЯ РАБОТА № 2. МОДЕЛИРОВАНИЕ ОСНОВНЫХ ОПЕРАЦИЙ ДЛЯ ДВУХ ЧИСЛОВЫХ МНОЖЕСТВ

2.1. Цель работы

Изучить способы численного моделирования основных операций для множеств и составить компьютерную программу для выполнения этих операций над двумя конечными множествами.

2.2. Краткие теоретические сведения [2, 5, 9–11]

Если множества A, B$\in$ P(U), то **объединением** A$\cup$B множеств A и B определяется равенством A$\cup$B = {x | x $\in$ A или x $\in$ B }. **Пересечение** множеств A и B называется также их произведением и обозначается A·B, а объединение – суммой: A+B. Множество A\B называется **разностью** и определяется равенством A\B = {x | x $\in$ A и x $\notin$ B }.

Симметричной разностью называется конструкция A$\oplus$B $\equiv$ AΔB $\equiv$ (A\B) $\cup$ (B\A). Множество $\overline{A} \equiv U \setminus A$ является дополнением A в U

Для наглядного представления соотношений между множествами используются **диаграммы Эйлера-Венна** (см. рис. 2.1).

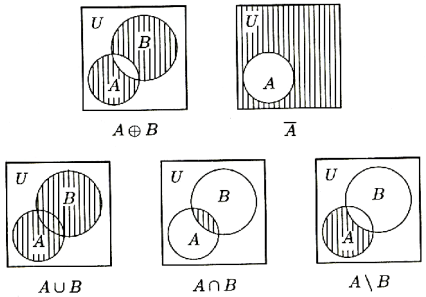


Рис. 2.1

Операции над множествами обладают следующими свойствами:

1. **Ассоциативность** операций пересечения и объединения

A∪(B∪C) = (A∪B)∪C, A∩(B∩C) = (A∩B)∩C.

2. **Коммутативность** операций пересечения и объединения

A∪B= B∪A, A∩B= B∩A.

3. **Идемпотентность** операций пересечения и объединения

A∪A= A, A∩A= A

4. Законы **дистрибутивности**

A∪(B∩C) = (A∪B)∩(A∪C), A∩(B∪C) = (A∩B) ∪ (A∩C).

5. Законы **поглощения**

A∪(A∩B) = A, A∩(A∪B) = A

6. Законы **де Моргана**

$\overline{(A \cup B)} = \overline{A} \cap \overline{B}$, $\overline{(A \cap B)} = \overline{A} \cup \overline{B}$.

7. Законы нуля и единицы (0≡∅, 1≡U)

A∪0=A, A∩0=0, A∪1=1, A∩1=A, $\overline{A} \cup A = 1$, $\overline{A} \cap A = 0$.

8. Закон **двойного поглощения**

$\overline{\overline{A}} = A$.

2.3. Порядок выполнения работы

Составить компьютерную программу для выполнения следующих операций двух конечных числовых множеств: разности, пересечения, объединения, симметричной разности. Программа должна

генерировать случайную последовательность входных данных, упорядочивать поля с данными и результатами по возрастанию.

Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- помеченных полей ввода для двух множеств;
- поля вывода результатов последней операции;
- индикатора для отображения типа последней выполненной операции;
- управляющих кнопок для активизации основных операций: разности (A\B и B\A), пересечения, объединения и симметричной разности;
- кнопки для формирования случайной последовательности входных данных исходных множеств A и B.

На рис. 2.2 приведен возможный вид интерфейса.

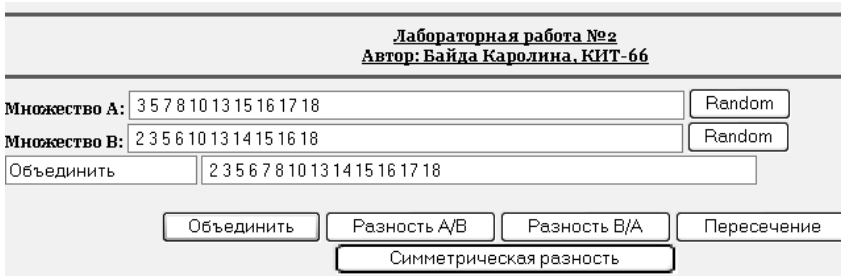


Рис. 2.2

Данный интерфейс может быть реализован такой последовательностью тегов:

<hr color=yellow size=3>   <u>

Лабораторная работа №2

*
Автор: Байда Каролина, КИТ-66</u> *

```
<hr color=yellow size=3> <b>Множество A:</b>
<input type=text name=txt1 size=66 value='1 2 3 4 5' onchange='f2()'>
<input type=button value='Random' onclick='rnd(1,20,10,1)'> <br>
<b>Множество B:</b>
<input type=text name=txt2 size=66 value='4 5 6 7 8'  onchange='f2()'>
<input type=button value='Random' onclick='rnd(1,20,10,2)'> <br>
<b> <input type=text name=txt4  size=20 value=''>
<input type=text name=txt3  size=66 value=''> <p> <center>
<input type=button name=but1 value='Объединить' onclick='g(0)'>
<input type=button name=but2 value='Разность A/B' onclick='g(1)'>
<input type=button name=but3 value='Разность B/A' onclick='g(2)'>
<input type=button name=but4 value='Пересечение' onclick='g(3)'>
<br> <input type=button name=but5 value='Симметрическая разность'
onclick='g(4)'> </center>
```

Алгоритм решения данной задачи может быть построен на функции $f_1(n)$, которая может быть построена будет модификации функции f() из 1-й лабораторной работы. Достаточно добиться, чтобы она вычисляла следующие величины в зависимости от входного параметра n:

$$f_1(n) = \begin{cases} A \cap B,\ n = 0; \\ A \setminus B,\ n = 1; \\ B \setminus A,\ n = 2. \end{cases}$$

Имея такую функцию f1(n), можно построить рабочую функцию g(k) по следующему правилу:

$$g(k) = \begin{cases} f_1(0) + f_1(1) + f_1(2),\ k = 0 \to A \cup B; \\ f_1(1),\ k = 1 \to A \setminus B; \\ f_1(2),\ k = 2 \to B \setminus A; \\ f_1(0),\ k = 3 \to B \cap A; \\ f_1(1) + f_1(2),\ k = 4 \to B \Delta A. \end{cases}$$

Возможное решение может иметь вид:

```
<script language="JavaScript">
```



```
function rnd (n1,n2,kmax,stxt) //Генератор последовательности случайных
{ s=" ";                       // натуральных чисел от n1 до n2 kmax штук
if ((n2-n1)<kmax)kmax=n2-n1;
n2-=n1
knum=0;
do
{ s0=Math.round(Math.random() *n2) +n1;
s1=" "+s0+" ";
if(s.indexOf(s1)==-1) {s+=s0+" ";knum++} }
while(knum<kmax)
eval("document.all.txt"+stxt+".value=s")
f2(); fsort(stxt);}

function f2() //Очищаем поля результата и индикатор последней операции
{document.all.txt3.value='';
document.all.txt4.value='';}

function fsort(n) //Сортировка числовых множеств по возрастанию
{s=eval("document.all.txt"+n+".value");
s=s.split(" ");
ns=s.length;
for (i=0; i<ns; i++) s[i]=1*s[i];
// Сортировка простым обменов
for (i=0; i<ns; i++)
for (j=0; j<ns; j++)
if(s[j]>s[j+1])
{p=s[j];
s[j]=s[j+1];
s[j+1]=p;}
s0="";
for (i=0; i<ns; i++) if (s[i]>0) s0+=s[i]+" ";
eval("document.all.txt"+n+".value=s0"); }
```

```
function g(k) //Сведение общей задачи к более элементарным
{var ind=new Array();
ind[0]='Объединить';ind[1]='Разность A и B';ind[2]='Разность B и A';
ind[3]='Пересечение'; ind[4]='Симметричная разность';
if (k==0)  document.all.txt3.value=f1(0)+f1(1)+f1(2);
if (k==1) document.all.txt3.value=f1(1);
if (k==2) document.all.txt3.value=f1(2);
if (k==3) document.all.txt3.value=f1(0);
if (k==4) document.all.txt3.value=f1(1)+f1(2);
document.all.txt4.value=ind[k];
fsort(3) ;}

function f1(n) // Поиск пересечения или разности двух множеств
{ ma=document.all.txt1.value;
mb=document.all.txt2.value;
if (n==2)
{temp=ma; ma=mb; mb=temp;}
ma=ma.split(' ');
mb=mb.split(' ');
s=""; s1=" "; flag=true;
for (i=0; i<ma.length; i++)
{flag=true;
mai=ma[i];
for(j=0; j<mb.length; j++)
if(mai==mb[j])
{s+=mai+" ";
flag=false;
break;}
if(flag) s1+=mai+" "; }
if(n==0) return s;
return s1;}
</script>
```

Функциональная блок-схема данного алгоритма может иметь вид, как показано на рис. 2.3.

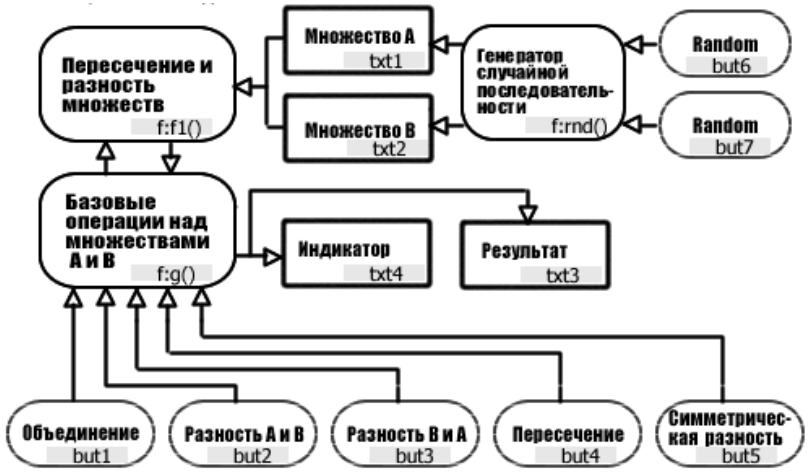


Рис. 2.3

С помощью текстового редактора Блокнот создать файл **prog2.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

2.4. Содержание отчета

Смотри пункт 1.4.

2.5. Контрольные вопросы [6,7]

1. Какой вид сортировки использован при написании функции fsort(n)?
2. Обязателен ли символ точка с запятой в конце оператора в языке JavaScript?
3. Можно ли с помощью данной программы проиллюстрировать равенства для множеств A и B: $A \cup (A \cap B) = A$, $A \cap (A \cup B) = A$?

4. Можно ли в рамках данной программы реализовать операцию дополнение $\overline{A}$ для множества A?
5. Что необходимо переделать в программе для того, чтобы проиллюстрировать равенства для множеств A, B и C: (A\B)\C = A\(B∪C)?
6. Кратко охарактеризуйте назначение параметров в функции генерации случайной последовательности чисел rnd(n1,n2,kmax,stxt).
7. Объясните назначение атрибута onchange в теге <input>.
8. Равны ли множества ∅ и {∅}?

3. ЛАБОРАТОРНАЯ РАБОТА № 3. ПОСТРОЕНИЕ МАТРИЦЫ БИНАРНОГО ОТНОШЕНИЯ

3.1. Цель работы

Изучить способы численного моделирования матрицы бинарного отношения и разработать компьютерную программу для построения матрицы бинарного отношения на двух заданных числовых множествах.

3.2. Краткие теоретические сведения [6, 9, 10]

Декартовым произведением множеств A и B называют множество M, содержащее все возможные пары, в которых на первом месте стоит элемент из A, на втором – элемент из B. Формально A×B = M, M={(a_i, b_j)| a_i∈A, b_j∈B}.

Рассмотрим два конечных множества A = {a_1, a_2, ..., a_m}, B = {b_1, b_2, ..., b_m} и бинарное отношение P ⊆ A×B. Определим матрицу (**смежности**) [P] = (p_{ij}) размера m×n бинарного отношения P по следующему правилу:

$$p_{ij} = \begin{cases} 1, \textit{если } (a_i, b_j) \in P, \\ 0, \textit{если } (a_i, b_j) \notin P. \end{cases}$$

Полученная матрица содержит полную информацию о связях между элементами и позволяет представлять эту информацию на

компьютере. Заметим, что любая матрица, состоящая из нулей и единиц, является матрицей некоторого бинарного отношения.

Обратное отношение P^{-1} определяется как P^{-1}={(a,b) | (b,a)∈P}. **Композицией** бинарных отношений P⊆A×B и Q⊆B×C называется множество P°Q={(x,y)|x∈A, y∈C, ∃z∈B, такой, что (x,z)∈P, (z,y)∈Q}. Пусть P – бинарное отношение на множестве A: $P \subseteq A^2$. Отношение P называется **рефлексивным**, если (x,x)∈P для всех x∈A. Отношение называется **симметричным**, если для любых x,y∈A из (x,y)∈P следует (y,x)∈P, т.е. P^{-1}=P, или $[P]=[P]^T$. Отношение P называется **антисимметричным**, если из (x,y)∈P и (y,x)∈P следует, что x=y. Отношение P называется транзитивным если из (x,y)∈P и (y,z)∈P следует (x,z)∈P, т.е. $P \circ P \subseteq P$. При этом матрица композиции $[P \circ P] = [P] \cdot [P]$, где умножение матриц производится по обычному правилу умножения матриц, но при сложении используются следующие правила: 0+0=0, 1+1=1, 1+0=0+1=1.

Бинарное отношение на некотором множестве называют:

- **эквивалентностью**, если оно рефлексивно, симметрично и транзитивно;
- **толерантностью**, если оно рефлексивно и симметрично;
- **порядком** (или частичными порядком), если оно рефлексивно, антисимметрично и транзитивно;
- **предпорядком** (или квазипорядком), если оно рефлексивно и транзитивно;
- **строгим порядком**, если оно и рефлексивно, антисимметрично и транзитивно;
- **строгим предпорядком**, если оно и рефлексивно и транзитивно.

3.3. Порядок выполнения работы

Составить компьютерную программу для построения матриц следующих бинарных отношений:

- “a > b” (a больше b);
- “a ≥ b” (a больше или равно b);
- “a = b” (a равно b);

- “a mod b=0” (a делится b);
- “a mod 2=0 или b mod 2=0“ (a или b является четным числом);
- “a mod 2=0 и b mod 2=0“ (a и b одновременно являются четными числами).

Здесь элемент a принадлежит множеству A (a∈A), b – B (b∈B).

Программа должна генерировать случайную последовательность входных данных, упорядочивать поля с входными данными по возрастанию, динамически строить булеву матрицу заданного отношения.

Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- помеченных полей ввода для двух множеств;
- кнопки для формирования случайной последовательности входных данных исходных множеств A и B;
- индикатора для отображения текущего отношения;
- кнопки для пересчета матрицы бинарного отношения;
- матрица бинарного отношения с двумя цветами ячеек.

На рис. 3.1 приведен возможный вид интерфейса.

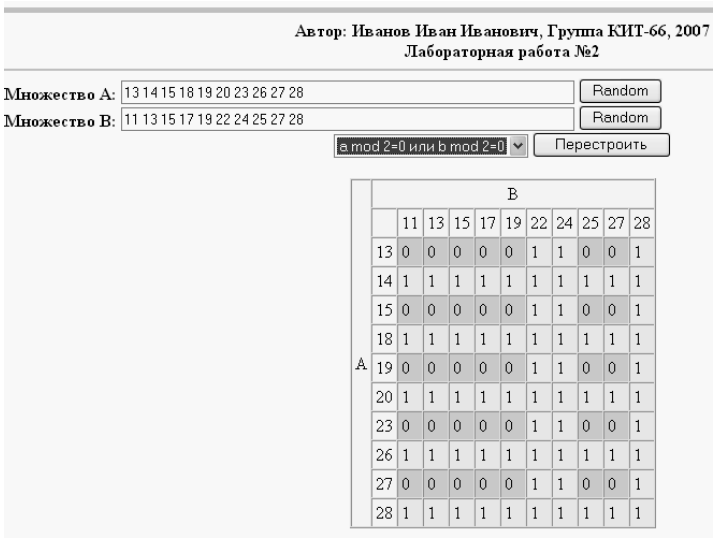


Рис. 3.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<center><hr color=gold size=5><b>Автор: Иванов Иван Иванович,
Группа КИТ-66, 2007
<br>Лабораторная работа №2</b><hr color=gold size=2></center>
<b>  Множество A: </b>
<input type=text name=txt1 size=66 value='' onchange='fsort(1);g()'>
<input type=button value='Random' onclick='rnd(10,30,10,1); g()'><br>
<b>  Множество B: </b>
<input type=text name=txt2 size=66 value='' onchange='fsort(2);g()'>
<input type=button value='Random' onclick='rnd(10,30,10,2); g()'>
<center><select name=sel1 onchange=g()>
<option>a>b</option><option>a>=b</option>
<option>a=b</option><option>a mod b=0</option>
<option>a mod 2=0 или b mod 2=0</option>
<option>a mod 2=0 и b mod 2=0</option></select>
 <input type=button name=but1 value='Перестроить' onclick='g()'>
</center><div ID="info"> </div>
```

Для правильного функционирования данного интерфейса необходимо написать функцию **g**, реализующую алгоритм построения бинарного отношения. Функции **rnd** и **fsort** могут быть взяты из предыдущей программы.

Возможное решение может иметь вид:

```
<script language="JavaScript">
w = new Array(20);
for(i=0;i<20;i++) w[i] = new Array(20);
rnd(10,30,10,1); rnd(10,30,10,2);
g();

function g() // Строим матрицу бинарного отношения
{document.all.info.innerHTML="";
ma=document.all.txt1.value; mb=document.all.txt2.value;
ma=ma.split(" ";);
mb=mb.split(" ");
na=ma.length-1;
nb=mb.length-1;
for (i=0;i<na;i++) ma[i]=1*ma[i];
for (j=0;j<nb;j++) mb[j]=1*mb[j];
for (i=0;i<na;i++)
for (j=0;j<nb;j++)
w[i][j]=0;
k=1*document.all.sel1.selectedIndex;

for (i=0;i<na;i++)
for (j=0;j<nb;j++)
{switch (k) // Выбор режима расчета
{case 0: {if (ma[i] > mb[j])  w[i][j]=1; break;}
case 1: {if (ma[i] >= mb[j])  w[i][j]=1; break;}
case 2: {if (ma[i] == mb[j])  w[i][j]=1; break;}
case 3: {if ((ma[i] % mb[j]) ==0)  w[i][j]=1; break;}
case 4: {if ( ( ma[i] % 2 ==0) || (mb[j] % 2 ==0) )  w[i][j]=1; break;}
case 5: {if ( ( ma[i] % 2 ==0) && (mb[j] % 2 ==0) )  w[i][j]=1; break;}}
```



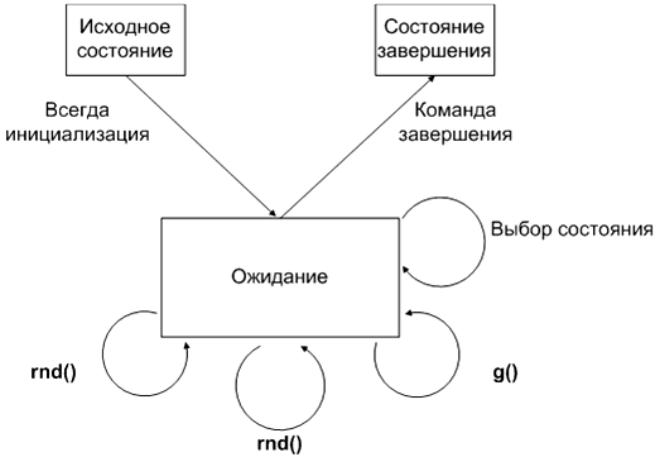
```
}  // Формируем таблицу результатов с помощью тегов
s="<br><table border=1 align=center cellpadding=3 cellspacing=0>"
s+="<tr bgcolor=ffffcc><td rowspan="+(na+2)+"><br>A</td><td
align=center colspan="+(nb+1)+">B</td></tr>";
s+="<tr bgcolor=ffffcc><td> </td>";
for (j=0;j<nb;j++) s+="<td>"+mb[j]+"</td>";
s+="</tr>"
for (i=0;i<na;i++)
{s+="<tr><td bgcolor=ffffcc>"+ma[i]+"</td>";
for (j=0;j<nb;j++)
{if (w[i][j]==1) s+="<td bgcolor=ccffcc>";
if (w[i][j]==0)s+="<td bgcolor=ffcccc>";
s+=w[i][j]+"</td>"}
s+="</tr>";}
s+="</table>";
document.all.info.innerHTML=s;}

function rnd(n1,n2,kmax,stxt) // Генератор последовательности
случайных
{ s=" ";                          // натуральных чисел от n1 до n2 kmax
штук
if ((n2-n1)<kmax) kmax=n2-n1;
n2-=n1; knum=0;
do
{s0=Math.round(Math.random() * n2) + n1
s1=""+s0+"";
if (s.indexOf(s1)==-1) {s+=s0+" ";knum++}}
while (knum<kmax)
eval("document.all.txt"+stxt+".value=s")
fsort(stxt)}

function fsort(n) // Сортировка числовых множеств по возрастанию
{s=eval("document.all.txt"+n+".value");
s=s.split(" ");
ns=s.length;
```

```
for (i=0;i<ns;i++)
s[i]=1*s[i];
for (i=0;i<ns;i++)
for (j=0;j<ns-i;j++)
if (s[j]>s[j+1])
{p=s[j];
s[j]=s[j+1];
s[j+1]=p;}
s0="";
for (i=0;i<ns;i++)
if (s[i]>0)s0+=s[i]+" ";
eval("document.all.txt"+n+".value=s0");}
</script>
```

С помощью текстового редактора Блокнот создать файл **prog3.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Блок-схему в виде переходов состояний** (об этом типе блок-схем подробно рассказано в книге [8], возможное решение представлено на рис. 3.2) программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.



Чертеж выполнила студентка Михайлова Юлия, группа КИТ-46Б.

Рис. 3.2

Меняя входные данные, провести исследование свойств соответствующих бинарных отношений.

3.4. Содержание отчета

Смотри пункт 1.4.

3.5. Контрольные вопросы [7, 11]

1. Пусть A и B – конечные множества, состоящие из *m* и *n* элементов соответственно. Сколько существует бинарных отношений между множествами A и B?
2. Построить бинарное отношение:
   - рефлексивное, симметричное, не транзитивное;
   - не рефлексивное, антисимметричное, не транзитивное;
   - рефлексивное, не симметричное, транзитивное.
3. Определите, какие из следующих отношений на множестве людей рефлексивны, симметричны или транзитивны:
   - «… иметь тех же родителей, что и …»;
   - «… являться братом …»;
   - «… старше или младше, чем …»;
   - «… не выше, чем …».
4. Структура завода включает администрацию, бухгалтерию, производственный цех, автотранспортный цех. Введено бинарное отношение T «руководящая структура – подчиненное подразделение». Задать это бинарное отношение на матрице смежности и определить его свойства.
5. Объясните механизм действия оператора eval в функции, написанной на языке JavaScript.
6. Каким образом в программе достигается динамический характер отображения матрицы бинарного отношения?
7. Как изменить приведенную в этой лабораторной работе программу, чтобы она определяла, обладает ли данное бинарное отношение свойствами рефлексивности, симметричности и транзитивности?

8. Перечислите теги, которые формируют таблицу? Используя теги, создайте таблицу размером три на четыре?

4. ЛАБОРАТОРНАЯ РАБОТА № 4. ГЕНЕРАЦИЯ РАЗМЕЩЕНИЙ

4.1. Цель работы

Изучить способы численного моделирования размещений и разработать компьютерную программу для генерации всех размещений для заданных значений параметров в лексикографическом порядке.

4.2. Краткие теоретические сведения [2, 10, 11]

Раздел математики, посвященный решению задач выбора и расположения элементов некоторого, обычно конечного, множества в соответствии с заданными правилами, называется **комбинаторикой**. Каждое такое правило определяет способ построения некоторой конструкции из элементов исходного множества, называемой **комбинаторной конфигурацией.** Главными целями комбинаторного анализа является изучение комбинаторных конфигураций, алгоритмов их построения и оптимизации, решение задач перечисления. Простейшими конфигурациями являются **перестановки**, **размещения, сочетания** и **разбиения**. Пусть дано множество M={a_1, a_2,…, a_n}. **Перестановкой** элементов множества M называется любой упорядоченный набор (**кортеж**) (a_{i1}, a_{i2}, …, a_{in}), состоящий из n различных элементов множества M. **Число всех перестановок P_n** равно $n!$. Определим число $m \le n$. **Размещением** из n элементов по m или упорядоченной (n,m)-выборкой, называется любой кортеж (a_{i1}, a_{i2}, …, a_{im}), состоящий из m попарно различных элементов множества M. **Число размещений** из n по m обозначается через A_n^m или $P(n,m)$ и равно $A_n^m = \dfrac{n!}{(n-m)!} = n(n-1)...(n-m+1)$.

(Предполагается, что 0!=1). **Сочетанием** из из n элементов по m или неупорядоченной (n,m)-выборкой, называется любое подмножество множества M, состоящее из m элементов. **Число сочетаний** из n элементов обозначается через C_n^m или $C(n,m)$ и равно $C_n^m = \dfrac{n!}{(n-m)!m!}$. Очевидно, что $A_n^m = m! \cdot C_n^m$. Число C_n^m обладает следующими свойствами:

- $C_n^m = C_n^{n-m}$;
- $C_n^m + C_n^{m+1} = C_{n+1}^{m+1}$;
- $(a+b)^n = \sum_{m=0}^{n} C_n^m a^m b^{n-m}$ для любых $a,b \in R$, $n \in N$ (бином Ньютона).

В силу последнего свойства числа C_n^m называются биноминальными коэффициентами.

Размещением с повторением из n элементов по m или упорядоченной (n,m)–выборкой с возвращениями называется любой кортеж (a_1, a_2, …, a_m) элементов множества M, для которого $|M| = n$. **Число размещений с повторениями** $\widetilde{P}(n,m)$ равно n^m. **Сочетанием с повторением** из n элементов по m или неупорядоченной (n,m)–выборкой с возвращениями называются множества, которые состоят из элементов, выбранных m раз из множества M, причем один и тот же элемент допускается выбирать повторно. **Число сочетаний с повторениями** из n элементов по m обозначается $\widetilde{C}(n,m)$ и вычисляется по формуле $\widetilde{C}(n,m) = C_{n+m-1}^m = \dfrac{(n+m-1)!}{m!(n-1)!}$.

4.3. Порядок выполнения работы

Составить компьютерную программу генерации всех размещений для заданных значений n и m в лексикографическом порядке. Программа должна позволять пользователю устанавливать значения n и m из заданного интервала, динамически строить множество размещений, вычислять значения A_n^m, $(m \le n)$.

Программу реализовать:

– интерфейсная часть на языке разметки гипертекста HTML [3];
– функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- выпадающего меню для выбора значения параметра ***n***;
- выпадающего меню для выбора значения параметра ***m***;
- динамически отображаемое множество размещений для заданных значений параметров ***m*** и ***n***;
- текущее значение A_n^m.

На рис. 4.1 приведен возможный вид интерфейса.

Рис. 4.1

Данный интерфейс может быть реализован такой последовательностью тегов:

* *
* Генерация размещений*
*A^m_n
*
n = <input type="button" value="+" style="font-size:8pt;width:15pt;"
onclick="document.all.nsel.selectedIndex+=1;if
(document.all.nsel.selectedIndex==-1)
document.all.nsel.selectedIndex=0;g()"><select name="nsel"
style="font-family:Tahoma;font-size:8pt;" onchange="g()">
<option value="1">1</option><option value="2">2</option>
<option value="3">3</option><option value="4" selected>4</option>
<option value="5">5</option><option value="6">6</option>

</select><input type="button" value="-" style="font-
size:8pt;width:15pt;" onclick="document.all.nsel.selectedIndex-=1;if
(document.all.nsel.selectedIndex==-1)
document.all.nsel.selectedIndex=0;g()">
m = <input type="button" value="+" style="font-size:8pt;width:15pt;"
onclick="document.all.msel.selectedIndex+=1;if
(document.all.msel.selectedIndex==-1)
document.all.msel.selectedIndex=0;g()"><select name="msel"
style="font-family:Tahoma;font-size:8pt;" onchange="g()">
<option value="1">1</option><option value="2">2</option>
<option value="3" selected>3</option><option value="4">4</option>
<option value="5">5</option><option value="6">6</option>
</select><input type="button" value="-" style="font-
size:8pt;width:15pt;" onclick="document.all.msel.selectedIndex-=1;if
(document.all.msel.selectedIndex==-1)
document.all.msel.selectedIndex=0;g()">
* *

Для правильного функционирования данного интерфейса необходимо написать функцию **g**, реализующую рекурсивный алгоритм построения размещений [12]. Текст решения на языке программирования Паскаль имеет вид:

Program GPlac;
Const n=4 ; m=3; *{ *Значения пит взяты в качестве примера. *}*
Var
A:Array[1..m] Of Integer; *{*Массив для хранения элементов размещения. *}*
S:Set Of Byte; *{*Для хранения использованных в размещении цифр. *}*
Procedure Solve(t:Integer);*{*Параметр t определяет номер позиции в размещении. *}*
Var i:Byte;
Begin
For i:=l To n Do *{*Перебираем цифры и находим 1-ю неиспользованную. *}*
If Not (i In S) Then Begin
S:=S+[i]; *{* Запоминаем её в множестве занятых цифр. *}*
A[t]:=i; *{* Записываем её* в *размещение. *}*
If t<m Then Solve(t+1) Else Print; *{*Если размещение не получено, то идем к следующей позиции, иначе выводим очередное размещение. *}*

```
S:=S-[i]; {* Возвращаем цифру в число неиспользованных. *}
End;
End;
```
Фрагмент основной программы.
```
Begin   S: = [];   Solve(1);  End.
```

Возможное решение может иметь вид:

```
<script language="JavaScript">
g(); // Вызов функции при загрузке страницы в браузере
function g()
{var n, m, s=' ', k=0, sum=' ';
if (document.all.nsel.selectedIndex<document.all.msel.selectedIndex)
document.all.msel.selectedIndex=document.all.nsel.selectedIndex
n=document.all.nsel.options[document.all.nsel.selectedIndex].value
m=document.all.msel.options[document.all.msel.selectedIndex].value
sum+='{<br>';
f(1);
sum+='<br>}<br>A<sup>'+m+'</sup><sub>'+n+'</sub> = '+k;
document.all.info1.innerHTML=sum;
function f(t)
{var i, j, d;
for(i=1;i<=n;i++)
{if (s.indexOf(' '+i+' ')==-1)
{s+=i+' ';
if(t<m)
f(t+1);
else
{k++;
sum+=s+'  ';
if (k%10==0) sum+='<br>';}
d=s.split(' ');
s=' ';
for(j=0;j<d.length;j++) if (i!=(1*d[j]) && (1*d[j])!=0) s+=d[j]+' ';
}}}}
</script>
```


С помощью текстового редактора Блокнот создать файл **prog4.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Блок-схему в виде структурные карт Константайна** (об этом типе блок-схем подробно рассказано в книге [8], возможное решение представлено на рис. 4.2) программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

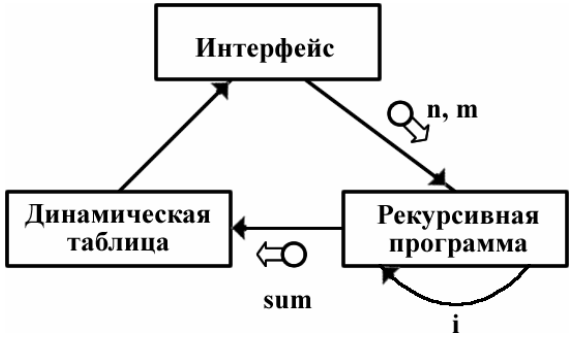


Рис. 4.2

4.4. Содержание отчета

Смотри пункт 1.4.

4.5. Контрольные вопросы [2]

1. Группе из пяти сотрудников выделено три путевки. Сколько существует способов распределения путевок, если:
   - все путевки различны;
   - все путевки одинаковы.
2. Сколько различных десятичных чисел можно написать, используя цифры 0, 1 и 2?
3. Сколько различных перестановок образуется из следующих слов: а) зебра; б) водород?
4. Чему равно значение суммы $\sum_{m=0}^{n} C_n^m$?

5. Можно ли с помощью разработанной программы построить все перестановки из *n* элементов?
6. Какой оператор в программе ограничивает рекурсивный спуск?
7. Что означает запись *document.all.info1.innerHTML=sum* в тексте программы?
8. С помощью, каких операторов в программе контролируется выполнение требования $m \le n$?

5. ЛАБОРАТОРНАЯ РАБОТА № 5. ПОСТРОЕНИЕ ТАБЛИЦЫ ИСТИННОСТИ

5.1. Цель работы

Изучить способы моделирования базовых логических операций и разработать компьютерную программу автоматического построения таблиц истинности для выражений булевой алгебры.

5.2. Краткие теоретические сведения [2, 10, 11]

Булева алгебра – это название области математики, занимающейся логическим анализом. Простейшая булева алгебра состоит из множества B={0,1} вместе с определенными на ней операциями дизъюнкции ($\vee$), конъюнкции ($\wedge$) и отрицания ($\neg$). Булевыми переменными называются переменные принимающие значения 0 и 1. Комбинируя булевы переменные с помощью базовых операций и круглых скобок, получаем **булевы выражения**. Действия логических операций задаются **таблицами истинности**, в каждой строке которых взаимно однозначно сопоставляется набор значений переменных, составляющих булевы выражения, и соответствующее этому набору значение полученного выражения (см. табл. 5.1)

Таблица 5.1

p	q	$\neg$p	(p$\wedge$q)	(p$\vee$q)
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Два булевых выражения называются **эквивалентными**, если они имеют одинаковые таблицы истинности. Вычисляя таблицы истинности, легко установить справедливость некоторых

эквивалентностей, которые принято называть **законами булевой алгебры**:

- **коммутативность** $p \wedge q = q \wedge p$, $p \vee q = q \vee p$;
- **ассоциативность** $p \wedge (q \wedge r) = (p \wedge q) \wedge r$, $p \vee (q \vee r) = (p \vee q) \vee r$;
- **дистрибутивности** $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$,
  $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$;
- **идемпотентности** $p \wedge p = p$, $p \vee p = p$;
- **поглощения** $p \wedge (p \vee q) = p$, $p \vee (p \wedge q) = p$;
- **де Моргана** $\neg(p \wedge q) = \neg p \vee \neg q$, $\neg(p \vee q) = \neg p \wedge \neg q$.

Булевой функцией от n переменных $p_1, p_2, \ldots, p_n$ называется такая функция $f: B^n \to B$, что $f(p_1, p_2, \ldots, p_n)$ – булево выражение.

Элементарной конъюнкцией называется конъюнкция литер. Если x – булева переменная, $\delta \in \{0,1\}$, то выражение $x^\delta = x$, если $\delta = 1$ и $x^\delta = \neg x$, если $\delta = 0$ называется **литерой**.

Минтермом называется булева функция, которая принимает значение 1 только на одном наборе значений аргументов. Произвольную булеву функцию можно записать как дизъюнкцию минтермов. Такая запись называется **дизъюнктивной нормальной формой** и для каждой булевой функции определена единственным образом с точностью до перестановки элементарных конъюнкций.

Множество функций, через которые можно выразить любую булеву функцию, называется **полной системой функций**.

Булево выражение можно упростить, используя **карту Карно,** прямоугольную таблицу, чьи строки и столбцы обозначены конъюнкциями булевых переменных и их отрицаний. В клетках этой таблицы, соответствующих минтермам данной дизъюнктивной нормальной формы, помещаются единицы. Обнаруженные в этой таблице пары минтермов можно сгруппировать и преобразовать в одно простое выражение.

5.3. Порядок выполнения работы

Составить компьютерную программу автоматического построения таблиц истинности для выражений булевой алгебры. Программа должна позволять пользователю вводить произвольное булево

выражение, определять булевы переменные и динамически строить таблицу истинности.

Программу реализовать:

– интерфейсная часть на языке разметки гипертекста HTML [3];

– функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле ввода булева выражения;
- кнопки для активизации процесса вычисления таблицы истинности;
- динамически отображаемую таблицу истинности.

На рис. 5.1 приведен возможный вид интерфейса.

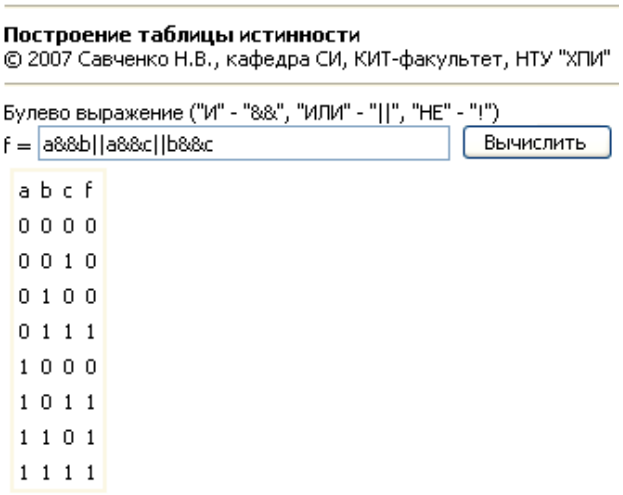


Рис. 5.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-size:8pt;font-family:Tahoma;}</style>
<body class="s1" onload="f()" TOPMARGIN=0>
<hr><b>Построение таблицы истинности</b>
```



```
<br>&copy; 2007 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ "ХПИ" <hr>Логическое выражение ("И" – "&&", "ИЛИ" – "||", "НЕ" – "!")
<br>f = <input type="text" name="txt1" value="a&&b||a&&c||b&&c" size=40 class="s1" onchange="f()"> 
<input type="button" name="b1" value="Вычислить" onclick="f()" class="s1"><br><table border=0 class="s1"><tr>
<td align=center valign=top><span ID="res"> </span></td>
</tr></table></body>
```

Для правильного функционирования данного интерфейса необходимо написать функцию **f**, реализующую алгоритм построения таблицу истинности булева выражения.

Возможное решение может иметь вид:

```
<script language="JavaScript">
var maxlname=50, tlname;
var lname = Array (maxlname);
function f()
{var i,j,si;
s=document.all.txt1.value;
fncol(s);
sres="<table border=0 class='s1' cellspacing=0 cellpadding=3 style='border:0 thin solid;'>";
sres+="<tr>";
for (i=0;i<tlname;i++) sres+="<td>"+lname[i]+"</td>";
sres+="<td> f </td></tr>";
tlname2=Math.pow(2,tlname);
for (i=0;i<tlname2;i++)
{sres+="<tr>";
z=flog(i,tlname);
dz=z.split("|");
for (j=0;j<tlname;j++)
{dzj="0"; if (dz[j]=="true") dzj="1";
sres+="<td>"+dzj+"</td>";
eval(lname[j]+"="+dz[j]);}
si="0"; if (eval(s)) si="1";
```

```
sres+="<td>"+si+"</td></tr>";}
sres+="</table>";
document.all.res.innerHTML=sres;}

function fncol(s) // Определяем количество булевых переменных и их имена
{var sum="",n=s.length;
var i,j,si,flag,ns=0;
for (i=0;i<n;i++)
{si=s.charAt(i);
if ( si=="&" || si=="|" || si=="!" || si=="(" || si==")")
sum+=" ";
else
sum+=si;}
dsum=sum.split(" ");
tlname=0;
for (i=0;i<dsum.length;i++)
if (dsum[i]!="")
{flag=true;
for (j=0;j<tlname;j++)
if (lname[j]==dsum[i]) {flag=false; break;}
if (flag) {lname[tlname]=dsum[i]; tlname++}}}

function flog(t,z) // Вычисляем булево выражение при конкретных значениях
                   // булевых переменных
{var sum1="",t1,z1=0,i;
t1=t;
while (t1!=0)
{if (t1%2==0)
sum1="false|"+sum1;
else
sum1="true|"+sum1;
t1=Math.floor(t1/2);
z1++;}
```



```
for (i=0;i<(z-z1);i++) sum1="false|"+sum1;
return sum1;}
</script>
```

С помощью текстового редактора Блокнот создать файл **prog5.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

5.4. Содержание отчета

Смотри пункт 1.4.

5.5. Контрольные вопросы [7, 11]

1. С помощью разработанной программы докажите закон дистрибутивности.
2. Покажите, что булево выражение $\neg(\neg p \vee q) \wedge (p \vee q)$ эквивалентно p.
3. Является ли булева функция $\neg p \wedge q \wedge r$ минтермом?
4. Найдите дизъюнктивную нормальную форму булевой функции $f=(p \wedge q) \vee (\neg q \wedge r)$.
5. Проверьте соотношения $\neg(p \wedge \neg q) \vee \neg r = \neg p \vee q \vee \neg r$.
6. Вычислите таблицу истинности булева выражения $(p \wedge (\neg q \vee r)) \vee (\neg p \wedge (q \vee \neg r))$ и найдите его дизъюнктивную нормальную форму.
7. Образует ли функция $\neg (q \wedge r)$ полную систему функций?
8. Доказать эквивалентность $p \wedge (p \vee r) \wedge (q \vee r)$ и $(p \wedge q) \vee (p \wedge r)$.

6. ЛАБОРАТОРНАЯ РАБОТА № 6. НАХОЖДЕНИЕ ОСТОВА МИНИМАЛЬНОГО ВЕСА ПО АЛГОРИТМУ ПРИМА–КРАСКАЛА

6.1. Цель работы

Изучить способы моделирования алгоритмов нахождения остова минимального веса и разработать компьютерную программу, реализующую алгоритм Прима–Краскала.

6.2. Краткие теоретические сведения [2, 10, 11]

Графом *G(V,E)* называется совокупность двух непустых множеств – непустого множества *V* (множества **вершин**) и множества *E* двухэлементных подмножеств множества *V* (*E* – множество **рёбер**). Число вершин графа *G* $p=|V|$, а число ребёр – $q=|E|$. Пусть v_1, v_2 – вершины, e = (v_1,v_2) – соединяющее их ребро. Тогда вершина v_1 и ребро *e* **инцидентны**, ребро e и вершина v_2 также инцидентны. Два ребра, инцидентные одной вершине, называются **смежными**; две вершины, инцидентные одному ребру, также называются смежными. Обычно граф изображают диаграммой: вершины – точками (или кружками), рёбра – линиями.

Известны различные способы представления графов в памяти компьютера, которые различаются объёмом занимаемой памяти и скоростью выполнения операций над графами. Представление графа с помощью квадратной булевой матрицы, отражающей смежность вершин, называется **матрицей смежности**. Ребро, ведущее из вершины в нее же, называется **петлей**. Граф без кратных ребер и петель называется **простым**. **Цепью** между вершинами v и u называется последовательность ребер, соединяющих v и u. Если граф ориентированный (орграф), в котором вместо ребер имеются **дуги** (ориентированные стрелки), то аналог цепи называется **путем**.

Связный граф – это граф, где существует цепь между любой парой вершин *v, u*; иногда такой граф называют **односвязным.** В **взвешенном графе** каждому ребру приписывается вес (или длина). Взвешенные графы также называют **сетями**, их часто обозначают *N(V,E,W)*,

Часть вершин и все инцидентные им ребра называются **подграфом**; все вершины и часть инцидентных им ребер называется **суграфом**. **Циклом** называется цепь из *v* в *v*. **Деревом** называется граф без циклов. **Остовным деревом** называется связный суграф графа, не имеющий циклов. **Полным графом** называется граф, в котором проведены все возможные ребра (в полном графе с *n* вершинами имеется *n (n–1)/2* ребер). Дерево с n вершинами имеет *n–1* ребер.

В терминах теории графов **задача Прима–Краскала** выглядит следующим образом:

Дан полный граф с n вершинами. Найти остовное дерево минимальной длины.

В таком виде задача была поставлена и решена Примом (1961). Краскал, одновременно и независимо, поставил и решил задачу не для плоского случая, где расстояния определяются по формуле

$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ (здесь (x_i,y_i) – декартовы координаты i-й вершины), а для произвольных положительных d_{ij}, $i,j = 1, \ldots, n$. При этом для некоторых пар индексов $d_{ij} = \infty$, что означает отсутствие ребра, т.е. рассматривается любой граф, а не только полный. Задача Прима–Краскала может быть решена с помощью жадного алгоритма, а именно:

в цикле n-1 раз делай: выбрать самое короткое еще не выбранное ребро при условии, что оно не образует цикла с уже выбранными.

Способ реализации алгоритма может выглядеть следующим образом. До построения дерева окрасим каждую вершину *i* в отличный от других цвет *i*. При выборе очередного ребра, скажем (*i,j*), где *i* и *j* имеют разные цвета, вершина *j* и все, окрашенные в ее цвет (т.е. ранее с ней соединенные) перекрашиваются в цвет *i*. Таким образом, выбор вершин разного цвета обеспечивает отсутствие циклов. После выбора *n-1* ребер все вершины получают один цвет.

Итак, более подробный алгоритм выглядит так.

1. *(ввод). Ввести матрицу расстояний* $D = \{d_{ij}\}$, $i,j = 1,\ldots,n$.
2. *(инициализация). Приписать разные цвета всем вершинам:* $col_i=i$; *длина дерева* $L:=0$.

3. *(общий шаг). В цикле по k:= 1 to n-1 do найти ребро минимальной длины между вершинами разного цвета; пусть это ребро (i,j).*
   Запомнить результат: Res1[k]:=i; Res2[k]=j.
   Перекрасить вершины: i1:=col[i]; j1:=col[j].
   В цикле по m:=1 to n do
   if col[m]=j1 then col[m]:=i1.
   Нарастить длину дерева: L:=L+ d[i,j].
   Конец цикла по k.
4. *(вывод). Вывести Res1, Res2.*

Оценим требуемую память и требуемое число операций для алгоритма Прима–Краскала. В варианте Прима надо хранить *2n* координат точек, в варианте Краскала – n^2 расстояний; в обоих вариантах удобно хранить *2(n-1)* номеров вершин, т. е. *n-1* ребер ответа. Всего требуется памяти $O(n^2)$.Для нахождения текущего минимального ребра надо просмотреть $O(n^2)$ чисел и сделать это надо *n-1* раз, так что временная сложность алгоритма $O(n^3)$.

6.3. Порядок выполнения работы

Составить компьютерную программу для автоматического построения остова минимального веса для графа, заданного матрицей смежности. Программа должна позволять пользователю вводить граф разной размерности, находить остов минимального веса путем перечисления соответствующих ребер.

Программу реализовать:

– интерфейсная часть на языке разметки гипертекста HTML [3];

– функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле матрицы смежности заданного графа;
- кнопки для активизации процесса нахождения минимального остовного дерева;
- динамически отображаемую таблицу результата, включающую перечень ребер дерева и общую длину;

- динамически формировать пустую матрицу смежности разной размерности;
- возможность принудительно приведения матрицы смежности к симметричному виду;
- встроенную проверку работоспособности алгоритма на тестовых примерах.

На рис. 6.1 приведен возможный вид интерфейса.

Рис. 6.1

Данный интерфейс может быть реализован такой последовательностью тегов:

<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<body class="s1" onload="fprim();fpoisk();" TOPMARGIN=2>
*<hr size=4>Нахождение остова минимального веса по алгоритму <u>Прима–Краскала</u>
© 2007 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ "ХПИ"<hr>*
<input type="button" value="Вычислить" onclick="fpoisk();" class="s1">
* <input type="button" value="Пример" onclick="fprim();" class="s1">*

```
<select name="sel3" class="s1">
<option value="1">Вариант 1</option>
<option value="2">Вариант 2</option>
</select><div style="margin-top:3pt;">
<inpu type="button" value="Матрица смежности" onclick="fcrmat()"
class="s1"> <select name="sel1" class="s1">
<option value="5">5</option><option value="6">6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option value="10">10</option>
</select> <input type="button" value="Симметризовать"
onclick="fsym()" class="s1"></div><table border=0 class="s1"><tr>
<td valign=top><span ID="mtrx"> </span></td>
<td align=center valign=top><span ID="res"> </span></td>
</tr></table></body>
```

Для правильного функционирования данного интерфейса необходимо написать функцию **fpoisk**, реализующую алгоритм Прима–Краскала построения минимального остовного дерева.

Возможное решение может иметь вид:

```
<script language="JavaScript">
var gl_n;

function fcrmat()
{var s="",n,i,j,v;
n=1*document.all.sel1.value;
gl_n=n;
s="<table class='s1'>";
s+="<tr align='center'><td> </td>";
for (i=0;i<n;i++)
s+="<td>"+(i+1)+"</td>";
s+="</tr>";
for (i=0;i<n;i++)
{s+="<tr><td>"+(i+1)+" </td>";
for (j=0;j<n;j++)
{v="b";
```



```
if (i==j) v="0";
s+="<td><input type='text' size='2' value='"+v+"'
name='m_"+i+"_"+j+"' class='s1'></td>";}
s+="</tr>";}
s+="</table>";
document.all.mtrx.innerHTML=s; document.all.res.innerHTML="";}

function fprim()
{var i,j,s;
with(document.all)
{if (sel3.selectedIndex==0) sel1.selectedIndex=2;
if (sel3.selectedIndex==1) sel1.selectedIndex=2;
fcrmat();
if (sel3.selectedIndex==0)
{m_0_1.value="9.5";  m_0_2.value="8.6"; m_0_3.value="10.8";
m_0_4.value="8.9";  m_0_5.value="7.1"; m_0_6.value="14";
m_1_2.value="14.6"; m_1_3.value="8.6"; m_1_4.value="17";
m_1_5.value="8.9";  m_1_6.value="6.4";  m_2_3.value="9.5";
m_2_4.value="15";   m_2_5.value="15.6"; m_2_6.value="20.6";
m_3_4.value="19.7"; m_3_5.value="15";   m_3_6.value="14.9";
m_4_5.value="9.5";  m_4_6.value="19.3"; m_5_6.value="9.8"; }
if (sel3.selectedIndex==1)
{m_0_1.value="7";  m_0_2.value="15"; m_0_3.value="12";
 m_0_5.value="10";m_1_2.value="13"; m_1_3.value="9";
 m_1_6.value="8";  m_2_3.value="7"; m_2_4.value="15";
 m_2_5.value="7"; m_3_4.value="9";  m_3_6.value="11";
m_4_5.value="10"; m_5_6.value="12";}}
fsym();}

function fpoisk()
{var n,i,j,i1,j1,k,l,m,dmin;
var temp;
n=gl_n;
w = new Array(n); // Весовая матрица
for(i=0;i<n;i++) w[i] = new Array(n);
```

```
p = new Array(n); // Перечень ребер дерева
for(i=0;i<n;i++) p[i] = new Array(2);
col = new Array(n); // Массив для хранения цвета вершин
for(i=0;i<n;i++) col[i] = i;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{w[i][j]=eval("document.all.m_"+i+"_"+j+".value");
if(w[i][j]=="b") w[i][j]=32000;
else
w[i][j]=1*w[i][j];}
// Алгоритм Прима–Краскала
l=0; k=0;
while (k<n-1)
{dmin=30000;
for(i1=0;i1<n;i1++)
for(j1=i1+1;j1<n;j1++)
if ((w[i1][j1]<dmin) && (col[i1]!=col[j1]))
{dmin=w[i1][j1]; i=i1; j=j1;}
l+=dmin; p[k][0]=i;p[k][1]=j; j1=col[j];
for (m=0;m<n;m++)
if (col[m]==j1) col[m]=col[i];
k++; }
s="<b>Результат</b><br><table border=0 class='s1'>";
for (m=0;m<n-1;m++)
s+="<tr><td>"+(p[m][0]+1)+" - " + (p[m][1]+1) + " :
"+w[p[m][0]][p[m][1]]+"</td></tr>";
s+="<tr><td align=right><hr>"+l+"</td></tr>";
s+="</table>"; document.all.res.innerHTML=s;}

function fsym()
{var i,j,n=gl_n;
for (i=1;i<n;i++)
for (j=0;j<i;j++)
eval("document.all.m_"+i+"_"+j+".value=document.all.m_"+j+"_"+i+".v
alue")}
```



```
</script>
```

С помощью текстового редактора Блокнот создать файл **prog6.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

6.4. Содержание отчета

Смотри пункт 1.4.

6.5. Контрольные вопросы [7, 11]

1. Как в программе **prog6.htm** задается вес ребра равным бесконечности?
2. Найдите минимальное остовное дерево графа, изображенного на рис. 6.2.

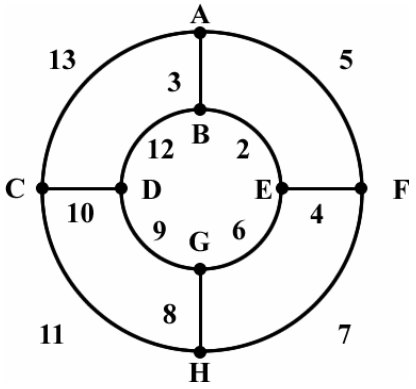


Рис 6.2

3. В табл. 6.1 приведены расстояния (в милях) между шестью городами Ирландии. Используя алгоритм поиска минимального остовного дерева, найдите сеть дорог минимальной общей длины, связывающую все шесть городов.

Таблица 6.1

	Атлон	Дублин	Голуэй	Лимерик	Слайго	Уэксфорд
Атлон	-	78	56	73	71	114
Дублин	78	-	132	121	135	96
Голуэй	56	132	-	64	85	154
Лимерик	73	121	64	-	114	116
Слайго	71	135	85	144	-	185
Уэксфорд	114	96	154	116	185	-

4. Докажите лемму об эстафете которая утверждает, что сумма степеней вершин произвольного графа G = (V, E) равна удвоенному числу его ребер.
5. Найти остов минимального веса взвешенного графа (рис. 6.3).

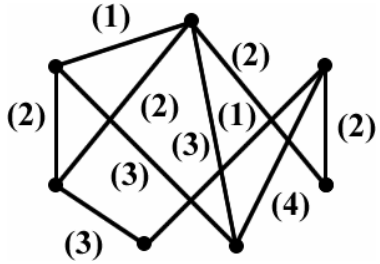


Рис 6.3

6. Семь команд сыграли по одному матчу между собой. Сколько всего матчей было сыграно?
7. Построить граф по матрице смежности $\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$.
8. В шахматном турнире по круговой системе участвуют семь школьников. Известно, что Ваня сыграл 6 партий, Толя – 5, Леша и Дима – по 3 партии, Семен и Илья – по 2, а Женя – 1. С кем сыграл Леша?
9. Для графа, заданного матрицей весов (рис. 6.4, 6.5), построить минимальный по весу остов и найти его вес.

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{array} \begin{array}{c} \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{array} \\ \begin{pmatrix} - & 5 & 11 & 14 & \infty & \infty & 8 \\ 5 & - & 5 & 7 & \infty & \infty & 8 \\ 11 & 5 & - & 4 & 8 & 6 & \infty \\ 14 & 7 & 4 & - & 7 & \infty & 11 \\ \infty & \infty & 8 & 7 & - & 3 & 5 \\ \infty & \infty & 6 & \infty & 3 & - & 6 \\ 8 & 8 & \infty & 11 & 5 & 6 & - \end{pmatrix} \end{array}$$

Рис. 6.4

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{array} \begin{array}{c} \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{array} \\ \begin{pmatrix} - & 10 & \infty & 5 & \infty & 6 & \infty \\ 10 & - & 6 & 1 & 4 & \infty & 5 \\ \infty & 6 & - & 3 & 1 & 2 & \infty \\ 5 & 1 & 3 & - & 3 & \infty & 5 \\ \infty & 4 & 1 & 3 & - & 4 & 2 \\ 6 & \infty & 2 & \infty & 4 & - & \infty \\ \infty & 5 & \infty & 5 & 2 & \infty & - \end{pmatrix} \end{array}$$

Рис. 6.5

7. ЛАБОРАТОРНАЯ РАБОТА № 7. НАХОЖДЕНИЕ КРАТЧАЙШИХ МАРШРУТОВ ПО АЛГОРИТМУ ФОРДА–БЕЛЛМАНА

7.1. Цель работы

Изучить способы моделирования алгоритмов нахождения кратчайших маршрутов на взвешенных графах и разработать компьютерную программу, реализующую алгоритм Форда–Беллмана.

7.2. Краткие теоретические сведения [2, 10, 11]

Пусть $G=(V;E)$ – взвешенный граф, имеющий n вершин и матрицу весов $W = (w_{ij})$, $w_{ij} \in R$. Задача состоит в нахождения взвешенного расстояния от фиксированной вершины $a_i \in V$ (называемой **источником**) до всех вершин графа G.

Будем предполагать, что в G отсутствуют контуры с отрицательным весом, поскольку, двигаясь по такому контуру достаточное количество раз, можно получить маршрут, имеющий вес, меньший любого заведомо взятого числа, и тем самым задача нахождения расстояния становится бессмысленной.

Определим **алгоритм Форда–Беллмана**. Зададим строку $D^{(1)}=(d_1^{(1)}, d_2^{(1)}, \ldots, d_n^{(1)})$, полагая $d_1^{(1)}=0$, $d_j^{(1)}= w_{ij}$, $i \neq j$. В этой строке $d_j^{(1)}$ $(i \neq j)$ есть вес w_{ij} дуги (a_i, a_j), если эта дуга существует, и $d_j^{(1)}=\infty$ в противном случае. Теперь определим строку $D^{(2)}=(d_1^{(2)}, d_2^{(2)}, \ldots, d_n^{(2)})$, полагая $d_j^{(2)}=min\{d_j^{(1)}, d_k^{(1)}+ w_{kj}\}$ для $k=1,\ldots, n$. Нетрудно заметить, что

$d_j^{(2)}$ – минимальный из весов (a_i,a_j) – маршрутов, состоящих не более чем из двух дуг. Продолжая процесс, на шаге s определим строку $D^{(s)}=(d_1^{(s)}, d_2^{(s)}, \ldots, d_n^{(s)})$, полагая $d_j^{(s)}=min\{d_j^{(s-1)},d_k^{(s-1)}+ w_{kj}\}$ для $k=1,\ldots, n$. Искомая строка w-расстояний получается при $s = n - 1$: $d_j^{(n-1)}=\rho_w(a_i,a_j)$. Действительно, на этом шаге из весов всех (a_i,a_j)-маршрутов, содержащих не более $n - 1$ дуг, выбирается наименьший, а каждый маршрут более чем с $n - 1$ дугами содержит контур, добавление которого к маршруту не уменьшает w – расстояния, так как мы предположили отсутствие контуров отрицательного веса. Работу алгоритма можно завершить на шаге k, если $D^{(k)}= D^{(k-1)}$.

Продемонстрируем работу алгоритма Форда–Беллмана на примере взвешенного графа с матрицей весов показанного на рис. 7.1.

$$W = \begin{pmatrix} 0 & 1 & \infty & \infty & 3 \\ \infty & 0 & 3 & 3 & 8 \\ \infty & \infty & 0 & 1 & -5 \\ \infty & \infty & 2 & 0 & \infty \\ \infty & \infty & \infty & 4 & 0 \end{pmatrix}$$

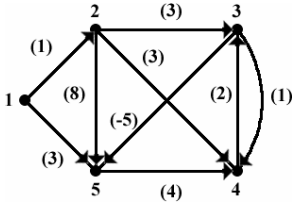


Рис. 7.1

В качестве источника выберем вершину 1. Тогда $D^{(1)}$=(0,1,∞,∞,3), $D^{(2)}$=(0,1,4,4,3), $D^{(3)}$=(0,1,4,4,-1), $D^{(4)}$=(0,1,4,3,-1). Таким образом, $\rho_w(1,1)=0$, $\rho_w(1,2)=1$, $\rho_w(1,3)=4$, $\rho_w(1,4)=3$, $\rho_w(1, 5)=-1$.

Отметим, что, зная расстояние от источника a_i до всех остальных вершин графа, можно найти и сами кратчайшие (a_i,a_j)-маршруты. Действительно, пусть a_i, b_1, b_2,..., b_r, a_j – кратчайший (a_i,a_j)-маршрут. Тогда по строке $D^{(n-1)}$ вершина $b_r= a_{k1}$ находится из соотношения $\rho_w(a_i,a_j)= \rho_w(a_i,a_{k1})+ w_{k1j}$, вершина $b_{r-1} = a_{k2}$ – из соотношения $\rho_w(a_i,a_{k1})= \rho_w(a_i,a_{k2})+ w_{k2k1}$ и т.д.

Для реализации алгоритма Форда–Беллмана требуется порядка $O(n^3)$ операций.

7.3. Порядок выполнения работы

Составить компьютерную программу для нахождения кратчайших маршрутов по алгоритму Форда–Беллмана. Программа должна позволять пользователю вводить произвольный граф выражение,

определять булевы переменные и динамически строить таблицу истинности.

Программу реализовать:

– интерфейсная часть на языке разметки гипертекста HTML [3];

– функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле матрицы смежности заданного графа;
- кнопки для активизации процесса нахождения кратчайших маршрутов от заданного источника;
- динамически отображаемую таблицу результата, включающую перечень ребер соответствующих ребер кратчайших маршрутов;
- динамически формировать пустую матрицу смежности разной размерности;
- встроенную проверку работоспособности алгоритма на тестовых примерах.

На рис. 7.2 приведен возможный вид интерфейса.

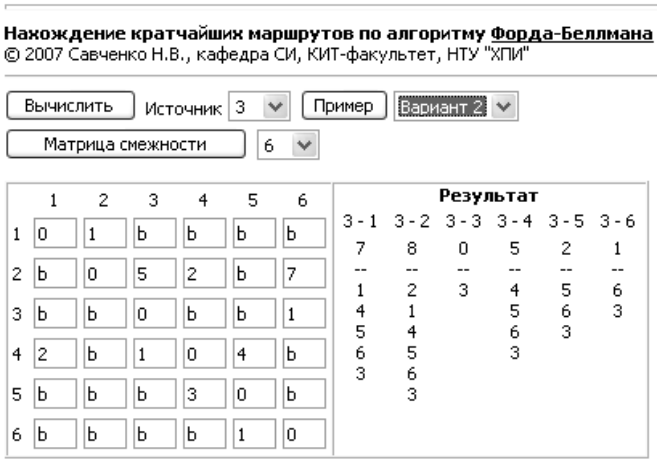


Рис. 7.2

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<body class="s1" onload="fprim();fpoisk();" TOPMARGIN=2>
<hr size=4><b>Нахождение кратчайших маршрутов по алгоритму
<u>Форда–Беллмана</u></b><br>&copy; 2007 Савченко Н.В.,
кафедра СИ, КИТ-факультет, НТУ "ХПИ"<hr>
<input type="button" value="Вычислить" onclick="fpoisk();"
class="s1">
 Источник <select name="sel2" class="s1"
onchange="fsel2();fpoisk()">
<option value="1">1</option><option value="2">2</option>
<option value="3">3</option><option value="4">4</option>
<option value="5">5</option><option value="6">6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option
value="10">10</option></select>
 <input type="button" value="Пример" onclick="fprim();fsel2()"
class="s1"><select name="sel3" class="s1">
<option value="1">Вариант 1</option>
<option value="2">Вариант 2</option>
</select><div style="margin-top:3pt;">
<input type="button" value="Матрица смежности" onclick="fcrmat()"
class="s1"> <select name="sel1" class="s1" onchange="fsel2();">
<option value="5">5</option><option value="6" selected>6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option value="10">10</option>
</select></div><br>
<table border=1 class="s1" cellpadding=0 cellspacing=0>
<tr><td valign=top><span ID="mtrx"> </span></td>
<td align=center valign=top><span
ID="res"> </span></td></tr>
</table></body>
```


Для правильного функционирования данного интерфейса необходимо написать функцию **fpoisk**, реализующую алгоритм Форда–Беллмана для нахождения кратчайших маршрутов.

Возможное решение может иметь вид:

```
<script language="JavaScript">
var gl_n;
function fcrmat()
{var s="",n,i,j,sb;
n=1*document.all.sel1.value;
gl_n=n;
s="<table class='s1'>";
s+="<tr align='center'><td> </td>";
for (i=0;i<n;i++)
s+="<td>"+(i+1)+"</td>";
s+="</tr>";
for (i=0;i<n;i++)
ë{s+="<tr><td>"+(i+1)+" </td>";
for (j=0;j<n;j++)
{sb="b";
if (i==j) sb="0";
s+="<td><input type='text' size='1' value='"+sb+"'
name='m_"+i+"_"+j+"' class='s1'></td>";}
s+="</tr>";}
s+="</table>";
document.all.mtrx.innerHTML=s;document.all.res.innerHTML="";}

function fprim()
{
with(document.all)
{if (sel3.selectedIndex==0) sel1.selectedIndex=0;
if (sel3.selectedIndex==1) sel1.selectedIndex=1;
fcrmat();
if (sel3.selectedIndex==0)
{m_0_0.value="0"; m_0_1.value="1";
m_0_4.value="3";m_1_1.value="0"; m_1_2.value="3";
```

```
m_1_3.value="3";m_1_4.value="8"; m_2_2.value="0"; m_2_3.value="1";
m_2_4.value="-5"; m_3_2.value="2"; m_3_3.value="0";
m_4_3.value="4"; m_4_4.value="0";
}
if (sel3.selectedIndex==1)
{m_0_0.value="0"; m_0_1.value="1"; m_1_1.value="0";
m_1_2.value="5"; m_1_3.value="2"; m_1_5.value="7";m_2_2.value="0";
m_2_5.value="1"; m_3_0.value="2";m_3_2.value="1"; m_3_3.value="0";
m_3_4.value="4";
m_4_3.value="3"; m_4_4.value="0";
m_5_4.value="1";m_5_5.value="0";}}

function fsel2()
{var s1,s2;
s1=1*document.all.sel1.selectedIndex+5;
s2=1*document.all.sel2.selectedIndex+1;
if (s2>s1) document.all.sel2.selectedIndex=0;}
function fpoisk()
{var n,i,j,k,dmin,t,m;
var temp;
n=gl_n;
k=1*document.all.sel2.selectedIndex;
w = new Array(n);
for(i=0;i<n;i++) w[i] = new Array(n);
d = new Array(n);
for(i=0;i<n;i++) d[i] = new Array(n);
p = new Array(n);
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{w[i][j]=eval("document.all.m_"+i+"_"+j+".value");
if(w[i][j]=="b") w[i][j]=32000;
else
w[i][j]=1*w[i][j];}
for (j=0;j<n;j++) d[1][j]=w[k][j];
d[1][k]=0;
```



```
for (m=2;m<n;m++)
{for (j=0;j<n;j++)
{dmin=d[m-1][j];
for (i=0;i<n;i++)
{t=d[m-1][i]+w[i][j]
if(w[i][j]==32000) t=32000;
if (t<dmin) dmin=t;}
d[m][j]=dmin;}}
for (i=0;i<n;i++) if (d[n-1][i]==32000) d[n-1][i]="b";
k=1*document.all.sel2.selectedIndex+1;
for (j=0;j<n;j++){
m=j+1;
s="<br>--<br>";
if (d[n-1][m-1]!="b")
{ s+=m;
while(m!=k)
{dm=d[n-1][m-1];
for (i=0;i<n;i++)
{ if ((w[i][m-1]!=0) && (w[i][m-1]!=32000))
if ((w[i][m-1]+d[n-1][i])==dm) {m=i+1; break;}}
s+=" <br> "+m;}}
p[j]=s;}
s="<b>Результат</b><br><table border=0 class='s1'><tr>";
for (i=0;i<n;i++) s+="<td align=center> "+k+" - "+(i+1)+ "
 </td>"
s+="</tr><tr>";
for (i=0;i<n;i++) s+="<td align='center' valign='top'>"+d[n-1][i]+p[i]+
"</td>"
s+="</tr></table>"
document.all.res.innerHTML=s;}
</script>
```

С помощью текстового редактора Блокнот создать файл **prog7.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на

контрольных примерах. **Функциональную блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

7.4. Содержание отчета

Смотри пункт 1.4.

7.5. Контрольные вопросы [7, 11]

1. Как в программе **prog7.htm** задается вес ребра равным бесконечности?
2. Для графа, заданного матрицей весов (рис. 7.3, 7.4), найти минимальный путь с начальной вершины с номером 1.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	–	4	∞	15	8	∞	∞
x_2	∞	–	5	∞	∞	∞	∞
x_3	∞	∞	–	∞	9	∞	7
x_4	∞	∞	4	–	10	−6	∞
x_5	∞	∞	∞	∞	–	7	16
x_6	∞	−18	7	∞	∞	–	6
x_7	∞	∞	∞	∞	∞	∞	–

Рис. 7.3

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	–	6	12	16	3	∞	∞
x_2	∞	–	5	∞	∞	13	∞
x_3	∞	∞	–	−5	∞	∞	9
x_4	∞	∞	∞	–	∞	∞	6
x_5	∞	−7	∞	∞	–	5	15
x_6	∞	∞	8	∞	∞	–	6
x_7	∞	∞	∞	∞	∞	∞	–

Рис. 7.4

3. Найти все кратчайшие маршруты из вершины 2 для взвешенного графа (рис. 7.5).

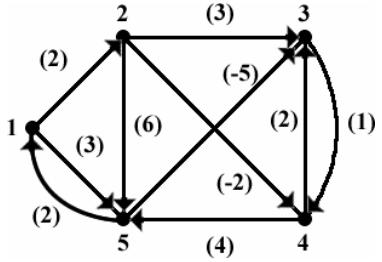


Рис 7.5

4. Проследите за работой алгоритма Форда–Беллмана на примере орграфа, изображенного на рис. 7.6, и найдите кратчайшие пути до каждой вершины:(а) от вершины A; (б) от вершины C.

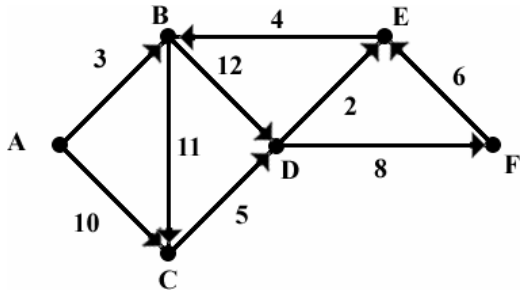


Рис. 7.6

5. С помощью алгоритма Форда–Беллмана найдите кратчайший путь от вершины S до всех остальных вершин в нагруженном графе из рис. 7.7. Найдите два кратчайших пути от S до T.

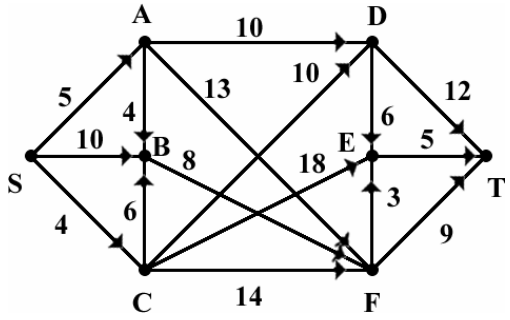


Рис. 7.7

6. Для графа, заданного матрицей весов (рис. 7.8, 7.9), найти минимальный путь с начальной вершины с номером 1.

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	-	7	∞	−8	∞	∞
x_2	∞	-	13	−9	10	∞
x_3	∞	∞	-	3	−4	−2
x_4	∞	∞	∞	-	9	∞
x_5	∞	∞	∞	∞	-	8
x_6	∞	∞	∞	∞	∞	-

Рис. 7.8

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	–	3	∞	∞	∞	7	∞
x_2	∞	–	∞	8	5	−10	∞
x_3	∞	4	–	3	6	∞	∞
x_4	∞	∞	∞	–	∞	∞	−5
x_5	∞	∞	∞	∞	–	6	13
x_6	∞	∞	7	6	∞	–	4
x_7	∞	∞	∞	∞	∞	∞	–

Рис. 7.9

8. ЛАБОРАТОРНАЯ РАБОТА № 8. НАХОЖДЕНИЕ КРАТЧАЙШИХ МАРШРУТОВ ПО АЛГОРИТМУ ДЕЙКСТРЫ

8.1. Цель работы

Изучить способы моделирования алгоритмов нахождения кратчайших маршрутов на взвешенных графах и разработать компьютерную программу, реализующую алгоритм Дейкстры.

8.2. Краткие теоретические сведения [2, 10, 11]

Алгоритм Дейкстры является более эффективным, чем алгоритм Форда–Беллмана, но используется только для взвешенных графов, в которых **веса всех дуг неотрицательны**. Пусть $G=(V;E)$ – взвешенный граф, имеющий n вершин и матрицу весов $W = (w_{ij})$, $w_{ij} \geq 0$; a_i – выделенный источник. Зададим строку $D^{(1)}=(d_1^{(1)}, d_2^{(1)}, ..., d_n^{(1)})$, полагая, как и в алгоритме Форда–Беллмана, $d_1^{(1)}=0$, $d_j^{(1)}= w_{ij}$, $i \neq j$. Обозначим через T_1 множество вершин $M\backslash\{a_i\}$. Предположим, что на шаге s уже определены строка $D^{(s)}=(d_1^{(s)}, d_2^{(s)}, ..., d_n^{(s)})$ и множество вершин T_s. Выберем теперь вершину $a_j \in T_s$ так, что $d_j^{(s)}=\min\{d_k^{(s)} | a_k \in T_s\}$. Положим $T_{s+1}= T_s\backslash\{a_j\}$, $D^{(s+1)}=(d_1^{(s+1)}, d_2^{(s+1)}, ..., d_n^{(s+1)})$, где $d_j^{(s+1)}=\min\{d_k^{(s)}, d_j^{(s)}+w_{jk}\}$, если $a_k \in T_{s+1}$, и $d_k^{(s+1)}= d_k^{(s)}$, если $a_k \notin T_{s+1}$. На шаге $s=n-1$ образуется строка $D^{(n-1)}$ w-расстояний между вершиной a_j и остальными вершинами графа: $d_j^{(n-1)}=\rho_w(a_i,a_j)$.

Для реализации алгоритма Дейкстры требуется порядка $O(n^2)$ операций. Продемонстрируем работу алгоритма Дейкстры на примере взвешенного графа с матрицей весов показанного на рис. 8.1.

$$W = \begin{pmatrix} 0 & 1 & \infty & \infty & \infty & \infty \\ \infty & 0 & 5 & 2 & \infty & 7 \\ \infty & \infty & 0 & \infty & \infty & 1 \\ 2 & \infty & 1 & 0 & 4 & \infty \\ \infty & \infty & \infty & 3 & 0 & \infty \\ \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

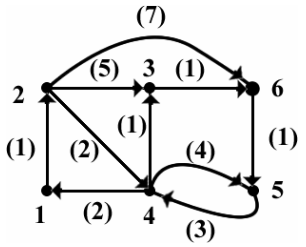


Рис. 8.1

Тогда по алгоритму Дейкстры $T_1 = \{2,3,4,5,6\}$, $D^{(1)}=(0,\underline{1},\infty,\infty,\infty,\infty)$, $T_2 = \{3,4,5,6\}$, $D^{(2)}=(0,1,6,\underline{3},\infty,\infty)$, $T_3 = \{3,5,6\}$, $D^{(3)}=(0,1,\underline{4},3,7,7)$, $T_4 = \{5,6\}$, $D^{(4)}=(0,1,\underline{4},3,7,\underline{5})$, $T_5 = \{5\}$, $D^{(5)}=(0,1,\underline{4},3,6,\underline{5})$ (в $D^{(s)}$ подчеркнута величина $d_j^{(s)}$, для которой $T_{s+1}= T_s\backslash\{a_j\}$). Таким образом, $\rho_w(1,1) = 0$, $\rho_w(1,2) = 1$, $\rho_w(1,3) = 4$, $\rho_w(1,4) = 3$, $\rho_w(1,5) = 6$, $\rho_w(1,6) =$ 5. D.

8.3. Порядок выполнения работы

Составить компьютерную программу для нахождения кратчайших маршрутов по алгоритму Дейкстры. Программа должна позволять пользователю вводить граф разной размерности, динамически находить кратчайшие маршруты для заданного источника.

Программу реализовать:

– интерфейсная часть на языке разметки гипертекста HTML [3];

– функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле матрицы смежности заданного графа;
- кнопки для активизации процесса нахождения кратчайших маршрутов от заданного источника;
- динамически отображаемую таблицу результата, включающую перечень ребер соответствующих ребер кратчайших маршрутов;
- динамически формировать пустую матрицу смежности разной размерности;
- встроенную проверку работоспособности алгоритма на тестовых примерах.

На рис. 8.2 приведен возможный вид интерфейса.

Рис. 8.2

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<body class="s1" onload="fprim();fpoisk();" TOPMARGIN=2>
<hr size=4><b>Нахождение кратчайших маршрутов по алгоритму
<u>Дейкстры</u></b>
<br>&copy; 2007 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ
"ХПИ"<hr>
<input type="button" value="Вычислить" onclick="fpoisk();"
class="s1">
 Источник <select name="sel2" class="s1"
onchange="fsel2();fpoisk()">
<option value="1">1</option><option value="2">2</option>
<option value="3">3</option><option value="4">4</option>
<option value="5">5</option><option value="6">6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option value="10">10</option>
</select>
 <input type="button" value="Пример" onclick="fprim();fsel2()"
class="s1">
<select name="sel3" class="s1">
<option value="1">Вариант 1</option>
<option value="2" selected>Вариант 2</option></select>
```



```
<div style="margin-top:3pt;">
<input type="button" value="Матрица смежности" onclick="fcrmat()"
class="s1">
 <select name="sel1" class="s1" onchange="fsel2();">
<option value="5">5</option><option value="6" selected>6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option value="10">10</option>
</select></div><table border=0 class="s1">
<tr><td valign=top><span ID="mtrx"> </span></td>
<td align=center valign=top><span ID="res"> </span></td>
</tr></table></body>
```

Для правильного функционирования данного интерфейса необходимо написать функцию **fpoisk**, реализующую алгоритм Дейкстры для нахождения кратчайших маршрутов.

Возможное решение может иметь вид:

```
<script language="JavaScript">
var gl_n;
function fcrmat()
{var s="",n,i,j;
n=1*document.all.sel1.value;
gl_n=n;
s="<table class='s1'>";
s+="<tr align='center'><td> </td>";
for (i=0;i<n;i++)
s+="<td>"+(i+1)+"</td>";
s+="</tr>";
for (i=0;i<n;i++)
{s+="<tr><td>"+(i+1)+" </td>";
for (j=0;j<n;j++)
s+="<td><input type='text' size='2' value='b' name='m_"+i+"_"+j+"'
class='s1'></td>";
s+="</tr>";}
s+="</table>";
document.all.mtrx.innerHTML=s;
document.all.res.innerHTML="";}
```

```
function fprim()
{with(document.all)
{if (sel3.selectedIndex==0) sel1.selectedIndex=0;
if (sel3.selectedIndex==1) sel1.selectedIndex=1;
fcrmat();
if (sel3.selectedIndex==0)
{m_0_0.value="0"; m_0_1.value="1"; m_0_4.value="3";
m_1_1.value="0";
m_1_2.value="3"; m_1_3.value="3"; m_1_4.value="8"; m_2_2.value="0";
m_2_3.value="1"; m_2_4.value="-5"; m_3_2.value="2";
m_3_3.value="0";
m_4_3.value="4"; m_4_4.value="0";}
if (sel3.selectedIndex==1)
{m_0_0.value="0"; m_0_1.value="1"; m_1_1.value="0";
m_1_2.value="5";
m_1_3.value="2"; m_1_5.value="7"; m_2_2.value="0"; m_2_5.value="1";
m_3_0.value="2"; m_3_2.value="1"; m_3_3.value="0"; m_3_4.value="4";
m_4_3.value="3"; m_4_4.value="0"; m_5_4.value="1";
m_5_5.value="0";}}}

function fsel2()
{var s1,s2;
s1=1*document.all.sel1.selectedIndex+5;
s2=1*document.all.sel2.selectedIndex+1;
if (s2>s1) document.all.sel2.selectedIndex=0;}

function fpoisk()
{var n,i,j,k,dmin,t,m,zi;
var temp;
n=gl_n;
k=1*document.all.sel2.selectedIndex;

w = new Array(n);
for(i=0;i<n;i++) w[i] = new Array(n);
```



```
d = new Array(n);
for(i=0;i<n;i++) d[i] = new Array(n);
z = new Array(n);
for(i=0;i<n;i++) z[i] = 0;
p = new Array(n);
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{w[i][j]=eval("document.all.m_"+i+"_"+j+".value");
if(w[i][j]=="b") w[i][j]=32000;
else
w[i][j]=1*w[i][j];}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{wij=1*w[i][j];
if (wij<0) {alert("Матрица весов не должна содержать
отрицательных значений!"); return;}}
for (j=0;j<n;j++) d[1][j]=w[k][j];
d[1][k]=0; z[k]=1;
for (m=2;m<n;m++)
{for (j=0;j<n;j++)
{if (z[j]==0)
{dmin=64000;
for (i=0;i<n;i++)
{
if (z[i]==0)
{
t=d[m-1][i]+w[i][j]
if(w[i][j]==32000) t=32000;
if (t<dmin) dmin=t;}}
d[m][j]=dmin;}
else
d[m][j]=d[m-1][j];}
dmin=64000;
for (i=0;i<n;i++)
if (z[i]==0)
```

```
{if (d[m][i]<dmin) {dmin=d[m][i];zi=i;}}
z[zi]=1;}
for (i=0;i<n;i++) if (d[n-1][i]==32000) d[n-1][i]="b";
k=1*document.all.sel2.selectedIndex+1;
for (j=0;j<n;j++)
{m=j+1;
s="<br>--<br>";
if (d[n-1][m-1]!="b")
{ s+=m;
while(m!=k)
{dm=d[n-1][m-1];
for (i=0;i<n;i++){
if ((w[i][m-1]!=0) && (w[i][m-1]!=32000))
if ((w[i][m-1]+d[n-1][i])==dm) {m=i+1; break;}}
s+=" <br> "+m;}}
p[j]=s;}
s="<b>Результат</b><br><table border=0 class='s1'><tr>";
for (i=0;i<n;i++) s+="<td align=center> "+k+" -
"+(i+1)+" </td>"
s+="</tr><tr>";
for (i=0;i<n;i++) s+="<td align='center' valign='top'>"+d[n-
1][i]+p[i]+"</td>"
s+="</tr></table>"
document.all.res.innerHTML=s;}
```

</script>

С помощью текстового редактора Блокнот создать файл **prog8.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

8.4. Содержание отчета

Смотри пункт 1.4.

8.5. Контрольные вопросы [7, 11]

1. Как в программе **prog8.htm** задается вес ребра равным бесконечности?
2. Для графа, заданного матрицей весов (рис. 8.3, 8.4), найти минимальный путь с начальной вершины с номером 1.

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	-	5	10	13	∞	∞
x_2	∞	-	8	9	13	∞
x_3	∞	∞	-	5	3	6
x_4	∞	∞	∞	-	8	10
x_5	∞	∞	∞	∞	-	9
x_6	∞	∞	∞	∞	∞	-

Рис. 8.3

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	-	11	∞	14	15	∞
x_2	∞	-	13	∞	∞	∞
x_3	∞	∞	-	∞	∞	13
x_4	∞	7	11	-	9	∞
x_5	∞	11	10	∞	-	14
x_6	∞	∞	∞	∞	∞	-

Рис. 8.4

3. С помощью алгоритма Дейкстры найдите кратчайший путь от вершины S до всех остальных вершин в нагруженном графе из рис. 8.5. Найдите два кратчайших пути от x0 до z.

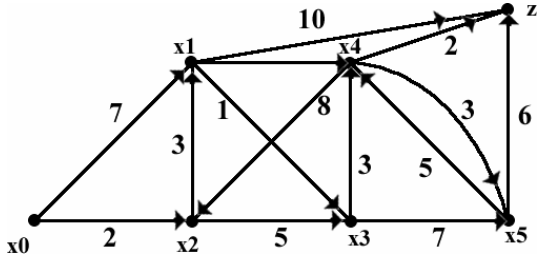


Рис. 8.5

Расчетно-графические задания

9. ВАРИАНТ № 1.
ВЫЧИСЛЕНИЕ ЧИСЛА СОЧЕТАНИЙ

9.1. Цель работы

Изучить особенности использования рекурсии в программировании и разработать компьютерную программу для генерации числа сочетаний.

9.2. Краткие теоретические сведения [2, 10, 11]

Рекурсивным называется способ построения объекта (понятия, системы, описание действия), в котором определение объекта включает аналогичный объект (понятие, систему, действие) в виде некоторой его части. Общеизвестный пример рекурсивного изображения - предмет между двумя зеркалами: в каждом из них виден бесконечный ряд отражений. В программировании можно привести следующие примеры использования этого понятия, а именно:

- рекурсивное определение в синтаксисе языка;
- рекурсивная структура данных – элемент структуры данных содержит один или несколько указателей на аналогичную структуру данных;
- рекурсивная функция – тело функции содержит прямой или косвенный (через другую функцию) собственный вызов.

Рекурсия не может быть безусловной, в этом случае она становится бесконечной. Рекурсия должна иметь внутри себя условие завершения, по которому очередной шаг ее уже не производится.

Особенности работы рекурсивной функции заключается в следующем. При обращении к рекурсивной функции копируется в память не весь текст функции, а только ее части, связанные с локальными данными (формальные, фактические параметры, локальные переменные и точка возврата). Алгоритмическая часть (операторы, выражения) рекурсивной функции и глобальные

переменные не меняются, поэтому они присутствуют в памяти компьютера в единственном экземпляре.

Каждый рекурсивный вызов порождает новый «экземпляр» формальных параметров и локальных переменных, причем старый «экземпляр» не уничтожается, а сохраняется в стеке по принципу вложенности. Здесь имеет место единственный случай, когда одному имени переменной в процессе работы программы соответствует несколько ее экземпляров. Происходит это в такой последовательности:

- в стеке резервируется место для формальных параметров, в которые записываются значения фактических параметров. Обычно это производится в порядке, обратном их следованию в списке;
- при вызове функции в стек записывается точка возврата – адрес той части программы, где находится вызов функции;
- в начале тела функции в стеке резервируется место для локальных (автоматических) переменных.

Рекурсивный алгоритм должен разрабатываться, не выходя за рамки текущего рекурсивного вызова. Принципы написания таких функций могут быть сформулированы следующим образом:

- рекурсивная функция разрабатывается как обобщенный шаг процесса, который вызывается в произвольных начальных условиях и приводит к следующему шагу в некоторых новых условиях;
- для шага процесса – рекурсивного вызова, необходимо определить инварианты – сохраняемые в процессе выполнения алгоритма условия и соотношения;
- начальные условия очередного шага должны быть формальными параметрами функции;
- начальные условия следующего шага должны быть сформированы в виде фактических параметров рекурсивного вызова;
- локальными переменными функции должны быть объявлены все переменные, которые имеют отношение к протеканию текущего шага процесса и к его состоянию;

- в рекурсивной функции обязательна проверка условий завершения рекурсии, при которых следующий шаг процесса не выполняется.

Размещением из n элементов по m или упорядоченной (n,m)-выборкой, называется любой кортеж $(a_{i1}, a_{i2}, \ldots, a_{im})$, состоящий из m попарно различных элементов множества M. **Число размещений** из n по m обозначается через A_n^m или $P(n,m)$ и равно

$$A_n^m = \frac{n!}{(n-m)!} = n(n-1)...(n-m+1) .$$

(Предполагается, что 0!=1). **Сочетанием** из из n элементов по m или неупорядоченной (n,m)–выборкой, называется любое подмножество множества M, состоящее из m элементов. **Число сочетаний** из n элементов обозначается через C_n^m или $C(n,m)$ и равно $C_n^m = \frac{n!}{(n-m)!m!}$. Очевидно, что $A_n^m = m! \cdot C_n^m$. Число C_n^m обладает следующими свойствами:

- $C_n^m = C_n^{n-m}$;
- $C_n^m + C_n^{m+1} = C_{n+1}^{m+1}$;
- $(a+b)^n = \sum_{m=0}^{n} C_n^m a^m b^{n-m}$ для любых $a,b \in \mathrm{R}$, $n \in \mathrm{N}$ (бином Ньютона).

В силу последнего свойства числа C_n^m называются биноминальными коэффициентами.

1.3. Порядок выполнения работы

Составить компьютерную программу для вычисления числа сочетаний при заданных значениях параметров n и k. Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- символ для обозначения числа сочетаний из n по k;
- выпадающих меню для выбора значений параметров в заданном диапазоне;
- кнопок для быстрого изменения текущих значений параметров;
- динамического поля вывода результата расчета.

На рис. 9.1 приведен возможный вид интерфейса.

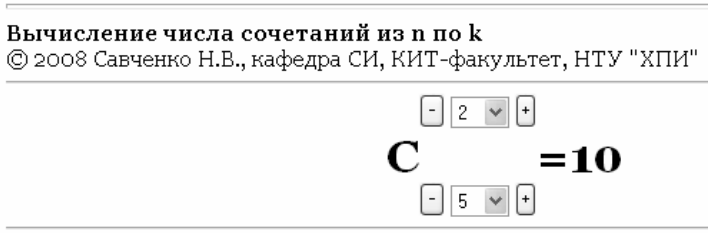


Рис. 9.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<div style="font-size:12pt;font-family:Georgia">
<hr size=4><b>Вычисление числа сочетаний из n по k</b>
<br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ
"ХПИ"<hr>
<table border="0" cellspacing="0" cellpadding="0" align=center>
<tr><td> </td><td>
<input type="button" name="but1" value="-" onclick="if
(document.all.sel1.selectedIndex>0) document.all.sel1.selectedIndex--;f()">
<select name="sel1" onchange="f()">
<option>0</option><option>1</option><option selected>2</option>
<option>3</option><option>4</option>
<option>5</option><option>6</option>
<option>7</option><option>8</option>
<option>9</option><option>10</option>
<option>11</option><option>12</option></select>
```

```
<input type="button" name="but2" value="+" onclick="if
(document.all.sel1.selectedIndex<12)
document.all.sel1.selectedIndex++;f()">
</td><td> </td>
</tr><tr style="font-size:24pt;font-weight:bold;font-family:Georgia">
<td align="right">C</td><td> </td>
<td width=200>=<span ID="cnk"></span></td>
</tr><tr><td> </td><td>
<input type="button" name="but3" value="-" onclick="if
(document.all.sel2.selectedIndex>0) document.all.sel2.selectedIndex--;f()">
<select name="sel2" onchange="f()">
<option>0</option><option>1</option>
<option>2</option><option>3</option>
<option>4</option><option selected>5</option>
<option>6</option><option>7</option>
<option>8</option><option>9</option>
<option>10</option><option>11</option><option>12</option></select>
<input type="button" name="but4" value="+" onclick="if
(document.all.sel2.selectedIndex<12)
document.all.sel2.selectedIndex++;f()">
</td><td> </td></tr></table></div><hr>
```

Для правильного функционирования данного интерфейса необходимо написать функцию ***c(n,k)***, реализующую рекурсивный алгоритм построения числа сочетаний [12].

Возможное решение может иметь вид:

```
<script language="JavaScript">
f();
function f(){ // Анализ входных параметров
var nt,kt;
nt=1*document.all.sel2.selectedIndex;
kt=1*document.all.sel1.selectedIndex;
if (kt>nt) {document.all.sel2.selectedIndex=kt; nt=kt;}
document.all.cnk.innerHTML=c(nt,kt);}

function c(n,k){ // Рекурсивная функция для вычисления числа
сочетаний
if (n==k || k==0) return 1;
else return c(n-1,k)+c(n-1,k-1);}
```



```
</script>
```

С помощью текстового редактора Блокнот создать файл **rgz1.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Блок-схему в виде структурные карт Константайна** (об этом типе блок-схем подробно рассказано в книге [8]) программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

9.4. Содержание отчета

Печатный отчет должен соответствовать требованиям по оформлению документов такого типа и содержать следующую информацию:

10. Титульный лист.
11. Название расчетно-графического задания.
12. Тема расчетно-графического задания.
13. Краткое описание метода выполнения.
14. Блок-схема алгоритма решения поставленной задачи.
15. Краткое объяснение интерфейса программы.
16. Листинг кода программы.
17. Результаты контрольных расчетов.
18. Краткие выводы.

9.5. Контрольные вопросы

1. Какие необходимо внести изменения в программу, чтобы она вычисляла число размещений из n по k. Как выглядит рекурсивная формула для вычисления числа размещений из n по k?

Из 20 студентов надо назначить 5 дежурных. Сколькими способа ми это можно сделать? Ответ: 15504. Какие необходимо внести изменения в программу чтобы решить эту задачу?

2. Скольким и способами можно составить бригаду из четырёх плотников, если имеются предложения от 10 человек? Ответ: 210.

3. Скольким и способами можно расселить 9 студентов в комнаты, каждая из которых рассчитана на трёх человек? Ответ: 81.
4. Скольким и способами пять девушек и трое юношей могут разбиться на две команды по четыре человека в команде, если в каждой команде должно быть хотя бы по одному юноше? Ответ: 30.
5. Найти натуральное число n, удовлетворяющее уравнению

$C_n^5 = 2C_{n-1}^5$.

6. Реализуйте на языке JavaScript алгоритм генерации всех сочетаний в лексикографическом порядке. Начальное сочетание равно 1, 2, ..., k, последнее – N-k+1, N-k+2, ..., N. Переход от текущего сочетания к другому осуществляется по следующему алгоритму. Просматриваем сочетание справа налево и находим первый элемент, который можно увеличить. Увеличиваем этот элемент на единицу, а часть сочетания (от этого элемента до конца) формируем из чисел натурального ряда, следующих за ним. На языке Паскаль это может выглядеть следующим образом:

```
Procedure GetNext;{* Предполагается, что текущее
сочетание  (хранится в массиве C) не является последним. *}
Var i,j:Integer;
Begin i:=k;
While  (C[i]+k-i+1>N) Do Dec(i);
{*Находим элемент, который можно увеличить. *}
Inc (C[i]); {* Увеличиваем на единицу. *}
For j:=i+1 To k Do C[j]:=C[j-1]+1;
{* Изменяем стоящие справа элементы. *}
End;
```

7. Реализуйте на языке JavaScript алгоритм генерации перестановок с повторениями.

10. ВАРИАНТ № 2. КОЛИЧЕСТВО РАЗЛИЧНЫХ ЦЕЛОЧИСЛЕННЫХ РЕШЕНИЙ УРАВНЕНИЯ

10.1. Цель работы

Изучить понятие сочетания с повторением и разработать компьютерную программу для нахождения количества различных целочисленных решений уравнений определенного типа.

10.2. Краткие теоретические сведения [5, 13, 14]

Предположим, что n объектов выбираются из k типов объектов с неограниченным повторением. Пусть a_i – объект типа i, тогда $n_1a_1+n_2a_2+...+n_ka_k$, где $n_i \geq 0$ для всех i, а $n_1+n_2+...+n_k=n$ представляет выбор n_i объектов типа i. Это можно записать в виде

$a_1a_1a_1...a_1|a_2a_2a_2a_2...a_2|...|a_ka_ka_ka_k...a_k$,

где каждое a_i повторено n_i раз. Поскольку место расположения каждого типа понятно, то выборку можно записать в виде

$xxx...x|xxxx...x|...|xxxx...x$.

Заметим, что число разделителей | на один меньше количества типов. Таким образом, имеем n объектов плюс k-1 разделителей, образующих $n+k-1$ мест для размещения x или |. Каждое расположение знаков x и | дает новый способ выбора n объектов из k типов объектов с неограниченным повторением. Поскольку существует $C_{n+k-1}^{n} = C_{n+k-1}^{k-1}$ способов выбора места для знака x или, что эквивалентно, для знака |, то следовательно эта величина определяет количество различных способов выбора n из k типов объектов с неограниченным повторением. Такие выборки называются **сочетаниями из k объектов по n с повторением** и их количество вычисляется по формуле $\widetilde{C}(n,k) = C_{n+k-1}^{k} = \frac{(n+k-1)!}{k!(n-1)!}$.

Рассмотрим следующую задачу. Сколько решений имеет уравнение

$n_1 + n_2 + n_3 + n_4 + n_5 = 25$,

где каждое n_i – неотрицательное число? Это эквивалентно вопросу о том, сколько существует различных выборок вида

$n_1 a_1 + n_2 a_2 + n_3 a_3 + n_4 a_4 + n_5 a_5$,

где имеется n_i объектов типа a_i и $n_1 + n_2 + n_3 + n_4 + n_5 = 25$, но количество таких выборок – это количество различных сочетаний из 5 элементов по 25 с повторениями. Следовательно существует $\frac{29!}{25!4!} = 23751$ различных решений искомого уравнения.

10.3. Порядок выполнения работы

Составить компьютерную программу для нахождения количества целочисленных решений уравнения вида $n_1 + n_2 + n_3 + n_4 = N$, где каждое n_i – неотрицательное число.

Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- выпадающие меню для изменения значения параметра N и граничных условий, накладываемых на переменные n_i;
- индикатор числа решений, получаемый на основе комбинаторного подсчета;
- место для записи всех корней заданного уравнения, получаемых путем простого перебора возможных значений переменных.

На рис. 10.1 приведен возможный вид интерфейса.

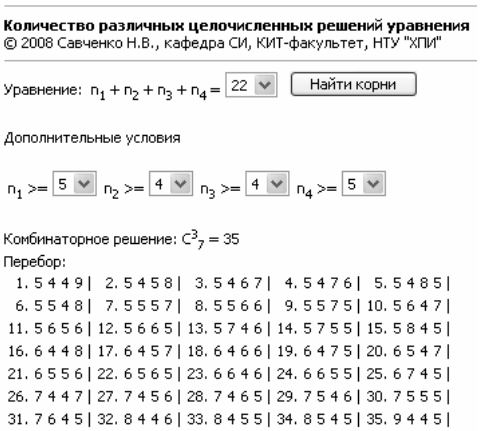


Рис. 10.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<body TOPMARGIN=0>
<div class="s1">
<hr size=0><b>Количество различных целочисленных решений
уравнения</b><br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-
факультет, НТУ "ХПИ"<hr size=0>Уравнение: 
n<sub>1</sub> + n<sub>2</sub> + n<sub>3</sub> + n<sub>4</sub> =
<select name=sel0 class=s1 onchange="f()">
<option value=21>21</option><option value=22>22</option>
<option value=23 selected>23</option>
<option value=24>24</option><option value=25>25</option>
</select>
  <input type=button value="Найти корни" onclick="f()"
class=s1>

<p>Дополнительные условия
<p> n<sub>1</sub> >=
<select name=sel1 class=s1 onchange="f()">
<option>0</option><option>1</option>
<option selected>2</option>
<option>3</option><option>4</option>
<option>5</option><option>6</option>
</select>
```

```
 n<sub>2</sub> >=
<select name=sel2 class=s1 onchange="f()">
<option>0</option><option>1</option><option>2</option>
<option selected>3</option><option>4</option>
<option>5</option><option>6</option>
</select>

 n<sub>3</sub> >=
<select name=sel3 class=s1 onchange="f()">
<option>0</option><option>1</option>
<option>2</option><option>3</option>
<option selected>4</option onchange="f()">
<option>5</option><option>6</option>
</select>

 n<sub>4</sub> >=
<select name=sel4 class=s1 onchange="f()">
<option>0</option><option>1</option>
<option>2</option><option>3</option>
<option>4</option><option selected>5</option>
<option>6</option>
</select>

<p>Комбинаторное решение: <span ID="info1"></span>
<br>Перебор: <span ID="info2"></span></div>
```

Для правильного функционирования данного интерфейса необходимо написать функцию вычисляющую корни методом простого перебора и функция для вычисления числа сочетаний. Возможное решение может иметь вид:

```
<script language="JavaScript">
f();
function f(){ // Очищаем поля вывода и запускаем рабочую функцию
document.all.info1.innerHTML=""
document.all.info2.innerHTML=""
g()}

function g(){
// Поиск корней простым перебором и вычисление количества
```



```
// корней комбинаторным методом
var n0,n1,n2,n3,n4,i1,i2,i3,i4,j=0;

n0=1*document.all.sel0.options[document.all.sel0.selectedIndex].value;
n1=1*document.all.sel1.selectedIndex;
n2=1*document.all.sel2.selectedIndex;
n3=1*document.all.sel3.selectedIndex;
n4=1*document.all.sel4.selectedIndex;
sum="<table border=0 class=s1><tr align=right>";
for(i1=n1;i1<=n0;i1++)
for(i2=n2;i2<=n0-i1;i2++)
for(i3=n3;i3<=n0-i1-i2;i3++)
for(i4=n4;i4<=n0-i1-i2-i3;i4++)
{
s=i1+i2+i3+i4;
if(s==n0)
{
j++;
sum+="<td>"+j+".</td><td>"+i1+"</td><td>"+i2+"</td><td>"+i3+"</td><td>"+i
4+" |</td>";
if (j%5==0) sum+="</tr><tr align=right>"
}
}
sum+="</table>"

document.all.info2.innerHTML=sum
n=n0-n1-n2-n3-n4+3;
if (n<3)
document.all.info1.innerHTML="Решений нет";
else
document.all.info1.innerHTML+="C<sup>3</sup><sub>"+n+"</sub> =
"+c(n,3);
}
function c(n,k) { // Вычисление числа сочетаний
if (n==k || k==0) return 1;
else return c(n-1,k)+c(n-1,k-1);
}</script>
```

С помощью текстового редактора Блокнот создать файл **rgz2.htm** с рабочей версией программы. Отладку программы проводить в

браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную** б**лок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

10.4. Содержание отчета

Смотри пункт 9.4.

10.5. Контрольные вопросы

1. Сколько существует целочисленных решений уравнения $n_1 + n_2 + n_3 + n_4 = 21$ таких, что $n_1 \geq 2$, $n_2 \geq 3$, $n_3 \geq 4$, $n_4 \geq 5$?
2. Сколько существует целочисленных решений уравнения $n_1 + n_2 + n_3 + n_4 = 23$ таких, что $n_1 \geq 2$, $n_2 \geq 3$, $n_3 \geq 4$, $n_4 \geq 5$? Ответ: 220.
3. Реализуйте на языке JavaScript алгоритм генерации множества всех подмножеств. Алгоритм может быть описан следующим образом. Формируем массив, состоящий из N нулей, и рассматриваем его как пустое множество. Для получения следующего подмножества из текущего подмножества обрабатываем текущий массив из чисел 0 или 1 следующим образом: справа (от первого элемента массива к последнему) ищем первое число, равное 0. Если такое число не найдено, то текущее подмножество является последним, то есть множеством, состоящим из всех элементов. На этом алгоритм заканчивает свою работу. Если же элемент, равный 0, найден, то он заменяется на 1, а все числа справа от него (если таковые имеются) заменяются на нули. На языке Паскаль это может выглядеть следующим образом:

```
const alphabet : string[26] ='ABCDEFGHIJKLMNOPQRSTUVWXYZ';
Var b : array[1..100] of integer; n,i,j: integer;
begin
n:=4;
for i:=1 to n+1 do b[i]:=0;
writeln('''');
while (b[n+1]=0) do begin i:=1;
while b[i]=1 do begin b[i]:=0; inc(i); end;
```



```
b[i]:=1;
for i:=1 to n do if b[i]=1 then write(alphabet[i]);
writeln;
end; end.
```

4. Реализуйте на языке JavaScript алгоритм генерации множества всех сочетаний с повторениями. Основная идея заключается в следующем: сочетания записываем в виде $(N-1)$-го нуля и M единиц, где единицы заменяют символы, а нули – выступают в роли разделителей. При таком подходе для решения задачи достаточно сгенерировать все перестановки из M единиц и $(N-1)$-го нуля.

11. ВАРИАНТ № 3. МОДЕЛИРОВАНИЕ РАБОТЫ СО СТЕКОМ

11.1. Цель работы

Изучить принципы организации стека и на основе объектно-ориентированного подхода разработать компьютерную программу для иллюстрации основных операций с этим объектом.

11.2. Краткие теоретические сведения [5, 13, 14]

Структурой данных называется совокупность физически и логически взаимосвязанных переменных и их значений. **Последовательностью** называется упорядоченное множество переменных, количество которых может меняться. **Стеком** называется последовательность элементов, включение элементов в которую и исключение из которой производится только с одного конца. Начало последовательности называется **дном стека**, конец последовательности, в который добавляются элементы и из которых они исключаются, – **вершиной стека**. Операция добавления нового элемента (запись в стек) имеет общепринятое название ***Push*** (погрузить), операция исключения – ***Pop*** (звук выстрела). Операции *Push* и *Pop* безадресные: для их выполнения никакой дополнительной информации о месте размещения элементов не требуется.

На рис. 11.1 приведено графическое изображение стека.

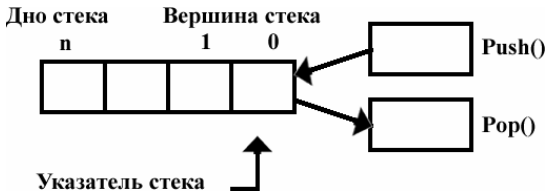


Рис. 11.1

Исключительная популярность стека в программировании объясняется тем, что при заданной последовательности записи элементов в стек (например, A–B–C) извлечение их происходит в обратном порядке (C–B–A). А именно эта последовательность действий соответствует таким понятиям, как вложенность вызовов функций, вложенность определений конструкций языка и т.д. Следовательно, везде, где речь идет о вложенности процессов, структур, определений, механизмом реализации такой вложенности является стек:

- при вызове функции адрес возврата (адрес следующей за вызовом команды) запоминается в стеке, таким образом создается "история" вызовов функций, которую можно восстановить в обратном порядке;
- при синтаксическом анализе вложенных друг в друга конструкций языка трансляторы используют магазинные (стековые) автоматы, стек при этом содержит не до конца проанализированные конструкции языка.

Для способа хранения данных в стеке имеется общепринятый термин – **LIFO** (last in – first out, "последний пришел – первый ушел").

Важное свойство стека – относительная адресация его элементов. На самом деле для элемента, сохраненного в стеке, важно не его абсолютное положение в последовательности, а положение относительно вершины стека или его указателя, которое отражает "историю" его заполнения. Поэтому адресация элементов стека происходит относительно текущего значения указателя стека.

В архитектуре практически всех компьютеров используется аппаратный стек. Он представляет собой обычную область внутренней (оперативной) памяти компьютера, с которой работает специальный регистр – указатель стека. С его помощью процессор выполняет

операции Push и Pop по сохранению и восстановлению из стека байтов и машинных слов различной размерности. Единственное отличие аппаратного стека от рассмотренной модели – это его расположение буквально "вверх дном", то есть его заполнение от старших адресов к младшим.

3.3. Порядок выполнения работы

Составить компьютерную программу для иллюстрации основных операций со стеком. Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле ввода данных для помещения в стек;
- поле вывода данных при извлечении их со стека;
- динамическое представление содержимого стека в виде линейного списка с общим входом–выходом.

На рис. 11.2 приведен возможный вид интерфейса.

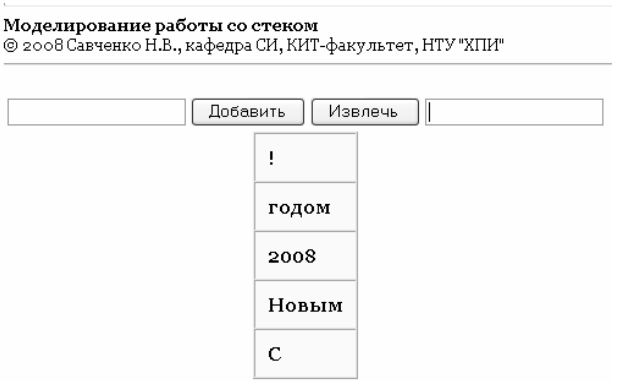


Рис. 11.2

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<div style="font-size:10pt;font-family:Georgia">
<hr size=4><b>Моделирование работы со стеком</b>
```

```
<br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ
"ХПИ"<hr><br><table><tr>
<td><input type="text" value="" name="txt1"></td>
<td><input type="button" value="Добавить" name="but1" onclick="if
(document.all.txt1.value!='') s.push(document.all.txt1.value);
document.all.txt1.value='';f_view();"> </td>
<td><input type="button" value="Извлечь " name="but2"
onclick="document.all.txt2.value=s.pop();f_view();"> </td>
<td><input type="text" value="" name="txt2"> </td>
</tr><tr><td> </td><td colspan=2><p><div ID="info"></div></td>
<td> </td></tr></table></div>
```

Для правильного функционирования данного интерфейса необходимо написать набор функций, позволяющих выполнить основне операции при работе со стеком и функцию для динамического отображения его содержимого. Возможное решение может иметь вид:

```
<script language="JavaScript">
function stack() // Конструктор стека
{this.store= new Object(); // Объект для хранения элементов в стеке
this.count=0; // Число элементов в стеке}

function push(value) // Добавляем элемент в стек
{this.count++;
this.store[this.count]=value;}
stack.prototype.push=push; // Определяем метод push для класса stack

function pop() // Извлекаем элемент из стека
{if(this.count>0) {
var value=this.store[this.count];
this.count--;
return value;}
else return '';}
stack.prototype.pop=pop;  // Определяем метод pop для класса stack

function getcount(value) // Число элементов в стеке
{return this.count;}
stack.prototype.getcount=getcount;
```



```
 // Определяем метод getcount для класса stack
var s= new stack(); // Объявляем стек

function f_view()  // Отображаем содержимое стека в виде таблицы
{var i,t;
t="<table border=1 align=center cellspacing='0' cellpadding='10'
bgcolor='ffffcc'>";
for (i=s.count;i>0;i--)
t+="<tr><td><b>"+s.store[i]+"</b></td></tr>";
t+="</table>";
document.all.info.innerHTML=t;}
</script>
```

Замечание. *Объекты позволяют сосредоточить в одном месте данные и действия, изолировав их от других объектов. В JavaScript класс объектов – это функция. Единственным отличием является то, что для доступа к свойствам и методам объекта используется элемент this. При создании экземпляра объекта исполняются все операторы тела конструктора объекта. Можно задать/изменить значение свойства вне конструктора объекта. Для того, чтобы сделать функцию методом объекта, достаточно присвоить ее имя свойству объекта. С помощью свойства prototype можно определить свойства и методы класса объектов. Свойства являются принадлежностью экземпляра объекта вне зависимости, определено ли оно как свойство класса или свойство экземпляра. Определять свойства и методы класса целесообразно в тех случаях, когда от всех экземпляров объекта требуется одинаковое поведение.*

С помощью текстового редактора Блокнот создать файл **rgz3.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную** б**лок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

11.4. Содержание отчета

Смотри пункт 9.4.

11.5. Контрольные вопросы

1. Допишите функцию, которая проверяет пуст ли стек или нет.
2. Разработайте метод top, который возвращает, но не удаляет, элемент в вершине стека.
3. Каким образом создаются объекты в программах на JavaScript?
4. Какое назначение метода *prototype* при работе с объектами в JavaScript?
5. Используя объекты, реализуйте на языке JavaScript работу с очередью.
6. Как создать линейный список в JavaScript?
7. Для хранения элементов стека используется обычный одномерный массив из N элементов. Реализуйте операции работы со стеком в этом случае. Как в этом случае проверять переполнение стека?

12. ВАРИАНТ № 4. ПРЕОБРАЗОВАНИЕ ПОСТФИКСНОГО ВЫРАЖЕНИЯ В ИНФИКСНОЕ

12.1. Цель работы

Изучить способы представления бинарных операций и разработать компьютерную программу для преобразования постфиксного арифметического выражения в инфиксное.

12.2. Краткие теоретические сведения [5, 13, 14]

В арифметике существуют определенные правила, касающиеся приоритета выполнения операций над целыми числами, что дает возможность, не приводя к путанице, сокращать число используемых в выражении скобок. Некоторые свойства целых чисел также позволяют уменьшить потребность в скобках.

Например, известно, что благодаря закону ассоциативности сложения выражение 3 + 5 + 7 не требует скобок, т.к. (3 + 5) + 7 = 3 + (5 + 7). Понятно также, что операции возведения в степень должны быть выполнены первыми, за ними следуют умножение и деление и, наконец, сложение и вычитание. Но даже при таком соглашении

необходимо использовать скобки. Исключить скобки можно, применив префиксную или суффиксную форму записи.

Выражение находится в **префиксной** (польской) записи, если в нем знак операции непосредственно предшествует операндам, на которые она действует. Выражение находится в **постфиксной** (обратной польской записи), если в нем знак операции следует непосредственно за операндами, на которые она действует. Выражение находится в **инфиксной** записи, если в нем знак операции находится между операндами, на которые она действует.

Например, инфиксное выражение $a + b$ в префиксной записи имело бы вид $+ab$, а в постфиксной записи, соответственно, вид $ab+$. Инфиксное выражение $a \times (b+c)$ имело бы в префиксной записи вид $\times a+bc$, а в постфиксной, соответственно, вид $abc+\times$.

Постфиксное выражение может быть заменено инфиксным посредством следующей процедуры:

1. Начинать считывание выражения слева направо.

2. Если считанный символ не является символом операции, то поместить его в стек и продолжать чтение.

3. Если считанный символ является символом операции, то:

а) вытолкнуть из стека два верхних элемента;

б) поместить символ операции между вторым и первым элементом из стека и поставить скобки;

в) поместить выражение, полученное на шаге (б), обратно в стек.

4. Продолжать чтение слева направо и выполнять шаги (2) и (3) до завершения.

12.3. Порядок выполнения работы

Составить компьютерную программу для преобразования постфиксного арифметического выражения в инфиксное. Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;

- поле ввода постфиксного арифметического выражения;
- поле вывода результата;
- протокола работы стека;
- набор тестовых заданий;
- кнопку для запуска процесса конвертации;
- кнопку для очистки рабочих полей и приведение программы в начальное состояние.

На рис. 12.1 приведен возможный вид интерфейса.

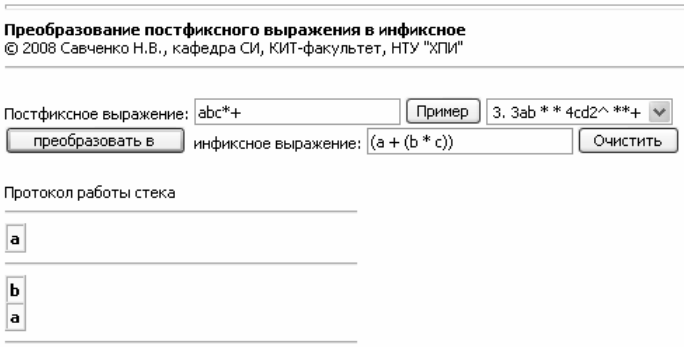


Рис. 12.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<div class="s1"><hr size=4><b>Преобразование постфиксного
выражения в инфиксное</b><br>&copy; 2008 Савченко Н.В., кафедра
СИ, КИТ-факультет, НТУ "ХПИ"<hr><br>Постфиксное выражение:
<input type="text" value="abc*+" size=25 name="txt1" class="s1"
onchange="fcl()"> <input type="button" value="Пример" name="but2"
onclick="f()" class="s1"> <select name="sel1" class="s1">
<option value="abc*+">1. abc*+</option> <option value="ab2^+cd2^+*">2.
ab2^+cd2^+*</option>
<option value="3ab * * 4cd2^ **+">3. 3ab * * 4cd2^ **+</option>
<option value="ab + 2^cd + :">4. ab + 2^cd + :</option>
<option value="a2^b + cd2^+*">5. a2^b + cd2^+*</option>
</select><br><input type="button" value="преобразовать в" name="but1"
onclick="g()" class="s1">  инфиксное выражение: <input type="text"
value="" size=25 name="txt2" class="s1"> <input type="button"
value="Очистить" name="but3" onclick="fcl()" class="s1">
```



```
<p><div ID="info"></div>
```

Для правильного функционирования данного интерфейса необходимо написать функцию ***g()***, позволяющую выполнить алгоритм преобразования постфиксного выражения в инфиксное. Реализация объекта стек взята из предыдущего задания.

Возможное решение может иметь вид:

```
<script language="JavaScript">

function fcl(){
document.all.info.innerHTML="Протокол работы стека";
document.all.txt2.value=""; // Очистить поле результата
s.count=0; // Обнулить стек
}
function f() // Выбор тестового примера
{fcl();
document.all.txt1.value=document.all.sel1.options[document.all.sel1.selected
Index].value;}

function stack() // Конструктор стека
{this.store= new Object(); // Объект для хранения элементов в стеке
this.count=0; // Число элементов в стеке}

function push(value) // Добавляем элемент в стек
{this.count++;
this.store[this.count]=value;}
stack.prototype.push=push; // Определяем метод push для класса stack

function pop() // Извлекаем элемент из стека
{if(this.count>0) {
var value=this.store[this.count];
this.count--;
return value;}
else return "";}
stack.prototype.pop=pop;  // Определяем метод pop для класса stack

function getcount(value) // Число элементов в стеке
{return this.count;}
```

```
stack.prototype.getcount=getcount;  // Определяем метод getcount для
класса stack

function f_view()  // Отображаем содержимое стека в виде таблицы
{var i,t;
t="<table border=1 cellspacing='0' cellpadding='2' bgcolor='ffffcc'
class='s1'>";
for (i=s.count;i>0;i--)
t+="<tr><td><b>"+s.store[i]+"</b></td></tr>";
t+="</table>"; document.all.info.innerHTML+="<hr width=30% align=left>"+t;}

var s= new stack(); // Объявляем стек
fcl();

function g() // Преобразование постфиксного выражения в инфиксное
{var i,n,ti,ts1,ts2,ts,t=document.all.txt1.value;
fcl();
n=t.length;
for (i=0;i<n;i++){
 ti=t.charAt(i);
 if (ti!=" ")
 { if (ti=="+" || ti=="*" || ti==":" || ti=="^" || ti=="-" || ti=="/")
 { ts1=s.pop();
 ts2=s.pop();
 if (ti=="^")
 ts =ts2+ti+ts1;
 else
 ts ="("+ts2 + " "+ti+" "+ts1+")";
 s.push(ts); }
 else
 { s.push(ti); } }
 f_view();}
document.all.txt2.value=s.pop();}
</script>
```

С помощью текстового редактора Блокнот создать файл **rgz4.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную** б**лок-схему** программы

нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

12.4. Содержание отчета

Смотри пункт 9.4.

12.5. Контрольные вопросы

1. Модернизируйте программу на предмет обработки ошибок в исходном выражении.
2. Замените следующее постфиксное выражение инфиксным:
   - *abc++3^;*
   - *ab+^2cd^+/.*
3. Какие необходимо произвести изменения в программе, чтобы она обрабатывала выражения алгебры логики?
4. Модернизируйте программу для преобразования префиксного выражения в инфиксное.
5. Разработайте программу вычисления значения арифметического выражения, записанного в постфиксной форме.

13. ВАРИАНТ № 5. ПРЕОБРАЗОВАНИЕ ИНФИКСНОГО ВЫРАЖЕНИЯ В ПОСТФИКСНОЕ

13.1. Цель работы

Изучить способы представления бинарных операций и разработать компьютерную программу для преобразования инфиксного арифметического выражения в постфиксное.

13.2. Краткие теоретические сведения [5, 13, 14]

Общие замечания о формах записи арифметических выражений смотри в предыдущем задании. Простейший алгоритм преобразования инфиксного выражения в постфиксное может быть описан следующим образом. Переход от инфиксного выражения к постфиксному может быть осуществлен посредством следующей процедуры:

1. Добавлять скобки в выражение, пока оно не станет находиться в полной скобочной записи.
2. Начиная с самых внутренних скобок, удалить пару скобок и поместить находящийся внутри скобок оператор на место соответствовавшей ему правой скобки. При наличии более одной самой внутренней пары скобок начинать нужно с левой пары.
3. Перейти к следующей паре скобок, удалить ее и поместить оператор, находившийся внутри скобок, на место соответствовавшей ему правой скобки.
4. Продолжать шаг (3), пока все скобки не будут удалены.

Арифметическое выражение находится в полной скобочной записи, если каждому оператору в нем соответствует пара скобок, в которые заключены оператор и те операнды, на которые он действует.

13.3. Порядок выполнения работы

Составить компьютерную программу для преобразования постфиксного арифметического выражения в инфиксное. Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле ввода инфиксного арифметического выражения;
- поле вывода результата;
- протокола работы стека;
- набор тестовых заданий;
- кнопку для запуска процесса конвертации;
- кнопку для очистки рабочих полей и приведение программы в начальное состояние.

На рис. 13.1 приведен возможный вид интерфейса.

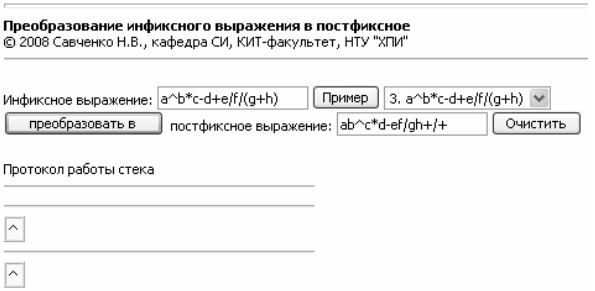


Рис. 13.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<div class="s1">
<hr size=4><b>Преобразование инфиксного выражения в постфиксное</b>
<br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ "ХПИ"<hr>
<br>Инфиксное выражение: <input type="text" value="a*b+c" size=20 name="txt1" class="s1" onchange="fcl()">
<input type="button" value="Пример" name="but2" onclick="f()" class="s1">
<select name="sel1" class="s1">
<option value="a*b+c">1. a*b+c</option>
<option value="(a+b)*(c+d)">2. (a+b)*(c+d)</option>
<option value="a^b*c-d+e/f/(g+h)">3. a^b*c-d+e/f/(g+h)</option>
</select>
<br><input type="button" value="преобразовать в" name="but1" onclick="g()" class="s1">
 постфиксное выражение: <input type="text" value="" size=20 name="txt2" class="s1">
<input type="button" value="Очистить" name="but3" onclick="fcl()" class="s1">
<p><div ID="info"></div></div>
```

Для правильного функционирования данного интерфейса необходимо написать функцию ***g()***, позволяющую выполнить алгоритм преобразования постфиксного выражения в инфиксное. Реализация объекта стек взята из предыдущего задания.

Возможное решение может иметь вид:

```
<script language="JavaScript">
function fcl()
{document.all.info.innerHTML="Протокол работы стека";
document.all.txt2.value=""; // Очистить поле результата
s.count=0; // Обнулить стек}

function f()
{fcl();
document.all.txt1.value=document.all.sel1.options[document.all.sel1.selected
Index].value;}

function stack() // Конструктор стека
{this.store= new Object(); // Объект для хранения элементов в стеке
this.count=0; // Число элементов в стеке}

function push(value) // Добавляем элемент в стек
{this.count++;
this.store[this.count]=value;}
stack.prototype.push=push; // Определяем метод push для класса stack

function pop() // Извлекаем элемент из стека
{if(this.count>0) {
var value=this.store[this.count];
this.count--;
return value;}
else return "";}
stack.prototype.pop=pop;  // Определяем метод pop для класса stack

function getcount(value) // Число элементов в стеке
{return this.count;}
stack.prototype.getcount=getcount;
// Определяем метод getcount для класса stack

function f_view()  // Отображаем содержимое стека в виде таблицы
{var i,t;
t="<table border=1 cellspacing='0' cellpadding='2' bgcolor='ffffcc'
class='s1'>";
for (i=s.count;i>0;i--)
t+="<tr><td><b>"+s.store[i]+"</b></td></tr>";
```



```
t+="</table>";
document.all.info.innerHTML+="<hr width=30% align=left>"+t;}
var s= new stack(); // Объявляем стек
fcl();
function prec(c)
{switch(c){case '^': return 4; case '*': return 3; case '/': return 3; case '-': return
2; case '+': return 2;case '(': return 1;}}
function g()
{var i,n,ti,t,z,sg,sg0; fcl();
sg="+-*/:^()"; sg0="+-*/:^"; t=document.all.txt1.value;
n=t.length;x="";
for (i=0;i<n;i++)
{ ti=t.charAt(i);
 if (ti!=" ") // Пропускаем пробелы
{ if (sg.indexOf(ti)<0) x+=ti;
 if (ti=="(") s.push(ti);  if (ti==")")
 { while (s.store[s.count]!="(")
 x+=s.pop();  ti=s.pop(); }
 if (sg0.indexOf(ti)>=0)
 { do  { exit=true;  if (s.count>0)
 if (prec(ti) <= prec(s.store[s.count]))
 { x+=s.pop();  exit=false; } }
 while (!exit);  s.push(ti); }} f_view();}
while (s.count>0) x+=s.pop(); document.all.txt2.value=x;}
</script>
```

С помощью текстового редактора Блокнот создать файл **rgz5.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную** б**лок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

13.4. Содержание отчета

Смотри пункт 9.4.

13.5. Контрольные вопросы

1. Модернизируйте программу на предмет обработки ошибок в исходном выражении.
2. Замените следующее инфиксное выражение постфиксным:
   - *(c+d) – (a+b)^2*;
   - *a – b^(c+d)*.
3. Какие необходимо произвести изменения в программе, чтобы она обрабатывала выражения алгебры логики?

14. ВАРИАНТ № 6.
ИССЛЕДОВАНИЕ СВОЙСТВ ГРУППОИДОВ

14.1. Цель работы

Составить компьютерную программу для исследования таблицы Кэли и изучить свойства конкретных группоидов.

14.2. Краткие теоретические сведения [5, 13, 14]

Всюду определённая (тотальная) функция $\varphi: M^n \to M$ называется **n-арной** (n-местной) **операцией** на M. Если операция φ – **бинарная** ($\varphi: M \times M \to M$), то будем писать $a\varphi b$ вместо $\varphi(a,b)$ или $a \otimes b$, где $\otimes$ – знак операции. Элементы a, b называются **операндами**, а знак операции – **оператором**. Такая форма записи называется инфиксной. Множество M вместе с набором операций Σ называется **алгебраической структурой**, **универсальной алгеброй** или просто **алгеброй**. Множество M называется **носителем**, вектор арностей – **типом**, а набор операций – **сигнатурой**. Кратко алгебру обозначают следующим образом <M; Σ>.

Некоторые часто встречающиеся свойства операций имеют специальные названия. Пусть задана алгебра <M; Σ> и $a,b,c \in M$; $\otimes,\oplus \in \Sigma$; $\otimes,\oplus: M \times M \to M$. Тогда:

- **ассоциативность** $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
- **коммутативность** $(a \otimes b) = (b \otimes a)$
- **дистрибутивность** $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
- **поглощение** $(a \otimes b) \oplus a = a$

- **идемпотентность** $a \otimes a = a$

Пример. Набор $\langle P(U); \cap, \cup, \neg, \varnothing, U \rangle$ с двухместными операциями $\cap$, $\cup$, одноместной – $\neg$, константами $\varnothing$=0 и U=1 является алгеброй Кантора. Двухэлементная булева алгебра определяется как <{0,1}; $\wedge$, $\vee$, $\neg$, 0, 1>.

Группоидом называется алгебра с одной бинарной операцией. Если группоид задан на конечном множестве, то его можно однозначно определить с помощью **таблицы Кэли (**или просто **таблицы умножения)**. Строки этой таблицы обозначают первый операнд, а столбцы – второй. Результат операции заносится в соответствующую ячейку таблицы.

Два группоида $F(\times)$ и $G(\cdot)$ называются **гомоморфизмом** между собой, если существуют взаимнооднозначная функция f ($F \to G$), такая что

$\forall a,b \in F\ f(a \times b) = f(a) \cdot f(b)$.

Гомоморфизм называется **мономорфизмом**, если функция f – инъекция, **эпиморфизмом**, если функция f – сюръекция, и **изоморфизмом**, если функция f – биекция.

Полугруппой называется группоид с ассоциативной операцией. Если любой элемент полугруппы можно получить путем умножения некоторого элемента на себя, то такая полугруппа называется **циклической**.

Моноидом называется полугруппа с **нейтральным элементом** e: $\exists e\ \forall a \in M\ a \otimes e = e \otimes a = a$.

Для мультипликативной операции нейтральный элемент называется **единицей** (**1**), а для аддитивной – **нулем** (0)**.** В любом моноиде нейтральный элемент единственный.

Группой называется моноид с симметричными элементами a*: $\forall a\ \exists a^* \in M\ a^* \otimes a = a \otimes a^* = e$.

Для мультипликативной операции симметричный элемент называется **обратным элементом** (**a^{-1}**), а для аддитивной – **противоположным элементом** (**-a**). В любой группе симметричный элемент единственный. Мощность носителя группы G называется **порядком группы** и обозначается $|G|$. Коммутативная группа называется **абелевой**.

Подгруппой группы G называется подмножество H множества G, которое есть группой относительно групповой операции исходной группы. Теорема Лагранжа утверждает, что порядок подгруппы конечной группы есть делителем порядка группы.

Взаимнооднозначная функция $f: M \to M$ называется **перестановкой** множества M. Перестановку удобно задавать таблицей из двух строк. В первой строке – значения аргументов, во второй – соответствующие значения функции. Такая таблица называется взаимнооднозначной **подстановкой**. **Произведением перестановок** называется их суперпозиция. Например, если $f = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 5\,2\,1\,4\,3 \end{pmatrix}$, $g = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 4\,1\,2\,3\,5 \end{pmatrix}$, то произведение перестановок $f \otimes g$ будет равно $f \otimes g = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 5\,1\,4\,3\,2 \end{pmatrix}$.

Таблицу **обратной подстановки** можно получить, если просто заменить местами строки таблицы исходной подстановки.

14.3. Порядок выполнения работы

Составить компьютерную программу для выполнения пересечения двух конечных множеств. Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле ввода исходной таблицы умножения с возможностью динамического изменения размерности;
- кнопок для активизации тестирования полноты исходной матрицы, проверки свойств умножения (ассоциативность и коммутативность), поиска нейтрального элемента, нахождения симметричных элементов;
- несколько вариантов тестовых начальных данных;

- поле для вывода результатов соответствующих проверок и отображения найденных элементов.

На рис. 14.1 приведен возможный вид интерфейса.

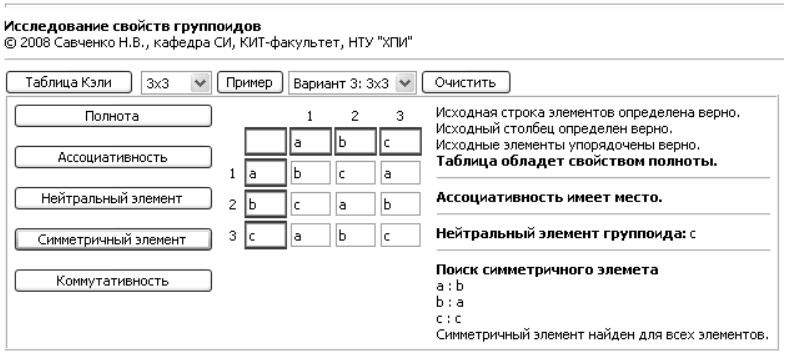


Рис. 14.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}
.s2 {font-family:Tahoma;font-size:8pt;width:120pt;}</style>
<body class="s1" onload="fprim();" TOPMARGIN=2>
<hr size=4><b>Исследование свойств группоидов</b>
<br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ
"ХПИ"<hr>
<input type="button" value="Таблица Кэли" onclick="fcrmat()" class="s1">
 <select name="sel1" class="s1">
<option value="4">3x3</option><option value="5">4x4</option>
<option value="6">5x5</option><option value="7">6x6</option>
<option value="8">7x7</option><option value="9">8x8</option>
<option value="10">9x9</option><option value="11">10x10</option>
</select>
<input type="button" value="Пример" onclick="fprim();" class="s1">
<select name="sel3" class="s1">
<option>Вариант 1: 6x6</option><option>Вариант 2: 3x3</option>
<option>Вариант 3: 3x3</option><option>Вариант 4: 5x5</option>
<option>Вариант 5: 3x3</option>
</select>
<input type="button" value="Очистить" onclick="fres0();" class="s1">
<br><table rules=rows cellspacing='0' cellpadding='5' class="s1">
<tr><td valign=top>
```

```
<input type="button" value="Полнота" onclick="f_test_sc()"
class="s2"><br><br>
<input type="button" value="Ассоциативность" onclick="f_test_as()"
class="s2"><br><br>
<input type="button" value="Нейтральный элемент" onclick="f_test_one()"
class="s2"><br><br>
<input type="button" value="Симметричный элемент"
onclick="f_test_sym()" class="s2">
<br><br><input type="button" value="Коммутативность"
onclick="f_test_com()" class="s2">
</td><td valign=top><span ID="mtrx"> </span></td>
<td valign=top><span ID="res1"> </span><span
ID="res2"> </span><span ID="res3"> </span><span
ID="res4"> </span><span ID="res5"> </span>
</td></tr></table> <span ID="prot"> </span> </body>
```

Для правильного функционирования данного интерфейса необходимо написать набор функций, позволяющих выполнить в полном объеме анализ свойств конкретной таблицы Кэли.

Возможное решение может иметь вид:

```
<script language="JavaScript">
var gl_n,w,flag1;

function fcrmat() {/// Формируем пустую таблицу
var s="",n,i,j,v;
n=1*document.all.sel1.value;
gl_n=n;
s="<table class='s1'>";
s+="<tr align='center'><td> </td>";
for (i=0;i<n;i++)
if (i==0)
s+="<td> </td>";
else
s+="<td>"+i+"</td>";
s+="</tr>";
for (i=0;i<n;i++) {
if (i==0) s+="<tr bgcolor=brown>";
else
s+="<tr>";
if (i==0) s+="<td bgcolor=white> </td>";
```



```
else
s+="<td>"+i+" </td>";
for (j=0;j<n;j++) {
v="";
if(j==0)
s+="<td bgcolor=brown>";
else
s+="<td>";
s+="<input type='text' size='2' value='"+v+"' name='m_"+i+"_"+j+"'
class='s1'></td>"; }
s+="</tr>"; }
s+="</table>";
document.all.mtrx.innerHTML=s;
fres0(); }

function fres0() {/// Очищаем поля результатов
document.all.res1.innerHTML=""; document.all.res2.innerHTML="";
document.all.res3.innerHTML=""; document.all.res4.innerHTML="";
document.all.res5.innerHTML="";}

function fprim() {/// Контрольные примеры
if (document.all.sel3.selectedIndex==0)
document.all.sel1.selectedIndex=3;
if (document.all.sel3.selectedIndex==1)
document.all.sel1.selectedIndex=0;
if (document.all.sel3.selectedIndex==2)
document.all.sel1.selectedIndex=0;
if (document.all.sel3.selectedIndex==3)
document.all.sel1.selectedIndex=2;
if (document.all.sel3.selectedIndex==4)
document.all.sel1.selectedIndex=0;
fcrmat();
if (document.all.sel3.selectedIndex==0) {
with (document.all) {
m_0_1.value="a"; m_0_2.value="b";
m_0_3.value="c"; m_0_4.value="d"; m_0_5.value="e"; m_0_6.value="f";
m_1_0.value="a"; m_1_1.value="e"; m_1_2.value="f";
m_1_3.value="a"; m_1_4.value="b"; m_1_5.value="c"; m_1_6.value="d";
m_2_0.value="b"; m_2_1.value="d"; m_2_2.value="c";
```

```
m_2_3.value="b"; m_2_4.value="a"; m_2_5.value="f"; m_2_6.value="e";
m_3_0.value="c"; m_3_1.value="a"; m_3_2.value="b";
m_3_3.value="c"; m_3_4.value="d"; m_3_5.value="e"; m_3_6.value="f";
m_4_0.value="d"; m_4_1.value="f"; m_4_2.value="e";
m_4_3.value="d"; m_4_4.value="c"; m_4_5.value="b"; m_4_6.value="a";
m_5_0.value="e"; m_5_1.value="c"; m_5_2.value="d";
m_5_3.value="e"; m_5_4.value="f"; m_5_5.value="a"; m_5_6.value="b";
m_6_0.value="f"; m_6_1.value="b"; m_6_2.value="a";
m_6_3.value="f"; m_6_4.value="e"; m_6_5.value="d"; m_6_6.value="c";
} }
if (document.all.sel3.selectedIndex==1){
with (document.all) {
m_0_1.value="1"; m_0_2.value="a"; m_0_3.value="b";
m_1_0.value="1"; m_1_1.value="1"; m_1_2.value="a"; m_1_3.value="b";
m_2_0.value="a"; m_2_1.value="a"; m_2_2.value="b"; m_2_3.value="1";
m_3_0.value="b"; m_3_1.value="b"; m_3_2.value="1"; m_3_3.value="a";
}}
if (document.all.sel3.selectedIndex==2){
with (document.all){
m_0_1.value="a"; m_0_2.value="b"; m_0_3.value="c";
m_1_0.value="a"; m_1_1.value="b"; m_1_2.value="c"; m_1_3.value="a";
m_2_0.value="b"; m_2_1.value="c"; m_2_2.value="a"; m_2_3.value="b";
m_3_0.value="c"; m_3_1.value="a"; m_3_2.value="b"; m_3_3.value="c";
}}
if (document.all.sel3.selectedIndex==3){
with (document.all){
m_0_1.value="0";
m_0_2.value="1"; m_0_3.value="2"; m_0_4.value="3"; m_0_5.value="4";
m_1_0.value="0"; m_1_1.value="0"; m_1_2.value="0"; m_1_3.value="0";
m_1_4.value="0"; m_1_5.value="0";
m_2_0.value="1"; m_2_1.value="0"; m_2_2.value="1"; m_2_3.value="2";
m_2_4.value="3"; m_2_5.value="4";
m_3_0.value="2"; m_3_1.value="0"; m_3_2.value="2"; m_3_3.value="4";
m_3_4.value="1"; m_3_5.value="3";
m_4_0.value="3"; m_4_1.value="0"; m_4_2.value="3"; m_4_3.value="1";
m_4_4.value="4"; m_4_5.value="2";
m_5_0.value="4"; m_5_1.value="0"; m_5_2.value="4"; m_5_3.value="3";
m_5_4.value="2"; m_5_5.value="1";
}}
```



```
if (document.all.sel3.selectedIndex==4){
with (document.all){
m_0_1.value="a"; m_0_2.value="b"; m_0_3.value="c";
m_1_0.value="a"; m_1_1.value="a"; m_1_2.value="b"; m_1_3.value="c";
m_2_0.value="b"; m_2_1.value="a"; m_2_2.value="b"; m_2_3.value="c";
m_3_0.value="c"; m_3_1.value="a"; m_3_2.value="b"; m_3_3.value="c";
}}}
function f_cre_w(){/// Проверяем полноту операции
var n=gl_n,i,j;
w = new Array(n); for(i=0;i<n;i++) w[i] = new Array(n);
for(i=0;i<n;i++) for(j=0;j<n;j++)
w[i][j]=eval("document.all.m_"+i+"_"+j+".value");}

function f_test_sc(){
var n=gl_n,i,j,k,wi,wij;
f_cre_w();
document.all.res1.innerHTML="";
flag1=true;
for(i=1;i<n-1;i++) {
wi=w[0][i];
for(j=i+1;j<n;j++)
if (wi==w[0][j]){flag1=false; break;}
if(!flag1) break; }
if(!flag1){fres0();
document.all.res1.innerHTML="Совпадающие элементы в исходной
строке при i="+i+" j="+j; return false;}
else
document.all.res1.innerHTML="Исходная строка элементов определена
верно."; flag1=true;
for(i=1;i<n-1;i++) {
wi=w[i][0];
for(j=i+1;j<n; j++)
if (wi==w[j][0]){flag1=false; break;}
if(!flag1) break;}
if(!flag1) {fres0();
document.all.res1.innerHTML="Совпадающие элементы в исходном
столбце при i="+i+" j="+j;
return false;}
else
```

```
document.all.res1.innerHTML+="<br>Исходный столбец определен
верно."
flag1=true;
for(i=1;i<n;i++)
if (w[0][i]!=w[i][0]){flag1=false;break;}
if(!flag1) {fres0();
document.all.res1.innerHTML="Неправильное определение исходных
элементов при i="+i;return false;}
else
document.all.res1.innerHTML+="<br>Исходные элементы упорядочены
верно."
flag1=true;
for(i=1;i<n;i++) {
for(j=1;j<n;j++) {flag=false;wij=w[i][j];
for(k=1;k<n;k++)
if (wij==w[0][k]) {flag=true;break}
if(!flag) {flag1=false;break;}}
if(!flag) break;}
if(!flag1) {fres0();
document.all.res1.innerHTML+="<br>Таблица содержит неизвестный
элемент при i="+i+" j="+j+"<br><b>Таблица не обладает свойством
полноты.</b>";return false;}
else
document.all.res1.innerHTML+="<br><b>Таблица обладает свойством
полноты.</b>";return true;
}

function f_test_as(){/// Проверяем ассоциативность операции
var n=gl_n,i,j,k,l,wij,wjk,wil,wlk,flag;
document.all.prot.innerHTML=""
if (f_test_sc()){
flag2=true;
for(i=1;i<n;i++) {
for(j=1;j<n;j++) {
for(k=1;k<n;k++){
wjk=w[j][k];
for(l=1;l<n;l++)
if (wjk==w[0][l]) break;
wil=w[i][l];wij=w[i][j];
```



```
for(l=1;l<n;l++)
if (wij==w[0][l]) break;
wlk=w[l][k];
if (wil!=wlk){flag2=false;
flag=w[0][i]+"("+w[0][j]+w[0][k]+") != ("+w[0][i]+w[0][j]+")"+w[0][k]}}
if (!flag2) break;}
if (!flag2) break}
if(!flag2) {
document.all.res2.innerHTML="<hr>Ассоциативность операции
нарушается, например, при<br>"+flag;
return false;}
else
{document.all.res2.innerHTML="<hr><b>Ассоциативность имеет
место.</b>";return true;}}}

function f_test_one(){/// Поиск нейтрального элемента
var n=gl_n,i,j,wi,flag;
if (f_test_sc()){
flag3=false;flag="";
for(i=1;i<n;i++) {
wi=w[0][i];
for(j=1;j<n;j++) {
flag3=false;
if (w[i][j]==wi && wi==w[j][i])
{if(flag=="") flag=w[0][j];
if(flag==w[0][j]) {
flag3=true;break;}}}}
if(!flag3) {document.all.res3.innerHTML="<hr>Нейтральный элемент не
найден.</b>";document.all.res4.innerHTML="";return "";}
else
document.all.res3.innerHTML="<hr><b>Нейтральный элемент
группоида: </b>"+flag;return flag;}}

function f_test_sym(){/// Поиск симметричных элементов
var n=gl_n,i,j,wi,flag,sym,nt=1;
if (f_test_sc()){
flag=f_test_one();
if (flag!=""){sym="<b>Поиск симметричного элемента</b>";
for(i=1;i<n;i++) {wi=w[0][i];sym+="<br>"+wi;
```

```
for(j=1;j<n;j++)
if (w[i][j]==flag && flag==w[j][i])
{sym+=" : "+w[0][j];nt++;}}
flag4=true; if(nt!=n) flag4=false;
document.all.res4.innerHTML="<hr>"+sym
if(flag4)
document.all.res4.innerHTML+="<br>Симметричный элемент найден
для всех элементов."
else
document.all.res4.innerHTML+="<br>Симметричный элемент найден не
для всех элементов."}}}

function f_test_com(){/// Проверяем коммутативность операции
var n=gl_n,i,j;
if (f_test_sc()){
flag5=true;
for(i=1;i<n;i++) {
for(j=i+1;j<n;j++) {
if(w[i][j]!=w[j][i]) {flag5=false;break;}}
if(!flag5) break;}
if(!flag5) {document.all.res5.innerHTML="<hr><b>Коммутативность
операции нарушается</b>, например, при
"+w[0][i]+w[0][j]+"!="+w[0][j]+w[0][i];
return false;}
document.all.res5.innerHTML="<hr><b>Коммутативность имеет
место.</b>"; return true;}}
</script>
```

С помощью текстового редактора Блокнот создать файл **rgz6.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную** **блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

14.4. Содержание отчета

Смотри пункт 9.4.

14.5. Контрольные вопросы

1. Является ли группоидом множество *{1, -1, i, -i}* с обычным умножением? Здесь *i* – мнимая единица.
2. Составьте таблицу Кэли для группоида, образованного из подстановок $\begin{pmatrix}1&2\\1&1\end{pmatrix}, \begin{pmatrix}1&2\\2&1\end{pmatrix}, \begin{pmatrix}1&2\\1&2\end{pmatrix}, \begin{pmatrix}1&2\\2&2\end{pmatrix}$.
3. Проверьте ассоциативность группоида

$\otimes$	1	2	3
1	1	1	1
2	2	2	2
3	3	3	3

4. Составьте таблицу Кэли для группоида, образованного из матриц

$$E=\begin{pmatrix}1&0\\0&1\end{pmatrix},\ A=\begin{pmatrix}-1&0\\0&-1\end{pmatrix},\ B=\begin{pmatrix}0&1\\1&0\end{pmatrix},\ C=\begin{pmatrix}0&-1\\-1&0\end{pmatrix}.$$

5. Исследуйте алгебру <{a, b, c, d, e}; $\otimes$>, определенную следующей таблицей Кэли

$\otimes$	a	b	c	d	e
a	c	d	a	b	e
b	d	c	b	b	e
c	a	a	b	a	c
d	b	a	a	b	d
e	a	b	e	e	c

6. Постройте таблицу Кэли для группы самосовмещения ромба.
7. Покажите, что конечные группы порядка 2 и 3 являются абелевыми.
8. Подмножество A полугруппы $S(\otimes)$ называется **идеалом** полугруппы, если $\forall x\in S,\ a\in A \rightarrow x\otimes a\in A \wedge a\otimes x\in A$. Покажите, что подмножество парных чисел есть идеалом полугруппы всех натуральных чисел за операцией умножения.

9. Докажите, что любая группа имеет один идемпотентный элемент.
10. Постройте таблицы Кэли для полугрупп классов остатков за модулем 8 как по сложению, так и по умножению.
11. Исследуйте алгебру <{a, b, c, d}; ⊗>, определенную следующей таблицей Кэли

⊗	a	b	c	d
a	a	a	b	a
b	c	d	a	b
c	a	c	d	d
d	d	a	d	a

15. ВАРИАНТ № 7. НАХОЖДЕНИЕ ЭЙЛЕРОВОГО ЦИКЛА В НЕОРИЕНТИРОВАННОМ ГРАФЕ

15.1. Цель работы

Изучить алгоритмы обхода неориентированного графа и разработать компьютерную программу для нахождения эйлерового цикла в неориентированном графе.

15.2. Краткие теоретические сведения [5, 13, 14]

Графом *G(V,E)* называется совокупность двух непустых множеств – непустого множества *V* (множества **вершин**) и множества *E* двухэлементных подмножеств множества *V* (*E* – множество **рёбер**). Число вершин графа *G* $p=|V|$, а число ребёр – $q=|E|$. Если некоторая пара вершин соединена несколькими ребрами, то такой граф называется **мультиграфом**. Ребро, ведущее из вершины в нее же, называется **петлей**. Граф без кратных ребер и петель называется **простым**.

Пусть v_1, v_2 – вершины, e = (v_1,v_2) – соединяющее их ребро. Тогда вершина v_1 и ребро *e* **инцидентны**, ребро e и вершина v_2 также инцидентны. Два ребра, инцидентные одной вершине, называются **смежными**; две вершины, инцидентные одному ребру, также называются смежными. **Степенью вершины** называется число ребер,

которые инцидентны данной вершине. Обычно граф изображают диаграммой: вершины – точками (или кружками), рёбра – линиями.

Известны различные способы представления графов в памяти компьютера, которые различаются объёмом занимаемой памяти и скоростью выполнения операций над графами. Представление графа с помощью квадратной булевой матрицы, отражающей смежность вершин, называется **матрицей смежности**.

Маршрутом в графе называется чередующаяся последовательность вершин и ребер, таких, что ребро e_i соединяет вершины v_i и v_{i+i}. Маршрут можно задать как последовательностью вершин, так и последовательностью ребер. **Длиной маршрута** называется число ребер, которые в нем содержатся. **Расстоянием** между двумя вершинами называется длина кратчайшего маршрута их соединяющих. **Цепью** будем называть маршрут, все ребра которого различны, а **простой цепью** назовем цепь, в которой все вершины, возможно, кроме крайних, различны.

Если первая и последняя вершины совпадают, то такая цепь будет являться **циклом**. Цикл, у которого все вершины различны (за исключением первой и последней), называется **простым циклом**. **Связный граф** – это граф, где существует цепь между любой парой вершин *v, u*; иногда такой граф называют **односвязным.**

Цикл, содержащий все ребра графа, называется **эйлеровым циклом**. Связный неориентированный граф является эйлеровым тогда и только тогда, когда каждая вершина имеет четную степень. Путь, который включает каждое ребро графа только один раз называется **эйлеровым путем**. Эйлеров путь который не является эйлеровым циклом называется **собственным эйлеровым путем**.

Опишем алгоритм построения эйлерового цикла в эйлеровом мультиграфе. Этот алгоритм задается следующими правилами.

1. Выбрать произвольно некоторую вершину a.
2. Выбрать произвольно некоторое ребро и, инцидентное a, и присвоить ему номер 1 (назовем это ребро пройденным).
3. Каждое пройденное ребро вычеркнуть и присвоить ему номер, на единицу больший номера предыдущего вычеркнутого ребра.

4. Находясь в вершине x, не выбирать ребро, соединяющее x с a, если имеется возможность иного выбора.
5. Находясь в вершине x, не выбирать ребро, которое является перешейком (т.е. ребром, при удалении которого граф, образованный не вычеркнутыми ребрами, распадается на две компоненты связности, каждая из которых имеет хотя бы по одному ребру).
6. После того как в графе будут занумерованы все ребра, образуется эйлеров цикл, причем порядок нумерации соответствует последовательности обхода ребер.

Алгоритм построения эйлерового цикла может быть также построен на основе алгоритма поиска в глубину [15]. Ребро считается обратным, если не ни одного непомеченного ребра выходящего из текущей вершины. Если такие ребра записывать по мере нахождения в стек, то по завершению обхода в глубину в нем будет хранится эйлеров цикл.

15.3. Порядок выполнения работы

Составить компьютерную программу для нахождения эйлерового цикла в неориентированном графе. Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле ввода матрицы смежности графа;
- поле ввода стартовой вершины цикла;
- поле вывода набора ребер эйлерового цикла;
- кнопку для активизации контрольного примера;
- кнопку для запуска процедуры симметризации матрицы смежности.

На рис. 15.1 приведен возможный вид интерфейса.

Рис. 15.1

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<html><style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<body class="s1" TOPMARGIN=2 onload="fprim();fpoisk();">
<hr size=4><b>Нахождение эйлерова цикла в неориентированном
графе</b><br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-
факультет, НТУ "ХПИ"<hr><input type="button" value="Найти"
onclick="fpoisk();" class="s1">Стартовая вершина <select
name="sel2" class="s1">
<option value="0">1</option><option value="1">2</option>
<option value="2">3</option><option value="3">4</option>
<option value="4">5</option><option value="5">6</option>
<option value="6">7</option><option value="7">8</option>
<option value="8">9</option><option value="9">10</option>
<option value="10">11</option><option value="11">12</option>
</select> <input type="button" value="Пример" onclick="fprim();"
class="s1"><select name="sel3" class="s1">
<option value="1">Конверт</option>
</select><div style="margin-top:3pt;">
<input type="button" value="Матрица смежности" onclick="fcrmat()"
class="s1"> <select name="sel1" class="s1">
<option value="4">4</option><option value="5">5</option>
<option value="6">6</option><option value="7">7</option>
<option value="8">8</option><option value="9">9</option>
<option value="10">10</option><option value="11">11</option>
```

```
<option value="12">12</option></select>
 <input type="button" value="Симметризовать" onclick="fsym()"
class="s1"></div><table border=0 class="s1">
<tr><td valign=top><span ID="mtrx"> </span></td>
<td align=center valign=top><span ID="res"> </span></td>
</tr></table></body></html>
```

Для правильного функционирования данного интерфейса необходимо написать функцию ***fpoisk()***, позволяющих выполнить поиск эйлерового цикла. Возможное решение может иметь вид:

```
<script language="JavaScript">
var gl_n;
function fcrmat() // Формируем пустую матрицу смежности
{
var s="",n,i,j,v;
n=1*document.all.sel1.value;
gl_n=n;
s="<table class='s1'>";

s+="<tr align='center'><td> </td>";
for (i=0;i<n;i++)
s+="<td>"+(i+1)+"</td>";
s+="</tr>";

for (i=0;i<n;i++)
{
s+="<tr><td>"+(i+1)+" </td>";
for (j=0;j<n;j++)
{
s+="<td><input type='text' size='2' value='"+0+"' name='m_"+i+"_"+j+"'
class='s1'></td>";
}
s+="</tr>";
}
s+="</table>";
document.all.mtrx.innerHTML=s;
document.all.res.innerHTML="";
}
function fprim() // Начальные данные для контрольного примера
{
```



```
var i,j,s;
if (document.all.sel3.selectedIndex==0)
document.all.sel1.selectedIndex=2;
fcrmat();

if (document.all.sel3.selectedIndex==0)
{
with (document) {
all.m_0_1.value="1"; all.m_0_2.value="1"; all.m_1_2.value="1";
all.m_1_3.value="1"; all.m_1_4.value="1"; all.m_2_3.value="1";
all.m_2_4.value="1"; all.m_3_4.value="1"; all.m_3_5.value="1";
all.m_4_5.value="1"; }
}
fsym();
}

function fsym() // Симметризуем матрицу смежности
{
var i,j,n;
n=gl_n;
for (i=1;i<n;i++)
for (j=0;j<i;j++)
eval("document.all.m_"+i+"_"+j+".value=document.all.m_"+j+"_"+i+".value");
}

function fpoisk() // Поиск эйлерового цикла

{
var i,star,pos=0,k,sum,j,one=0;
n=gl_n;
stack = new Array(100);
path0 = new Array(100);
path1 = new Array(100);

w = new Array(n); // Матрица смежности графа
for(i=0;i<n;i++) w[i] = new Array(n);

for(i=0;i<n;i++)
for(j=0;j<n;j++)
```

```
{
w[i][j]=eval("document.all.m_"+i+"_"+j+".value");
if(w[i][j]=="b") w[i][j]=32000;
else
w[i][j]=1*w[i][j];
}
start=1*document.all.sel2.selectedIndex
if (start>n) start=0;

// Проверяем: является ли граф эйлеровым
one=0;

for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
if (w[i][j]==1) one++;
if (one%2==1)
{
alert("Граф не является эйлеровым!");
return -1;
}
}

stack[0]=start; // Заносит в стек стартовую вершину

k=1; // Позиция вершины стека

while(k!=0)
{
p=0;
// Находит вершину с минимальным номером, смежную с
// вершиной, номер которой находится в стеке в позиции k

for (i=0; i<n; i++)
if (w[stack[k-1]][i]==1)
{
p=1;
break;
}
```



```
if (p!=0)
{
stack[k]=i; // Заносит вершину в стек
// Помечает ребро как прямое
w[stack[k-1]][i]=2;
w[i][stack[k-1]]=2;
k++;
// Сдвигает позицию вершины стека на единицу вверх
}
else
{
// Заносим в стек обратное ребро

path0[pos] = stack[k-1]+1;
path1[pos] = stack[k-2]+1;
pos++; // Сдвигаем на единицу вверх вершину стека
k--; // Сдвигаем на единицу вниз вершину стека
}
}
// Выводим результат

sum="<b>Результат</b><br><br>"
for (i=0; i<pos-1; i++)
sum+=path0[i] + " - " + path1[i] + "<br>";
document.all.res.innerHTML=sum;
}
</script>
```

С помощью текстового редактора Блокнот создать файл **rgz7.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную** б**лок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

15.4. Содержание отчета

Смотри пункт 9.4.

15.5. Контрольные вопросы

1. Как программе проверяется критерий для возможности построения эйлерового цикла?
2. Для графа, изображенного на рис. 15.2, постройте эйлеров цикл. Если это невозможно, то укажите, какое минимальное количество ребер надо удалить для того, чтобы это стало возможным.

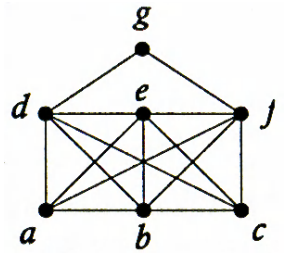


Рис. 15.2

3. Для графа, изображенного на рис. 15.3, постройте эйлеров цикл.

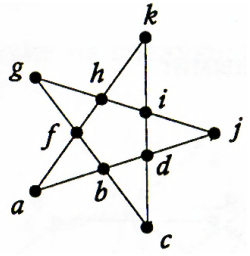


Рис. 15.3

4. Для графа, изображенного на рисунке рис. 15.4, постройте эйлеров цикл.

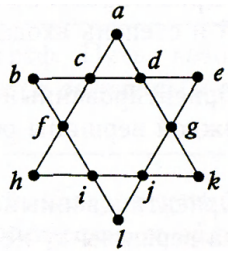


Рис. 15.4

5. Докажите, что граф имеет собственный эйлеров путь тогда и только тогда, когда он связный и ровно две его вершины имеют нечетную степень.

16. ВАРИАНТ № 8.
ОБХОД ГРАФА В ШИРИНУ (ВОЛНОВОЙ АЛГОРИТМ)

16.1. Цель работы

Изучить способы моделирования алгоритмов обхода неориентированных графов и разработать компьютерную программу, реализующую волновой алгоритм.

16.2. Краткие теоретические сведения [2, 10, 11]

Обход (поиск) в ширину является алгоритмом, который необходим при решении многих задач. Поиск в ширину эффективно просматривает вершины и ребра графа, задавая им некоторые метки.

Рассмотрим действие данного алгоритма на примере некоторого графа G (рис.16.1).

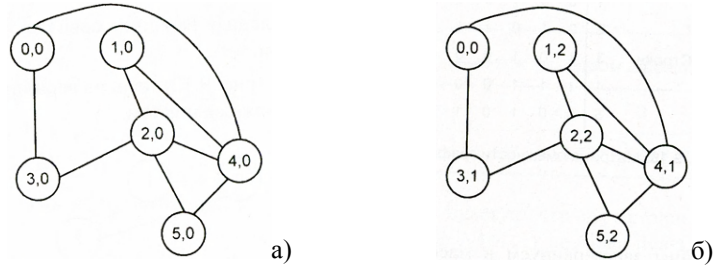


Рис. 16.1. а) Шаг 1 алгоритма поиска в ширину (вершины помечены следующим образом – номер вершины, начальная метка). б) Метки вершин после выполнения волнового алгоритма

На первом шаге сделаем метки всех вершин нулевыми. Теперь необходимо выбрать стартовую вершину поиска. Пусть это будет вершина с номером 0. Каждой вершине, инцидентной данной вершине, присвоим метку 1. Присваивание метки происходит только в том случае, если вершина не помечена. Таким образом, вершины 3 и 4 получили метку 1. Теперь рассмотрим поочередно окружение всех вершин с меткой 1, и каждой вершине из этого окружения присвоим

метку 2. Таким образом, вершины 2, 1 и 5 получат метку 2. Далее, рассматривая окружение вершин 2, 1 и 5, алгоритм не найдет ни одной помеченной вершины. Данный результат означает, что поиск в ширину выполнен, и вершины получили свои метки.

Процесс расстановки меток можно сравнить с процессом распространения волны. Поэтому часто поиск в ширину называют **волновым алгоритмом**. После того как все вершины получили свои метки, можно определить **расстояние от стартовой вершины до любой другой**. Оно будет равно метке вершины, расстояние до которой необходимо определить. Следовательно, расстояние от вершины 0 до вершин 3 и 4 равно единице, а расстояние до вершин 1, 2 и 5 равно двум.

Для реализации волнового алгоритма зададим граф G с помощью матрицы смежности. Для каждой вершины зарезервируем в массиве label ячейку, в которую будет помещена метка данной вершины. n ячеек этого массива заполним большим числом, например, 32767, чтобы затем удобнее было производить сравнение. В качестве стартовой вершины можно взять произвольную вершину графа. После того как мы объявили стартовую вершину, она заносится в очередь и получает метку 0. Далее просмотрим нулевую строку матрицы смежности. Если некоторый элемент не равен нулю, то соответствующий номер столбца элемента помещается в очередь. Эти вершинам присваивается метка 1. После этого вершина 0 удаляется из очереди. Затем извлекается следующий элемент из очереди и алгоритм анализа повторяется. Алгоритм останавливается в том случае, если очередь становится пустой.

16.3. Порядок выполнения работы

Составить компьютерную программу для обхода графа в ширину (волновой алгоритм). Программа должна позволять пользователю вводить граф разной размерности, динамически проставлять метки вершин для заданного источника.

Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле матрицы смежности заданного графа;
- кнопки для активизации процесса обхода графа в ширину от заданного источника;
- динамически отображаемую таблицу результата, включающую перечень меток для каждой вершины;
- динамически формировать пустую матрицу смежности разной размерности;
- встроенную проверку работоспособности алгоритма на тестовых примерах.

На рис. 16.2 приведен возможный вид интерфейса.

Рис. 16.2

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<body class="s1" onload="fprim();fpoisk();" TOPMARGIN=2>
<hr size=4><b>Обход графа в <u>ширину</u> (волновой алгоритм)</b>
<br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ "ХПИ"
<hr><input type="button" value="Вычислить" onclick="fpoisk();" class="s1">
 Источник <select name="sel2" class="s1" onchange="fsel2();fpoisk()">
```

```
<option value="1">1</option><option value="2">2</option>
<option value="3">3</option><option value="4">4</option>
<option value="5">5</option><option value="6">6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option value="10">10</option></select>
 <input type="button" value="Пример" onclick="fprim();fsel2()"
class="s1">
<select name="sel3" class="s1">
<option value="1">Вариант 1</option>
<option value="2">Вариант 2</option>
</select>
<div style="margin-top:3pt;">
<input type="button" value="Матрица смежности" onclick="fcrmat()"
class="s1">
 <select name="sel1" class="s1" onchange="fsel2();">
<option value="5">5</option><option value="6" selected>6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option value="10">10</option></select>
</div><table border=0 class="s1">
<tr><td valign=top><span ID="mtrx"> </span></td>
<td align=center valign=top><span ID="res"> </span></td>
</tr></table></body>
```

Для правильного функционирования данного интерфейса необходимо написать функцию **fpoisk**, реализующую волновой алгоритм для обхода неориентированного графа.

Возможное решение может иметь вид:

```
<script language="JavaScript">
var gl_n;

function fcrmat() // Формируем матрицу для ввода данных
{
var s="",n,i,j;
n=1*document.all.sel1.value;
gl_n=n;
s="<table class='s1'>";

s+="<tr align='center'><td> </td>";
for (i=0;i<n;i++)
s+="<td>"+(i+1)+"</td>";
```



```
s+="</tr>";

for (i=0;i<n;i++)
{
s+="<tr><td>"+(i+1)+" </td>";
for (j=0;j<n;j++)
s+="<td><input type='text' size='2' value='0' name='m_"+i+"_"+j+"'
class='s1'></td>";
s+="</tr>";
}
s+="</table>";
document.all.mtrx.innerHTML=s;
document.all.res.innerHTML="";
}

function fprim() // Данные для примера
{
if (document.all.sel3.selectedIndex==0)
document.all.sel1.selectedIndex=1;
fcrmat();
if (document.all.sel3.selectedIndex==0){
with (document.all) {
m_0_3.value="1"; m_0_4.value="1";
m_1_2.value="1"; m_1_4.value="1";
m_2_1.value="1"; m_2_3.value="1"; m_2_4.value="1"; m_2_5.value="1";
m_3_0.value="1"; m_3_2.value="1";
m_4_0.value="1"; m_4_1.value="1"; m_4_2.value="1"; m_4_5.value="1";
m_5_2.value="1"; m_5_4.value="1"; } }
}

function fsel2()
{
var s1,s2;
s1=1*document.all.sel1.selectedIndex+5;
s2=1*document.all.sel2.selectedIndex+1;
if (s2>s1) document.all.sel2.selectedIndex=0;
}

function fpoisk() // Осуществляем обход графа в ширину
```

```
{
var n,i,j,start,m=0,p,k,cur;

n=gl_n; // Количество вершин в графе
start=1*document.all.sel2.selectedIndex;
w = new Array(n); // Объявляем матрицу смежности графа
for(i=0;i<n;i++) w[i] = new Array(n);

// Формируем матрицу смежности графа
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
w[i][j]=1*eval("document.all.m_"+i+"_"+j+".value");
if (w[i][j]==1) m++;
}
m/=2; // Количество ребер в графе

label = new Array(m); // Объявляем массив меток
fifo  = new Array(m); // Объявляем структуру данных "очередь"

for(i=0;i<m;i++)
{
fifo[i]=0; label[i]=32767;
}
p=0; // Указатель на начало очереди
k=1; // Указатель на конец очереди
fifo[p]=start; // Заносим стартовую вершину в очередь
label[start]=0; // Помечаем стартовую вершину меткой 0

while (p!=k)
{
cur=fifo[p]; // Выделяем очередную вершину из очереди
p++; // Сдвигаем указатель начала на единицу

for (i=0; i<n; i++)
if (w[cur][i]==1 && label[i]>label[cur]+1)
{
fifo[k]=i; // Заносим вершину в очередь
k++; // Сдвигаем указатель конца очереди
```



```
label[i]=label[cur]+1; // Помечаем вершину
}
}
// Выводим результат
s="<b>Результат</b><br><table
class=s1><tr><td><b>Вершина</b></td><td><b>Метка</b></tad></tr>";
for (i=0; i<n; i++)
s+="<tr align=center><td>"+(i+1)+"</td><td>"+label[i]+"<td></tr>";
s+="</table>";
document.all.res.innerHTML=s;
}</script>
```

С помощью текстового редактора Блокнот создать файл **rgz8.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

16.4. Содержание отчета

Смотри пункт 9.4.

16.5. Контрольные вопросы [7, 11]

4. Как будет выглядеть результат работы программы **rgz8.htm** для полных графов?
5. Известно, что во всяком дереве с количеством вершин больше или равно двум, содержатся не менее двух висячих вершин. Как этот факт отразится на результате работы волнового алгоритма?
6. Позволяют ли результаты работы программы **rgz8.htm** определить эксцентриситет заданной вершины?
7. Чему равен диаметр графа для тестового случая в программе **rgz8.htm?**
8. Найти матрицу расстояний, диаметр, радиус, центральные и периферийные вершины графа, изображенного на рис. 16.3.

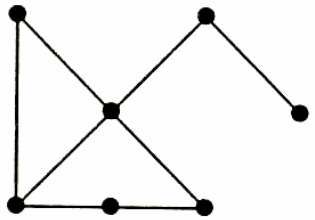


Рис. 16.3

17. ВАРИАНТ № 9.
ОБХОД ГРАФА В ГЛУБИНУ

17.1. Цель работы

Изучить способы моделирования алгоритмов обхода неориентированных графов и разработать компьютерную программу, реализующую алгоритм поиска в глубину.

17.2. Краткие теоретические сведения [2, 10, 11]

Обход (поиск) в глубину помечает вершины и ребра графа некоторым образом. Рассмотрим действие данного алгоритма на примере некоторого графа G (рис. 17.1).

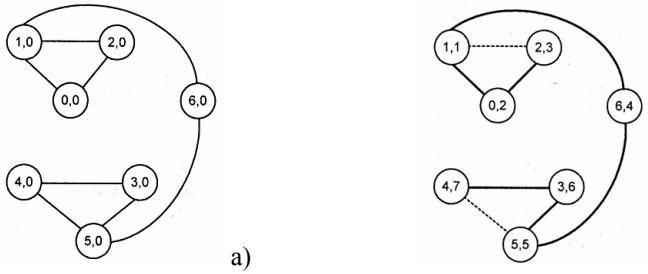


Рис. 17.1. а) Шаг 1 алгоритма поиска в глубину (вершины помечены следующим образом – номер вершины, начальная метка). б) Метки вершин после выполнения поиска в глубину. Пунктиром обозначены обратные ребра

На первом шаге сделаем метки всех вершин нулевыми. Выберем в качестве стартовой вершины – вершину с номером 1. Так как с нее начат поиск, то она уже пройдена, и следовательно, получает метку k=1. На третьем шаге выберем ребро, которое соединяет вершину 1 и непомеченную вершину с минимальным номером. В данном случае это

будет вершина с номером 0. Увеличим k на единицу и помечаем вершину 0 меткой k. При этом ребро (1,0) считается **пройденным прямым ребром**. Затем снова выбираем непомеченную вершину с минимальным номером и помечаем меткой k+1, т.е. вершина 2 будет помечена меткой 3, а ребро (0,2) будет отмечено как пройденное. Из вершины 2 выходит одно непомеченное ребро – это ребро (2,1), но вершина 1 уже помечена. В данной ситуации необходимо пометить ребро (2,1) как **обратное**. Двигаясь, таким образом, алгоритм возвращается в вершину 1, из которой был начат обход, и завершает свою работу.

17.3. Порядок выполнения работы

Составить компьютерную программу для обхода графа в глубину. Программа должна позволять пользователю вводить граф разной размерности, динамически проставлять метки вершин для заданного источника.

Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Интерфейсная часть должна включать:

- заголовок с указанием названия работы, информации об авторе, времени разработки;
- поле матрицы смежности заданного графа;
- кнопки для активизации процесса обхода графа в глубину от заданного источника;
- динамически отображаемую таблицу результата, включающую перечень меток для каждой вершины;
- динамически формировать пустую матрицу смежности разной размерности;
- встроенную проверку работоспособности алгоритма на тестовых примерах.

На рис. 17.2 приведен возможный вид интерфейса.

Рис. 17.2

Данный интерфейс может быть реализован такой последовательностью тегов:

```
<style>.s1 {font-family:Tahoma;font-size:8pt;}</style>
<body class="s1" onload="fprim();fpoisk();" TOPMARGIN=2>
<hr size=4>
<b>Обход графа в <u>глубину</u> </b>
<br>&copy; 2008 Савченко Н.В., кафедра СИ, КИТ-факультет, НТУ "ХПИ"
<hr><input type="button" value="Вычислить" onclick="fpoisk();" class="s1">
 Источник <select name="sel2" class="s1" onchange="fsel2();fpoisk()">
<option value="1">1</option><option value="2">2</option>
<option value="3">3</option><option value="4">4</option>
<option value="5">5</option><option value="6">6</option>
<option value="7">7</option><option value="8">8</option>
<option value="9">9</option><option value="10">10</option></select>
 <input type="button" value="Пример" onclick="fprim();fsel2()" class="s1">
<select name="sel3" class="s1">
<option value="1">Вариант 1</option><option value="2">Вариант 2</option>
</select><div style="margin-top:3pt;">
<input type="button" value="Матрица смежности" onclick="fcrmat()" class="s1"> <select name="sel1" class="s1" onchange="fsel2();">
<option value="5">5</option><option value="6" selected>6</option>
<option value="7">7</option><option value="8">8</option>
```



```
<option value="9">9</option><option value="10">10</option></select>
</div><table border=0 class="s1"> <tr><td valign=top>
<span ID="mtrx"> </span></td><td align=center valign=top>
<span ID="res"> </span></td></tr></table></body>
```

Для правильного функционирования данного интерфейса необходимо написать функцию **fpoisk**, реализующую поиск в глубину для неориентированного графа.

Возможное решение может иметь вид:

```
<script language="JavaScript">
var gl_n;
function fcrmat()
{var s="",n,i,j;
n=1*document.all.sel1.value;
gl_n=n;
s="<table class='s1'>";
s+="<tr align='center'><td> </td>";
for (i=0;i<n;i++)
s+="<td>"+(i+1)+"</td>";
s+="</tr>";
for (i=0;i<n;i++)
{s+="<tr><td>"+(i+1)+" </td>";
for (j=0;j<n;j++)
s+="<td><input type='text' size='2' value='0' name='m_"+i+"_"+j+"' class='s1'></td>";
s+="</tr>";}
s+="</table>";
document.all.mtrx.innerHTML=s;
document.all.res.innerHTML="";}
function fprim() {
if (document.all.sel3.selectedIndex==0)
{document.all.sel1.selectedIndex=2;
document.all.sel2.selectedIndex=1;}
fcrmat();
if (document.all.sel3.selectedIndex==0){
with (document.all) {
m_0_1.value="1"; m_0_2.value="1";
m_1_0.value="1"; m_1_2.value="1"; m_1_6.value="1";
m_2_0.value="1"; m_2_1.value="1";m_3_4.value="1"; m_3_5.value="1";
m_4_3.value="1"; m_4_5.value="1";
m_5_3.value="1"; m_5_4.value="1"; m_5_6.value="1";
```

```
m_6_1.value="1"; m_6_5.value="1"; }}}
function fsel2()
{var s1,s2;
s1=1*document.all.sel1.selectedIndex+5;
s2=1*document.all.sel2.selectedIndex+1;
if (s2>s1) document.all.sel2.selectedIndex=0;}
function fpoisk()
{var n,i,j,start,m,p,k,cur;
n=gl_n; // Количество вершин в графе
start=1*document.all.sel2.selectedIndex;
w = new Array(n); // Объявляем матрицу смежности графа
for(i=0;i<n;i++) w[i] = new Array(n);
// Формируем матрицу смежности графа
for(i=0;i<n;i++)
for(j=0;j<n;j++)
w[i][j]=1*eval("document.all.m_"+i+"_"+j+".value");
label = new Array(n); // Объявляем массив меток
lifo  = new Array(n); // Объявляем структуру данных "стек"
for(i=0;i<n;i++)
{label[i]=0; lifo[i]=0;}
lifo[0]=start; // Заносим стартовую вершину в стек
label[start]=1; // Присваиваем стартовой вершине метку 1
k=1; // Указатель на вершину стека
m=1; // Начальное значение метки

while (k!=0)
{p=0;
// Находим вершину с минимальным номером, смежную с
// вершиной, номер которой находится в стеке в позиции k
for (i=0; i<n; i++)
if (w[lifo[k-1]][i]==1) {p=1; break;}
if (p!=0)
{ if (label[i]==0)
 { lifo[k]=i; // Заносим вершину в стек
 m++; // Увеличивает значение метки
 label[i]=m; // Помечает вершину
 // Помечаем ребро как прямое
 w[lifo[k-1]][i]=2;
 w[i][lifo[k-1]]=2;
```



```
 k++; // Сдвигает позицию вершины стека на единицу вверх
 }
else
{ // Помечаем ребро как обратное
 w[i][lifo[k-1]]=3;
 w[lifo[k-1]][i]=3;
}}
else
 // Сдвигаем позицию вершины стека на единицу вниз,
 // тем самым удаляя вершину из стека
k--;}

// Выводим результат
s="<b>Результат</b><br><table
class=s1><tr><td><b>Вершина</b></td><td><b>Метка</b></tad></tr>";
for (i=0; i<n; i++)
s+="<tr align=center><td>"+(i+1)+"</td><td>"+label[i]+"<td></tr>";
s+="</table>";
document.all.res.innerHTML=s;}</script>
```

С помощью текстового редактора Блокнот создать файл **rgz9.htm** с рабочей версией программы. Отладку программы проводить в браузере Internet Explorer. Текст программы в обязательном порядке должен содержать комментарии. Провести тестирование программы на контрольных примерах. **Функциональную блок-схему** программы нарисовать с использованием программы Microsoft Visio, входящей в комплект Microsoft Office.

17.4. Содержание отчета

Смотри пункт 9.4.

17.5. Контрольные вопросы [7, 11]

1. Как образом реализована работа со стеком в программе **rgz9.htm?**
2. Как будет вести себя программа **rgz9.htm** в случае если исходный граф не связный?
3. Как будет вести себя программа **rgz9.htm** в случае если исходный граф полный?

4. Как модифицировать программу **rgz9.htm** чтобы она определяла какое минимальное количество ребер необходимо добавить в граф, чтобы он стал связным?
5. Задан связный неориентированный граф. Напишите программу, которая определяет, какое минимальное количество ребер можно исключить из графа, чтобы он по-прежнему оставался связным.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАЗРАБОТКИ

Разработать предложенную тему и составить компьютерную программу для выполнения конкретных операций над объектами темы.

Программу реализовать:

- интерфейсная часть на языке разметки гипертекста HTML [3];
- функциональная часть на си-подобном языке JavaScript [4].

Отчет кроме пунктов, перечисленных в разделе 1.4, должен содержать также краткие теоретические сведения и не менее 5 контрольных вопросов по теме задания. Подготовьте условия 3-4 заданий, которые могут, выполнены с использованием разработанной программы.

1. Алгоритм Уоршала вычисления транзитивного замыкания отношения [5].
2. Вычисление значений булевой функции по сокращенному дереву решений [5].
3. Метод резолюций [5].
4. Генерация перестановок.
5. Генерация перестановок с повторениями.
6. Генерация подмножеств.
7. Проверка неориентированного графа на связность [16].
8. Проверка неориентированного графа на регулярность [16].
9. Проверка неориентированного графа на двудольность [16].
10. Раскраска вершин графа [16].
11. Центроид дерева [5].
12. Кодировка Прюфера [16].

13. Кодировка Гапта [16].
14. Потоки в сетях [16].
15. Гамильтоновы циклы [16].
16. Задача коммивояжера [5].
17. Упаковка по методу Лемпела-Зива [5].
18. Простой столбцевой перестановочный шифр [16].
19. Перестановочный шифр с ключевым словом [16].
20. Шифр Полибия [16].
21. Алгоритм Фано [5].
22. Алгоритм Хаффмена [5].
23. Очередь [16].
24. Списки [16].
25. Деревья [5].

Список используемой и рекомендованной литературы

1. Малые жанры русского фольклора: Хрестоматия: Учеб. пос. для филол. спец. вузов / Сост. В.Н. Морохин. – М.: Высш. шк., 1986. – 399 с.

2. Судоплатов С.В., Овчинникова Е.В. Дискретная математика. – Новосибирск: Изд-во НГТУ, 2007. – 256 с.

3. Нидерст Дж. Web-мастеринг для профессионалов. – СПб.: Питер, 2001. – 576 с.

4. Федоров А.Г. JavaScript для всех. – М.: Компьютер Пресс, 1998.– 384 с.

5. Новиков Ф.А. Дискретная математика для программистов. – СПб.: Питер, 2006. – 368 с.

6. Белоусов А.И., Ткачев С.Б. Дискретная математика. – М.: Изд-во: МГТУ им. Н. Э. Баумана, 2006. – 744 с.

7. Кочетков П.А. Введение в дискретную математику. – М.: МГИУ, 2007. – 88 с.

8. Иванова Г.С. Технология программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2002. – 320 с.

9. Шапорев С.Д. Дискретная математика: Курс лекций и практических занятий. – СПб.: БХВ-Петербург, 2006. – 400 с.

10. Бондаренко М.Ф., Белоус Н.В., Руткас А.Г. Компьютерная дискретная математика. – Харьков: «Компания СМИТ», 2004. – 480 с.

11. Хаггарти Р. Дискретная математика для программистов. – М.: Техносфера, 2005. – 400 с.

12. Окулов С.М. Программирование в алгоритмах. – М.: БИНОМ, 2002. – 341 с.

13. Бородін О.І., Потьомкін Л.В., Сліпенко А.К. Основні поняття сучасної алгебри. – Київ: Радянська школа, 1976. – 103 с.

14. Андерсон Д, А. Дискретная математика и комбинаторика: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 960 с.

15. Динман М.И. С++. Освой на примерах. – СПб.: БХВ-Петербург, 2006. – 384 с.

16. Кирсанов М.Н. Графы в Maple. Задачи, алгоритмы, программы. – М.: ФИЗМАТЛИТ, 2007. – 168 с.

Краткий словарь терминов

- **Алеф-нуль** – кардинальное число, используемое для обозначения мощности счетного множества.

- **Антирефлексивность** – свойство бинарного отношения: отношение R антирефлексивно, если не найдется элемента x такого, чтобы было xRx.

- **Антисимметричность** – свойство бинарного отношения: отношение R антисимметрично, если не найдется двух элементов x и y так, чтобы было xRy, x≠y и yRx.

- **Антитранзитивность** (нетранзитивность) – свойство бинарного отношения: отношение R нетранзитивно, если оно не является транзитивным, но найдутся элементы x, y и z так, что xRy, yRz и xRz;

- **Антитранзитивность** (интранзитивность) – свойство бинарного отношения: отношение R интранзитивно, если не найдется трех элементов x, y, z так, чтобы было xRy, yRz и xRz; напр., интранзитивным окажется отношение <x является отцом y-ка>.

- **Ассоциативность**, **сочетательность** – свойство бинарной алгебраической операции: операция ⊗ ассоциативна, если при всех x, y и z имеет место равенство (x ⊗ y) ⊗ z = x ⊗ (y ⊗ z); Понятие ассоциативности обобщается также на случай n-арной операции.

- **Бесконечное множество** – множество, которое эквивалентно своему собственному подмножеству; напр., множество N натуральных чисел бесконечно, так как оно эквивалентно множеству всех четных натуральных чисел.

- **Биекция** – биективное отображение, взаимно однозначное соответствие соответствие, являющееся одновременно однозначным, инъективным и сюръективным; отображение, являющееся одновременно инъекцией и сюръекцией;

обозначается X–Y (читается: биекция между множествами Y и X).

- **Бинарная операция** – бинарная алгебраическая операция – алгебраическая операция, в которой число операндов равно двум, т.е. отображение прямого квадрата множества в это же множество; напр., в множестве M определена бинарная операция со знаком операции ⊗ (или просто операция ⊗), если для любых a, b ∈ M определен элемент a⊗b ∈ M.

- **Бинарное отношение**, **отношение** – подмножество прямого квадрата некоторого множества; если R – бинарное отношение на множестве M, т. е. R⊂M×M, то вместо (x,y)⊂R обычно пишут xRy; бинарное отношение иногда удобно понимать как двуместный предикат на данном множестве.

- **Бином**, **двучлен** – сумма или разность двух одночленов

- **Биномиальная формула**, **Ньютона бином** – формула разложения степени бинома по степеням его одночленов: $(a+b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k}$.

- **Булеан** – совокупность B(M) всех подмножеств рассматриваемого множества M; обозначается также 2^M.

- **Булева алгебра**, **булева решетка** – множество, в котором определены две коммутативные, ассоциативные, взаимно дистрибутивные и поглощающие бинарные алгебраические операции ∨, ∧ и унарная операция ' – (дополнение), так, что при всех x, y выполнены условия: (x ∧ x') ∨ y = y, (x ∨ x') ∧ y = y.

- **Булева функция**, **функция алгебры логики** – функция, аргументы которой, равно как и сама функция, принимают значения из некоторого двухэлементного множества.

- **Булево кольцо** – ассоциативное кольцо с единицей, все элементы которого идемпотентны.

- **Внешнее произведение множеств A и B** – множество A×B всевозможных упорядоченных пар вида (a, b), где a∈A и b∈B.

Аналогично определяется внешнее произведение множеств для трех, четырех и вообще произвольного множества множеств. Внешнее произведение множеств называют также прямым произведением, декартовым произведением (по имени Р. Декарта) или картезианским произведением (от слова «картезиус» – латинизированного имени Декарта). Пример. Пусть A – множество целых чисел, тогда внешнее произведение множеств $A \times B$ – множество всех двумерных целочисленных векторов.

- **Гипотеза** – недоказанное предложение.

- **Гомоморфизм** – отображение алгебраической системы в другую алгебраическую систему, сохраняющее рассматриваемые операции и/или отношения.

- **Группа** – моноид, каждый элемент которого является обратимым. операция моноида называется в этом случае групповой операцией.

- **Группоид** – универсальная алгебра с одной бинарной операцией.

- **Дизъюнктивная нормальная форма** (ДНФ) – выражение, состоящее из некоторой дизъюнкции конъюнкций.

- **Дистрибутивность**, **распределительность** – свойство бинарной алгебраической операции распределиться относительно другой бинарной алгебраической операции.

- **Законы де Моргана** – отрицание конъюнкции равносильно дизъюнкции отрицаний. Отрицание дизъюнкции равносильно конъюнкции отрицаний.

- **Изоморфизм** – биекция одной системы на другую систему такого же типа, сохраняющая структуру этих систем, т. е. операции, упорядоченность, топологию и т. п.

- **Изоморфизма проблема** – задача отыскания алгоритма, позволяющего по любой паре алгебраических систем из данного класса установить, изоморфны они или нет.

- **Изоморфные системы** – системы, между которыми можно указать изоморфизм.

- **Импликация** – бинарная логическая операция: импликация A $\rightarrow$ B или A $\subset$ B или A $\Rightarrow$ B высказываний A (посылки или антецедента) и B (заключения или консеквента) ложна тогда и только тогда, когда посылка A истинна, а заключение B ложно.

- **Инъекция** – вложение, инъективное соответствие (отображение) такое соответствие (отображение) между множествами X и Y ($X \rightarrow Y$), при котором различным элементам из X соответствуют различные элементы в Y.

- **Истина** – одно из возможных истинностных значений.

- **Истинностная таблица** – таблица, выражающая истинностное значение сложного высказывания через истинностные значения входящих в него высказываний.

- **Кантора теорема** – множество, состоящее из всех подмножеств данного непустого множества M, не эквивалентно ни самому M, ни его подмножеству.

- **Кардинальное число** – символ, обозначающий мощность множества; в случае конечного множества натуральное число: число элементов в множестве.

- **Квазигруппа** – такой группоид с бинарной операцией $\otimes$, в котором уравнения $a \otimes x = b$ и $y \otimes a = b$ имеют единственные решения для любых фиксированных элементов a и b.

- **Кватернион** – гиперкомплексное число вида $z = a \otimes 1 + b \otimes i + c \otimes j + b \otimes k$, где умножение гиперкомплексных единиц 1, i, j, k определено таблицей

$\otimes$	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

Множество всех кватернионов образует тело.

- **Класс** – синоним термина множество.

- **Кольцо** – аддитивная абелева группа, в которой определена еще операция умножения так, что умножение дистрибутивно относительно сложения.

- **Коммутативная группа** – абелева группа.

- **Коммутативное кольцо** – кольцо, умножение в котором является коммутативным.

- **Коммутативность, перестановочность** – свойство бинарной алгебраической операции: операция $\otimes$ коммутативна, если при всех x и y имеет место равенство $x \otimes y = y \otimes x$.

- **Континуума мощности множество** – множество, эквивалентное множеству всех подмножеств множества натуральных чисел; напр., множество R действительных чисел, множество точек отрезка [0,1], множество всех точек евклидова пространства R^n.

- **Континуума мощность** – кардинальное число $c=2^{\text{алеф-нуль}}$.

- **Континуум–гипотеза** – гипотеза (Г. Кантор, 1878), что не существует несчетного множества, мощность которого меньше мощности континуума.

- **Конъюнктивная нормальная форма** (КНФ) – выражение, состоящее из некоторой конъюнкции дизъюнкций.

- **Кэли таблица** – таблица умножения конечного группоида.

- **Логика** – наука о законах и формах мышления, об условиях истинности наших знаний.

- **Логическая операция** – операция построения сложного высказывания из данных высказываний; в математической логике определяются следующие основные логические операции: дизъюнкция, импликация, квантор, конъюнкция, отрицание, эквивалентность.

- **Логическая формула** – выражение, составленное из элементарных формул при помощи логических операций.

- **Логическая функция** – функция, определенная на прямой степени множества истинностных значений и принимающая значения в этом множестве.

- **Логически равносильные формулы** – логические формулы A и B, при которых формула $(A \to B) \wedge (B \to A)$ окажется общезначимой; обозначается $A \equiv B$ или $A \Leftrightarrow B$.

- **Логическое значение** – истинностное значение.

- **Логическое исчисление** – формализация содержательной логической теории.

- **Логическое следствие** – высказывание, являющееся истинным при любой интерпретации символов, при которой истинны посылки.

- **Логическое сложение** – дизъюнкция.
- **Логическое умножение** – конъюнкция.
- **Ложь** – одно из возможных истинностных значений.
- **Лупа** – квазигруппа, в которой имеется нейтральный элемент.

- **Множеств теория** – раздел математики, изучающий общие свойства множеств и операций над ними; дескриптивная теория множеств.

- **Множество** – одно из основных понятий математики: набор каких-либо различных объектов или элементов, рассматриваемое как одно целое; если множество задано перечислением его элементов, то такое множество обозначается в виде {a, b, ...}, если же имеется правило для определения принадлежности элементов множеству, то множество обозначается в виде {x :...} или {x|...}, где за двоеточием или вертикальной чертой указываются условия, которым должен удовлетворять элемент x, чтобы принадлежать рассматриваемому множеству; синонимы: набор, совокупность, система, комплект, класс.

- **Моноид, полугруппа с единицей** – полугруппа, в которой имеется нейтральный элемент соответствующей операции; напр., мультипликативная полугруппа N натуральных чисел или аддитивная полугруппа N, если нуль считать натуральным числом.

- **Мощность множества** – класс эквивалентности равномощных множеств, в который принадлежит рассматриваемое множество.

- **Несчетное множество** – бесконечное множество, не являющееся счетным множеством.

- **Операция, действие** – отображение прямого произведения некоторых множеств $M_1, M_2, \ldots, M_n$ в некоторое множество M: $M_1 \times M_2 \times \ldots \times M_n \rightarrow M$. Если число компонент этого прямого произведения равно n, то операция называется n-арной операцией; компоненты X_n оригинала $(x_1, x_2, \ldots, x_n) \in M_1 \times M_2 \times \ldots \times M_n$ называются операндами соответствующей операции ω, а образ $\omega(x_1, x_2, \ldots, x_n)$ – результатом или значением операции.

- **Отношение порядка, порядок, упорядоченность** – бинарное отношение, обладающее свойствами рефлексивности, антисимметричности и транзитивности (нестрогая упорядоченность), или же свойствами антирефлексивности и транзитивности (строгая упорядоченность); нестрогую упорядоченность обозначают обычно символами $\geq, \leq, \supseteq$ или $\subseteq$, а строгую упорядоченность – символом $<, >, \subset$ или $\supset$.

- **Отношение равенства** – бинарное отношение, совпадающее с диагональю прямого квадрата рассматриваемого множества.

- **Отношение эквивалентности, эквивалентность** – бинарное отношение, обладающее свойствами рефлексивности, симметричности и транзитивности.

- **Отображение, однозначное соответствие** – такое соответствие между множествами X и Y, при котором любому элементу $x \in X$ (прообразу) соответствует точно один элемент

$y \in Y$ (образ); обозначается $X \rightarrow Y$ (читается: отображение множества X в множество Y; если соответствие сюръективно, то: отображение множества X на множество Y);

- **Перестановка** – определение нового линейного упорядочения в линейно упорядоченном множестве; применение подстановки к конечному линейно упорядоченному множеству.

- **Перестановки** – различные линейно упорядоченные множества, состоящие из всех элементов некоторого заданного основного множества; если основное множество содержит n различных элементов, то число P_n всех перестановок выражается формулой $P_n = n!$

- **Перестановки с повторениями** – перестановки такого основного множества, среди n элементов которого имеется только *k* различных: первый элемент множества имеется в m_1 экземплярах, второй элемент в m_2 экземплярах и т. д., причем $m_1 + m_2 + \ldots + m_k = n$ число всех перестановок с повторениями выражается в виде полиноминального коэффициента $P_n^{m_1 m_2 \ldots m_k} = \dfrac{n!}{m_1! m_2! \ldots m_k!}$.

- **Полугруппа, ассоциативная система** – множество, в котором определена ассоциативная бинарная алгебраическая операция.

- **Размещения** – различные линейно упорядоченные равномощные подмножества некоторого заданного основного множества; если основное множество содержит n различных элементов, то число A^m_n всех m-элементных размещений или m-размещений выражается в виде убывающего факториала $A^m_n = n \cdot (n-1) \cdot \ldots \cdot (n-m+1)$.

- **Рефлексивность** – свойство бинарного отношения содержать диагональ прямого квадрата рассматриваемого множества: отношение R рефлексивно, если xRx при всех x.

- **Симметричность** – свойство бинарного отношения: отношение R симметрично, если из xRy всегда следует yRx.

- **Совершенная дизъюнктивная нормальная форма** – полная дизъюнктивная нормальная форма такая дизъюнктивная нормальная форма данной формулы, каждая элементарная конъюнкция которой содержит все рассматриваемые переменные, причем каждое по одному разу.

- **Совершенная конъюнктивная нормальная форма** – полная конъюнктивная нормальная форма такая конъюнктивная нормальная форма данной формулы, каждая элементарная дизъюнкция которой содержит все рассматриваемые переменные, причем каждое по одному разу.

- **Сочетания** – различные равномощные подмножества некоторого заданного основного множества; если основное множество содержит n различных элементов, то число C_n^m всех *m*-элементных сочетаний или m-сочетаний выражается в виде биномиального коэффициента $C_n^m = \frac{n!}{m!(n-m)!}$.

- **Сочетания с повторениями** – сочетания в случае основного множества, содержащего n различных элементов, по сколько угодно экземпляров каждого; число *m*-элементных сочетаний с повторениями выражается в виде: $C_{n+m-1}^m = \frac{n+m-1!}{m!(n-1)!}$.

- **Счетное множество** – множество, эквивалентное множеству N всех натуральных чисел; напр., множество всех целых чисел Z, множество всех рациональных чисел Q, множество всех рациональных точек плоскости [пространства].

- **Сюръекция, сюръективное соответствие (отображение)** – такое соответствие (отображение) между множествами X и Y ($X \to Y$), при котором каждый элемент множества Y соответствует некоторому элементу множества Y.

- **Транзитивное замыкание** – бинарное отношение R^+, определенное как объединение всех степеней данного бинарного отношения R: xR^+y тогда и только тогда, когда найдется натуральное число n такое, что xR^ny.

- **Транзитивность** – свойство бинарного отношения: отношение R транзитивно, если из xRy и yRz всегда следует xRz.

- **Элементарная дизъюнкция** – дизъюнкция, все члены которой являются элементарными формулами или их отрицаниями.

- **Элементарная конъюнкция** – конъюнкция, все члены которой являются элементарными формулами или их отрицаниями.

Словарь терминов по теории графов

- **Ациклический граф.** Ориентированный граф, не содержащий никакого ориентированного цикла.
- **Вершина графа.** Либо конец какого-нибудь ребра графа, либо изолированная точка графа.
- **Гамильтонова линия.** Элементарный цикл, проходящий по всем вершинам графа.
- **Грань многоугольного графа G.** Часть плоскости, ограниченная каким-нибудь минимальным циклом из G или максимальным циклом C_1 графа G; в последнем случае это часть плоскости, лежащая вне C_1; ее называют также бесконечной гранью.
- **Граф.** Фигура, состоящая из точек (называемых вершинами) и отрезков, соединяющих некоторые из этих вершин. Соединяющие отрезки могут быть прямолинейными или криволинейными; она называются ребрами графа.
- **Граф G***, двойственный многоугольному графу G. Многоугольный граф, каждая вершина которого соответствует определенной грани графа G, а каждая грань – определенной вершине графа G. Две вершины графа G* соединены ребром в том и только в том случае, когда соответствующие грани графа G имеют общее ребро.
- **Двудольный граф.** Граф, вершины которого можно разделить на два непересекающихся множества так, что вершины одного и того же множества не соединены между собой ребрами.
- **Дерево.** Связный граф, не имеющий циклов.
- **Додекаэдр.** Многогранник, ограниченный 12 пятиугольными гранями.

- **Дополнение G графа G.** Граф G состоит из всех ребер (и их концов), которые необходимо добавить к G для того, чтобы получился полный граф.
- **Изолированная вершина.** Вершина, из которой не исходит ни одного ребра.
- **Изоморфные графы.** Графы G_1, и G_2 изоморфны, если между их вершинами можно установить такое взаимно однозначное соответствие, что пары вершин графа G_1 в том и только в том случае соединены ребром, когда соединены ребром соответствующие пары вершин графа G_2. В случае ориентированных графов это соответствие должно сохранять ориентацию ребер.
- **Икосаэдр.** Многогранник, ограниченный 20 треугольными гранями.
- **Инцидентность ребра и вершины.** Ребро называется инцидентным вершине, если она является одним из его концов.
- **Корень дерева.** Любая вершина, которую мы выбираем за начальную точку дерева.
- **Кратные ребра.** Если две вершины графа соединены более чем одним ребром, то каждое такое ребро называется кратным.
- **Лес.** Граф, все связные компоненты которого являются деревьями (граф без циклов).
- **Максимальный цикл C_1 многоугольного графа G.** Цикл, окружающий весь граф G.
- **Минимальный цикл многоугольного графа G**. Цикл, образованный граничными ребрами одного из многоугольников, составляющих G.
- **Многогранник.** Трехмерное тело, граница которого состоит из плоских многоугольников.
- **Многоугольный граф (многоугольная сеть).** Плоский граф, ребра которого образуют множество смежных, не налегающих друг на друга многоугольников.
- **Нечетная вершина.** Вершина, степень которой нечетна.
- **Нуль-граф.** Граф, состоящий только из изолированных вершин: граф, не имеющий ребер.
- **Обратный граф G* для данного ориентированного графа G**. Граф G* получается из G изменением направлений всех его. ребер.
- **Однородный граф степени r.** Граф, степени всех вершин которого одинаковы и равны r. (В случае ориентированных

графов требуется, чтобы степени ρ и ρ* были одинаковы во всех вершинах и равны друг другу.)
- **Октаэдр.** Многогранник, ограниченный восемью треугольными гранями.
- **Ориентированный граф.** Граф, на котором указаны направления всех его ребер,
- **Перешеек.** Другой термин для связывающего ребра.
- **Петля.** Ребро графа, оба конца которого совпадают.
- **Плоский граф.** Граф, который можно начертить на плоскости так, чтобы его ребра пересекались только в его вершинах.
- **Полный граф.** Граф, любые две вершины которого соединены ребром. Полный граф, у которого n вершин, имеет n(n–1)/2 ребер.
- **Правильный граф.** Многоугольный однородный граф, такой, что двойственный к нему граф G* тоже является однородным.
- **Правильный многогранник.** Многогранник, все грани которого являются равными правильными многоугольниками и в каждой вершине которого сходится одно и то же число ребер.
- **Ребро графа.** Кривая, соединяющая две вершины графа и не содержащая других вершин.
- **Связная компонента вершины A.** Все вершины графа, которые можно соединить с точкой A цепями, и все инцидентные им ребра.
- **Связный граф.** Граф, каждая вершина которого может быть соединена некоторой цепью с любой другой его вершиной.
- **Связывающее ребро.** Ребро, удаление которого приводит к увеличению числа связных компонент графа.
- **Смешанный граф.** Граф, на котором имеются как ориентированные, так и неориентированные ребра.
- **Степень ρ(A) вершины A.** Число ребер, сходящихся в вершине A. Для ориентированного графа ρ(A) означает число выходящих ребер, а ρ*(A) – число входящих ребер в вершине A; в этом случае имеются две степени.
- **Тетраэдр**. Многогранник, ограниченный четырьмя треугольными гранями.
- **Цепь.** Линия на графе, не проходящая ни по какому ребру более одного раза.
- **Цикл.** Замкнутая цепь.
- **Циклическое ребро.** Ребро, не являющееся связывающим.
- **Цикломатическое число графа G.** Число ребер графа G минус число его вершин плюс единица.

- **Четная вершина.** Вершина, степень которой четна.
- **Эйлеров граф.** Граф, содержащий эйлерову линию.
- **Эйлерова линия.** Цепь, проходящая по всем ребрам графа в точности по одному разу.
- **Элементарная цепь.** Цепь, не проходящая ни через одну из своих вершин более одного раза.
- **Элементарный цикл.** Цикл, не проходящий ни через одну из своих вершин более одного раза.

Кратко о тегах [3]

♦ Разрыв строки:

*Строка 1
Строка 2*

♦ Параграфы:

Строка 1<p>Строка 2<p align=right>Строка 3

♦ Выделенный текст:

Строка 1

♦ Наклонный шрифт:

Строка 1

♦ Подчеркнутый шрифт:

<u>Строка 1</u>

♦ Заголовки:

<h1>Строка 1</h1>
<h2 align=center>Строка 2</h2>
<h6 align=right>Строка 3</h6>

♦ Неупорядоченный список:

*
Строка 1
Строка 2
Строка 3
*

♦ Упорядоченный список:

*
Строка 1
Строка 2
Строка 3
*

♦ Ссылка:

Курс: «Основы дискретной математики»

♦ Закладка:

Закладка 1

♦ Графическое изображение:

<img src='../img/s12.gif' alt='Надпись' align='left' width='240' height='180'>

♦ Таблица:

<table align='center' rules='all' width=90% cellspacing=1 cellpadding=0 bgcolor='ffffcc'><tr><td colspan=2 align='center'>1</td></tr><tr><td align='left'>2</td><td align='right'>3</td></tr></table></table>

♦ Комментарии:

<!-- Комментарии -->

♦ Бегущая строка:

<marquee bgcolor='yellow' direction=left height=20 width=200 loop=20 behavior=slide>Текст сообщения</marquee>

♦ Разделительные линии:

<hr width=50%>
<hr width=100>
<hr width=15 size=15>
<hr width=15 size=15 noshade>

♦ Интервал между строками:

*Строка 1
Строка 2
Строка 3
*

♦ Высота символа:

Строка 1

♦ Интервал между буквами:

Слово

♦ Задание шрифта:

Строка 1

♦ Отступ в красной строке:

* Строка 1
Строка 2 *

♦ Поле ввода:

<input type='text|password|checkbox|radio|submit|reset|image'>

Дополнение 1. Диаграммы переходов состояний

Диаграмма переходов состояний является графической формой предоставления конечного автомата - математической абстракции,

используемой для моделирования детерминированного поведения технических объектов или объектов реального мира.

На этапе анализа требований и определения спецификаций диаграмма переходов состояний демонстрирует поведение разрабатываемой программной системы при получении управляющих воздействий. Под управляющими воздействиями или сигналами в данном случае понимают управляющую информацию, получаемую системой извне. Например, управляющими воздействиями считают команды пользователя и сигналы датчиков, подключенных к компьютерной системе. Получив такое управляющее воздействие, разрабатываемая система должна выполнить определенные действия и или остаться в том же состоянии, или перейти в другое состояние взаимодействия с внешней средой.

Для построения диаграммы переходов состояний необходимо в соответствии с теорией конечных автоматов определить: основные состояния, управляющие воздействия (или условия перехода), выполняемые действия и возможные варианты переходов из одного состояния в другое. Условные обозначения, используемые при построении диаграмм переходов состояний, показаны на рис. Д.1.1.

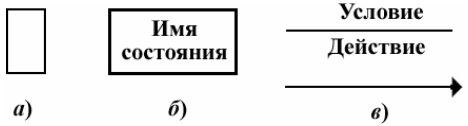


Рис. Д.1.1. Условные обозначения диаграмм переходов состояний: а – терминальное состояние; б – промежуточное состояние; в – переход

Если программная система в процессе функционирования активно не взаимодействует с окружающей средой (пользователем или датчиками), например, использует примитивный интерфейс и выполняет некоторые вычисления по заданным исходным данным, то диаграмма переходов состояний обычно интереса не представляет. В этом случае она демонстрирует только последовательно выполняемые переходы: из исходного состояния в состояние ввода данных, затем после выполнения вычислений – в состояние вывода и, наконец, в состояние завершения работы (рис. Д.1.2.).

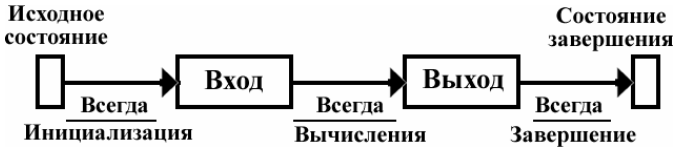


Рис. Д.1.2. Диаграмма переходов состояний программного обеспечения, активно не взаимодействующего с окружающей средой

Для интерактивного программного обеспечения с развитым пользовательским интерфейсом основные управляющие воздействия – команды пользователя, для программного обеспечения реального времени – сигналы от датчиков и/или оператора производственного процесса. Общим для этих типов программного обеспечения является наличие состояния ожидания, когда программное обеспечение приостанавливает работу до получения очередного управляющего воздействия. Для интерактивного программного обеспечения наиболее характерно получение команд различных типов (рис. Д.1.3), а если это еще и программное обеспечение реального времени – однотипных сигналов (либо от многих датчиков, либо требующих продолжительной обработки).

В отличие от интерактивных систем для систем реального времени обычно установлено более жесткое ограничение на время обработки полученного сигнала программного обеспечения. Такое ограничение часто требует выполнения дополнительных исследований поведения системы во времени, например, с использованием сетей Петри или Марковских процессов.

К программному обеспечению, требующему уточнения особенностей поведения посредством построения диаграммы переходов состояний, относится и программное обеспечение, ориентированное на работу в сети. При этом отдельно строят модели поведения сервера и клиента, представляя сообщения, передаваемые между ними, в виде управляющих воздействий.

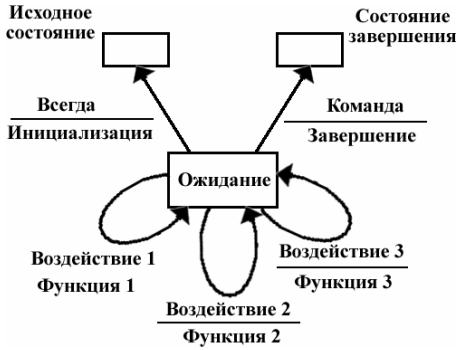


Рис. Д1.3. Пример диаграммы переходов состояний программного обеспечения, активно взаимодействующего с окружающей средой.

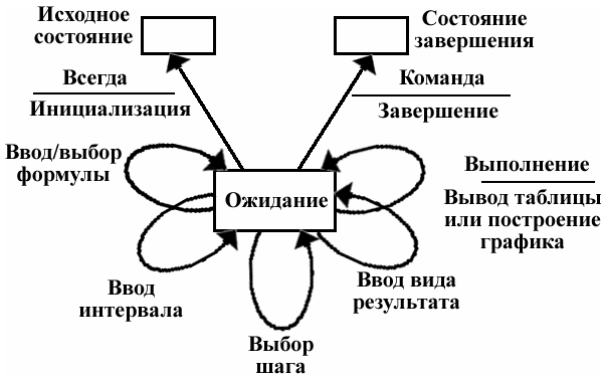


Рис. Д1.4. Диаграмма переходов состояний программы построения графиков (таблиц) функций

Пример. Рассмотрим диаграмму переходов состояний для программы построения графиков функций одной переменной. Программа относится к классу интерактивных, соответственно на этапе анализа и определения спецификаций целесообразно уточнить поведение программы на уровне интерфейса с пользователем, тем более, что наличие простого интерфейса оговорено в техническом задании. Один из возможных вариантов диаграммы переходов состояний программы представлен на рис. Д.1.4. Полученную диаграмму переходов состояний следует согласовать с заказчиком программного обеспечения.

Дополнение 2. Структурные карты Константайна

На структурной карте отношения между модулями представляют в виде графа, вершинам которого соответствуют модули и общие области данных, а дугам – межмодульные вызовы и обращения к общим областям данных.

Различают четыре типа вершин (рис. Д.2.1):

- модуль – подпрограмма,
- подсистема – программа,
- библиотека – совокупность подпрограмм, размещенных в отдельном модуле,
- область данных – специальным образом оформленная совокупность данных, к которой возможно обращение извне.

При этом отдельные части программной системы (программы, подпрограммы) могут вызываться последовательно, параллельно или как сопрограммы (рис. Д.2.2).

Чаще всего используют *последовательный* вызов, при котором модули, передав управление, ожидают завершения выполнения вызванной программы или подпрограммы, чтобы продолжить прерванную обработку.

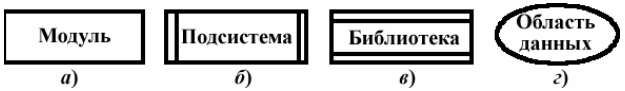


Рис. Д.2.1. Обозначения вершин по стандартам IBM, ISO и ANSI: *а* – модуль; *б* – подсистема; *в* – библиотека; *г* – область даннях

Под *параллельным* вызовом понимают распараллеливание вычислений на нескольких вычислителях, когда при активизации другого процесса данный процесс продолжает работу (рис. Д.2.3.а*)*. На однопроцессорных компьютерах в мультипрограммных средах в этом случае начинается попеременное выполнение соответствующих программ. Параллельные процессы бывают синхронные и асинхронные. Для синхронных процессов определяют точки синхронизации – моменты времени, когда производится обмен информацией.

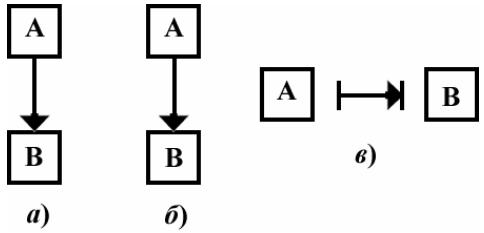


Рис. Д.2.2. Обозначения типа вызова: *а* – последовательный вызов; *б* – параллельный вызов; *в* – вызов сопрограммы между процессами

Асинхронные процессы обмениваются информацией только в момент активизации параллельного процесса. Под *вызовом сопрограммы* понимают возможность поочередного выполнения двух одновременно запущенных программ, например, если одна программа подготовила пакет данных для вывода, то вторая может ее вывести, а затем перейти в состояние ожидания следующего пакета. Причем в мультипрограммных системах основная программа, передав данные, продолжает работать, а не переходит в состояние ожидания, как изображено на рис. Д.2.3, *б.*

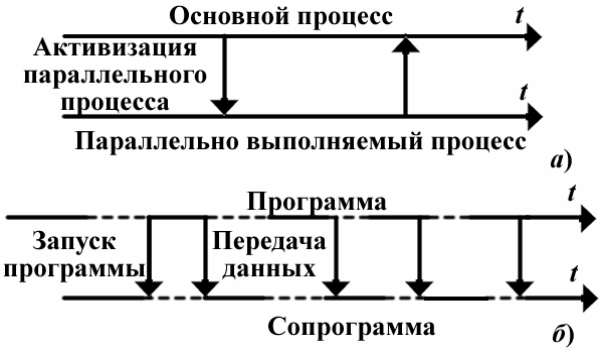


Рис. Д.2.3. Диаграммы реализации параллельного вызова *(а)* и вызова сопрограммы *(б):* выполнение; – – – – ожидание

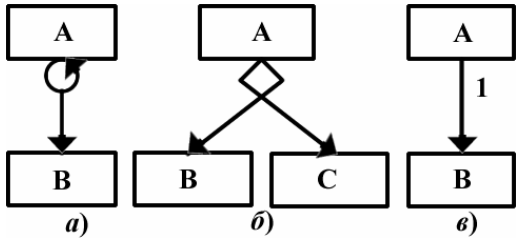


Рис. Д.2.4. Обозначения особых условий вызова: *а* – циклический; *б* – условный; *в* – однократный

Если стрелка, изображающая вызов, касается блока, то обращение происходит к модулю целиком, а если входит в блок, то – к элементу внутри модуля. При необходимости на структурной карте можно уточнить особые условия вызова (рис. Д.2.4): циклический вызов, условный вызов и однократный вызов – при повторном вызове основного модуля однократно вызываемый модуль не активизируется. Связи по данным и управлению обозначают стрелками, параллельными дуге вызова, направление стрелки указывает направление связи (рис. Д.2.5). Структурные карты Константайна позволяют наглядно представить результат декомпозиции программы на модули и оценить ее качество, т.е. соответствие рекомендациям структурного программирования (сцепление и связность).

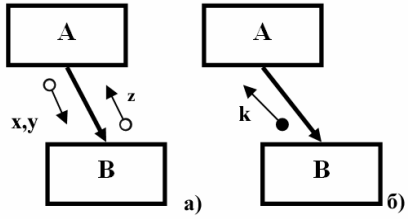


Рис. Д.2.5. Обозначение типа связи: *а* – по данным; *б* – по управлению

Для остальных подпрограмм имеют место особые условия вызова и типы связей (рис. Д.2.5.).

ОДЕРЖАНИЕ