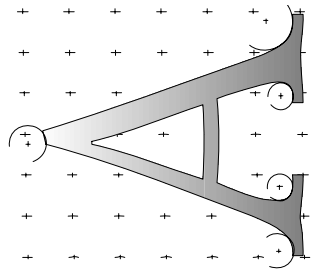
№ 16 (521)
16–31 августа 2006



ИНФОРМАТИК
Методическая газета для учителей информатики

«Жаркое лето-2006»
Олимпиады школьников по информатике

СОДЕРЖАНИЕ НОМЕРА

Заключительный этап XVIII Всероссийской олимпиады школьников по информатике

Председатель жюри олимпиады В.М. КИРЮХИН

Очередная, восемнадцатая по счету, Всероссийская олимпиада школьников по информатике завершилась в этом году в г. Кисловодске Ставропольского края, где в период с 21 по 27 апреля прошел заключительный этап олимпиады. Этому событию предшествовали школьные, муниципальные и региональные олимпиады школьников по информатике, а также федеральный окружной этап, который проводился только в Северо-Западном федеральном округе в г. Петрозаводске, Республика Карелия.

По результатам проведения первых четырех этапов олимпиады были отобраны самые лучшие школьники, победители и призеры региональных этапов и федерального окружного этапа олимпиады, которые и боролись за звание чемпиона страны по информатике. В общей сложности в заключительном этапе олимпиады приняли участие 198 школьников 7—11-х классов из 67 субъектов РФ. Если количество участников, как всегда, определялось техническими возможностями организаторов олимпиады, и именно исходя из этого определялись нормы представительства субъектов РФ, то увеличение числа субъектов РФ, представленных на заключительном этапе олимпиады, является отрадным фактом. По сравнению с прошлым годом, когда 58 субъектов РФ прислали своих представителей в г. Новосибирск, увеличение количества регионов в этом году является существенным и свидетельствует о том, что большая работа по вовлечению других регионов в олимпиадное движение по информатике, которая была проделана центральной методической комиссией по информатике за прошедший год, не прошла даром.

Как и в прошлые годы, олимпиада в г. Кисловодске проводилась в соответствии с Положением о Всероссийской олимпиаде школьников, утвержденным Приказом Министерства образования Российской Федерации от 30 октября 2003 г. № 4072. В связи с этим порядок организации и проведения заключительного этапа в этом году практически не изменился по сравнению с прошлым годом. Единственное существенное изменение коснулось только порядка формирования норм представительства субъектов РФ, которое было вызвано решением Центрального оргкомитета Всероссийской олимпиады школьников от 21 февраля 2006 г. № 1 об отмене бонусных мест субъектам РФ за одиннадцатиклассниками, ставшими победителями и призерами заключительного этапа олимпиады в прошлом году. Кроме того, было уменьшено до одного человека представительство тех субъектов РФ, число обучающихся в которых не превышает 50 тыс. человек.

С учетом этого факта квоты субъектов Российской Федерации определялись пропорционально общей численности обучающихся в субъекте РФ и с учетом результатов проведения федерального окружного этапа олимпиады в Северо-Западном федеральном округе. Кроме того, учитывалось также, что и московская городская олимпиада по информатике в соответствии с существующим Положением приравнена к федеральному окружному этапу. Сверх квоты персональные приглашения имели все победители и призеры заключительного этапа Всероссийской олимпиады по информатике прошлого года из числа учащихся невыпускных классов.

Самой многочисленной на прошедшей олимпиаде была команда Москвы — 14 участников. По 8 участников было представлено от Санкт-Петербурга, Московской области, а также от Республик Татарстан и Башкортостан. Такое представительство названных субъектов РФ в значительной степени определялось отличной работой с одаренными школьниками 7—9-х классов. Привлечение к участию в заключительном этапе олимпиады школьников, обучающихся в невыпускных классах, позволяет расширить представительство региона в будущем году в случае, если эти школьники станут призерами олимпиады. Если субъекты РФ заинтересованы в расширении своего представительства в будущем, они должны формировать состав своей команды как из лучших одиннадцатиклассников, так и лучших школьников невыпускных классов.

Непосредственным организатором заключительного этапа Всероссийской олимпиады школьников по информатике было Министерство образования Ставропольского края. Техническая и информационная поддержка была возложена на Северо-Кавказский государственный технический университет. Партнерами олимпиады выступили такие компании, как R-Style, “Кирилл и Мефодий”, РТКомм, National Instruments, Microsoft, Borland, и другие.

Ставропольский край не случайно был выбран местом проведения заключительного этапа Всероссийской олимпиады по информатике — ведь в крае активно ведется процесс информатизации системы образования. Не случайно и ответственность за техническую поддержку олимпиады легла на СевКавГТУ.

О вузе давно идет слава как о лидере в области информационных и коммуникационных технологий. На базе СевКавГТУ создан единственный в регионе Отраслевой центр технической поддержки функционирования сети станций спутниковой связи в образовательных учреждениях Южного федерального округа. Университет располагает развитой корпоративной информационной сетью, связывающей все его подразделения и филиалы. В рамках Межведомственной программы "Создание национальной сети компьютерных телекоммуникаций для науки и высшей школы" данный вуз был определен головной организацией по созданию регионального сегмента компьютерной телекоммуникационной сети системы образования и науки в крае.

Ставропольский край активно участвует также в реализации проекта "Информатизация системы образования" (ИСО), как один из семи пилотных регионов страны. Успешно идет здесь работа по созданию системы межшкольных методических центров, обеспечивающих внедрение современных образовательных и информационных технологий непосредственно во все учреждения образования края. В рамках этого проекта на СевКавГТУ возложено телекоммуникационное обеспечение функционирования системы межшкольных методических центров, и это говорит о высоком классе подготовки его специалистов.

В процессе подготовки к олимпиаде был тщательно изучен опыт проведения мероприятия такого масштаба в прошлом году в Новосибирске и заранее были сформированы все службы, способные обеспечить участников олимпиады всеми необходимыми ресурсами. Местом проведения олимпиады был определен санаторий Министерства обороны в г. Кисловодске, необходимое компьютерное оборудование было специально зарезервировано Министерством образования края из поставок ноутбуков по проекту ИСО, а сотрудники СевКавГТУ во главе с проректором по информационным технологиям и телекоммуникациям Г.В. Слюсаревым проделали большую работу по созданию необходимой технологической базы, соответствующей современному уровню проведения соревнования по информатике такого масштаба.

Как следствие этого, к началу олимпиады была создана сеть, обеспечивающая одновременную работу 250 рабочих станций; организованы внешние каналы связи с использованием современного телекоммуникационного оборудования, обеспечена информационная поддержка олимпиады на официальном сайте ***http://olymp.ncstu.ru*** в сети Интернет. Все это позволило во время проведения олимпиады использовать самые передовые информационные и телекоммуникационные технологии — беспроводный доступ, ip-телефонию, web-технологии.

Торжественно и красочно прошла церемония открытия заключительного этапа XVIII Всероссийской олимпиады школьников по информатике, а точнее, олимпиад — ведь параллельно с олимпиадой по информатике проводился и заключительный этап VII Всероссийской олимпиады по физической культуре. Было зачитано приветствие министра образования и науки Российской Федерации А.А. Фурсенко к участникам двух олимпиад. Тепло приветствовала всех собравшихся в зале министр образования Ставропольского края А.Ф. Золотухина, а проректор СевКавГТУ Г.В. Слюсарев отметил, что эта олимпиада к тому же являлась самой высотной — ведь участники находились почти на тысячеметровой высоте над уровнем моря. А закончилась церемония открытия самым настоящим... "шаропадом": на зрителей сверху посыпался ливень из разноцветных воздушных шариков.

Следуя давним традициям, нынешняя олимпиада проходила в два тура. На каждом туре для решения представленных участникам задач было отведено 5 часов. В распоряжение участников олимпиады были предоставлены рабочие места, оснащенные компьютерами RoverBook Voyager H591 на базе процессора Intel Celeron-M 1.6 ГГц с оперативной памятью 512 Мб, дополненными стандартной клавиатурой US/РУС и стандартной мышью.

На рабочем месте каждого участника олимпиады было инсталлировано следующее программное обеспечение:

- Windows XP Professional SP2;
- Far manager;
- Borland Pascal 7.0;
- Free Pascal 2.0.2;
- Borland Delphi 7.0;
- Borland C/C++ 3.1;
- MinGW 3.4.2 (GNU C/C++);
- Eclipse CDT 3.0.2;
- Microsoft Visual Studio 2003;
- MSDN January 2004;
- Java SDK .5.0_06.

Во время тура не разрешалось пользоваться личными компьютерами, калькуляторами, электронными записными книжками, средствами связи (пейджерами, мобильными телефонами и т.п.), принесенными электронными носителями информации (дискетами, CD- и DVD-дисками, модулями флэш-памяти и т.п.), а также учебной литературой и заготовленными личными записями.

В отведенное для каждого тура время требовалось решить три задачи. Решение каждой из них предполагало выполнение всех основных этапов решения задач с использованием компьютеров, начиная с формализации поставленной задачи и разработки наилучшего алгоритма ее решения и

кончая написанием и отладкой соответствующей программы на одном из допустимых языков программирования. Разные задачи можно было решать с использованием разных языков программирования.

Допускалось использование в решениях задач любых внешних модулей и заголовочных файлов, включенных в стандартную поставку соответствующего компилятора, так чтобы исходные тексты решений компилировались приведенными ниже командами.

Компилятор	Команда компиляции
Borland Pascal 7.0	bpc <имя файла>
Free Pascal 2.0.2	fpc <имя файла>
Borland C++ 3.1	bcc −ml <имя файла>
GNU C++ 3.4.2	g++ −O2 −x c++ <имя файла>
GNU C 3.4.2	gcc −O2 −x c <имя файла>
Borland Delphi 7.0	dcc32 −cc <имя файла>
Microsoft Visual C 7.0	cl /O2 <имя файла>
Microsoft Visual C++ 7.0	cl /O2 /GX <имя файла>
Microsoft Visual Basic	vbc <имя файла>
Java 5.0_06	javac <имя файла>

В решениях задач участников запрещалось:

- использовать расширенную и дополнительную память в программах на Borland Pascal и Borland C/C++;
- создавать каталоги и временные файлы при работе программы;
- осуществлять чтение и запись векторов прерываний;
- использовать сетевые средства;
- производить любые другие действия, нарушающие работу проверяющей системы.

Отбором и подготовкой задач для олимпиады занимались авторитетное жюри и научный комитет. В состав жюри олимпиады входили известные в стране ученые и педагоги: д. т. н., профессор В.А. Кузнецов (Петрозаводск), д. ф.-м. н., профессор В.В. Прохоров (Екатеринбург), к. п. н., доцент А.В. Алексеев (Ханты-Мансийск), к. ф.-м. н., доцент Е.В. Андреева (Москва), к. ф.-м. н., доцент В.Д. Лелюх (Нижний Новгород), к. ф.-м. н. А.В. Чернов (Москва), к. ф.-м. н., доцент Т.Г. Чурина (Новосибирск), Г.А. Корнеев (Санкт-Петербург), А.С. Лопатин (Санкт-Петербург), М.Р. Мирзаянов (Саратов), А.С. Станкевич (Санкт-Петербург), а также представители СевКавГТУ В.А. Галкина — декан факультета информационных систем и технологий, д. ф.-м. н., профессор, и В.А. Толпаев — заведующий кафедрой, д. ф.-м. н., доцент. Как и прежде, возглавлял работу жюри олимпиады к. т. н., доцент В.М. Кирюхин.

В состав научного комитета, по давно устоявшейся традиции, вошли студенты и аспиранты — победители школьных и студенческих международных олимпиад по информатике и программированию прошлых лет. Возглавлял работу научного комитета А.С. Станкевич.

При выборе олимпиадных задач жюри исходило из следующих соображений. С одной стороны, поскольку по статусу данная олимпиада является именно по информатике, а не по программированию, то задачи должны предоставлять школьникам возможность проявить свои способности на всех этапах решения задач с использованием компьютера, начиная с формализации и кончая тестированием и отладкой программы.

С другой стороны, задачи должны быть разумной сложности, и в процессе их решения не должны использоваться специальные знания. Сложность задач должна быть таковой, чтобы сильные школьники смогли продемонстрировать свои знания и умения в полной степени, а большая часть участников смогла бы получить хотя бы частные решения для представленных задач. Кроме того, нельзя было забывать, что уровень задач всероссийской олимпиады должен в той или иной степени соответствовать международному уровню, поскольку из состава участников заключительного этапа будет отобрана сборная команда России, которой в этом году предстоит защищать в Мексике честь нашей страны на международной олимпиаде.

Как и на прошлых всероссийских олимпиадах по информатике, проверка решений участников осуществлялась с использованием автоматической тестирующей системы. Во время тура в рамках тестирующей системы всем участникам была доступна возможность не только отправки своих решений на проверку, но и проверки работы своих программ на тестах из условия. Для многих участников это имело большое значение, так как позволяло выявить и исправить нелепые ошибки, которые часто допускаются по невнимательности, например, неверные имена входного или выходного файлов, неверный формат вывода, выход с ненулевым кодом возврата, незакрытие выходного файла и т.п. Также отсекались решения, которые сразу выделяли себе больший объем памяти, чем указано в условии задачи.

В течение каждого тура участники имели возможность посылать свои решения на проверку столько раз, сколько они считали нужным. Из всех решений одной и той же задачи, успешно прошедших предварительное тестирование в течение тура, к окончательной проверке принималось только решение, посланное последним. Если решение не проходило предварительное тестирование, то участники получали сообщение о типе допущенной ошибки.

Поскольку многие участники олимпиады впервые столкнулись с необходимостью взаимодействия с автоматической системой проверки своих решений, то сразу после открытия олимпиады был проведен пробный тур с целью предварительного ознакомления всех участников с порядком сдачи своих решений для тестирования во время соревнований. Для этих целей участникам также были предложены три задачи, которые носили тренировочный характер и больше были ориентированы на освоение технических особенностей используемого на олимпиаде программного обеспечения, чем на составление алгоритмов.

Проверка решений всех участников олимпиады осуществлялась сразу после окончания каждого тура с использованием автоматической системы проверки и специально разработанной жюри и научным комитетом олимпиады системы тестов. Система тестов для каждой задачи представляла собой набор входных файлов, которые по очереди подавались на вход проверяемой программы. Автоматическая система проверки анализировала в этом случае поведение программы для каждого теста (завершилась она нормально или же произошла ошибка во время ее исполнения, не были нарушены принятые ограничения по времени работы программы и объему занимаемой памяти) и в случае ее нормального завершения проверяла выданный ответ.

При составлении системы тестов жюри и научный комитет руководствовались следующими правилами:

- в состав тестов должны обязательно входить простейшие тесты, позволяющие поощрить участников, которые хоть немного продвинулись в решении задачи;
- тесты должны анализировать решения участников для всех частных случаев, и пропуск какого-либо случая обязательно должен штрафоваться;
- тесты должны проверять работу программы при максимально возможных значениях входных данных, указанных в тексте задачи, поскольку именно в этих случаях возникают переполнения типов данных, выход за пределы массива, превышение предельного объема используемой памяти и установленного времени работы программы;
- тесты должны дифференцировать решения участников с разной вычислительной сложностью, если такие случаи предполагаются жюри и научным комитетом.

Система оценки решений участников предусматривала начисление определенного количества баллов за каждый успешно прошедший тест. Общее количество тестов для каждой задачи и вес каждого теста в баллах определялись и утверждались жюри перед началом проверки решений участников. При этом учитывалось, что общее число баллов за каждую задачу в случае прохождения всех тестов не должно было превышать 100 баллов. Таким образом, максимально возможное число баллов, которое мог набрать каждый участник олимпиады на одном туре, составляло 300 баллов, а по итогам всего соревнования — 600 баллов.

После объявления результатов проверки решений второго тура и разбора всех олимпиадных задач каждый участник имел возможность подать апелляцию в жюри олимпиады. Поскольку процесс проверки решений был прекрасно организован, в чем немалая заслуга членов научного и технического комитетов, то апелляций практически не было. Те немногочисленные вопросы, которые и возникали по ходу соревнования, оперативно решались в процессе тестирования.

Общий анализ результатов проверки всех решений участников показал, что жюри очень квалифицированно подошло к формированию пакета олимпиадных задач. Все предложенные на олимпиаде задачи в целом оказались сбалансированными и по сложности, и по тематике. Отрадным является тот факт, что для всех шести задач получены полные решения, а процент нулевых оценок был очень низким.

В целом представленный на олимпиаде набор задач позволил проявить свои возможности как сильным, так и слабым участникам. Об этом также говорят общие показатели решения каждой из шести задач, которые приведены в следующей таблице.

Название задачи	Средний балл	Максимальный балл	Количество максимальных баллов	Количество нулевых баллов
“Банковские карты” (№ 1)	35,7	100	29	19
“Файловый менеджер” (№ 2)	19,8	100	9	66
“Приключение” (№ 3)	23,1	100	3	22
“Автобусы” (№ 4)	26,6	100	15	24
“Полимино” (№ 5)	16,6	100	1	48
“Треугольная реформа” (№ 6)	13,4	100	2	78

Оценивая результаты проверки каждой из задач прошедшей олимпиады, можно сделать следующие выводы. Наилучшие результаты были показаны участниками при решении задачи № 1 "Банковские карты". Жюри специально предложило эту задачу на первом туре, чтобы дать возможность как можно большему количеству школьников показать достойные результаты, и, как оказалось, не ошиблось. В итоге средняя оценка этой задачи составила около 36 баллов; по сравнению с другими эту задачу решило максимальное количество участников, и только 19 участников (около 9%) получили нулевую оценку.

Второй по показанным результатам оказалась задача второго тура "Автобусы". Ее полностью решили 15 участников, что является вторым результатом среди всех задач, а нулевые результаты показали 24 участника, т.е. около 12%. Хотя по количеству нулевых оценок эта задача превосходит задачу № 3 "Приключение", но по среднему баллу ее показатели лучше. Кстати, жюри долго обсуждало задачи второго тура. Рассматривались варианты и с более сложными задачами, но в конце концов было принято решение дать именно эту задачу, как наиболее доступную для нынешнего состава участников.

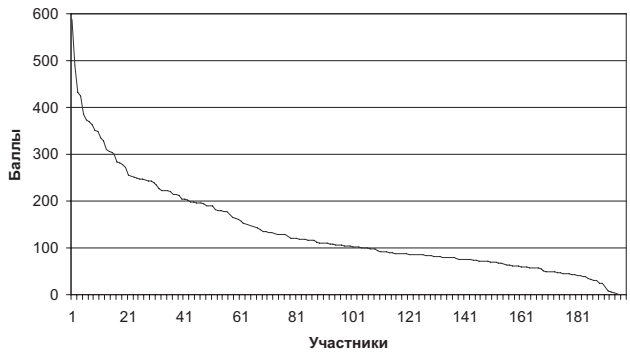
Результаты проверки решений задачи № 3 "Приключение" в среднем немного уступают задаче № 4 "Автобусы". Ее средний балл оказался чуть меньше — около 23, однако число нулевых оценок составляет всего 22, что является вторым результатом среди всех задач. К сожалению, несмотря на хороший средний балл, полностью эту задачу решили только три участника.

Следующими по показанным результатам стали задача № 2 "Файловый менеджер" и задача № 5 "Полимино". В среднем задачу № 2 участники олимпиады решили лучше, так как у нее средний балл составляет около 20, а у задачи № 5 — около 17, однако по предельным показателям задача № 2 выглядит предпочтительней. Так, полностью решили задачу № 2 девять человек, а задачу № 5 полностью решил только один участник — Банников Андрей из Республики Башкортостан. Зато по количеству нулевых оценок задача № 2 проигрывает задаче № 5. Число участников, получивших нулевые оценки у задачи № 2, составляет 66 (около 33%), а у задачи № 5 — 48 (около 24%).

Наиболее трудной для участников олимпиады, как и предполагало жюри, оказалась задача № 6 "Треугольная реформа". Средний балл за нее составляет 13,4%. Жюри специально выбрало эту геометрическую задачу, чтобы дать сильным школьникам посоревноваться между собой, и эти надежды оправдались. Два участника получили за эту задачу полный балл, однако и количество нулевых оценок было самым большим — 78. Следует отметить, что многие школьники не сдали свои решения этой задачи на проверку, а может быть, даже и не пытались ее решать и сконцентрировались на двух других задачах на втором туре, решив, что именно они дадут им необходимые для удачного выступления баллы.

Если сравнивать результаты решения задач по турам, то приведенные выше показатели говорят о том, что второй тур получился для участников чуть сложнее. Так, максимальный результат за первый тур составил 296 баллов, а за второй тур — 292 балла. В обоих случаях лучший результат показал Денисов Денис из Республики Карелия. Аналогичная ситуация и с нулевыми оценками за тур. На первом туре их было 8, а на втором — 9. Вообще же по результатам двух туров нулевых оценок было всего 3, что является хорошим показателем для олимпиады такого уровня.

Об общем уровне подготовки участников олимпиады и о сложности олимпиадных задач можно судить по распределению количества набранных баллов по участникам. На рисунке представлено такое распределение, полученное по итоговым результатам.



Итоговое распределение участников по баллам

По итогам соревнований абсолютным чемпионом нынешней олимпиады с большим отрывом от преследователей стал **Денисов Денис**, ученик 11-го класса гимназии № 37 г. Петрозаводска, Республика Карелия. В прошлом году он был четвертым, а в этом году он стал лучшим и на федеральном окружном этапе олимпиады, проходившем в марте этого года в Петрозаводске.

Второе место занял **Копелиович Сергей**, учащийся 11-го класса физико-математического лицея № 30 г. Санкт-Петербурга. Как и победитель олимпиады, он за прошедший год также достиг больших успехов и сумел диплом второй степени, полученный на прошлой олимпиаде, обменять на диплом первой степени, завоевав при этом почетное второе место. Сергей является и обладателем золотой медали международной олимпиады по информатике 2005 года.

Третий результат на олимпиаде показал **Климовский Арсений**, учащийся 11-го класса специализированного учебно-научного центра Московского государственного университета им. М.В. Ломоносова. Он упорно шел к этому результату. Как и Сергей Копелиович, в прошлом году он завоевал диплом второй степени, весь год занимался информатикой, и результаты оказались налицо.

Важным положительным итогом прошедшей олимпиады является открытие новых имен. Большое количество участников являлись учащимися невыпускных классов, многие из которых показали совсем неплохие результаты. Так, два ученика 9-го класса — Лернер Эмиль из Республики Татарстан и Кумок Аким из Московской области, а также пять школьников из 10-х классов были награждены дипломами первой степени, десять десятиклассников — дипломами второй степени, а один восьмиклассник, три девятиклассника и 12 десятиклассников — дипломами третьей степени. Заметим, что они все участвовали в общем конкурсе с одиннадцатиклассниками. Самым молодым на олимпиаде был учащийся 7-го класса Булычев Антон из Ленинградской области. Интересно также, что Епифанов Владислав из Нижнего Новгорода, несмотря на то что он еще учится в 8-м классе, уже в четвертый раз принимал участие в соревнованиях такого уровня (на этой олимпиаде он завоевал диплом третьей степени). В первый раз он принял участие во всероссийской олимпиаде по информатике в 2002 году в Перми, когда учился еще в третьем классе.

В общей сложности победителям и призерам олимпиады было вручено 84 диплома, из них 15 дипломов первой степени, 27 дипломов второй степени и 42 диплома третьей степени. Итоговые результаты победителей и призеров олимпиады представлены в таблице на с. 9—11, а результаты выступления всех участников можно найти на портале всероссийских олимпиад *http://rusolymp.ru/* и сайте олимпиады *http://olymp.ncstu.ru/*.

Не менее торжественно и красочно, чем открытие, прошло закрытие заключительного этапа XVIII Всероссийской олимпиады школьников по информатике. Первой с успешным окончанием олимпиады поздравила всех присутствующих в зале министр образования Ставропольского края А.Ф. Золотухина. Представитель Министерства образования и науки РФ Т.В. Бешененко отметила высокий уровень проведения заключительного этапа олимпиады по информатике и пожелала всем успехов в реализации своих творческих возможностей. Завершилось закрытие олимпиады награждением чемпиона страны этого года Денисова Дениса. Под бурные овации зала министр образования Ставропольского края А.Ф. Золотухина и председатель жюри В.М. Кирюхин вручили ему главный приз — ноутбук, предоставленный главным спонсором олимпиады — компанией R-Style, и золотую медаль победителя.

Все победители и призеры олимпиады были награждены почетными дипломами, медалями и призами от спонсоров. Специальный приз, как лучший участник из Ставропольского края, получил учащийся 10-го класса лицея № 14 из г. Ставрополь Артем Шурупов.

Несколько дней олимпиады для ее участников пролетели незаметно. Они нашли единомышленников и друзей, познакомились с прекрасным городом-курортом Кисловодском. Каждый на память об интеллектуальном соревновании увез компакт-диск с фотографиями, задачами и другой информацией об олимпиаде.

Олимпиада стала настоящей проверкой на профессионализм, как для юных участников, так и для организаторов. И эту проверку как те, так и другие выдержали достойно. Олимпиада была проведена на самом высоком уровне и соответствовала всем требованиям к соревнованиям такого уровня, причем не только национальным, но и международным. В этом бесспорная заслуга всех ее организаторов и участников.

Много интересных событий подарила нам нынешняя олимпиада. В плане демонстрации возможностей современных информационных технологий и их интеграции в образование, и особенно при работе с одаренными школьниками, она не имеет аналогов.

Так, во время проведения заключительного этапа олимпиады в Кисловодске впервые любой школьник России, и не только России, смог в режиме реального времени принять участие в интернет-турах Всероссийской олимпиады и оценить свой потенциал — ведь предлагались те же самые задачи, что и участникам олимпиады, и на их решение отводилось такое же количество времени. Этой возможностью воспользовались 714 человек из 42 субъектов РФ и ближнего зарубежья, причем больше всего было представителей Ставропольского края.

Аналогично, как и на самой олимпиаде, проходила проверка решений участников интернет-туров, при этом на портале СевКавГТУ для этих целей впервые использовалась система автоматической проверки задач по информатике нового поколения, разработанная при поддержке компании "Кирилл и Мефодий". А победителем интернет-олимпиады, набрав 257 баллов из 600 возможных, стал Рудоль Дмитрий, учащийся 10-го класса из Республики Беларусь.

Также каждый желающий мог на сайте олимпиады следить за ходом проведения самой олимпиады в режиме реального времени. Для этого использовалась новая разработка научно-производственного центра "Видикор" — система "VidicoR", созданная на основе передовых научных исследований Института математики и механики УрО РАН под руко-

водством члена жюри олимпиады д. ф.-м. н., профессора В.В. Прохорова. По комплексу характеристик эта система находится на уровне лучших мировых образцов по каждой из областей применения (видеотелефония, видеоконференцсвязь, передача телевизионного трафика, IP-телевидение и др.).

Всероссийская олимпиада по информатике 2006 года закончилась, но наших лучших школьников в этом году ждут новые испытания. Многие победители и призеры прошедшей олимпиады приглашены для участия в летних учебно-тренировочных сборах с 22 июня по 3 июля в г. Малоярославце Калужской области, по результатам которых и решится вопрос, кто поедет с 13 по 20 августа 2006 года в Мексику защищать честь нашей страны на XVIII международной олимпиаде по информатике. Хотелось бы пожелать тем четырем лучшим, кто будет включен в состав национальной команды, удачно выступить в Мексике и завоевать как можно больше медалей самого высокого достоинства!

В заключение хотелось бы рассказать всем читателям газеты "Информатика", что в мае 2006 г. начал функционировать федеральный портал Всероссийских олимпиад школьников ***rusolymp.ru***. На этом портале будет размещена полная информация обо всех проводимых в стране олимпиадах школьников, включая нормативно-справочные документы, тексты олимпиадных задач и их решения, методические рекомендации по подготовке к олимпиадам и решению олимпиадных задач и многое другое. Сейчас идет информационное наполнение сайтов олимпиад, имеющих международное продолжение (информатика, физика, математика, химия, биология и география). К концу года планируется представить на этом портале Всероссийские олимпиады школьников практически по всем предметам.

ПОБЕДИТЕЛИ И ПРИЗЕРЫ ЗАКЛЮЧИТЕЛЬНОГО ЭТАПА XVIII ВСЕРОССИЙСКОЙ ОЛИМПИАДЫ ШКОЛЬНИКОВ ПО ИНФОРМАТИКЕ

Место	Ф.И.О. участника	Субъект РФ	Класс	Сумма баллов	Степень диплома
1	Денисов Денис Валерьевич	Республика Карелия	11	588	I
2	Копелиович Сергей Владимирович	Санкт-Петербург	11	490	I
3	Климовский Арсений Андреевич	Москва	11	433	I
4	Банников Андрей Владимирович	Республика Башкортостан	11	424	I
5	Колосов Андрей Дмитриевич	Москва	10	386	I
6	Лернер Эмиль Эдуардович	Республика Татарстан	9	374	I
7	Фонарев Антон Вячеславович	Москва	11	369	I
8	Корнаков Илья Владимирович	Москва	11	363	I
9	Кумок Аким Сергеевич	Московская область	9	351	I
10	Разенштейн Илья Петрович	Нижегородская область	10	348	I
11	Петров Юрий Сергеевич	Санкт-Петербург	10	334	I
12	Богатый Иван Семеонович	Москва	10	329	I
13	Давыдов Олег Сергеевич	Санкт-Петербург	10	310	I
14	Грибовский Сергей Владимирович	Москва	11	307	I
15	Акиньшин Андрей Александрович	Алтайский край	11	305	I
16	Махлин Антон Юрьевич	Москва	10	299	II
17	Исенбаев Владислав Вольдемарович	Свердловская область	11	283	II
18	Гатилов Степан Юрьевич	Новосибирская область	11	281	II
19	Горбунов Валентин Камилевич	Республика Татарстан	11	277	II
20	Фоминых Федор Михайлович	Свердловская область	11	272	II

Место	Ф.И.О. участника	Субъект РФ	Класс	Сумма баллов	Степень диплома
21	Кошелев Владимир Константинович	Алтайский край	10	255	II
22	Алюшин Виктор Михайлович	Москва	11	253	II
23	Шиловский Сергей Сергеевич	Вологодская область	11	252	II
24	Крюк Дмитрий Сергеевич	Омская область	10	250	II
25	Викулов Андрей Викторович	Москва	11	246	II
25	Долгих Евгений Алексеевич	Пермский край	11	246	II
27	Смирнов Арсений Юрьевич	Санкт-Петербург	10	244	II
28	Феськов Антон Геннадьевич	Санкт-Петербург	10	242	II
28	Ахметзянов Дамир Искандерович	Республика Башкортостан	11	242	II
30	Еньков Алексей Сергеевич	Республика Татарстан	11	239	II
31	Батаев Владимир Андреевич	Москва	11	235	II
32	Новиков Антон Александрович	Пермский край	11	227	II
33	Ефремов Алексей Владимирович	Кировская область	10	223	II
33	Авдеев Вадим Александрович	Москва	11	223	II
35	Локоть Андрей Сергеевич	Ростовская область	11	222	II
36	Новиков Константин Викторович	Москва	10	221	II
37	Майоров Павел Олегович	Челябинская область	10	215	II
38	Кузькоков Владислав Евгеньевич	Новосибирская область	11	214	II
39	Киселев Павел Алексеевич	Московская область	10	213	II
40	Кочуров Александр Валерьевич	Самарская область	11	205	II
41	Гренкин Глеб Владимирович	Приморский край	10	204	II
42	Семенко Алексей Евгеньевич	Новосибирская область	11	202	II
43	Трефилов Александр Петрович	Саратовская область	11	197	III
43	Шаповалов Максим Алексеевич	Алтайский край	11	197	III
45	Булычев Иван Дмитриевич	Ленинградская область	11	196	III
45	Третьяк Елена Викторовна	Московская область	11	196	III
47	Поромов Сергей Сергеевич	Санкт-Петербург	10	195	III
48	Сергушичев Алексей Александрович	Вологодская область	10	193	III
49	Василенко Олег Сергеевич	Челябинская область	10	189	III
49	Ульянов Николай Николаевич	Москва	10	189	III
49	Надточий Александра Михайловна	Самарская область	10	189	III
52	Ермишин Федор Дмитриевич	Саратовская область	10	182	III

Место	Ф.И.О. участника	Субъект РФ	Класс	Сумма баллов	Степень диплома
53	Семипятный Владислав Константинович	Московская область	10	180	III
53	Николаев Кирилл Викторович	Курская область	10	180	III
55	Васильев Дмитрий Сергеевич	Республика Башкортостан	11	178	III
56	Бикинеев Артем Олегович	Тульская область	11	177	III
57	Высоцкий Петр Иванович	Республика Саха (Якутия)	11	172	III
58	Трушников Михаил Андреевич	Пермский край	11	165	III
59	Скорынин Павел Александрович	Челябинская область	11	163	III
60	Андреев Геннадий Владимирович	Иркутская область	11	162	III
61	Родиков Денис Евгеньевич	Смоленская область	11	157	III
62	Торопов Александр Владимирович	Свердловская область	11	154	III
63	Трусфус Михаил Валерьевич	Республика Татарстан	11	152	III
64	Михальков Олег Валентинович	Новосибирская область	11	150	III
65	Палевич Владимир Александрович	Удмуртская Республика	11	146	III
66	Мельников Сергей Вячеславович	Вологодская область	9	145	III
67	Дручинин Виктор Валерьевич	Оренбургская область	11	143	III
68	Трифонов Дмитрий Сергеевич	Московская область	11	138	III
69	Прозоров Илья Сергеевич	Удмуртская Республика	11	135	III
70	Ющенко Павел Александрович	Московская область	10	134	III
71	Епифанов Владислав Юрьевич	Нижегородская область	8	133	III
72	Храпов Вениамин Игоревич	Саратовская область	11	132	III
73	Судариков Игорь Валерьевич	Краснодарский край	11	131	III
74	Петров Александр Владимирович	Чувашская Республика	11	129	III
75	Кузнецов Павел Михайлович	Красноярский край	11	128	III
75	Фондаратов Валентин Сергеевич	Санкт-Петербург	9	128	III
75	Борисов Никита Олегович	Орловская область	11	128	III
78	Боник Григорий Борисович	Московская область	11	125	III
79	Анищенко Александр Михайлович	Воронежская область	11	121	III
80	Матвеякин Андрей Михайлович	Краснодарский край	10	120	III
80	Колобаев Вячеслав Владимирович	Челябинская область	10	120	III
82	Корниенко Дмитрий Александрович	Республика Татарстан	9	119	III
82	Второв Александр Александрович	Калужская область	11	119	III
82	Рисков Георгий Константинович	Тульская область	10	119	III

Задачи XVIII Всероссийской олимпиады школьников по информатике

Разбор задач подготовлен научным комитетом олимпиады под руководством А.С. СТАНКЕВИЧА

Все задачи олимпиады тестировались при следующих ограничениях:

Максимальное время работы на одном тесте:	2 *секунды*
Максимальный объем используемой памяти:	64 *мегабайта*

Задача 1. Банковские карты

Автор задачи — Г.А. Корнеев, оригинальная идея С.Г. Волченкова

Банк "Кисловодск" переходит на новый вид банковских карт. Для этого производятся одинаковые заготовки, на которых есть специальное место для идентификации клиента. Изначально на этом месте записывается кодовое число X. В банке с помощью специального прибора можно стирать некоторые цифры числа X. Оставшиеся цифры, будучи записанными подряд, должны образовывать номер счета клиента. Например, при $X = 12013456789$ номера счетов 5, 12, 17 или 12013456789 получить можно, а номера 22 или 71 получить нельзя.

Способ распределения номеров счетов в банке очень прост. Счетам присваиваются последовательно номера 1, 2, ... Очевидно, что при таком способе в какой-то момент впервые найдется номер счета N, который нельзя будет получить из цифр X указанным выше способом. Руководство банка хочет знать значение N.

Напишите программу, которая находила бы N по заданному X.

Формат входных данных

Во входном файле задано натуральное число X без ведущих нулей ($1 \le X < 10^{1000}$).

Формат выходных данных

В выходном файле должно содержаться искомое N без ведущих нулей.

Примеры

cards.in	cards.out
239	1
12013456789	22

Решение

Задачу можно переформулировать так: найти минимальное натуральное число N, которое нельзя получить из заданного натурального числа X вычеркиванием некоторых цифр.

Основным понятием, помогающим нам понять решение этой задачи, будет понятие блока. *Блок* — это отрезок цифр числа X, содержащий все цифры от 0 до 9. Ясно, скажем, что если число X можно разбить на 5 блоков, то из него можно вычеркиванием цифр получить любое пятизначное число. Оказывается, с небольшими оговорками верно и обратное.

Напомним, что *суффиксом* строки называется ее отрезок с некоторого места и до конца. Например, суффиксами строки 12345 являются строки 12345, 2345, 345, 45 и 5.

Итак, выделим в числе X минимальный (самый короткий) блок-суффикс. То есть возьмем несколько (как можно меньше) последних цифр X, образующих блок. Например, если $X = 12013456789$, то в этот блок войдут цифры 2013456789, а если $X = 112233445566778899001122334455667788990 0$, то в блок войдут цифры 1223344556677889900. Повторим эту операцию для оставшихся цифр X, потом еще раз, и так далее, пока это возможно. Число X окажется представлено в виде $S_0B_1B_2B_3...B_L$, где B_i — блоки, а S_0 — *неполный блок*, то есть некоторый (возможно, пустой) отрезок числа X, содержащий не все цифры от 0 до 9. Кроме того, по построению блоков B_i они удовлетворяют следующему условию: никакой суффикс блока B_i (не совпадающий со всем B_i) не является блоком, то есть все суффиксы B_i являются неполными блоками.

Теперь перейдем к решению нашей задачи. Пусть d_0 — это минимальная из цифр от 1 до 9, не встречающаяся в S_0. Пусть S_1 — это суффикс блока B_1 после первого вхождения в него цифры d_0. Пусть d_1 — это минимальная из цифр от 0 до 9, не встречающаяся в S_1. И так далее, последовательно строим S_2, d_2, ..., S_L, d_L. Тогда ответом является число $N = d_0d_1...d_L$.

Приведем пример.

Пусть $X = 5129312564339071801234567 89$. Тогда $B_2 = 0123456789$, $B_1 = 25643390718$, $S_0 = 512931$. Получаем $d_0 = 4$, значит, $S_1 = 3390718$, значит, $d_1 = 2$, значит, $S_2 = 3456789$, значит, $d_2 = 0$. Ответ $N = 420$.

Почему же этот метод всегда работает верно? Во-первых, на каждом шаге очередное d_i можно выб-

рать, так как S_i — это неполный блок (для S_0 это верно по построению, а S_i — это суффикс B_i, а значит, неполный блок). Во-вторых, на каждом шаге очередное S_i можно выбрать, так как B_i — это блок. Тем самым, метод всегда завершает работу. Осталось проверить, что найденный им ответ — правильный.

Проверим, что число N нельзя получить из числа X. Действительно, первая цифра числа N (d_0) отсутствует в S_0, значит, ее придется брать из B_1 или еще правее. Но тогда вторую цифру числа N (d_1) придется брать из B_2 или правее, и так далее, и цифру d_L брать неоткуда.

Теперь докажем, что любое меньшее число можно получить. Пусть $M < N$. Если в M меньше цифр, чем в N, то его можно получить, взяв первую цифру из B_1, вторую из B_2, и так далее. Пусть в M столько же цифр, сколько в N: $M = e_0e_1...e_L$. Пусть k — номер первой цифры, где M и N различаются: $e_0 = d_0$, $e_1 = d_1$, ..., $e_{k-1} = d_{k-1}$. Тогда из того, что $M < N$, получаем $e_k < d_k$. Выберем цифру e_0 из B_1, цифру e_1 из B_2, и так далее, цифру e_{k-1} из B_k. Тогда по построению цифры d_k все меньшие цифры можно найти в B_k правее цифры d_{k-1}, а значит, мы сможем взять цифру e_k из B_k. Далее берем цифру e_{k+1} из B_{k+1} и так далее, и получаем число M.

В решении остался неразобранным один случай: когда в S_0 встречаются все цифры от 1 до 9 (но не встречается цифра 0, иначе этот блок был бы полон). Оказывается, тогда необходимо взять $d_0 = 0$, но так как число не может начинаться с 0, необходимо дописать к нему слева цифру 1. Например, если $X = S_0 = 123456789$, то получаем $N = 10$. Доказательство правильности этого приема оставим в качестве упражнения.

Время работы этого решения линейно по длине X. Программа получается довольно простая и не требует знания каких-либо сложных алгоритмов или конструкций языка.

В этой задаче возможны и другие решения, например, можно применить бинарный поиск по ответу и научиться проверять, все ли числа, меньшие N, можно получить из X, однако реализовать такие решения гораздо сложнее, и они имеют большее (хотя и удовлетворяющее ограничениям в условии задачи) время работы.

Задача 2. Файловый менеджер

Автор задачи — М.Р. Мирзаянов

Петя работает над очень большим проектом. Проект содержит N файлов. В процессе работы Пете часто приходится просматривать и редактировать файлы. Для ускорения работы Петя использует файловый менеджер Fur Manager, который отображает список имен файлов проекта в некотором порядке.

В текущей версии Fur Manager'а для перемещения по списку имен файлов есть следующие возможности:

1) можно нажать клавишу ↓, при этом курсор перемещается на следующий файл в списке, для последнего файла следующим считается первый;

2) можно нажать клавишу ↑, при этом курсор перемещается на предыдущий файл в списке, для первого файла предыдущим считается последний;

3) можно нажать клавишу Alt и, удерживая ее, набрать последовательность латинских букв. После этого клавишу Alt следует отпустить, и в этот момент курсор переместится на ближайший файл, имя которого начинается с заданной последовательности символов. Ближайший файл — это файл, на который можно переместиться за наименьшее количество нажатий клавиши ↓. Если заданная последовательность является началом имени текущего файла, или файла, имя которого начинается с этой последовательности, не существует, то курсор останется на месте.

Первая и вторая из описанных возможностей файлового менеджера требуют по одному нажатию клавиши, а третья — одного нажатия (нажатие клавиши Alt) плюс количество нажатий, равное длине набранной последовательности латинских букв.

Файлы пронумерованы от 1 до N в порядке их следования. После загрузки Fur Manager'а курсор находится на первом файле.

Петя знает, что ему сначала придется редактировать файл с номером a_1, затем с номером a_2 и так далее, а последним — файл с номером a_k. В последовательности a_1, a_2, ..., a_k один и тот же номер может встречаться несколько раз. При каждом перемещении от одного файла к другому Петя хочет нажимать как можно меньше клавиш.

Требуется написать программу, которая выдает искомую последовательность нажатий клавиш.

Формат входных данных

В первой строке входного файла записано целое число N ($1 \le N \le 1000$) — количество файлов в проекте.

В следующих N строках записаны имена файлов, по одному в каждой строке. Файлы перечислены в том порядке, в котором они отображаются файловым менеджером. Имена состоят только из строчных латинских букв. Длина каждого имени не превосходит 2000 символов. Все имена файлов различны.

Далее в следующей строке записано целое число k ($1 \le k \le 10$).

Последняя строка входного файла содержит k целых чисел a_1, a_2, ..., a_k ($1 \le a_i \le N$) — номера

редактируемых файлов. Редактирование файлов выполняется в том порядке, в котором они встречаются в последовательности a_1, a_2, ..., a_k.

Формат выходных данных

Выходной файл должен содержать описание искомой последовательности нажатий клавиш в виде k блоков информации:

- первый блок информации описывает перемещение от файла с номером 1 к файлу с номером a_1;
- второй блок информации описывает перемещение от файла с номером a_1 к файлу с номером a_2;
- ...
- k-й блок информации описывает перемещение от файла с номером a_{k-1} к файлу с номером a_k.

Каждый блок информации выглядит следующим образом.

В первой строке блока записывается число L — наименьшее количество нажатий клавиш, необходимое для перемещения от очередного файла последовательности к следующему.

Следующие L строк блока описывают нажимаемые клавиши. Каждая из строк содержит описание одной клавиши:

- если нажимается клавиша ↓, то в строке записывается слово `down`;
- если нажимается клавиша ↑, то в строке записывается слово `up`;
- если нажимается клавиша Alt, то в строке записывается слово `Alt`;
- при нажатии клавиши с латинской буквой выводится соответствующая ей латинская буква.

Если существует несколько оптимальных способов перемещения, то требуется вывести любой из них.

Примеры

fur.in	fur.out
6 submit monitor monitorx monyator subversion sub 5 6 3 3 5 2	1 up 3 Alt m down 0 2 down down 2 Alt m

fur.in	fur.out
8 abc abv abba auto test auvto ioi olympiad 2 4 6	3 Alt a u 2 down down

Решение

Будем называть имена файлов строками.

Рассмотрим полный ориентированный граф с N вершинами, где вершина i соответствует файлу номер i. Вес ребра (i, j) — минимальное число нажатий, с помощью которых можно добраться из позиции курсора номер i в позицию j, используя только одну возможность менеджера (нажать клавишу ↑, нажать клавишу ↓ или нажать клавишу Alt и последовательность символов). Ребра, реализующие первые две возможности (*up*, *down*), соединяют все вершины, номера которых отличаются на 1. Вес таких ребер равен 1.

Для остальных пар ребер, если мы можем попасть из позиции i в позицию j с помощью Alt + + <Текст>, то вес такого ребра (i, j) — минимальная длина последовательности нажатий клавиш, которая переводит курсор с i-го файла на j-й (длина <Текста> + 1). Если из i в j нельзя попасть только с помощью однократного нажатия Alt вместе с некоторым текстом, то вес ребра будет $+\infty$.

Таким образом, чтобы найти последовательность нажатий минимальной длины, перемещающую курсор из позиции A в позицию B, надо найти кратчайший путь из A в B в нашем графе, например, с помощью алгоритма Дийкстры. А общая схема решения задачи в целом — k раз находить кратчайшие пути из одной вершины в другую на предварительно построенном графе.

Самая главная сложность этой задачи состоит в том, чтобы быстро найти веса ребер для последней возможности (Alt + <Текст>).

Пусть мы знаем, как быстро вычислять массив $common[i][j] =$ <максимальный общий префикс строк номер i и j>. Тогда $d[i][j]$ — вес ребра (i, j) — равен максимуму среди $(common[t][j] + 2)$ $i \le t < j$ (если $i > j$, то t пробегает значения $i, i+1, ...N, 1, ..., j-1$). Это верно, так как надо нажать сначала Alt, потом

$max(common[t][j])$ букв, потом еще одну. Если нажать менее чем $d[i][j]$ клавиш, то курсор попадет на какую-нибудь строку между i и j. Если $d[i][j]$ получилось больше, чем длина j-го слова + 1, то из i в j нельзя попасть только с помощью одного нажатия Alt, $d[i][j]$ присваиваем $+\infty$.

Например, для набора

```
aaa
baa
aa
bbbb
bbcc
```

$d[1][5] = 4$, $d[1][3] = +\infty$.

Вычисление матрицы d по известной матрице *common* несложно реализовать за $O(N^2)$ следующим образом:

$d[j-1][j] = common[j-1][j] + 2$,
$d[j-2][j] = max(common[j-2][j] + 2, d[j-1][j])$,
...
$d[i][j] = max(common[i][j] + 2, d[i+1][j])$.

Теперь обсудим три способа быстро найти $common[i][j]$ для всех i, j.

а) $O(LN\log(N))$, где L — максимальная длина слов.

Отсортируем все имена в лексикографическом порядке, пусть строка i имеет номер $P(i)$ в отсортированном списке.

Найдем $c[P(i)]$ — размер наибольшего общего префикса строк $P(i)$, $P(i+1)$. Тогда $common[i][j]$ — минимум среди всех $c[t]$, $P(i) \le t < P(j)$. Действительно, если $P(i)$ и $P(j)$ имеют общий префикс длиной L, то все $c[t] \ge L$, причем хотя бы один раз достигается точное равенство.

Тогда

$common[P(j-1)][P(j)] = c[P(j-1)]$,
$common[P(j-2)][P(j)] = max(c[j-2], common[P(j-1)][P(j)])$,
...
$common[P(i)][P(j)] = max(c[P(i)], common[P(i+1)][P(j)])$.

б) $O(N^2 + NL)$ (предложено одним из участников).

Будем искать *common* методом динамического программирования.

Пусть $B[i+1][j]$ — такое $1 \le t \le j$, что $common[i+1][t]$ максимально.

Допустим, вычислены все $common[a][b]$ и $B[a][b]$, $a \le i$, $b \le i$. Научимся находить $common[i+1][j+1]$ и $B[i+1][j+1]$ при условии, что для всех пар $(i+1, t)$, $t \le j$ все уже вычислено. Возьмем $A = B[i+1][j]$. Первые $min(common[i+1][A], common[A][j+1])$ символов у строк $(i+1)$ и $(j+1)$ совпадают. Тогда если

$common[j+1][A] < common[i+1][A]$, то
$common[i+1][j+1] = common[j+1][A]$ и
$B[i+1][j+1] = A$.

В противном случае находим $common[i+1][j+1]$ непосредственно сравнивая символы с номерами $common[i+1][A]+1, \ldots, L$ у строк $(i+1)$, $(j+1)$, пока не найдется несовпадение. При этом

$B[i+1][j+1] = max(common[i+1][j+1], B[i+1][j])$.

Таким образом, для каждого $(i+1)$ непосредственных сравнений будет не более L, остальных действий $O(N)$, так что заявленная оценка верна.

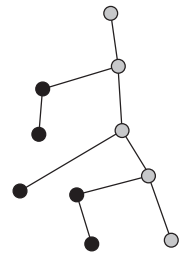
в) $O(S + N^2)$, где S — суммарная длина всех имен файлов.

Допишем к концу каждого имени файла символ "$". Построим сжатый бор из таких расширенных строк (подробнее о борах см. книгу Дэна Гасфилда "Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология"). Тогда каждой строке будет соответствовать лист в боре (у которого путевая метка равна расширенной строке). В противном случае получим, что префиксом какой-то строки A будет S\$, где S — строка без листа, тогда $A = S$, чего не может быть по условию.

Тогда $common[i][j]$ — длина путевой метки наименьшего общего предка для пары листьев, соответствующих строкам i, j (обозначается $LCA(i, j)$).

Вспомним, как алгоритм поиска в глубину раскрашивает вершины в процессе работы: вначале все вершины белые; когда вершина только начинает обрабатываться, ее красят в серый цвет; после того, как все ее сыновья (или вершины, в которые из нее ведут ребра) станут черными (обработанными), она становится черной.

Представим себе, что мы обходим в глубину из корня произвольное дерево.



Предложение. Пусть вершину j только что перекрасили в серый цвет (это значит, что все ее сыновья еще белые). Тогда $LCA(i, j)$ для любой черной вершины i будет $q(i)$ — наиболее удаленная от корня серая вершина, являющаяся предком i.

Так как конфигурация серых и черных вершин меняется по ходу выполнения алгоритма, то будем различать $q(i)$ для разных моментов времени (в частности, когда i не является черной, $q(i)$ не определена).

Итак, будем обходить в глубину из корня наш бор как дерево. В массиве $q[i]$ в любой момент времени будет храниться $q(A)$ в текущий момент времени, где A — лист, соответствующий строке i. Также будет храниться список листьев, уже ставших черными. Тогда при перекрашивании очередного листа в серый цвет надо пробежаться по списку уже

черных и проставить соответствующие значения *common*. При перекрашивании листа в черный цвет надо добавить его в список черных. Если при обработке любой вершины u оказывается, что $q[i]$ для какого-то i ссылается на черную вершину, то $q[i]$ — значение $q(A)$ для предыдущего момента времени, и $q[i]$ надо изменить на u.

Подробнее об алгоритме поиска *LCA* можно узнать из книги Т.Кормена, Ч.Лейзерсона, Р.Ривеста "Алгоритмы: построение и анализ" среди задач к разделу "Обход графов в глубину".

Задача 3. Приключение

Авторы — М.Р. Мирзаянов, В.Б. Гольдштейн

Теплым весенним днем группа из N школьников-программистов гуляла в окрестностях города Кисловодска. К несчастью, они набрели на большую и довольно глубокую яму. Как это случилось — непонятно, но вся компания оказалась в этой яме.

Глубина ямы равна H. Каждый школьник знает свой рост по плечи h_i и длину своих рук l_i. Таким образом, если он, стоя на дне ямы, поднимет руки, то его ладони окажутся на высоте $h_i + l_i$ от уровня дна ямы. Школьники могут, вставая друг другу на плечи, образовывать вертикальную колонну. При этом любой школьник может встать на плечи любого другого школьника. Если под школьником i стоят школьники j_1, j_2, ..., j_k, то он может дотянуться до уровня $h_{j1} + h_{j2} + ... + h_{jk} + h_i + l_i$.

Если школьник может дотянуться до края ямы (то есть $h_{j1} + h_{j2} + ... + h_{jk} + h_i + l_i \geq H$), то он может выбраться из нее. Выбравшиеся из ямы школьники не могут помочь оставшимся.

Найдите наибольшее количество школьников, которые смогут выбраться из ямы до прибытия помощи, и перечислите их номера.

Формат входных данных

В первой строке входного файла записано натуральное число N ($1 \leq N \leq 2000$) — количество школьников, попавших в яму.

Далее в N строках указаны по два целых числа: рост i-го школьника по плечи h_i ($1 \leq h_i \leq 10^5$) и длина его рук l_i ($1 \leq l_i \leq 10^5$).

В последней строке указано целое число — глубина ямы H ($1 \leq H \leq 10^5$).

Формат выходных данных

В первой строке выведите K — максимальное количество школьников, которые смогут выбраться из ямы. Если $K > 0$, то во второй строке в произвольном порядке выведите их номера, разделяя их пробелами. Школьники нумеруются с единицы в том порядке, в котором они заданы во входном файле. Если существует несколько решений, выведите любое.

Примеры

advent.in	advent.out
2 10 4 5 2 20	0
6 6 7 3 1 8 5 8 5 4 2 10 5 30	4 1 4 2 5

Решение

Назовем рослостью человека сумму его роста по плечи плюс длина его рук. Таким образом, рослость — максимальная глубина ямы, из которой человек может выбраться самостоятельно.

Вначале решим более простую задачу: могут ли все люди выбраться из ямы? Если да, то кто-то из них выбирается последним, таким образом, его рослости достаточно, чтобы выбраться из ямы самостоятельно. Таким образом, последним может выбираться человек с максимальной рослостью. При этом этот человек может помогать другим, встав в самый низ колонны из людей, подняв ее на высоту своего роста. Таким образом, если все люди выбираются из ямы, то они могут выбираться в порядке увеличения рослости (самый рослый — в основании). Для этого им потребуется создать колонну, в основании которой стоит самый рослый человек, на нем следующий по рослости и так далее.

Вернемся к решению исходной задачи.

Пусть нам известно, какие люди могут выбраться из ямы, а какие — нет. Тогда мы можем образовать из всех людей колонну, причем те люди, которые не смогут выбраться, стоят в основании колонны, поднимая остальных на сумму своих высот. При этом люди, которые выбираются из ямы, стоят по увеличению рослости, если смотреть сверху вниз.

Построим людей в колонну по увеличению рослости (самый рослый — внизу) и занумеруем их сверху вниз. Пусть среди первых i человек из ямы могут выбраться j, при условии, что все остальные люди им помогают. Переместим оставшихся $(i - j)$ человек в основание колонны, при этом она поднимется на сумму их ростов. Таким образом, нам выгодно, чтобы суммарный рост выбравшихся был минимален, так как при этом суммарный рост людей, переставленных в основание колонны, максимален.

Обозначим за P исходную высоту колонны (сумму ростов всех людей), а через p_{ij} — максимально возможную высоту оставшейся части колонны, если мы рассмотрели i человек, и из ямы выбираются j из них. Будем пересчитывать p_{ij} при помощи динамического программирования. В начале проинициализируем $p_{00} = P$, а остальные p_{0j} — значением $-\infty$.

Вычислим p_{ij}. Существуют два варианта.

Человек с номером i выбрался из ямы, т.е. он достает до верха. Для этого должно выполняться $p_{i-1j-1} + l_i \geq H$ (рост b_i самого человека в формулу не входит, так как он уже включен в p_{i-1j-1}). В этом случае p_{ij} равно $p_{i-1j-1} - b_i$.

Человек с номером i не выбрался из ямы. В этом случае p_{ij} может быть равно p_{i-1j}.

Итого p_{ij} равно максимуму из двух вариантов и может быть посчитано за $O(1)$. Таким образом, все значения p_{ij} можно вычислить за $O(N^2)$.

Искомое число K равно максимальному j, для которого $p_{N,j}$ не равно $-\infty$. Выбраться из ямы i-й также сможет, только если $p_{Ni} > -\infty$.

Менее эффективные решения оценивались из меньшего числа баллов.

Например, существует решение, в котором в качестве параметра динамического программирования используются высота ямы или сумма ростов людей. Например, можно было строить колонну из людей снизу вверх. При этом люди обрабатываются по убыванию рослости, и подсчитывается максимально возможная глубина ямы a_{ijk}, из которой могут выбраться первые j человек из первых i, если суммарный рост выбравшихся людей равен k. При этом также требуется рассмотреть два варианта.

Если i-й человек выбирается, то a_{ijk} может быть равно a_{i-1jk}.

Если i-й человек не выбирается, то его рослость минимальна из рассмотренных и он стоит на самом верху колонны. Таким образом, a_{ijk} может быть равно $\min(a_{i-1j-1k-bi}, k + l_i)$.

Значения a_{ijk} также могут быть вычислены динамическим программированием за $O(N^2P)$. При этом ответом является максимальное j, такое, что $(P - k) + a_{Njk} > H$. Такое решение набирало около 50 баллов.

Также можно было подсчитывать b_{ijk} — минимальный суммарный рост выбравшихся j людей из первых i, которые могут выбраться из ямы глубины k. Эту величину также можно пересчитывать динамическим программированием. При этом требуется $O(N^2H)$ времени. Такое решение набирало 70 баллов.

Задача 4. Автобусы

Автор — В.А. Матюхин

Новый президент Тридевятой республики начал свою деятельность с полной ревизии системы общественного транспорта страны. В результате на основе социологических опросов населения было составлено идеальное ежедневное расписание движения междугородних автобусов, утвержденное Парламентом республики.

Более того, было решено заменить весь автобусный парк одинаковыми новыми, очень дорогими, но гораздо более надежными, красивыми и удобными машинами.

Автобусная сеть страны охватывает N городов, занумерованных целыми числами от 1 до N.

Идеальное расписание содержит M ежедневных рейсов, i-й рейс начинается в городе F_i в момент времени X_i и заканчивается в некотором другом городе G_i в момент времени Y_i. Продолжительность каждого рейса ненулевая и строго меньше 24 часов. Рейс i выполняется одним из автобусов, находящихся в момент времени X_i в городе F_i.

Новые автобусы не требуют ремонта и могут работать круглосуточно, поэтому автобус, прибывший в некоторый момент времени в некоторый город, всегда готов в тот же самый момент времени или позже отправиться в путь для обслуживания любого другого рейса из данного города. Автобус может выехать из города, только выполняя какой-либо рейс из расписания.

Предполагается, что расписание будет действовать неограниченное время, поэтому может оказаться так, что его невозможно обслужить никаким конечным числом автобусов.

Определите наименьшее количество новых автобусов, достаточное для обеспечения движения по расписанию в течение неограниченного периода времени.

Формат входных данных

В первой строке содержатся целые числа N и M ($1 \leq N, M \leq 100\,000$) — количество городов и рейсов автобусов соответственно.

В каждой из следующих M строк содержится описание рейса автобуса: номер города отправления F_i, время отправления X_i, номер города назначения G_i ($F_i \neq G_i$), время прибытия Y_i, отделенные друг от друга одним пробелом. Времена задаются в формате `HH:MM`, где `HH` — часы от 00 до 23, `MM` — минуты от 00 до 59.

Формат выходных данных

В выходной файл выведите одно число — минимально необходимое количество автобусов. Если расписание невозможно обслуживать в течение не-

ограниченного периода времени конечным числом автобусов, выведите число -1.

Примеры

buses.in	buses.out
2 1 1 09:00 2 15:30	-1
5 5 1 09:00 2 14:30 3 23:45 1 06:50 2 14:30 3 20:50 4 09:00 5 21:00 5 10:00 4 20:00	3

Решение

Прежде всего определим, в каких случаях потребуется конечное число автобусов. Построим ориентированный граф, вершинами которого будут города, а ребрами — рейсы автобусов. Чтобы автобусы не скапливались в городе, количество прибывающих и отъезжающих автобусов должно совпадать, поэтому в полученном графе количество входящих ребер должно совпадать с количеством исходящих ребер. Если это свойство не выполняется, значит, автобусов бесконечное число.

С этого момента будем решать задачу в предположении, что количество автобусов конечно. Сначала разберем случай, когда отсутствуют рейсы, которые захватывают полночь. Изначально будем считать, что во всех городах общее количество автобусов равно нулю. Проследим за движением автобусов в течение первого дня. В тот момент, когда приходит время отправления автобуса по расписанию, мы отправляем любой из автобусов, который находится в городе. Если на данный момент в городе автобусы отсутствуют, добавляем новый автобус и увеличиваем ответ на единицу. Можно заметить, что количества автобусов утром и вечером совпадают, так как в течение дня из города уезжает столько же автобусов, сколько в него возвращается. Поэтому дополнительные автобусы для обеспечения движения по расписанию в последующие дни не потребуются.

Рассмотрим второй случай. Будем считать, что каждый из рейсов ровно в полночь делает остановку в виртуальном $(N + 1)$-м городе. Тогда рейс, для которого время отправления превышает время прибытия, распадается на две части: (*время отправления* — 24:00) и (00:00 — *время прибытия*). В результате такого преобразования задача сводится к предыдущему случаю. Другой подход заключается в том, что можно изначально на каждом рейсе, который захватывает полночь, запустить по автобусу и после этого свести задачу к предыдущему случаю.

Существует несколько способов программно реализовать перечисленные выше идеи. При решении задачи можно отдельно не разбирать случай бесконечного количества автобусов и рейсы, которые захватывают полночь. Однако при этом появляется возможность допустить ошибку, которую в дальнейшем будет трудно найти и исправить.

Правильное решение задачи требует $O(N + M)$ памяти и работает за время $O(N + M\log M)$. Заведем специальный массив, в котором будем хранить текущее количество автобусов в городе. Каждый рейс преобразуем в два события: отправление и прибытие автобуса. Отсортируем все события по времени их возникновения. Если два события возникают одновременно, сначала будем рассматривать прибытие автобуса, а только потом отправление. Такая модификация позволит автобусу отправиться сразу же после прибытия. Будем просматривать события одно за другим. Если автобус прибывает в город, увеличиваем количество автобусов в этом городе на единицу, если отправляется — уменьшаем на единицу. Если при отправлении в городе отсутствуют автобусы, увеличиваем ответ на единицу. Понятно, что такой алгоритм будет работать за линейное время, если не учитывать сортировку данных. Это решение набирает полный балл.

Можно непосредственно имитировать передвижение автобусов в течение дня, рассматривая дискретные моменты времени с интервалом в одну минуту. Тогда в каждый момент времени автобус либо находится в движении, либо простаивает. Если в указанный момент времени должен отправиться один из автобусов по расписанию, находим автобус, который простаивает, либо заводим новый автобус. Такое решение на некоторых тестах не укладывается в ограничение по времени и набирает 70 баллов.

Еще одно решение основано на "склеивании" рейсов и выделении циклов в графе. Если есть два рейса из города A в город B и из города B в город C, заменяем их одним рейсом из города A в город C, пересчитываем при этом продолжительность рейса. В общем случае через город B может проходить несколько рейсов. В этом случае, чтобы общее количество автобусов было минимальным, необходимо рейсы разбить на пары так, чтобы минимизировать суммарное время простоя. Продолжаем склейку до тех пор, пока все рейсы не превратятся в петли (циклы). Следует отметить, что в некоторых тестах правильное решение может в одном городе содержать несколько петель и склейка таких маршрутов приводит к ухудшению результата. Эту идею можно довести до правильного решения, но трудоемкость этого перехода по сложности такая же, как правильное решение задачи. Решение, которое при склеивании перебирает все пары рейсов, проходя-

щих через заданный город, на некоторых тестах не укладывается в ограничения по времени и набирает 70 баллов.

Решение, которое правильно определяет случай бесконечного количества автобусов и проходит тесты небольшой размерности, набирает 20 баллов. Решение, которое не учитывает рейсы, захватывающие полночь, набирает 40 баллов.

Стоит отметить, что решение задачи достаточно сложно отлаживать. Основная проблема в том, что не всегда возможно с первого взгляда вручную найти правильный ответ к задаче, даже на маленьком тесте. Это может казаться парадоксальным, но в некоторых случаях требуется автобусов больше, чем количество рейсов:

Входные данные	Выходные данные
1 2 1 10:00 2 18:00 2 17:00 1 11:00	3

В общем случае, так как автобус не может простаивать более суток, количество автобусов не превышает удвоенного количества рейсов.

Задача 5. Полимино

Автор задачи — Г.А. Корнеев

Известный математик Соломон В. Голомб предложил название *полимино* для связной фигуры, вырезанной из клетчатой бумаги по линиям сетки. Фигура называется связной, если из любой ее клетки можно добраться в любую другую, переходя из клетки в клетку через их общую сторону. Шахматист, добавил Голомб, сказал бы, что из любой клетки полимино можно дойти ладьей в любую другую. На *рис.* 1 приведены примеры восьми полимино.

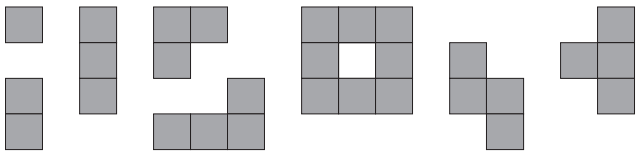
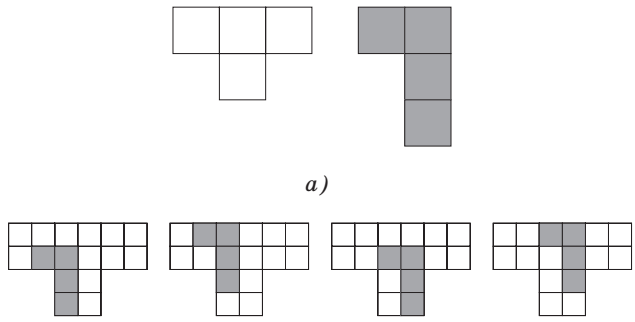


Рис. 1

Саша увлекается полимино. Для своих экспериментов она вырезает новое полимино из бумаги в клеточку или из старых полимино, оставшихся после предыдущих попыток. Далеко не всегда из старого полимино (*рис.* 2*а*, слева) можно вырезать новое (*рис.* 2*а*, справа). Поэтому Саша может перед вырезанием нового полимино разделить каждую клетку старого полимино на K^2 одинаковых квадратных клеток меньшего размера (см. *рис.* 2*б*, здесь $K = 2$).



а)

б)

Рис. 2

Сашу заинтересовало, сколько существует различных способов вырезать новое полимино из старого при заданном значении K, если повороты, отражения и переворачивания как нового полимино, так и старого недопустимы.

Например, на *рис.* 2*б* приведены все возможные способы вырезания полимино, приведенного на *рис.* 2*а*, при $K = 2$.

Напишите программу, которая ответит на интересующий Сашу вопрос.

Формат входных данных

Первая строка входного файла содержит число K ($1 \le K \le 10\,000$).

Далее следуют описания двух полимино, сначала нового, затем старого. Каждое полимино задается следующим образом — в первой строке описания задаются размеры H (высота) и W (ширина) минимально возможного прямоугольника, в котором можно разместить данное полимино. Следующие H строк содержат по W символов описания клеток. При этом клетка, входящая в полимино, обозначается символом "X" (прописная латинская буква "икс"), а не входящая — символом "." (точка). Количество клеток в каждом полимино не превышает 300.

Формат выходных данных

Выходной файл должен содержать одно число — количество различных способов вырезать заданное новое полимино из старого, каждая клетка которого разбита на K^2 клеток.

Примеры

poly.in	poly.out
2 3 2 XX .X .X 2 3 XXX .X.	4

poly.in	poly.out
2 2 2 XX XX 2 3 XXX .X.	7
1 2 2 XX XX 1 1 X	0

Решение

Пусть H_1, W_1 — размеры минимального охватывающего прямоугольника нового полимино, N — количество клеток в новом полимино, а H_2, W_2, M — размеры минимального охватывающего прямоугольника старого полимино и количество клеток в нем, соответственно.

Для того чтобы найти число способов вырезать новое полимино из исходного, увеличенного в K раз, можно перебрать все возможные сдвиги одного относительно другого и проверить, что фигуры, приложенные друг к другу таким образом, совпадают. Под сдвигом одного полимино относительно другого будем понимать сдвиг фиксированной клетки одного полимино относительно фиксированной клетки другого. Однако это неэффективное решение с асимптотикой $O(K^2MN)$, которое не укладывается в ограничение по времени.

Рассмотрим K^2 возможных сдвигов нового полимино относительно одного из увеличенных квадратов исходного. Для каждого сдвига рассмотрим минимальную конфигурацию клеток, такую, что после увеличения в ней будет встречаться нужное полимино с текущим сдвигом. Назовем такую конфигурацию шаблоном.

Пример: на *рис.* 3 показано полимино, сдвинутое относительно "клетки" 3×3, и шаблон для этого сдвига.

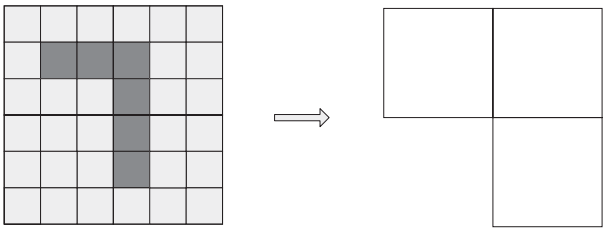
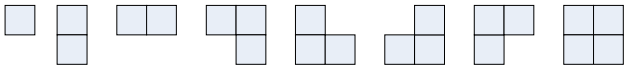


Рис. 3

Теперь необходимо лишь проверить, сколько раз встречается этот шаблон в исходном полимино. Количество клеток в шаблоне есть $O(N/K)$, построить его можно за $O(N)$, найти число вхождений шаблона в исходное полимино можно за $O(NM/K)$. Поэтому асимптотика этого решения $O(NMK + K^2N)$.

Однако при больших значениях K и это решение не укладывается в ограничение по времени. Нетрудно заметить, что при $K \geq \max(W_1, H_1)$ шаблон для любого сдвига содержит не более 4 клеток. То есть возможны следующие варианты шаблонов:



Для каждого из этих шаблонов за $O(M)$ можно подсчитать количество вхождений его в исходное полимино. Осталось лишь для каждой такой конфигурации найти количество сдвигов, при которых она станет шаблоном для нового полимино. Для первых трех типов шаблонов несложно выписать готовые формулы:

Шаблон	Количество сдвигов
	$(K - H_1 + 1)(K - W_1 + 1)$
	$(K - W_1 + 1)(H_1 - 1)$
	$(K - H_1 + 1)(W_1 - 1)$

Теперь необходимо подсчитать число сдвигов для остальных конфигураций шаблонов. Для этого рассмотрим все возможные сдвиги (X, Y) прямоугольника, содержащего полимино, относительно "клетки" размера $K \times K$ так, как показано на *рис.* 4.

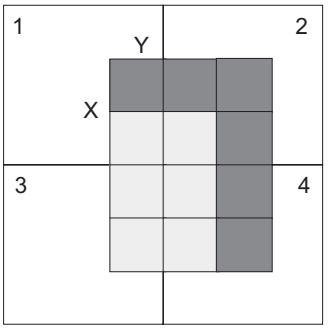


Рис. 4

Число всех возможных сдвигов есть $O(W_1H_1)$. Для каждого такого сдвига можно получить соответствующий шаблон. Для этого необходимо для каждого из

четырех прямоугольников определить, есть ли в нем клетки полимино. Это можно сделать за $O(1)$, если предварительно для каждой клетки прямоугольника, содержащего полимино, определить, есть ли клетки, находящиеся левее и выше данной, правее и выше, левее и ниже, правее и ниже данной (за $O(W_1H_1)$ для всех клеток прямоугольника). Решение, которое для каждого сдвига строит шаблон за $O(N)$, также укладывается в ограничение по времени.

Пусть A_i — количество сдвигов полимино относительно квадрата $K \times K$, таких, что i-я конфигурация клеток будет шаблоном для каждого из этих сдвигов. Пусть B_i — количество вхождений i-го шаблона в исходное полимино. Тогда ответом будет сумма $A_i \times B_i$, для i от 1 до 8.

Таким образом, при $K < \max(H_1, W_1)$ асимптотика решения $O(NMK + K^2N)$, при $K \geq \max(H_1, W_1)$ — $O(W_1H_1 + M)$.

Задача 6. Треугольная реформа

Авторы задачи — Е.В. Андреева, Ю.Е. Егоров

Король Полигонии Трианг IV помешан на реформах. Чтобы войти в мировую историю, он решил провести территориальную реформу в своей стране. Страна Полигония имеет форму простого многоугольника, то есть ее граница не имеет самопересечений и самокасаний. В Полигонии большую роль играют *внутренние диагонали* — отрезки, соединяющие вершины государственной границы и полностью проходящие по территории страны, не касаясь границы. При этом форма Полигонии такова, что никакие две внутренние диагонали не лежат на одной прямой.

Вместо традиционного деления государства на *административные округа* король Трианг IV решил разделить свою страну на *административные треугольники* внутренними диагоналями. Для сокращения управляющего аппарата король повелел подготовить такой план деления страны, в котором количество административных треугольников будет минимальным.

Требуется написать программу, выполняющую деление Полигонии внутренними диагоналями на минимально возможное число административных треугольников.

Формат входных данных

В первой строке входного файла записано число N ($3 \leq N \leq 20$) — количество вершин многоугольника, образующего границу Полигонии. В следующих N строках находятся по 2 целых числа, по абсолютной величине не превосходящие 10 000 — координаты вершин в порядке обхода многоугольника против часовой стрелки. Гарантируется, что никакие три последовательные вершины многоугольника не лежат на одной прямой, и он не имеет самопересечений и самокасаний. Также гарантируется, что никакие две диагонали, содержащиеся внутри многоугольника, не лежат на одной прямой.

Формат выходных данных

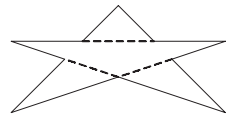
В первую строку выходного файла выведите наименьшее количество административных треугольников, на которое можно разбить Полигонию.

Во вторую строку выведите количество различных внутренних диагоналей K, с помощью которых можно произвести данное разбиение.

В каждую из следующих K строк выведите 4 целых числа — координаты начала и конца соответствующей диагонали разбиения, полностью лежащей внутри многоугольника и не проходящей по его границе.

Если искомых разбиений несколько, выведите любое из них.

Примеры

reform.in	reform.out	Рисунок к тесту
4 0 0 1 0 1 1 0 1	2 1 1 1 0 0	
10 -6 0 0 2 6 0 3 3 6 4 2 4 0 6 -2 4 -6 4 -3 3	4 3 2 4 -2 4 0 2 3 3 -3 3 0 2	

Решение

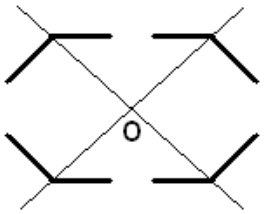
Фактически задача состоит в том, чтобы разрезать многоугольник на минимальное количество треугольников, производя разрезы только вдоль внутренних диагоналей.

Основную техническую проблему в решении представляет ответ на вопрос: является ли некоторая диагональ внутренней? Решать эту проблему можно несколькими способами. Прежде всего надо убедиться, что диагональ не пересекается со сторонами многоугольника (в противном случае она не является ни внутренней, ни внешней). Затем можно, например, проверить, принадлежит ли середина этой диагонали внутренней области

многоугольника. Сделать это возможно, подсчитав число пересечений луча, начинающегося в этой точке и заканчивающегося на бесконечности, с границей многоугольника. При нечетном числе пересечений точка внутренняя, а при четном — внешняя.

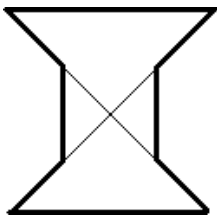
Перейдем к решению основной задачи. Для начала упомянем, что, разрезая вдоль произвольных прямых линий, а не только вдоль диагоналей, количество частей уменьшить невозможно. Ограничение на тип разрезов было предложено лишь для устранения проблем, связанных с точностью вычислений.

Теперь исследуем, имеет ли смысл разрезать многоугольник вдоль двух пересекающихся внутренних диагоналей:



Предположим, что вдоль одной из диагоналей разрез уже сделан. Тогда мы имеем два отдельных многоугольника, для каждого из которых требуется решить поставленную задачу. Но ни в одном из них точка O не является вершиной, а значит, по приведенному выше утверждению, можно решить для них задачу, не проводя разрезы через точку O. Значит, всегда можно ликвидировать такие пары пересекающихся разрезов.

Впрочем, это не значит, что такие пары разрезов строго увеличивают количество треугольников; вот один из контрпримеров:



Итак, каждая проведенная внутренняя диагональ делит наш многоугольник на два отдельных независимых многоугольника.

Теперь приведем следующие факты из геометрии.

Факт 1. В любом (простом) многоугольнике с более чем тремя вершинами найдется внутренняя диагональ.

Факт 2. Любой N-угольник можно разрезать диагоналями на $N - 2$ треугольника.

Построим на базе этих фактов следующий алгоритм.

Алгоритм разделения N-угольника на $N - 2$ треугольника

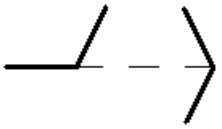
При $N = 3$ треугольник сам является единственной искомой областью.

При $N > 3$ найдем произвольную внутреннюю диагональ и разрежем многоугольник вдоль нее. Пусть одна из частей оказалась M-угольником. Тогда другая часть будет иметь $N - M + 2$ вершин (убедитесь в этом!). Запустим наш алгоритм рекурсивно в обеих частях. В одной из них мы получим $M - 2$ треугольника, а в другой — $N - M$ треугольников. Итого имеем $N - 2$ треугольника, что нам и требовалось.

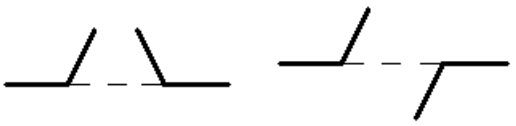
Результат работы приведенного выше алгоритма может оказаться неоптимальным с точки зрения исходной задачи. Тонкое место в рассуждении — это утверждение о количестве вершин в первой и второй частях. Улучшение может иметь место, если количество вершин в одной или в обеих частях окажется меньше. Это произойдет, если три или четыре вершины, идущие подряд в новом многоугольнике, окажутся на одной прямой. Тогда одну или две из них можно будет перестать рассматривать как вершину.

Это приводит нас к следующим важным определениям.

Определение 1. Диагональю первого типа будем называть внутреннюю диагональ, лежащую на одной прямой ровно с одной стороной многоугольника, имеющей с ней общий конец.



Определение 2. Диагональю второго типа будем называть внутреннюю диагональ, лежащую на одной прямой с двумя сторонами многоугольника, имеющими с ней общий конец.



Определение 3. Диагонали обоих типов будем называть особенными диагоналями.

Таким образом, если в предыдущем алгоритме в какой-то момент выбрать не обычную внутреннюю диагональ, а особенную, то суммарное количество вершин в новых двух многоугольниках уменьшится на номер типа этой диагонали. А значит, на это же число уменьшится и количество треугольников в разбиении.

Теперь видна задача оптимизации: в процессе работы алгоритма требуется использовать множество диагоналей с максимальной суммой номеров типов. К сожалению, использовать абсолютно все особенные диагонали получается не всегда: напри-

мер, две пересекающиеся особенные диагонали использовать вместе невыгодно.

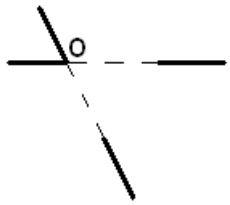
Исследуем подробно все случаи взаимного расположения двух особенных диагоналей.

• Если диагонали пересекаются (строго пересекаются, а не касаются), то их не следует использовать вместе.

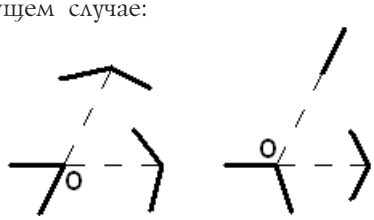
• Если диагонали не имеют общих точек, то их можно использовать вместе.

Особенная внимательность требуется в случае, когда две особенные диагонали выходят из одной вершины. В этом случае имеет значение их тип.

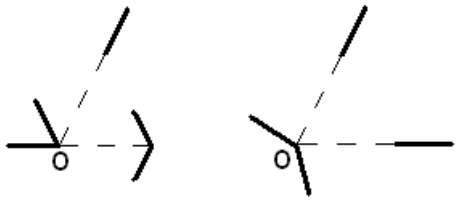
• Если это две диагонали второго типа, то их использовать вместе не следует (после разрезания вдоль одной из них точка *O* перестает быть вершиной в новых многоугольниках):



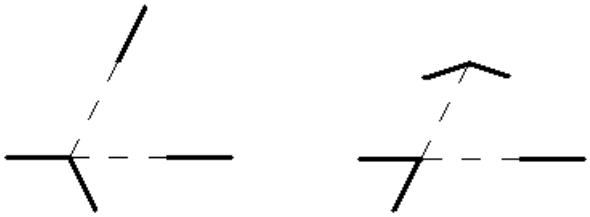
• Для двух диагоналей первого типа есть четыре различных случая. В первых двух случаях две диагонали нельзя брать вместе по той же причине, что и в предыдущем случае:



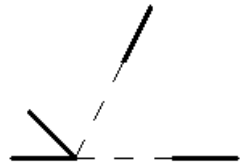
• Еще в двух случаях точка *O* является вершиной в обеих частях и диагонали можно брать вместе, улучшая при этом ответ на задачу:



• Если эти две особенные диагонали разных типов, то возникают три различных случая. В первых двух диагонали выбирать вместе нельзя:



• И лишь при следующем взаиморасположении диагоналей можно использовать их вместе:



Зная это, поступим следующим образом: построим граф, в котором вершины — это особенные диагонали, а ребра между вершинами проведены тогда, когда их можно использовать вместе. Кроме того, введем вес вершины, равный типу соответствующей особенной диагонали.

Здесь стоит оценить размер этого графа. Каждая особенная диагональ является продолжением некоторой, хотя бы одной, стороны. В то же время, продолжением каждой стороны может являться не более одной особенной диагонали, поскольку иначе появятся две внутренние диагонали многоугольника, лежащие на одной прямой. Отсюда ясно, что количество особенных диагоналей не превышает количества вершин многоугольника.

В построенном графе требуется найти множество вершин, попарно соединенных ребрами между собой, с наибольшим суммарным весом. Эта задача называется "Задачей о максимальной клике" и является *NP*-трудной.

Экспоненциальное решение задачи придумать несложно. Например, можно перебрать все возможные множества вершин в этом графе (их 2^m, где m — количество особенных диагоналей) и для каждого из них проверить, все ли вершины этого множества соединены между собой. Из всех подходящих множеств выбрать множество с наибольшим весом, оно и будет искомым.

Получив требуемый набор особенных диагоналей, решение изначальной задачи доводится до конца следующим образом.

Разрежем многоугольник вдоль всех особенных диагоналей из найденного набора. Многоугольник при этом разобьется на некоторое количество меньших многоугольников, в каждом из которых нет особенных диагоналей (иначе их можно было бы добавить в наш максимальный набор). Для каждого из этих многоугольников, следовательно, можно запустить первичный алгоритм, который правильно и оптимально разрежет его на треугольники.

Время работы полученного алгоритма составляет $O(2^N N^2)$.

Имеется также решение с этой асимптотикой, оформленное в виде динамического программирования.

Рассмотрим некоторый многоугольник, появившийся в процессе разрезания исходного многоуголь-

ника внутренними диагоналями. Множество вершин этого многоугольника — подмножество вершин изначального, причем порядок обхода этих вершин определяется однозначно.

Таким образом, имеется не более чем 2^N многоугольников, подлежащих исследованию, и их удобно идентифицировать масками из N битов.

Применим динамическое программирование, чтобы для каждого такого многоугольника подсчитать количество треугольников в его минимальной триангуляции, зная ответы для всех многоугольников с меньшим количеством вершин.

Для треугольника подсчет очевиден.

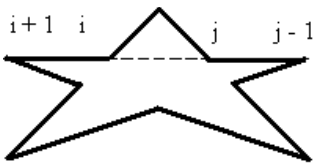
Для произвольного выпуклого многоугольника подсчет также не представляет сложности.

Теперь рассмотрим некоторый невыпуклый многоугольник. Выберем в нем минимальную по номеру вершину, угол при которой превышает 180° (в невыпуклом многоугольнике такой угол всегда найдется).

В любой триангуляции из этой вершины должна выходить хотя бы одна диагональ. Проведем из нее все возможные внутренние диагонали. В каждом из случаев многоугольник разделится на два других многоугольника, обработанных ранее. Среди всех вариантов выберем вариант с минимальным суммарным количеством треугольников. Такое решение набирало полный балл.

Однако существует и эффективное решение исходной задачи, основанное на классическом динамическом программировании. Заведем двумерный массив a размером $N \times N$, элемент a_{ij} которого будет обозначать, на какое минимальное число треугольников можно разбить многоугольник, образованный вершинами i, $i + 1$, ..., j (а если $i > j$, то вершинами i, $i + 1$, ..., N, 1, ..., j), если диагональ, соединяющая вершины i и j, войдет в оптимальное разбиение.

Находить значение каждого элемента такого массива будем следующим образом. Для треугольников оно равно 1. Если вершин в соответствующем многоугольнике больше, то рассмотрим все вершины k, лежащие между i и j, в порядке обхода многоугольника против часовой стрелки. Найдем такое k, что диагонали, соединяющие вершины i и k, а также j и k, являются внутренними, а сумма $a_{ik} + a_{kj}$ минимальна. Тогда $a_{ij} = a_{ik} + a_{kj} + 1$. Поправку в работу этого алгоритма вносят диагонали первого и второго типа: например, если диагональ, соединяющая вершины i и j, — второго типа, то или $a_{ij} = a_{i+1j-1}$, или $a_{ji} = a_{j-1i+1}$.



Московская личная олимпиада по программированию 2005/2006 уч. года для 7–9-х классов

Авторы задач — Е.В. АНДРЕЕВА, В.М. ГУРОВИЦ, В.А. МАТЮХИН
Автор разбора — В.М. ГУРОВИЦ

От редакции. Мы уже рассказывали на страницах нашей газеты о формах проведения Московской городской олимпиады по информатике и способах отбора на эту олимпиаду. В этом учебном году количество различных соревнований было еще увеличено. В частности, была добавлена олимпиада для учащихся 7—9-х классов. Отбор участников этой олимпиады производился через тот же заочный отборочный интернет-тур, что и на основную олимпиаду. Только критерии отбора были существенно снижены. Несколько школьников приняли участие в олимпиаде по результатам окружных соревнований, которые также проводились в общем зачете, т.е. для школьников 7—11-х классов одновременно. Участие в отборочных соревнованиях было не напрасным: несмотря на свой юный возраст, все участники олимпиады уверенно работали с файлами, и только один ученик не смог набрать никаких баллов.

Правила олимпиады совпадали с правилами "настоящих" олимпиад по информатике, в частности, со Всероссийской олимпиадой. Однако задачи были существенно проще и на решение 5 задач отводилось 4 часа. Наверное, ученикам 10—11-х классов большинство задач показались бы не олимпиадными, а учебными. Кроме того, условия задач были сформулированы предельно кратко и понятно. Тем не менее никто из участников не решил полностью все 5 предложенных задач, хотя каждая задача была кем-то решена. А первая — самая простая — задача была не полностью решена даже победителем олимпиады!

Подобные соревнования направлены на возрождение интереса к программированию, на стимулирование занятия программированием в основной школе. Организаторы считают первый опыт весьма успешным и стремятся его расширить: в июне 2006 г. была проведена и первая командная олимпиада для учеников 8-х и более младших классов, ведь именно им предстоит соревноваться в олимпиаде для 7—9-классников в следующем учебном году. Пожелаем им, а также их учителям и наставникам успехов!

Задача A. Строки в книге

В книге на одной странице помещается K строк. Таким образом, на 1-й странице печатаются строки с 1-й по K-ю, на второй — с $(K + 1)$-й по $(2K)$-ю и т.д. Напишите программу, которая по номеру строки в тексте определяет номер страницы, на которой будет напечатана эта строка и порядковый номер этой строки на странице.

Формат входных данных

Входной файл содержит число K — количество строк, которое печатается на странице, и число N — номер строки ($1 \le K \le 200$, $1 \le N \le 20\,000$).

Формат выходных данных

В выходной файл выведите два числа — номер страницы, на которой будет напечатана эта строка, и номер строки на странице.

Примеры

a.in	a.out
50 1 20 25 15 43	1 1 2 5 3 13

Решение

Подсчитаем сначала, сколько страниц будет заполнено целиком, если мы напечатаем только первые N строк. Для этого разделим N на K с остатком. Неполное частное как раз и будет равно искомому числу — количеству страниц, на которых напечатано по K строк. Если остаток от деления равен нулю, то строка с номером N окажется последней на странице с номером $N \operatorname{div} K$. Если же остаток отличен от нуля, то эта строка окажется уже на следующей странице — на странице с номером $(N \operatorname{div} K) + 1$ — и ее порядковый номер будет в точности равен этому

остатку — N mod K. Осталось написать короткую программу:

```
read(K, N);
if N mod K = 0 then write(N div K, ' ', K)
else write (N div K + 1, ' ', N mod K);
```

Можно решить эту задачу и без использования условного оператора:

```
read(K, N);
write((n − 1) div k) + 1, ' ',
      (n − 1) mod k + 1);
```

Проверьте самостоятельно, что эти формулы дают верный результат.

Задача B. Палиндромная последовательность

Последовательность чисел назовем *симметричной*, если она одинаково читается как слева направо, так и справа налево. Например, следующие последовательности являются симметричными:

```
1 2 3 4 5 4 3 2 1
1 2 1 2 2 1 2 1
```

Вашей программе будет дана последовательность чисел. Требуется определить, какое минимальное количество и каких чисел надо приписать в конец этой последовательности, чтобы она стала симметричной.

Формат входных данных

Во входном файле записано сначала число N — количество элементов исходной последовательности. Далее записано N чисел — элементы этой последовательности. $1 \le N \le 100$, элементы последовательности — натуральные числа от 1 до 9.

Формат выходных данных

В выходной файл выведите сначала число M — минимальное количество элементов, которое надо добавить к последовательности, а потом M чисел (каждое — от 1 до 9) — числа, которые надо добавить к последовательности.

Примеры

b.in	b.out
9 1 2 3 4 5 4 3 2 1 5 1 2 1 2 2 5 1 2 3 4 5	0 3 1 2 1 4 4 3 2 1

Решение

Заметим, что в худшем случае нам придется дописать $N - 1$ чисел. Например, если дана последовательность 1 2 3 ... $(N - 1)$ N, то кратчайшая симметричная последовательность будет выглядеть так: 1 2 3 ... $(N - 1)$ N $(N - 1)$ ... 3 2 1. В самом лучшем случае наша последовательность уже является симметричной, и дописывать ничего не нужно. Таким образом, ответом в задаче является число от 0 до $N - 1$.

Опишем один из алгоритмов, решающих нашу задачу. Для начала проверим, является ли данная нам последовательность симметричной, используя вызов функции, которая будет проверять, является ли симметричной последовательность $a[1]$, $a[2]$, ..., $a[k]$. Для этого нужно сравнить числа $a[1]$ и $a[k]$, $a[2]$ и $a[k - 1]$, и т.д. до $a[k \text{ div } 2]$ и $a[k \text{ div } 2 + 1]$.

Если в какой-то из пар числа окажутся неравными, то последовательность симметричной не является. Напишем соответствующий код:

```
function palindrom(k: integer): boolean;
var i: integer;
begin
  palindrome := true;
  for i := 1 to k div 2 do
    if a[i] <> a[k + 1 − i] then begin
      palindrom := false;
      break
    end
end;
```

Теперь вернемся к исходной задаче. Если исходная последовательность не симметрична, то проверим, достаточно ли добавить к ней одно число. Ясно, что для того, чтобы получилась искомая последовательность, необходимо в конец дописать число $a[1]$:

```
a[N + 1] := a[1];
```

После этого опять проверим, получилась ли у нас симметричная последовательность, вызывая функцию `palindrom(N + 1)`. Если нас вновь постигла неудача, допишем в конец исходной последовательности уже два числа:

```
a[N + 2] := a[1]; a[N + 1] := a[2];
```

И т.д. до тех пор, пока функция `palindrom` на очередном шаге не вернет результат `true`.

Покажем, как организовать этот процесс, используя цикл **for**:

```
[1] for i := 0 to N − 1 do begin
[2]   for j := 1 to i do
      {Добавляем в конец i чисел}
[3]     a[N + j] := a[i + 1 - j];
[4]   if palindrom(N + i) then begin
     {Печатаем количество добавленных чисел}
[5]     writeln(i);
[6]     for j := 1 to i do
          {Печатаем добавленные числа}
[7]       write(a[N + j], ' ');
[8]     halt
[9]   end
[10] end;
```

Обратите внимание, что мы принудительно завершаем работу программы оператором **halt**, если при очередной итерации цикла получается симметричная последовательность. Можно было обойтись без этого, используя цикл **while**.

Теперь попробуем несколько модифицировать приведенный алгоритм. Дело в том, что мы выполняем лишние операции, дописывая в конец те же числа, что стоят в начале последовательности, и затем сравнивая эти числа в функции palindrom. Ясно, что достаточно проверять лишь числа, которые не дописывались в конец.

Для этого в цикле внутри функции palindrom нужно начинать проверять числа не с a[1], а с числа, которое сравнивается с числом a[N], то есть с числа a[k + 1 - N]:

```
for i := k + 1 – N to k div 2 do ...
```

Тогда и в основной программе нет нужды дополнять массив новыми числами: строки [2] и [3] можно удалить, а строку [7] заменить такой:

```
write(a[i + 1 - j],' ');
```

Отметим, что максимальная возможная длина палиндрома — 199. Это нужно учитывать при объявлении массива a.

Задача C. Кинотеатр

X мальчиков и *Y* девочек пошли в кинотеатр и купили билеты на подряд идущие места в одном ряду. Напишите программу, которая выдаст, как нужно сесть мальчикам и девочкам, чтобы рядом с каждым мальчиком сидела хотя бы одна девочка, а рядом с каждой девочкой — хотя бы один мальчик.

Формат входных данных

Во входном файле записаны два числа — *X* и *Y* (оба числа натуральные, не превосходящие 100).

Формат выходных данных

В выходной файл выведите какую-нибудь строку, в которой будет ровно *X* символов B (обозначающих мальчиков) и *Y* символов G (обозначающих девочек), удовлетворяющую условию задачи. Пробелы между символами выводить не нужно.

Если рассадить мальчиков и девочек согласно условию задачи невозможно, в выходной файл должна быть записана строка NO SOLUTION.

Примеры

c.in	c.out
5 5 5 3 100 1	BGBGBGBGBG BGBGBBGB NO SOLUTION

Решение

Заметим, что задача не всегда имеет решение: например, если мальчиков слишком много, а девочек мало (или, и того хуже, нет вовсе!), то посадить каждого мальчика рядом с девочкой не удастся. Поскольку рядом с каждой девочкой могут сидеть не более двух мальчиков, то количество мальчиков не может превосходить количества девочек более чем вдвое. То есть, если $X > 2Y$, то задача решения не имеет. Также задача не имеет решения и в случае $Y > 2X$.

Покажем, что в остальных случаях школьников можно рассадить требуемым образом. Если мальчиков и девочек поровну, то одна из искомых рассадок очевидна: BGBG...BG.

Пусть мальчиков больше, чем девочек, на k человек: $X = Y + k$. Тогда можно рассадить школьников так: $\underbrace{BGB\ BGB...BGB}_{k\text{ троек}}\ \underbrace{BG\ BG...BG}_{Y-k\text{ пар}}$.

В тройках $2k$ мальчиков и k девочек, в парах — $(Y - k)$ мальчиков и $(Y - k)$ девочек. Итого: $2k + (Y - k) = Y + k = X$ мальчиков, $k + (Y - k) = Y$ девочек. Поскольку $X \le 2Y$, то $k \le Y$, то есть $Y - k \ge 0$. Таким образом, предложенная рассадка всегда возможна. Случай, когда девочек больше, чем мальчиков, разбирается аналогично.

Остается написать программу.

```
read(X, Y);
if (X > 2 * Y) or (Y > 2 * X) then
  write('NO SOLUTION')
else if X >= Y then begin
       k := X – Y;
       for i := 1 to k do
         write('BGB');
       for i := 1 to Y – k do
         write('BG')
     end
     else begin
       k := Y – X;
       for i := 1 to k do
         write('GBG');
       for i := 1 to X – k do
         write('GB')
     end;
```

Обратим внимание на некоторые особенности работы этой программы. Если $Y = X$, то $k = 0$ и первый цикл **for** не выполняется ни разу: все школьники рассаживаются парами "мальчик — девочка". Если же $X = 2Y$ (или $Y = 2X$), не выполняется второй цикл, поскольку $Y - k$ ($X - k$, соответственно) равно нулю. В этом случае все школьники рассаживаются тройками.

Задача D. Количество слов

Во входном файле записана строка текста, в которой могут встречаться:

- прописные и строчные (т.е. большие и маленькие) латинские буквы,
- пробелы,
- знаки препинания: точка, запятая, восклицательный и вопросительный знаки,
- символ "—", обозначающий в некоторых случаях тире, а в некоторых — дефис.

Слово — это последовательность подряд идущих латинских букв и знаков "дефис", ограниченная с обоих концов. В качестве ограничителей могут выступать начало строки, конец строки, пробел, знак препинания, тире. Тире отличается от дефиса тем, что слева и справа от знака "дефис" пишутся буквы, а хотя бы с одной стороны от тире идет либо начало строки, либо конец строки, либо пробел, либо какой-либо знак препинания, либо еще одно тире.

Напишите программу, определяющую, сколько слов в данной строке текста.

Формат входных данных

Во входном файле записана строка длиной не более 200 символов.

Формат выходных данных

В выходной файл выведите одно число — количество слов, которые содержатся в исходной строке.

Примеры

d.in	d.out
Hello , world! www.olympiads.ru Gyro-compass - this is a ...	2 3 4

Решение

В каждом слове есть первая буква. Поэтому вместо того, чтобы считать слова, можно считать первые буквы слов. Какая буква является первой буквой слова? Любая, слева от которой стоит не буква. Исключение составляет буква, слева от которой стоит символ "—", а перед ним — буква. Осталось написать программу, которая перебирает все символы строки и считает, сколько среди них первых букв.

```
read(s);
for i := 1 to length(s) do
{s[i] — буква}
  if (s[i] in ['a'..'z','A'..'Z'])
  {s[i - 1] — не буква}
   and not(s[i - 1] in ['a'..'z','A'..'Z'])
  {s[i - 1] — не дефис}
   and not ((s[i - 1] = '-') and
   (s[i - 2] in ['a'..'z','A'..'Z']))
   then n := n + 1;
write(n);
```

У этой программы есть небольшой дефект. Когда мы проверяем, например, символ `s[1]`, нам приходится обращаться к несуществующим символам `s[0]` и `s[-1]`. Чтобы избежать этого, можно изначально добавить в начало строки два символа, которые не повлияют на количество слов в строке, например, `s:='!!' + s`, и начинать искать первые буквы слов, начиная с третьего символа строки:

```
for i := 3 to length(s) do ...
```

Задача E. Метро

Метрополитен состоит из нескольких линий метро. Все станции метро в городе пронумерованы натуральными числами от 1 до N. На каждой линии расположено несколько станций. Если одна и та же станция расположена сразу на нескольких линиях, то она является станцией пересадки и на этой станции можно пересесть с любой линии, которая через нее проходит, на любую другую (опять же проходящую через нее).

Напишите программу, которая по данному вам описанию метрополитена определит, с каким минимальным числом пересадок можно добраться со станции A на станцию B. Если данный метрополитен не соединяет все линии в одну систему, то может так получиться, что со станции A на станцию B добраться невозможно, в этом случае ваша программа должна это определить.

Формат входных данных

Во входном файле записано сначала число N — количество станций метро в городе ($2 \le N \le 100$). Далее записано число M — количество линий метро ($1 \le M \le 20$). Далее идет описание M линий. Описание каждой линии состоит из числа P_i — количества станций на этой линии ($2 \le P_i \le 50$) и P_i чисел, задающих номера станций, через которые проходит линия (ни через какую станцию линия не проходит дважды).

В конце файла записаны два различных числа: A — номер начальной станции и B — номер станции, на которую нам нужно попасть. При этом если через станцию A проходит несколько линий, то мы можем спуститься на любую из них. Так же если через станцию B проходит несколько линий, то нам не важно, по какой линии мы приедем.

Формат выходных данных

В выходной файл выведите минимальное количество пересадок, которое нам понадобится. Если добраться со станции A на станцию B невозможно, выведите в выходной файл одно число -1 (минус один).

Примеры

e.in	e.out
5 2 4 1 2 3 4 2 5 3 3 1	0
5 5 2 1 2 2 1 3 2 2 3 2 3 4 2 4 5 1 5	2
10 2 6 1 3 5 7 4 9 6 2 4 6 8 10 7 3 8	1
4 2 1 2 3 4 1 3	-1

Решение

Идею решения этой задачи можно кратко описать так. Рассмотрим все линии, на которых расположена станция *A*. До любой станции на любой из этих линий можно добраться со станции *A* *без пересадок*. Пометим все эти линии числом 0. Теперь рассмотрим все непомеченные линии, у которых есть общие станции с линиями, помеченными числом 0. До любой станции на любой из этих линий можно добраться со станции *A*, используя *одну пересадку*, но нельзя добраться без пересадок. Пометим эти линии числом 1. Далее аналогично рассмотрим все непомеченные линии, у которых есть общие станции с линиями, помеченными числом 1. На любую станцию этих линий можно попасть, используя *две пересадки*, поэтому пометим их числом 2, и т.д. до тех пор, пока мы впервые не пометим одну из линий, на которых есть станция *B*. Число, которым помечена эта линия, и будет являться ответом в нашей задаче. (Такой алгоритм называют *волновым*.)

Теперь займемся реализацией этого алгоритма. Проще всего написать программу, используя структуру данных *множество* (**set**). Пусть `L[i]` — множество станций метро, лежащих на `i`-й линии (то есть `L` — массив множеств), `current` — множество станций, на которых можно оказаться, используя не более `step` пересадок. Построим множество `next` станций, на которых можно оказаться, используя не более `(step + 1)` пересадок. Во-первых, все станции из множества `current` входят во множество `next`. Во-вторых, если на какую-то станцию можно попасть за не более чем `step` пересадок, и эта станция принадлежит в том числе и линии `i`, то тогда на любую станцию линии `i` можно попасть за не более чем `(step + 1)` пересадок. То есть, если множество `current * L[i]` не пусто (звездочкой обозначено пересечение множеств, то есть их общая часть), все элементы множества `L[i]` нужно включить во множество `next`:

```
next := next + L[i]; {объединение множеств}
```

Осталось заметить, что если метро состоит из *M* линий, то либо со станции *A* на станцию *B* можно попасть не более чем с $(M - 1)$-й пересадкой, либо нельзя попасть вовсе (*A* и *B* лежат в разных *компонентах связности* соответствующего графа).

Теперь напишем тело программы.

```
begin
{Чтение данных из входного файла}
  read(N, M);
  for i := 1 to M do begin
    read(k);
    for j := 1 to k do begin
      read(t);
   {Добавляем станцию t во множество L[i]}
      Include(L[i], t)
    end
  end;
  read(A, B);
{Инициализация}
  emptyset := []; {Пустое множество}
  step := 0;
  current := [A];
{Основной цикл}
  for step := 0 to M - 1 do begin
    next := current;
    for i := 1 to M do
      if current * L[i] <> emptyset then
        next := next + L[i];
    if B in next then begin
      write(step); halt
    end;
    current := next
  end;
{Пути от станции A до станции B нет}
  write(-1);
end.
```

Готовимся к олимпиадам по информатике

От редакции. Задачи олимпиад по информатике, несмотря на большое разнообразие, очень часто можно отнести к той или иной теме. В данном выпуске мы рассмотрим три таких темы: динамическое программирование, алгоритмы на графах и вычислительную геометрию. Задачи на эти темы стали уже почти обязательным "олимпиадным минимумом", хотя соответствующие алгоритмы не входят в школьную программу по информатике.

В настоящей тематической подборке использованы задачи различных соревнований. Материалы городской олимпиады (г. Нижний Новгород, 2004—2005 уч. г.) подготовлены предметной комиссией олимпиады под председательством В.Д. Лелюха. Задачи командных соревнований школьников Свердловской области по программированию и их решения были собраны в одноименной книге А.В. Ипатовым и любезно предоставлены редакции автором. Протестировать решение этих задач можно на замечательном сайте ***acm.timus.ru***. Кроме того, в подборке приведены задачи различных московских соревнований двух последних лет, тесты к которым можно найти на сайте ***www.olympiads.ru***, а также задача Всероссийской олимпиады школьников по информатике 2005 г.

Все упомянутые задачи публикуются в нашей газете впервые. Мы надеемся, что данные материалы помогут педагогам в занятиях с одаренными школьниками, а будущим участникам соревнований — выступать на олимпиадах по информатике и программированию на равных с остальными.

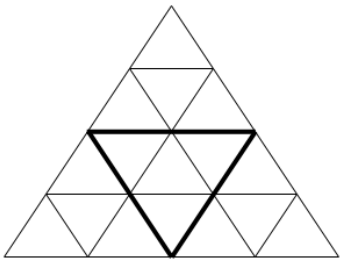
Динамическое программирование

Задача "Количество треугольников" (Московская командная олимпиада 2005–2006 уч. г.)

Автор задачи — А.А. Петров
Автор разбора — В.М. Гуровиц

Рассмотрим фигуру, аналогичную показанной на рисунке (большой равносторонний треугольник, составленный из маленьких равносторонних треугольников). На рисунке приведена фигура, состоящая из четырех уровней треугольников.

Напишите программу, которая будет определять, сколько всего в ней треугольников (необходимо учитывать не только "маленькие" треугольники, а вообще все треугольники — в частности, треугольник, выделенный жирным, а также вся фигура являются интересующими нас треугольниками).



Формат входных данных

Во входном файле записано одно число N — количество уровней в фигуре ($1 \le N \le 100\,000$).

Формат выходных данных

Выведите в выходной файл количество треугольников в такой фигуре.

Примеры

Входной файл	Выходной файл
1 2 4	1 5 27

Решение

Заметим, что все возможные треугольники — равносторонние, и одна из их сторон — горизонтальная. Такие треугольники могут быть двух типов: горизонтальная сторона снизу (треугольник ориентирован как вся фигура) или горизонтальная сторона сверху (треугольник ориентирован как выделенный треугольник на рисунке из условия).

Подсчитаем сначала количество треугольников первого типа. Заметим, что каждый такой треугольник однозначно задается своим основанием, а значит, парой его концов — узлов сетки, лежащих на одной горизонтали. Если в некоторой горизонтали k узлов, то пару точек можно выбрать $k(k-1)/2$ способами, то есть существуют $k(k-1)/2$ треугольников с таким основанием. Если фигура состоит из n уровней, то в верхней горизонтали имеются 2 узла, в следующей — 3, ..., в последней — $n(n+1)/2$. Таким образом, количество треугольников первого типа равно $\frac{1}{2}(1\cdot 2 + 2\cdot 3 + \ldots + n(n+1))$. Для предложенных в за-

даче ограничений на входные данные можно вычислить эту сумму, используя цикл. Но можно вычислить эту сумму в явном виде, не используя компьютер.

Чуть сложнее подсчитать количество треугольников второго типа. Покажем один из возможных способов это сделать. Для удобства обозначим $S_n = 1 + 2 + 3 + ... + n = n(n+1)/2$. Подсчитаем отдельно количество треугольников со стороной 1, со стороной 2, со стороной 3 и т.д. Из рисунка видно, что эти количества равны S_{n-1}, S_{n-3}, S_{n-5}, ... соответственно (слагаемые каждой из сумм S_i — это количества треугольников соответствующего размера в одном горизонтальном ряду). Таким образом, общее количество треугольников второго типа равно $S_{n-1} + S_{n-3} + S_{n-5} + ... = (n(n-1) + (n-2)(n-3) + (n-4)(n-5) + ...)/2$ (последнее слагаемое в этой сумме равно S_2 или S_1 в зависимости от четности n). Эту сумму также можно сосчитать, используя цикл, а можно упростить, записав ответ в явном виде. Общее количество треугольников обоих типов равно $k(k+1)(4k+1)/2$, если $n = 2k$, и $(k+1)(4k^2+7k+2)/2$, если $n = 2k+1$. Доказать это можно по индукции.

Задача "Представление числа"
(Московская командная олимпиада 2005–2006 уч. г.)

Автор задачи и разбора — А.Ю. Гусаков

Учительница математики попросила школьников составить арифметическое выражение так, чтобы его значение было равно данному числу N, и записать его в тетради. В выражении могут быть использованы натуральные числа, не превосходящие K, операции сложения и умножения, а также скобки. Петя очень не любит писать и хочет придумать выражение, содержащее как можно меньше символов. Напишите программу, которая поможет ему в этом.

Формат входных данных

В первой строке входного файла записаны два натуральных числа: N ($1 \le N \le 10\,000$) — значение выражения и K ($1 \le K \le 10\,000$) — наибольшее число, которое разрешается использовать в выражении.

Формат выходных данных

В единственной строке выходного файла выведите выражение с данным значением, записывающееся наименьшим возможным количеством символов.

Если решений несколько, выведите любое из них.

Примечание

При подсчете длины выражения учитываются все символы: цифры, знаки операций, скобки.

Примеры

Входной файл	Выходной файл
7 3 15 20 176 1	3+1+3 15 (1+1+1+1)*(1+1+1+1)* (1+1+(1+1+1)*(1+1+1))

Решение

Прежде чем перейти к описанию идеи решения, введем несколько определений.

Будем называть выражение C *выражением первого типа*, если оно является числом от 1 до K, или если последней операцией при его вычислении является сложение, то есть $C = A + B$, где A и B — некоторые выражения.

Будем называть выражение C *выражением второго типа*, если оно является числом от одного до K или если последней операцией при его подсчете является умножение, то есть $C = A \cdot B$, где A и B — некоторые выражения. Например, $(2 + 3 \cdot 4) + 5 \cdot 7$ — выражение первого типа, $(2 + 3) \cdot (3 + 2)$ — выражение второго типа, 1 — выражение, относящееся как к первому, так и ко второму типу.

Теперь обсудим идею решения. Пусть `a[1,m]` равно наименьшей длине выражения первого типа, значение которого равно `m`; `a[2,m]` равно наименьшей длине выражения второго типа, значение которого равно `m`. Легко видеть, что при `m` ≤ `K` значения `a[1,m]` и `a[2,m]` равны длине числа `m`. Пусть теперь `m > K`, и у нас уже вычислены все значения `a[1,1]`, ..., `a[1,m-1]`, `a[2,1]`, ..., `a[2,m-1]`.

Сначала вычислим `a[1,m]`. Поскольку соответствующее выражение представимо в виде $C = A + B$, нам достаточно перебрать все возможные значения и типы выражений A и B и выбрать оптимальную комбинацию. Иными словами, `a[1,m] = min(a[i1,t] + a[i2,m - t] + 1)`, где `i1`, `i2` — 1 или 2, а `t` ≤ `m/2` (можно считать, что значение A меньше или равно $m/2$, иначе A и B можно поменять местами). В переменных `how[1,m][1]`, `how[1,m][2]`, `how[1,m][3]` будем хранить те значения `i1`, `t`, `i2` соответственно, при которых достигается минимальное значение.

Теперь вычислим `a[2,m]`. Соответствующее выражение представимо в виде $A \cdot B$, где A и B — либо выражения второго типа, либо выражения первого типа, взятые в скобки (либо какая-то их комбинация). Будем считать, что значение A меньше или равно, иначе поменяем местами A и B. Таким образом, `a[2,m] = min(a[i1,t] + a[i2,m/t] + 1 + 2*(4-i1-i2))`, где `i1`, `i2` — 1 или 2, а `t` ≤ $\sqrt{m}$ и `m` делится на `t`.

Замечание. Число `4 - i1 - i2` равно количеству выражений первого типа среди A и B, а значит, равно количеству пар скобок, которое придется добавить.

В переменных `how[2,m][1]`, `how[2,m][2]`, `how[2,m][3]` будем хранить те значения `i1`, `t`, `i2` соответственно, при которых достигается минимум.

Теперь мы умеем вычислять значения `a[1,1]`, ..., `a[1,N]`, `a[2,1]`, ..., `a[2,N]`. Естественно, что искомое выражение является выражением первого или второго типов, поэтому его длина равна `min(a[1,N], a[2,N])`. Осталось только научиться его выводить. Для этого напишем рекурсивную процедуру `Save(last, n)`, которая будет выписывать выражение типа `last`, значением которого является `n`. Во-первых, если `n` ≤ `K`, то нужно просто вывести число `cur`. Иначе нужно посмотреть на значение `last`.

Если `last = 1` (то есть выражение имеет вид $A + B$), то сначала выпишем первое слагаемое — `Save(how[last,n][1], how[last,n][2])`, затем поставим знак '+' и выпишем второе слагаемое — `Save(how[last,n][3], n - how[last,n][2])`.

Если `last = 2` (то есть имеем дело с произведением), то нужно действовать аналогично случаю `last = 1`, но при этом не забыть, что если какой-то из множителей первого типа, то его надо окружить скобками.

Количество операций, выполняемых алгоритмом, пропорционально N^2.

Оказывается, что приведенную динамическую схему можно упростить. Достаточно для каждого числа `m` хранить информацию лишь о самом коротком выражении, независимо от его типа. При этом, если для данного числа самое короткое выражение может быть как выражением типа 1, так и выражением типа 2, то нужно выбирать произведение, а не сумму. Обоснование правильности этого алгоритма оставим читателю.

Задача "Кафе"
(Московская командная олимпиада 2004–2005 уч. г.)

Автор задачи — А.П. Лахно
Автор разбора — В.Ю. Антонов

Около Петиного университета недавно открылось новое кафе, в котором действует следующая система скидок: при каждой покупке более чем на 100 рублей покупатель получает купон, дающий право на один бесплатный обед (при покупке на сумму 100 рублей и меньше такой купон покупатель не получает).

Однажды Пете на глаза попался прейскурант на ближайшие N дней. Внимательно его изучив, он решил, что будет обедать в этом кафе все N дней, причем каждый день он будет покупать в кафе ровно один обед. Однако стипендия у Пети небольшая, и поэтому он хочет по максимуму использовать предоставляемую систему скидок так, чтобы его суммарные затраты были минимальны. Требуется найти минимально возможную суммарную стоимость обедов и номера дней, в которые Пете следует воспользоваться купонами.

Формат входных данных

В первой строке входного файла записано целое число N ($0 \le N \le 100$). В каждой из последующих N строк записано одно целое число, обозначающее стоимость обеда в рублях на соответствующий день. Стоимость — неотрицательное целое число, не превосходящее 300.

Формат выходных данных

В первой строке выдайте минимальную возможную суммарную стоимость обедов. Во второй строке выдайте два числа: K_1 и K_2 — количество купонов, которые останутся неиспользованными у Пети после этих N дней, и количество использованных им купонов соответственно.

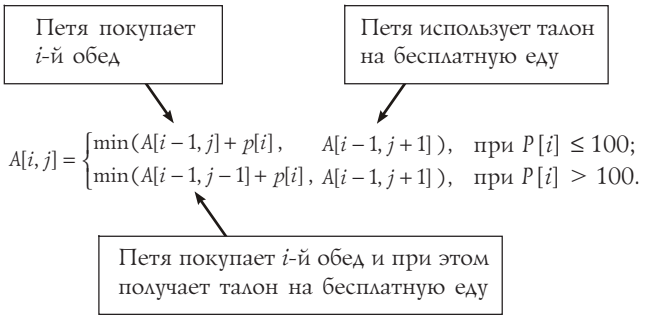
В последующих K_2 строках выдайте в возрастающем порядке номера дней, когда Пете следует воспользоваться купонами. Если существует несколько решений с минимальной суммарной стоимостью, то выдайте то из них, в котором значение K_1 максимально (на случай, если Петя когда-нибудь еще решит заглянуть в это кафе). Если таких решений несколько, выведите любое из них.

Пример

Входной файл	Выходной файл
5 35 40 101 59 63	235 0 1 5

Решение

Заведем двумерный массив A, в ячейке $A[i,j]$ которого будет храниться минимальная сумма, которую мог потратить Петя за первые i дней, при условии, что после этого у него в запасе осталось j талонов на бесплатную еду. Положим $A[0,0] = 0$; $A[0,i] = \infty$ при $i > 0$. Пусть $P[i]$ — цена обеда в i-й день. Для заполнения ячейки $A[i,j]$ можно воспользоваться следующей формулой:



Ответом будет являться минимальное из чисел (назовем его Ans), хранящихся в N-м столбце. Рассмотрим все элементы N-го столбца, значения которых совпадают с Ans. Понятно, что K_1 — максимальный номер строки из всех номеров строк этих элементов. Как же посчитать параметр K_2, который также нужно вывести? Для этого дополнительно будем хранить в $B[i,j]$ общее количество купонов на бесплатные обеды, полученных Петей (считается по формулам, сходным с формулами для вычисления $A[i,j]$). Можно не заводить второй массив, а считать K_2 непосредственно перед выводом ответа.

Остается вывести дни, в которые были потрачены купоны. Для ячейки массива $A[i,j]$ по числам в столбце $A[i-1]$ можно определить, был ли использован купон в i-й день. А зная K_1, можно вывести все дни, в которые были потрачены купоны, и подсчитать их количество K_2.

Задача "Скобки"
(Московская командная олимпиада 2004–2005 уч. г.)

Авторы задачи — Р.А. Жуйков, М.О. Трухина
Автор разбора — В.Ю. Антонов

Назовем строку S правильной скобочной последовательностью, если она состоит только из символов '**{**', '**}**',

‘[’, ‘]’, ‘(’, ‘)’ и выполнено хотя бы одно из следующих трех условий:

1) S — пустая строка;

2) S можно представить в виде $S = S_1 + S_2 + S_3 + ... + S_N$ ($N > 1$), где S_i — непустые правильные скобочные последовательности, а знак ‘+’ обозначает конкатенацию (приписывание) строк;

3) S можно представить в виде $S =$ ‘{’ + C + ‘}’ или $S =$ ‘[’ + C + ‘]’ или $S =$ ‘(’ + C + ‘)’, где C является правильной скобочной последовательностью.

Дана строка, состоящая только из символов ‘{’, ‘}’, ‘[’, ‘]’, ‘(’, ‘)’. Требуется определить, какое минимальное количество символов надо вставить в эту строку для того, чтобы она стала правильной скобочной последовательностью.

Формат входных данных

В первой строке входного файла записана строка, состоящая только из символов ‘{’, ‘}’, ‘[’, ‘]’, ‘(’, ‘)’. Длина строки не превосходит 100 символов.

Формат выходных данных

Вывести в первую строку выходного файла единственное неотрицательное целое число — ответ на поставленную задачу.

Примеры

Входной файл	Выходной файл
{()} ([{}])	2 0

Решение

Пусть задана строка S, состоящая из символов ‘{’, ‘}’, ‘[’, ‘]’, ‘(’, ‘)’. Заведем двумерный массив A, в элементе $A[i,j]$ которого будем хранить минимальное количество скобок, которые нужно добавить к подстроке строки S с i-го по j-й символ ($S[i]..S[j]$), чтобы получить правильную скобочную последовательность. Если $i > j$, то положим $A[i,j] = 0$. Будем находить решения, постепенно увеличивая разность между j и i, учитывая, что $A[i,i] = 1$ для любого i (подумайте, почему). При увеличении разности между j и i вычисление $A[i,j]$ осуществляется по следующим правилам:

1) если $S[i]$ — закрывающая скобка, то в правильной скобочной последовательности до нее должна быть открывающая скобка, т.е. $A[i,j] = 1 + A[i+1,j]$;

2) если $S[i]$ — открывающая скобка, то возможно несколько вариантов восстановления правильной скобочной последовательности:

а) среди остальных скобок существует пара для этой, тогда: $A[i,j] = min(A[i+1,k-1] + A[k+1,j])$, где минимум берется по всем k, таким, что $S[k]$ — закрывающая скобка, соответствующая открывающей скобке $S[i]$;

б) соответствующую закрывающую скобку нужно добавить к строке, т.е. $A[i,j] = min(A[i+1,k] + A[k+1,j])$ при $i \le k \le j$ (нужную закрывающую скобку вставляем после k-й скобки).

$A[i,j]$ равен минимуму из этих двух вариантов.

Ответом будет являться $A[1,Length(S)]$.

Задача “Числа”

(I городская олимпиада, г. Нижний Новгород, 2004–2005 уч. г.)

Дана последовательность чисел $a_1, a_2, ..., a_N$. За одну операцию разрешается удалить любое (кроме крайних) число, заплатив за это штраф, равный произведению этого числа на сумму соседних. Требуется удалить все числа, кроме крайних, с минимальным суммарным штрафом.

Например.

Начальная последовательность: 1 50 51 50 1

Удаляем четвертое число, штраф $50(1 + 51) = 2600$, получаем 1 50 51 1

Удаляем третье число, штраф $51(50 + 1) = 2601$, получаем 1 50 1

Удаляем второе число, штраф $50(1 + 1) = 100$.

Итого штраф 5301.

Формат входных данных

В первой строке входного файла расположено одно число N ($1 \le N \le 100$) — количество чисел в последовательности.

Во второй строке находятся N целых чисел $a_1, a_2, ..., a_N$; никакое из чисел не превосходит по модулю 100.

Формат выходных данных

Выведите в выходной файл одно число — минимальный суммарный штраф.

Пример

Входной файл	Выходной файл
5 1 50 51 50 1	5301

Решение

Заведем квадратную матрицу M размера $N \times N$. В ячейке M_{ij} будем хранить минимальный штраф за удаление всех чисел с i-го по j-е не включительно (элементы матрицы, у которых $i \ge j$, нам не будут нужны). Подсчитывать эти штрафы будем следующим образом: поскольку среди чисел между i-м и j-м какое-то (k-е) мы должны удалить последним, то переберем все k от i до j не включительно и посчитаем наименьший штраф за удаление всех чисел между i и j при условии, что k-е удаляется последним. Этот штраф равен сумме наименьших штрафов за удаление всех чисел c i по k (M_{ik}), с k по j (M_{kj}) и штрафа за последнее удаление — $(a_i + a_j)a_k$. Тогда очевидно, что

$$M_{ij} = \min_{i<k<j}(M_{ik} + M_{kj} + (a_i + a_j)a_k).$$

Несложно посчитать элементы матрицы с индексами, отличающимися на 1: так как удалять между соседними числами нечего, то $M_{i,i+1} = 0$.

Осталось только вычислить все элементы матрицы (например, в порядке увеличения разности между индексами) и посмотреть на число M_{1N}, которое и будет ответом на задачу.

Кроме того, надо аккуратно обработать случай $N = 1$. Вышеприведенный алгоритм для этого случая не подходит, но очевидно, что, так как удалять при $N = 1$ нечего, ответ здесь — 0.

Задача "Казино"

(XVIII Всероссийская олимпиада школьников по информатике, 2005 г.)

Вновь открытое казино предложило оригинальную игру.

В начале игры крупье выставляет в ряд несколько фишек разных цветов. Кроме того, он объявляет, какие последовательности фишек игрок может забирать себе в процессе игры. Далее игрок забирает себе одну из заранее объявленных последовательностей фишек, расположенных подряд. После этого крупье сдвигает оставшиеся фишки, убирая разрыв. Затем игрок снова забирает себе одну из объявленных последовательностей и так далее. Игра продолжается до тех пор, пока игрок может забирать фишки.

Рассмотрим пример. Пусть на столе выставлен ряд фишек *rrrgggbbb*, и крупье объявил последовательности *rg* и *gb*. Игрок, например, может забрать фишки *rg*, лежащие на третьем и четвертом местах слева. После этого крупье сдвинет фишки, и на столе получится ряд *rrggbbb*. Еще дважды забрав фишки *rg*, игрок добьется того, что на столе останутся фишки *bbb* и игра закончится, так как игроку больше нечего забрать со стола. Игрок мог бы действовать и по-другому — на втором и третьем ходах забрать не последовательности *rg*, а последовательности *gb*. Тогда на столе остались бы фишки *rrb*. Аналогично, игрок мог бы добиться того, чтобы в конце остались ряды *rrr* или *rbb*.

После окончания игры полученные фишки игрок меняет на деньги. Цена фишки зависит от ее цвета.

Требуется написать программу, определяющую максимальную сумму, которую сможет получить игрок.

Формат входных данных

В первой строке входного файла записано число K ($1 \le K \le 26$) — количество цветов фишек. Каждая из следующих K строк начинается со строчной латинской буквы, обозначающей цвет. Далее в той же строке через пробел следует целое число X_i ($1 \le X_i \le 150$, $i = 1..K$) — цена фишки соответствующего цвета.

В $(K + 2)$-й строке описан ряд фишек, лежащих на столе в начале игры. Ряд задается L строчными латинскими буквами ($1 \le L \le 150$), которые обозначают цвета фишек ряда.

В следующей строке содержится число N ($1 \le N \le 150$) — количество последовательностей, которые были объявлены крупье. В следующих N строках записаны эти последовательности. Гарантируется, что сумма длин этих N строк не превосходит 150 символов и все они непустые.

Формат выходных данных

В выходной файл выведите единственное целое число — максимальную сумму денег, которую может получить игрок.

Пример

Входной файл	Выходной файл
6 a 1 b 4 d 2 x 3 f 1 e 3 fxeeabadd 2 aba ed	16

Решение

Во-первых, заметим, что не любая стратегия игрока приводит к нужному результату. Например, если в примере, разбиравшемся в условии задачи (*rrrgggbbb*), стоимость буквы *b* будет больше стоимости буквы *r*, то только один из путей приводит к оптимальному результату: три раза забрать буквы *gb*. После некоторого размышления можно понять, что всевозможные так называемые "жадные" решения — делать самый выгодный ход, делать самый длинный ход, делать самый левый ход и т.п. — также в большинстве случаев неверны. Перебирать же все варианты, для каждого из них — опять все варианты, и так далее — оказывается слишком медленным подходом.

Для решения этой задачи применим динамическое программирование. В первой половине решения мы для каждого куска начальной последовательности фишек определим, можем ли мы забрать его целиком, не затрагивая при этом остальные фишки. Иными словами, для всех l и r найдем величину $a[l,r]$, равную 1, если фишки с номерами (то есть позициями в начальном расположении, считая слева) с l по r игрок может за несколько действий все забрать себе, и 0 — в противном случае. Затем по этим данным во второй половине решения мы восстановим ответ к задаче.

Во избежание путаницы будем называть последовательности, объявленные крупье (в отличие от начальной последовательности), *словами*.

Вторая половина решения проще первой, поэтому начнем с нее. Пусть нам известны величины $a[l,r]$. Пусть тогда $b[l,r]$ — это максимальная сумма денег, которую можно получить, играя только на отрезке с l-й фишки по r-ю (то есть не трогая остальные фишки). Ясно, что если $a[l,r] = 1$, то $b[l,r]$ — это просто сумма стоимостей всех фишек с l-й по r-ю. Если же $a[l,r] = 0$, то уже не удастся забрать все фишки, соответственно, какая-то из фишек должна остаться — пусть это фишка с номером k. Тогда заметим, что задача распадается на две — для фишек слева от k-й и для фишек справа от нее, т.е. что $b[l,r] = b[l,k-1] + b[k+1,r]$ (здесь и далее мы считаем, что для любого m $b[m,m-1] = 0$). Но так как номер оставшейся фишки не фиксирован, то в действительности $b[l,r]$ равно максимуму этих величин по всем k:

$$b[l,r] = \max_{k=l,l+1,\ldots,r} (b[l,k-1] + b[k+1,r]) \qquad (*)$$

(напомним, что эта формула верна при $a[l,r] = 0$). Теперь мы видим, что для нахождения величин $b[l,r]$ можно применить динамическое программирование: если мы будем перебирать r от 1 до длины начальной последовательности, а затем l от r до 1, то к моменту вычисления величины $b[l,r]$ все величины, которые необходимы для ее нахождения по формуле (*), уже вычислены. Таким образом, зная величины $a[l,r]$, мы можем найти $b[l,r]$. Легко видеть, что на этом шаге требуется порядка L^3 действий (напомним, что L — это длина начальной последовательности).

Теперь вернемся к первой половине решения, а именно — нахождению величин $a[l,r]$. Как узнать, можно ли забрать себе данную последовательность (в нашем случае — кусок начальной) целиком? Во-первых, возможно, эту последовательность можно разделить на две части, каждую из которых можно забрать себе целиком. То есть, если найдется k такое, что $a[l,k] = 1$ и $a[k + 1,r] = 1$, то и $a[l,r] = 1$. Если же так сделать не удается, то можно заметить, что если мы все-таки заберем всю последовательность целиком, то ее первую и последнюю фишки мы заберем только на последнем шаге. Пусть на последнем шаге мы заберем некоторое слово S. Тогда задача сводится к тому, чтобы выкинуть из последовательности некоторые куски (можно считать, что для этих кусков величина a уже посчитана, поэтому мы знаем, какие из них можно выкинуть, а какие — нет), оставив первую и последнюю фишки на месте, так, чтобы осталось слово S.

Для решения этой задачи опять применим динамическое программирование. А именно, пусть $c[l,r,p,q] = 1$, если из отрезка с l-й фишки по r-ю можно так выкинуть несколько кусков, чтобы остались первые q фишек слова номер p, и при этом l-я и r-я фишки остались на месте, и 0 — иначе (тогда нас интересует величина $c[l,r,p,length(p)]$, где $length(p)$ — длина слова номер p). Во-первых, ясно, что если первая фишка слова номер p не совпадает с l-й фишкой начальной последовательности, или q-я — с r-й, то $c[l,r,p,q] = 0$. В противном случае пусть $(q - 1)$-я фишка слова получится из k-й фишки последовательности. Тогда, во-первых, должно быть $c[l,k,p,q - 1] = 1$ (чтобы получить первые $q - 1$ фишек), а, во-вторых, $a[k + 1, r - 1] = 1$ (чтобы выкинуть кусок между $(q - 1)$-й фишкой и q-й). Но так как число k не фиксировано, то получаем, что $c[l,r,p,q] = 1$ тогда и только тогда, когда найдется такое k. Тем самым получен алгоритм для подсчета c. Оценим время его работы. На подсчет одного элемента массива c требуется порядка L действий (по количеству возможных k). Нам необходимо подсчитать элементы вида $c[l,r,p,length(p)]$. Из вышеприведенных рассуждений легко заметить, что для их подсчета нам потребуется подсчет только элементов вида $c[l,...,...,...]$, а элементов такого вида порядка $L \cdot S$ (где S — сумма длин всех слов), так как для параметра r есть порядка L вариантов, а для пары параметров (p, q) — порядка S вариантов. Тем самым мы для любых l и r можем найти $a[l,r]$ за порядка $L \cdot L \cdot S = L^2 \cdot S$ действий, а общее время работы получается порядка $L^2 \cdot L^2 \cdot S = L^4 \cdot S$.

Однако легко заметить, что если подсчитать все величины $c[l,r,p,q]$ сначала, а потом лишь использовать найденные значения, то время работы будет меньше. Действительно, всего в массиве c $L^2 \cdot S$ элементов, и на подсчет каждого из них уйдет порядка L действий, тем самым общее время работы будет порядка $L^3 \cdot S$.

Алгоритмы на графах

Задача "День объединения"

(I командное соревнование школьников Свердловской области по программированию, 2000 г.)

Автор задачи — А.Ботов
Автор разбора — А.Ипатов

Исландия — страна, расположенная на множестве мелких островков. Прошло уже много времени с того момента, когда был построен первый мост. Сейчас это государство полностью соединено мостами, и с любого острова можно попасть на любой другой, не пользуясь лодкой. Недавно жители решили добавить к уже большому списку праздников еще один — День Объединения. Как вы, наверное, уже догадываетесь, так они решили назвать тот день, в который жители перестали нуждаться в лодках для перемещения между любыми двумя островами. К сожалению, было построено очень много мостов, и разобраться, когда же настал этот день, оказалось не совсем просто.

Жители Исландии будут вам очень благодарны, если вы напишете программу, которая определит по датам постройки мостов, когда же наступил этот знаменательный день. Итак, надо определить тот день, в который все острова оказались соединенными (пусть даже через другие острова) мостами.

Формат входных данных

Во входном файле для этой задачи вы сможете найти: в первой строке два числа N и K — количество островов и мостов соответственно ($2 \le N \le 1000$, $1 \le K \le 10\,000$). В последующих K строках вам будет дана информация о мостах в следующем виде: "*a b date*", где *date* — дата в формате *dd/mm/yyyy*, а числа a и b — номера островов, которые соединил построенный мост. Записи в файле упорядочены по времени, последний построенный мост — в конце файла. Гарантируется, что после постройки последнего моста в Исландии с любого острова можно попасть на любой другой, пользуясь только мостами (возможно несколькими).

Формат выходных данных

Выходной файл должен содержать единственную дату, записанную в том же формате, что и во входном файле, соответствующую тому дню, когда жители Исландии смогли ходить друг к другу в гости пешком.

Пример

Входной файл	Выходной файл
5 10 1 3 04/10/1607 5 4 25/05/1665 2 5 13/12/1746 5 4 06/06/1799 2 4 01/04/1815 3 1 22/11/1836 3 5 17/08/1855 4 1 15/11/1897 5 1 21/04/1931 2 3 14/05/1967	17/08/1855

Решение

Рассмотрим все острова и мосты между ними (такая модель в математике называется графом, острова называются вершинами графа, а мосты — ребрами графа). Будем говорить, что группа островов связна, если с любого острова группы можно попасть на любой другой ее остров, пользуясь только мостами. Так, все острова можно разбить на связные группы (назовем их компонентами связности), причем до построения первого моста каждый остров представлял собой отдельную компоненту связности, а после построения последнего все острова лежат в одной компоненте связности. Назовем остров изолированным, если с него нельзя попасть ни на какой другой остров, пользуясь только мостами. В начале все острова являются изолированными. Назовем вновь построенный мост ab полезным, если до его постройки нельзя было попасть с острова a на остров b, пользуясь только мостами. В противном случае назовем мост ab бесполезным. Построение каждого полезного моста уменьшает количество компонент связности ровно на 1, следовательно, всего полезных мостов $N - 1$, и День Объединения наступает после построения последнего полезного моста. Рассмотрим алгоритм раскраски, решающий данную задачу: заведем массив из N элементов $color[i]$, указывающий цвет вершины(острова) i — номер компоненты связности, в которой она лежит. При этом будем нумеровать только компоненты, состоящие более чем из одной вершины, то есть для изолированных вершин в массиве $color$ будет стоять 0. Затем последовательно считываем мосты, определяя, является ли следующий мост полезным, или нет. Мост ij является бесполезным, если $color[i] = color[i] > 0$. Иначе увеличиваем счетчик полезных мостов $useful$. При этом выполняем следующие действия: если обе вершины не покрашены, то красим их цветом с номером $useful$. Если одна вершина покрашена, а вторая — нет, то красим вторую цветом первой. Если обе вершины покрашены, но в разные цвета, красим все вершины, покрашенные в цвет второй, цветом первой.

Фактически мы воспользовались алгоритмом построения остова в графе.

Задача "Генеалогическое дерево"

(II командное соревнование школьников Свердловской области по программированию, 2000 г.)

Автор задачи — Л.Волков
Автор разбора — А.Ипатов

Система родственных отношений у марсиан достаточно запутана. Собственно говоря, марсиане почкуются когда им угодно и как им угодно, собираясь для этого разными группами, так что у марсианина может быть и один родитель, и несколько десятков, а сотней детей сложно кого-нибудь удивить. Марсиане привыкли к этому, и такой жизненный уклад кажется им естественным.

А вот в Планетарном Совете запутанная генеалогическая система создает серьезные неудобства. Там заседают достойнейшие из марсиан, и поэтому, чтобы никого не обидеть, во всех обсуждениях слово принято предоставлять по очереди, так, чтобы сначала высказывались представители старших поколений, потом те, что помладше, и лишь затем уже самые юные и бездетные марсиане. Однако соблюдение такого порядка на деле представляет собой совсем не простую задачу. Не всегда марсианин знает всех своих родителей, что уж тут говорить про бабушек и дедушек! Но когда по ошибке сначала высказывается прапрадед, а потом только молодо выглядящий прапрадед — это настоящий скандал.

Ваша цель — написать программу, которая определила бы раз и навсегда такой порядок выступлений в Планетарном Совете, который гарантировал бы, что каждый член Совета получает возможность высказаться раньше любого из своих потомков.

Формат входных данных

В первой строке входного файла к этой задаче находится единственное число N, $1 \le N \le 100$ — количество членов Марсианского Планетарного Совета. По многовековой традиции все члены Совета нумеруются натуральными числами от 1 до N. Далее во входном файле следуют ровно N строк, причем I-я строка содержит список детей члена Совета с порядковым номером I. Список детей представляет собой последовательность порядковых номеров детей, разделенных пробелами и следующих в произвольном порядке. Список детей может быть пустым. Список детей (даже если он пуст) оканчивается нулем.

Формат выходных данных

Выходной файл должен в своей единственной строке содержать последовательность номеров выступающих, разделенных пробелами. Если несколько последовательностей удовлетворяют условиям задачи, то можно вывести любую из них.

Пример

Входной файл	Выходной файл
5 0 4 5 1 0 1 0 5 3 0 3 0	2 4 5 3 1

Решение

Проще найти обратный порядок выступления, то есть такой порядок, что любой марсианин выступает в Совете позже своих потомков. Поймем, что в любой группе марсиан есть марсианин, не имеющий детей. Если бы это было не так, мы бы взяли цепочку марсиан A_1, A_2, A_3, ..., A_k, ..., где A_k — сын A_{k-1}. В силу того, что марсиан конечное число, эта цепочка оборвется на A_m. Марсианин A_m не имеет детей.

Теперь возьмем марсианина, не имеющего детей, и заставим его выступать первым. В оставшейся группе вновь найдем марсианина, не имеющего детей среди марсиан этой группы, и заставим его выступать вторым и т.д. Всегда будет получаться, что имеющий детей марсианин выступает в Совете после них. Данный алгоритм решает задачу топологической сортировки вершин в ориентированном графе родственных отношений.

После того, как мы найдем такой порядок выступления марсиан, обратим его и получим нужный нам порядок.

Есть несколько реализаций этого алгоритма, использующих различные структуры данных. Мы для работы будем использовать такую структуру, как матрица смежности графа, — двумерный массив логического типа $son[i][j] = \begin{cases} 1\,(true), & j \text{ — сын } i; \\ 0\,(false), & j \text{ — не сын } i. \end{cases}$

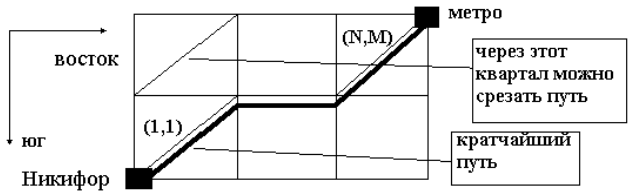
Эта реализация не является самой быстрой, но зато очень проста в понимании. Заведем еще массив $used[i]$, отмечающий, выступал ли уже марсианин с номером i, или нет, и массив $n\_of\_sons[i]$, показывающий количество детей у i-го марсианина. После ввода данных матрица смежности заполнена, и $n\_of\_sons[i]$ = *количеству элементов true в строке* $son[i]$. Приведем основной фрагмент решения задачи:

```
for i := 1 to n do
  begin
    for j := 1 to n do
      if not used[j] and (n_of_sons[j] = 0)
         then break;
    backorder[i] := j;
    used[j] := true;
    for k := 1 to n do
      if son[k][j] then
        begin
          dec(n_of_sons[k]);
          son[k][j] := false
        end;
  end;
for i := n downto 1 do
  write(backorder[i],' ');
```

Задача "Метро"

(IV командное соревнование школьников Свердловской области по программированию, 2001 г.)

Автор задачи — Л.Волков
Автор разбора — А.Ипатов



Многие программисты "СКБ Контур" любят добираться до работы на метро — благо головной офис расположен совсем недалеко от станции "Уралмаш". Ну а поскольку сидячий образ жизни требует активных физических нагрузок в свободное от работы время, многие сотрудники — в том числе и Никифор — ходят от дома до метро пешком.

Никифор живет в таком районе нашего города, где улицы образуют правильную сетку кварталов; все кварталы являются квадратами с длиной стороны, равной 100 метрам. Вход на станцию метро расположен на одном из перекрестков; Никифор начинает свой путь с другого перекрестка, который расположен южнее и западнее входа в метро. Естественно, что, выйдя из дома, Никифор всегда идет по улицам, ведущим либо на север, либо на восток. Некоторые кварталы, которые встречаются ему на пути, он может также пересечь по диагонали, ведущей из юго-западного угла квартала в северо-восточный. Таким образом, некоторые из маршрутов (ведущих всегда на север, восток или северо-восток) оказываются короче других. Никифора интересует, сколько времени понадобится ему на преодоление кратчайшего маршрута; для этого ему нужно знать его длину. Вы должны написать программу, которая по имеющейся информации о виде сетки кварталов рассчитывает длину кратчайшего маршрута из юго-западного угла в северо-восточный.

Формат входных данных

В первой строке входного файла находятся два целых числа N и M ($0 < N, M \le 1000$) — размер сетки кварталов с запада на восток и с юга на север соответственно. Никифор начинает путь с перекрестка, который находится к юго-западу от квартала с координатами (1, 1); станция метро находится к северо-востоку от квартала с координатами (N, M). Во второй строке входного файла находится целое число K ($0 \le K \le 100$) — количество кварталов, через которые можно пройти наискось. Далее следуют K строк с парами натуральных чисел, разделенных пробелами — координатами таких кварталов.

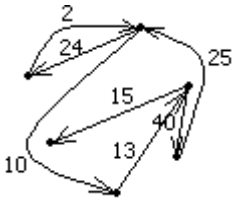
Формат выходных данных

В выходной файл требуется вывести длину кратчайшего пути от дома Никифора до станции метро в метрах, округленную до целых метров.

Пример

Входной файл	Выходной файл
3 2 3 1 1 3 2 1 2	383

Решение



Переведем эту задачу на язык графов. Ориентированным взвешенным графом назовем набор точек, некоторые пары которых соединены стрелками. По стрелке можно пройти из одной точки в другую. Будем далее называть точки вершинами, а стрелки — дугами. Каждая дуга имеет длину — некоторое положительное число. Длиной пути из вершины A в вершину B называется сумма длин всех дуг пути, ведущего из A в B. Задача о наименьшем пути состоит в нахождении пути из A в B с наименьшей длиной. Граф удобно хранить в виде матрицы смежности.

Составим следующий граф: вершинами графа назовем точки $(0, 0)$, (N, M) и все точки вида $(K - 1, L - 1) - (K, L)$, если квартал (K, L) можно срезать. Будем считать, что из точки (K_1, L_1) в точку (K_2, L_2) ведет дуга, если $K_1 \le K_2$ и $L_1 \le L_2$. Длина этой дуги равна $(K_2 + L_2 - K_1 - L_1)\cdot 100$. Кроме того, заменим длину дуг, соответствующих срезаемым кварталам, с 200 на $100\sqrt{2}$. В таком графе нужно найти кратчайший путь из вершины $(0, 0)$ в (N, M).

Формирование графа выглядит так:

```
for i := 1 to 2*k + 2 do
   for j := 1 to 2*k + 2 do
     matr[i,j] := infinity;
for i := 1 to k do
  readln(cutx[i],cuty[i]);
for i := k + 1 to 2*k do
  begin
    cutx[i] := cutx[i - k] - 1;
    cuty[i] := cuty[i - k] - 1
  end;
for i := 1 to 2*k do
  for j := 1 to 2*k do
    if (cutx[i] <= cutx[j]) and
       (cuty[i] <= cuty[j]) then
      matr[i,j] := (cutx[j] - cutx[i] +
          cuty[j] - cuty[i]) * 100;
for i := 1 to k do
  matr[i + k,i] := 141.42135;
cutx[2*k + 1] := 0;
cuty[2*k + 1] := 0;
cutx[2*k + 2] := n;
cuty[2*k + 2] := m;
for i := 1 to 2*k + 1 do
   matr[2*k + 1,i] := (cutx[i] +
cuty[i])*100;
for i := 1 to 2*k + 2 do
   matr[i,2*k + 2] := (m + n - cutx[i] -
cuty[i])*100;
```

Константу *infinity* положим равной 10^7. Теперь, когда матрица смежности заполнена, задача принимает вид стандартной задачи поиска кратчайшего пути в графе. Приведем здесь алгоритм Форда — Беллмана, решающий эту задачу. Этот алгоритм не является самым быстрым для подобной задачи, но, пожалуй, является самым простым. Идея алгоритма такова: будем работать в одномерном массиве $dist[i]$, обозначающем длину текущего самого короткого пути из $(0, 0)$ в $cutx[i]$. Будем пробегать все пары (i, j). В случае, если (длина текущего пути до i) + (длина дуги из i в j) меньше, чем длина текущего самого короткого пути до j, то изменим значение $dist[j]$. Этот процесс будем повторять, пока в массиве *dist* что-то будет меняться. Вот реализация этого алгоритма:

```
for i := 1 to 2*k + 2 do
  dist[i] := infinity;
dist[2*k + 1] := 0.0;
ind := true;
while ind do
begin
  ind := false;
  for i := 1 to 2*k + 2 do
    for j := 1 to 2*k + 2 do
      if dist[i] + matr[i,j] < dist[j]
        begin
          ind := true;
          dist[j] := dist[i] + matr[i,j]
        end
end;
writeln(dist[2*k + 2]:0:0);
```

Задача "Круговая порука"

(VI командное соревнование школьников Свердловской области по программированию, 2002 г.)

Автор задачи — Л.Волков
Автор разбора — А.Ипатов

Кто-то из N мальчиков и девочек опять разбил мамину любимую кружку. Мама рассердилась и перенумеровала мальчиков и девочек числами от 1 до N. После этого она подошла к мальчику (или девочке) под номером 1 и грозно спросила: "Кто разбил кружку?" Он (или она) честно ответил: "Я разбил кружку!" И был строго наказан.

Вы, конечно, понимаете, что рассказанная выше история идеализирована. На самом деле (правда это была или неправда — мы не знаем) мальчик (или девочка) под номером 1 ответил: "Да это не я! Это девочка (или мальчик) под номером K_1!". Потом мама подошла к номеру 2 и задала тот же вопрос...

Некоторые дети пытались ответить маме честно. Некоторые отвечали просто так, лишь бы что-нибудь сказать. Некоторые же заранее сговорились не выдавать маме преступника: каждый из группы юных заговорщиков называл кого-то другого — по кругу. В итоге мама совсем замучилась. Отчаявшись запомнить, что же говорил по поводу чашки каждый из отпрысков, она записала все их "показания" на листочек. Теперь она

собирается серьезно расследовать дело. В первую очередь она решила установить, не сговорились ли какие-нибудь мальчики и девочки по кругу, так, как это было описано выше. Вы должны написать программу, чтобы помочь ей в такой ситуации, — ведь, судя по всему, эта разбитая чашка не первая и не последняя...

Формат входных данных

В первой строке входного файла записано число T ($0 < T < 1001$) — количество тестов в файле. Каждый тест состоит из двух строк: в первой строке указано число N — количество детей ($0 < N < 25\,001$). Во второй строке через пробел записаны N чисел — показания опрошенных детей. Мама записывала в эту строку на i-е место номер того ребенка, которого i-й ребенок назвал виноватым, или число 0, если i-й ребенок вдруг признавался в том, что он разбил кружку.

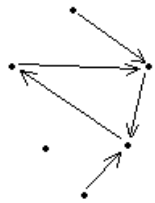
Формат выходных данных

В выходной файл следует вывести по одной строке для каждого теста. В эту строку надо записать YES, если показания детей по крайней мере выглядят непротиворечивыми: ровно один ребенок признался в том, что он разбил кружку, и нет группы детей, которая указывает друг на друга по кругу. В противном случае необходимо вывести NO.

Пример

Входной файл	Выходной файл
4 4 2 0 2 2 4 2 0 2 1 5 2 3 4 1 3 3 0 3 2	YES YES NO NO

Решение



В случае если массив показаний детей, назовем его *evd*, содержит больше одного нуля или не содержит их вовсе, то показания нужно считать противоречивыми. Рассмотрим основной случай, где ровно один ребенок признался в том, что разбил кружку. Каждый из остальных детей указал на кого-то другого. Рассмотрим ориентированный граф, в котором вершины обозначают детей, а дуги — их показания. Наша задача — проверить, содержит ли этот граф хотя бы один цикл. Поскольку каждый ребенок указывает ровно на одного, данный граф можно заменить неориентированным, имея в виду, что один граф не содержит циклов тогда и только тогда, когда и другой не содержит их. В теории графов доказывается следующая теорема: граф из $n - 1$ ребра и n вершин не содержит циклов тогда и только тогда, когда является связным. Таким образом, если граф является связным, то ответ задачи — YES, иначе — NO. Алгоритм проверки графа на связность был описан в решении задачи "День объединения".

Но мы сейчас рассмотрим другой алгоритм решения этой задачи, не требующий введения графа вовсе. Рассмотрим любого из детей, не сознавшихся в содеянном поступке. Покрасим его в первый цвет и перейдем к ребенку, на которого тот указывает. Покрасим его в такой же цвет и будем продолжать этот процесс, пока не дойдем до ребенка, который сознался, либо не встретим ребенка, уже покрашенного в первый цвет. Во втором случае в показаниях детей имеется цикл, и можно выдавать ответ NO. В первом случае найдем еще не покрашенного ребенка, покрасим его во второй цвет и продолжим аналогичный процесс окрашивания детей во второй цвет, пока не встретим ранее покрашенного ребенка (возможно, в первый цвет). В этом случае вновь найдем ребенка, не покрашенного ни в один из цветов, и начнем красить детей в третий цвет. Алгоритм закончит работу либо в случае, если все дети уже покрашены в какой-либо из цветов, либо в процессе окрашивания в цвет i мы дошли до ребенка, уже покрашенного в этот цвет. В первом случае ответ задачи — YES, во втором — NO.

Цвет ребенка будем отмечать тоже в массиве *evd*, но отрицательными числами: -1, -2 и т.д. Приведем реализацию этого алгоритма.

```
read(N);
nulls := 0;
for j := 1 to N do
  begin
    read(evd[j]);
    if (evd[j] = 0) then
      inc(nulls)
  end;
if nulls <> 1 then
  writeln('NO')
else
 begin
  ind := true;
  color := -1;
  while ind do
   begin
    p := 1;
    while (evd[p] <= 0) and (p <= N) do
      inc(p);
    if p > N then break;
    while evd[p] > 0 do
      begin
        p1 := evd[p];
        evd[p] := color;
        p := p1
      end;
    ind := (evd[p] = color);
    dec(color)
   end;
  if ind then writeln('NO')
         else writeln('YES')
 end;
```

Для решения задачи достаточно вызвать алгоритм T раз.

Задача "Монополия"

(Московская олимпиада по информатике 2005–2006 уч. г.)

Автор задачи и разбора — А.П. Лахно

В Тридесятом государстве есть N фирм, занимающихся разработкой программного обеспечения. Однажды известный олигарх Тридесятого государства Иванушка решил монополизировать эту отрасль. Для этого он хочет купить максимальное число программистских фирм Тридесятого государства.

Он разослал предложения всем N компаниям и через некоторое время получил от каждой из них согласие или отказ. Однако он знает, что в бизнесе очень многое зависит от взаимного доверия партнеров.

В результате небольшого исследования Иванушка установил, между какими компаниями существует взаимное доверие (причем всегда если компания A доверяет компании B, то компания B доверяет компании A).

Теперь при желании Иванушка может повторно разослать предложения всем компаниям, включив в письма список компаний, давших согласие участвовать в его проекте. При этом каждая компания, независимо от своего первоначального мнения, дает согласие, если в списке есть хотя бы одна компания, которой она доверяет, и отказ в противном случае. Таким образом, некоторые компании, которые изначально не согласились участвовать в проекте, могут теперь дать свое согласие, а некоторые из давших согласие — наоборот, отказаться. В результате этого у Иванушки формируется новый список, который он опять может разослать фирмам. Он может сколь угодно долго повторять операцию, каждый раз рассылая текущий список. Иванушка может остановить процесс в любой момент и заключить договора с теми, кто после последней рассылки дал согласие.

Напишите программу, которая определит, какое максимальное число компаний может объединить Иванушка под своим началом.

Будем считать, что Иванушка — честный предприниматель и он никогда не подтасовывает рассылаемые им списки.

Формат входных данных

В первой строке входного файла записано число N — количество фирм ($1 \le N \le 2000$). Далее идут N чисел, описывающих ответ фирмы на первое предложение Иванушки (1 — согласие, 0 — отказ). Далее записано число M ($0 \le M \le 200\,000$) — количество пар компаний, между которыми существует доверие. Далее записано M пар чисел, задающих номера фирм, между которыми существует взаимное доверие.

Формат выходных данных

Выведите в выходной файл одно число — максимальное число фирм, которое сможет купить Иванушка.

Примеры

Входной файл	Выходной файл
7 1 0 0 0 0 0 1 6 1 2 1 3 1 4 4 5 5 6 2 5	4
3 0 0 0 0	0

Решение

Ситуация, рассматриваемая в задаче, описывается неориентированным графом, вершинами которого являются фирмы, а ребра задаются отношением взаимного доверия.

Изначально вершины, соответствующие согласившимся фирмам, отмечены. Далее, на каждой итерации создается новый список отмеченных вершин: в него включаются те и только те вершины, у которых на предыдущем шаге была хотя бы одна отмеченная смежная вершина.

Рассмотрим ключевые закономерности изменения списка отмеченных вершин.

Пусть на некотором шаге имеются две смежные вершины, хотя бы одна из которых отмечена. Тогда после одной итерации отмеченная вершина может "погаснуть", но после второй она обязательно вновь станет отмеченной. Заметим, что эта вершина заведомо будет попадать в список отмеченных через каждые две итерации.

Если же имеются две отмеченные смежные вершины, то они останутся отмеченными после любого числа итераций.

Первое решение

Основываясь на этих закономерностях, приходим к следующему алгоритму решения задачи. Раскрасим имеющийся граф в два цвета с помощью поиска в глубину: первую вершину каждой компоненты связности красим в белый, все смежные с ней — в черный, все смежные с ними — снова в белый и т.д. Если при этом в какой-то компоненте связности есть цикл нечетной длины, то возникает коллизия: компоненту связности нельзя раскрасить в два цвета так, чтобы все вершины, смежные с белыми, были черными, а все вершины, смежные с черными, — белыми.

Параллельно с раскраской для каждой вершины запоминаем номер компоненты связности, в которой она лежит. Для компонент связности, содержащих циклы нечетной длины, помечаем, что их раскрасить не удалось. Кроме того, для каждой компоненты связности запоминаем число белых вершин и число черных вершин в ней.

Проходим по первоначальному списку отмеченных вершин.

Если компонента связности, в которой лежит отмеченная вершина, содержит циклы нечетной длины, то, начиная с некоторой итерации, все вершины компоненты будут попадать в список отмеченных на каждом шаге.

Если исходный список отмеченных вершин содержит как белую, так и черную вершину какой-то компоненты связности, то, так же, как и в предыдущем случае, начиная с некоторой итерации, все вершины компоненты будут попадать в список отмеченных на каждом шаге.

Если список вообще не содержит отмеченных вершин из какой-то компоненты, то ни одна вершина этой компоненты никогда не попадет в список отмеченных.

Для остальных компонент в первоначальном списке содержатся отмеченные вершины только одного цвета (для каждой компоненты — своего). Начиная с некоторой итерации, на каждом четном шаге в список отмеченных попадают вершины таких компонент, отмеченных тем же цветом, что и отмеченные вершины в исходном списке, а на нечетных шагах — вершины противоположного цвета. Единственным исключением здесь являются компоненты связности, состоящие ровно из одной вершины, — эта вершина в любом случае никогда не попадет в список отмеченных после первого шага.

С учетом всего вышесказанного, ответ задачи вычисляется как максимум из трех чисел: количества отмеченных вершин в исходном списке и количеств отмеченных вершин на четных и нечетных шагах с достаточно большим номером, которые, в свою очередь, несложно вычисляются с помощью ранее посчитанных величин, указанных в начале разбора.

Второе решение

Приведем вариант решения, несколько менее естественного, но существенно более изящного. Построим граф, вершинами которого являются пары вида (четность итерации, номер фирмы), а ребра соединяют вершины с разной четностью, соответствующие взаимно доверяющим фирмам. Пометим исходно вершины, соответствующие четной итерации и изначально согласившимся фирмам. Запустим теперь из помеченных вершин обход в глубину или ширину. Ответом задачи будет максимум из количества изначально согласившихся фирм и количеств обойденных вершин с четным и нечетным числом итераций соответственно.

Асимптотика обоих решений определяется асимптотикой процедуры обхода графа, т.е. $O(N + M)$.

В ограничениях задачи было достаточно разумно при отсутствии правильного решения написать по крайней мере решение, моделирующее процесс. При использовании отсечения по времени и аккуратном кодировании такое решение тоже набирало полный балл.

Заметим, что при моделировании, как это ни странно, не достаточно N итераций. Приведем простой пример входных данных: цепочка из пяти вершин, у которой, кроме того, соединены пятая и третья вершины. Изначально отмечена только первая вершина:

```
5
1 0 0 0 0
5
1 2
2 3
3 4
4 5
3 5
```

В данном случае ответ достигается лишь на шестой итерации при пяти вершинах. В общем же случае оптимальное решение будет найдено не более чем за $2N$ итераций.

Задача "Роботы"

(Московская командная олимпиада по информатике 2005–2006 уч. г.)

Автор задачи — Е.В. Андреева
Автор разбора — А.В. Фонарев

В подземелье есть N залов, соединенных туннелями. В некоторых залах находятся роботы, которые одновременно получили команду собраться в одном месте.

Роботы устроены так, что, получив команду, они все начали двигаться с такой скоростью, что туннель между двумя любыми залами любой робот преодолевает за 1 минуту. Роботы не могут останавливаться (в том числе и в залах), а также менять направление движения, находясь в туннелях. Однако, попав в зал, робот может из него пойти по тому же туннелю, по которому он пришел в этот зал.

Напишите программу, вычисляющую, через какое минимальное время все роботы смогут собраться вместе (в зале или в туннеле).

Формат входных данных

Во входном файле записаны сначала числа N — количество залов ($1 \le N \le 400$) и K — количество туннелей. Далее записано K пар чисел, каждая пара описывает номера залов, соединяемых туннелем (по туннелю можно перемещаться в обе стороны). Между двумя залами возможно несколько туннелей. Туннель может соединять зал с самим собой. Далее записано число M ($1 \le M \le 20\,000$) — количество роботов. Затем идет M чисел, задающих номера залов, где вначале расположены роботы. В одном зале может быть несколько роботов.

Формат выходных данных

В выходной файл выведите минимальное время в минутах, через которое роботы могут собраться вместе. Если роботы никогда не смогут собраться вместе, выведите одно число -1 (минус один).

Примеры

Входной файл	Выходной файл
4 5 1 2 2 3 3 4 1 4 1 3 3 1 2 4	1

Входной файл	Выходной файл
3 3 1 2 2 3 3 1 3 1 2 3	1.5

Решение

Переведем задачу на язык графов. Вершинами нашего графа будут залы, а ребрами — туннели. Граф взвешенный, вес каждого ребра — 1. Тогда роботам требуется за кратчайшее время встретиться в вершине или на ребре.

Сразу отметим два факта:

1. Если робот может прийти в вершину за время t, он может повторно попасть туда же за время $t + 2$.

2. Если роботы встречаются в тоннеле (на ребре), то это происходит в середине, причем часть роботов идут по ребру в одну сторону, а часть в другую (иначе они бы уже встретились в одной вершине).

Теперь становится понятно, как найти минимальное время, за которое роботы могут собраться в данной вершине v: смотрим, за какое наименьшее четное (нечетное) время роботы придут в v (это наибольшая из длин кратчайших четных (нечетных) путей каждого робота до v). Первое, что приходит в голову, — разбить каждое ребро на две равные части, т.е. добавить вершины, которые будут соответствовать серединкам. Но мы так делать не будем, т.к. это решение требует некоторой аккуратности. Если мы знаем время T_i, за которое данный робот добирается до вершины i, то до центра ребра uv робот доберется за $min(T_u, T_v) + 1/2$. Появляется следующая идея: для каждого робота найдем кратчайшие четный и нечетный пути до всех вершин графа. После чего для каждой вершины найдем время сбора в ней. То же самое сделаем с каждым ребром (сбор в центре) и возьмем общий минимум. Если собраться нигде не удалось (ясно, что это бывает только в случае, когда найдутся два робота в разных компонентах связности графа), ответом будет -1.

Самый интересный момент решения — поиск кратчайших путей четной и нечетной длины. Для этого немножко модифицируем обход в ширину, вычисляя сразу 2 величины — четное и нечетное расстояния. Граф будем хранить в виде списков смежных вершин. При считывании входных данных можно сразу удалить кратные ребра, но ни в коем случае нельзя удалять петли (если в вершине есть петля, это позволяет роботу вернуться туда же не за 2 хода, а за 1). Все расстояния в задаче полуцелые (т.е. становятся целыми при умножении на 2), а значит, если считать, что вес ребра — 2, то все вычисления можно проводить с помощью целых типов данных (это дает хороший выигрыш во времени). В конце же просто разделим ответ на 2.

Теперь можно привести основной фрагмент решения задачи.

```
  var m: array[1..400,1..400, 0..1] of
integer;
      l: array[1..400,0..400] of integer;
      n, k, nr, i, j, t: integer;
      r: array[1..400] of integer;
      a, b: array[1..20000] of integer;
```

$m[i][j][k]$ — матрица кратчайших путей: i — вершина, из которой мы ищем путь, j — пункт назначения, k — четность пути (0 или 1); $l[i][j]$ — список смежных вершин: i — вершина, для которой хранится список, $l[i]$ — сам список ($l[i][0]$ — его длина); $a[i]$ и $b[i]$ — концы ребер.

Далее описан модифицированный обход графа в ширину. Он использует очередь, реализованную при помощи массива o. Если путь данной четности до вершины не существует, значение соответствующей ячейки будет равно бесконечности (число, большее 10^6). За одну итерацию цикла из очереди берется очередная вершина, обновляются расстояния для смежных вершин. Те, для которых расстояния обновились и которые не находятся в очереди, туда заносятся. Массив u показывает, находится ли данная вершина в очереди.

```
procedure bfs(v: integer);
var p1, p2, cv: integer;
begin
 fillchar(u, sizeof(u), 0);
 p1 := 1;
 p2 := 1;
 o[p1] := v;
 u[v] := true;
 while p1 <= p2 do begin
  cv := o[p1];
  inc(p1);
  u[cv] := false;
  for i := 1 to l[cv][0] do begin
   if m[v][l[cv][i]][0] > m[v][cv][1] + 2 then
    begin
     m[v][l[cv][i]][0] := m[v][cv][1] + 2;
     if not u[l[cv][i]] then begin
      u[l[cv][i]] := true;
      inc(p2);
      o[p2] := l[cv][i]
     end
    end;
   if m[v][l[cv][i]][1] > m[v][cv][0] + 2 then
    begin
     m[v][l[cv][i]][1] := m[v][cv][0] + 2;
     if not u[l[cv][i]] then begin
      u[l[cv][i]] := true;
      inc(p2);
      o[p2] := l[cv][i]
     end
    end
  end
 end
end;
```

Вычислительная геометрия

Задача "Найди прямую"
(Московская олимпиада по информатике 2005–2006 уч. г.)

Автор задачи и разбора — Б.О. Василевский

На плоскости дано множество отрезков. Требуется найти прямую, которая пересекла бы наибольшее возможное количество из данных отрезков и при этом проходила бы как минимум через две точки с целочисленными координатами.

Считается, что прямая пересекает отрезок, если она имеет с ним хотя бы одну общую точку (т.е. она может проходить через конец отрезка, внутреннюю точку отрезка, либо содержать весь отрезок).

Формат входных данных

Во входном файле записано сначала число N — количество отрезков ($1 \le N \le 1000$). Далее идет N четверок чисел X_{i1}, Y_{i1}, X_{i2}, Y_{i2}, задающих координаты концов отрезков. Все эти числа целые, по модулю не превосходящие 10 000.

Заданные отрезки могут пересекаться, иметь общие части, один из них может полностью содержаться внутри другого. Отрезки имеют ненулевую длину.

Формат выходных данных

В выходной файл выведите координаты каких-нибудь двух точек, через которые проходит прямая, пересекающая наибольшее количество отрезков. Координаты точек должны быть целыми и не должны по модулю превышать 10^7.

Примеры

Входной файл	Выходной файл	Пояснение (кол-во отрезков, пересекаемых прямой)
3 0 0 1 0 0 1 1 1 0 2 1 2	0 -1 1 4	3
2 0 0 1 0 0 0 1 0	0 0 1 0	2
2 0 0 1 0 2 0 1 0	1 0 1 1	2
5 -1 0 3 4 2 3 5 6 0 2 2 -2 8 5 9 2 8 5 9 2	10 3 1 4	4

Решение

Степенью прямой назовем количество данных отрезков, которые она пересекает; концы отрезков будем называть *вершинами*.

Прежде всего обсудим, каким может быть относительное расположение искомой прямой и данных отрезков. Рассмотрим прямую с максимальной степенью.

Если прямая не горизонтальная, будем двигать ее влево горизонтально, пока она не пройдет через какую-нибудь вершину. Иначе можно двигать ее вертикально вверх. Новая прямая будет иметь такую же степень, как и начальная прямая, так как количество пересечений не может увеличиться, а уменьшается оно только при прохождении через концы отрезков.

Итак, можно считать, что искомая прямая должна проходить через какую-либо вершину. Теперь будем вращать эту прямую против часовой стрелки вокруг вершины, через которую она проходит, пока прямая не пройдет через вторую вершину. Аналогично, степень полученной прямой равна степени исходной. И теперь прямая проходит через две различных вершины.

На этих идеях основано решение с алгоритмической сложностью $O(N^3)$, где N — количество отрезков: перебираем все пары вершин; для каждой пары считаем степень прямой, через эти вершины проходящей; среди них выбираем прямую с наибольшей степенью. Но такое решение не набирает полного балла.

Решение с алгоритмической сложностью $O(N^2 \log N)$ использует только первое рассуждение: искомая прямая проходит через вершину. Таким образом, если для каждой вершины найти прямую наибольшей степени, которая проходит через нее, то, выбрав среди всех степеней максимум, получим ответ.

Научимся искать прямую с наибольшей степенью, проходящую через вершину $P(xp, yp)$. Любое упоминание прямой теперь подразумевает, что она проходит через P. Сразу договоримся не рассматривать отрезки, проходящие через P, так как они не влияют на поиск прямой максимальной степени. Главное — не забыть их подсчитать и учесть при вычислении степени прямой.

Общая идея такова: будем вращать прямую $y = yp$ вокруг P против часовой стрелки, подсчитывая пересечения.

Ориентированной площадью $[\mathbf{A}, \mathbf{B}]$ пары векторов $\mathbf{A} = (x1, y1)$ и $\mathbf{B} = (x2, y2)$ назовем число, равное $x1 \cdot y2 - y1 \cdot x2$. По абсолютной величине она равна площади параллелограмма, у которого в качестве сторон взяты $\mathbf{A}$ и $\mathbf{B}$, отложенные от одной точки. Она равна нулю тогда и только тогда, когда $\mathbf{A}$ и $\mathbf{B}$ коллинеарны. В противном случае знак $[\mathbf{A}, \mathbf{B}]$ дает нам информацию о направлении кратчайшего поворота от вектора $\mathbf{A}$ к вектору $\mathbf{B}$. Если система координат выбрана стандартным образом, то $[\mathbf{A}, \mathbf{B}]$ больше нуля тогда и только тогда, когда кратчайший поворот — против часовой стрелки, см. *рис.* 1 (соответственно, $[\mathbf{A}, \mathbf{B}]$ меньше нуля тогда и только тогда, когда кратчайший поворот — по часовой стрелке, см. *рис.* 2). В литературе по вычис-

лительной геометрии эту же величину часто называют *векторным произведением* векторов **A** и **B**.

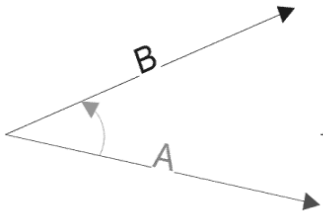


Рис. 1

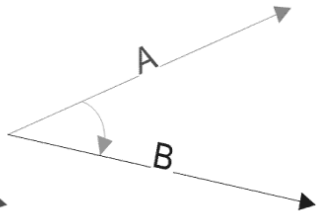


Рис. 2

Назовем *началом* отрезка *XY* ту его вершину, которая встретится при вращении нашей прямой раньше остальных точек этого отрезка; *концом* — ту, которая встретится последней. Это можно определить, вычислив знак ориентированной площади $S = [\boldsymbol{PX}, \boldsymbol{PY}]$: если *X* — начало, то $S > 0$. В случае, если *X*, *Y* и *P* лежат на одной прямой, началом будем считать любую из вершин, концом — оставшуюся.

Рассмотрим для начала более простой случай: по отношению к *P* все отрезки лежат справа и сверху. Чтобы реализовать "вращение прямой", надо отсортировать все вершины по возрастанию *полярного угла* (то есть по углу между лучом из точки *P* в вершину и осью абсцисс), в случае совпадения будем считать, что конец всегда больше начала.

Сделаем небольшие пояснения. Сортировка точек по полярному углу проходит почти так же, как и сортировка чисел, только для сравнения двух точек *X*, *Y* используется знак ориентированной площади $S = [\boldsymbol{PX}, \boldsymbol{PY}]$. "*X* меньше *Y*", если $S > 0$ ("*X* больше *Y*", если $S < 0$).

Прямая $y = yp$ не пересекает ни одного отрезка в силу того, что все концы отрезков лежат выше нее.

Шаг 1. Считаем переменную *C* равной 0.

Шаг 2. Просматриваем все вершины так, как они идут в отсортированном списке.

```
Для каждой вершины Q (
  Если Q — начало, то (
    Увеличиваем C на 1.
    Если C больше найденного максимума,
    то обновляем максимум, запоминаем,
    что соответствующая прямая проходит
    через Q.
    ) иначе уменьшаем C на 1.
  )
)
```

Понятно, что после просмотра вершины *Q* степень всех прямых между *PQ* и *PS* (где *S* — следующая за *Q* вершина в отсортированном списке) равна *C* (любая из этих прямых пересекает все отрезки, которые уже "начались", но еще не "закончились", их ровно *C* по построению). Среди всех таких значений *C* надо выбрать наибольшее. В качестве прямой с соответствующей степенью можно брать, к примеру, *PQ*, как и описано в алгоритме.

Сложность общего случая (когда отрезки расположены произвольно) заключается в следующем: пусть мы сравниваем точки как обычно — по полярному углу. Тогда привычное свойство транзитивности сравнения "из $a \le b$, $b \le c$ следует $a \le c$" не выполняется, например, для вершин $a = (0, 1)$, $b = (1/2, -1/2)$, $c = (-1/2, -1/2)$, $P = (0, 0)$. Поэтому любая сортировка за время $N\log N$, основанная на таком свойстве, будет сортировать точки неправильно. А ведь именно за счет сортировки за время $N\log N$ мы имеем оценку $O(N^2\log N)$, а не $O(N^3)$.

Существует несколько способов выйти из этой ситуации, но наиболее простой, на наш взгляд, следующий.

Исходная прямая — по-прежнему $y = yp$. Разделим отрезки на два типа:

1) пересекающие $y = yp$,

2) не пересекающие $y = yp$.

Идея в том, чтобы перед сортировкой подвергать симметрии относительно *P* все вершины, лежащие ниже *P* (не забывая, начало это или конец). Но делать такую симметрию надо аккуратно.

Пусть отрезок *XY* после возможной симметрии его вершин, описанной выше, перешел в отрезок *X'Y'*. Если ориентированная площадь $[\boldsymbol{PX}, \boldsymbol{PY}]$ отличается по знаку от ориентированной площади $[\boldsymbol{PX'}, \boldsymbol{PY'}]$, то визуально начало и конец "поменялись местами" (мы по-прежнему называем *X'* началом, если *X* — начало, и, наоборот, — концом, если *X* — конец). То есть при вращении наша прямая сначала встретит конец *X'Y'*, а потом начало (см. *рис.* 3). В этом случае надо учитывать, что прямые из области 3 (см. *рис.* 3) тоже пересекают *XY* (в алгоритме — шаг 3). Кстати, такое возможно, только если *XY* — первого типа.

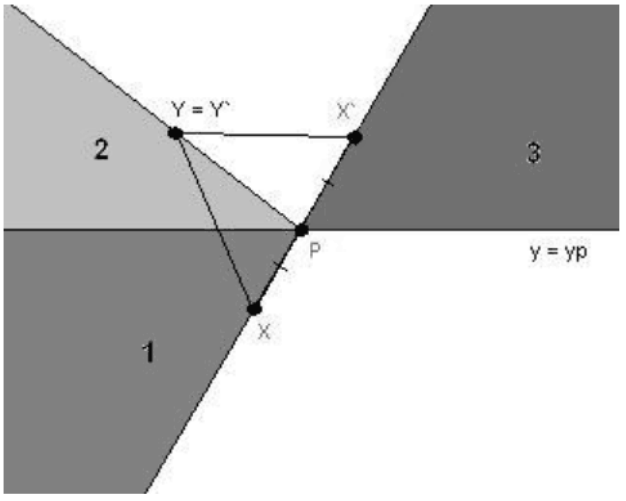


Рис. 3

С отрезками второго типа таких неудобств не будет.

Замечание 1. Если приступать к сортировке, проделав предварительно лишь описанные преобразования, мы не получим правильного порядка. Пусть, например, $P(0, 0)$, $A(-1, 0)$, $B(1, 0)$, *A* и *B* — начала. Тогда вершины *A* и *B* будут считаться

равными (подумайте, почему). Чтобы избежать этого, можно подвергать симметрии еще и точки, лежащие на $y = yp$, но левее P. Тогда никакие две вершины не будут лежать с P на одной прямой и одновременно по разные стороны относительно P.

Замечание 2. Чтобы проверить, лежит ли P на отрезке AB, надо сначала убедиться, что все эти три точки лежат на одной прямой. Если это верно, можно посмотреть скалярное произведение векторов PA и PB: если оно меньше или равно 0, то P принадлежит AB (0 может быть, если один из векторов нулевой), в противном случае — нет.

Для каждой точки X через X' будем обозначать следующее: если X ниже P или X принадлежит $y = yp$ и левее P, то X' симметрична X относительно P. В противном случае $X = X'$.

Чтобы алгоритм получился полным, вспомним о существовании отрезков, проходящих через P. По-прежнему считаем, что концы отрезка ему принадлежат.

Шаг 1. Считаем переменную C равной 0. Список для сортировки считаем пустым.

Шаг 2 (создание списка для сортировки).

```
Для каждого отрезка XY (
  Если P принадлежит XY, то (
    Увеличить C на 1
    ) иначе (
       Если XY — первого типа, то (
         Если знаки [PX, PY] и [PX', PY']
         различаются (оба числа должны
         быть отличны от нуля),
         то увеличить C на 1.
       )
      )
    Пусть S = [PX, PY].
    Если S >= 0, то X' помечаем как начало,
                    Y' — как конец.
    Если S < 0, то Y' помечаем как начало,
                   X' — как конец.
    Добавляем в список X'
    Добавляем в список Y'
  )
)
```

Шаг 3 (сортировка и поиск прямой с максимальной степенью). Теперь отсортируем полученный список по полярному углу и далее будем действовать, как в простом случае (единственное отличие — стартовое значение C может быть отлично от нуля).

Задача "Разрезание многоугольника"

(I городская олимпиада, г. Нижний Новгород, 2004–2005 уч. г.)

На плоскости заданы многоугольник и прямая; прямая не проходит через вершины многоугольника.

Напишите программу, которая определит, на сколько частей делит эта прямая многоугольник.

Формат входных данных

В первой строке входного файла находится одно число N ($3 \le N \le 1025$) — число вершин многоугольника. Далее следуют N строк, задающие вершины многоугольника: в i-й из этих строк находятся два целых числа x_i и y_i ($|x_i|, |y_i| \le 10\ 000$) — координаты i-й вершины многоугольника. Вершины занумерованы от 1 до N в порядке обхода против часовой стрелки.

В $(N + 2)$-й строке входного файла находятся четыре целых числа p_x, p_y, q_x, q_y, задающие прямую (все числа не превосходят по модулю 10 000). Прямая проходит через точки (p_x, p_y) и (q_x, q_y); эти точки различны.

Формат выходных данных

В выходной файл выведите одно целое число — ответ на задачу.

Пример

Входной файл	Выходной файл
4 0 0 1 0 1 1 0 1 -1 -1 3 4	2

Решение

В этой задаче требовалось найти количество частей, на которые прямая разделяет многоугольник.

Очевидным образом количество точек пересечения прямой и границы многоугольника четно. Действительно, ни одна вершина, а следовательно, ни одна сторона многоугольника не лежит на секущей прямой. Кроме того, если мы будем двигаться по прямой из бесконечности, то сколько раз мы "войдем" в многоугольник, то есть пересечем его границу, проходя извне внутрь, столько же раз и "выйдем", то есть пересечем его границу, проходя изнутри вовне.

Ясно, что каждая пара "вход—выход" в/из многоугольника увеличивает количество частей на 1. Так как сначала часть была одна — сам многоугольник, то окончательная формула для количества частей следующая: (*количество пересечений*)/2 + 1.

Теперь посчитаем количество пересечений. Проще всего для каждого ребра многоугольника проверить, лежат ли его концы по разные стороны от прямой. Таким образом, для каждого ребра $(x_i; y_i)$—$(x_{i+1}; y_{i+1})$ и прямой, заданной двумя точками $(p_x; p_y)$ и $(q_x; q_y)$, имеем условие того, что точки A_i $(x_i; y_i)$ и $A_{i+1}(x_{i+1}; y_{i+1})$ лежат по разные стороны от прямой $P(p_x; p_y)$—$Q(q_x; q_y)$: векторные произведения векторов $\boldsymbol{PA_i}(x_i - p_x; y_i - p_y)$ и $\boldsymbol{PA_{i+1}}(x_{i+1} - p_x; y_{i+1} - p_y)$ на вектор $\boldsymbol{PQ}(q_x - p_x; q_y - p_y)$ имеют разные знаки:

$$[\boldsymbol{PA_i}, \boldsymbol{PQ}] \times [\boldsymbol{PA_{i+1}}, \boldsymbol{PQ}] < 0.$$

Но в программировании именно такой способ проверять, что величины имеют разный знак, очень часто является ошибкой. Дело в том, что каждое векторное произведение в нашей задаче по модулю может достигать 800 000 000, то есть их произведе-

ние уже не влезет в четырехбайтовый целый тип данных. В данном случае лучше записать более длинное условие: либо первое произведение меньше нуля, а второе больше, либо наоборот. Другой способ состоит в том, чтобы вместо векторных произведений перемножать их знаки.

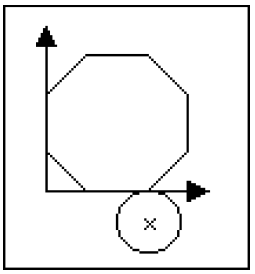
Задача "Точность попадания снаряда"

(VI командное соревнование школьников Свердловской области по программированию, 2002 г.)

Автор задачи — А.Ботов
Автор разбора — А.Ипатов

"Знаете ли вы, что неточность попадания снаряда можно компенсировать его диаметром?"

С.В. Сизый



В этой задаче вам предлагается с помощью компьютера определить самый маленький диаметр, которым можно компенсировать неточность попадания снаряда в каждом конкретном случае. Будем считать, что все цели являются выпуклыми многоугольниками. Попаданием считается ситуация, когда круглая воронка, остающаяся от снаряда, задевает хотя бы одну точку цели (диаметр воронки равен диаметру снаряда).

Формат входных данных

В первой строке входного файла находятся 3 числа — координаты попадания центра снаряда и количество сторон многоугольника N ($3 \le N \le 100$), следующие N строк содержат координаты его вершин, перечисленные по часовой стрелке. Все координаты являются целыми числами из диапазона $[-1\,000\,000; 1\,000\,000]$.

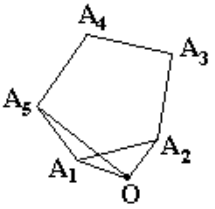
Формат выходных данных

В выходном файле должно быть записано единственное число — минимальный диаметр снаряда, который поразит цель. Число следует округлить до трех знаков после запятой.

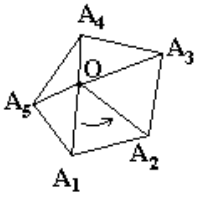
Пример

Входной файл	Выходной файл
2 -1 8 0 1 1 0 2 0 3 1 3 2 2 3 1 3 0 2	2.000

Решение



Первое, что нам нужно научиться определять, — это лежит ли центр снаряда в мишени. Если это так, то ответ задачи — 0.000. Если нет, то расстояние от точки до многоугольника равно минимуму расстояний от этой точки до его сторон.

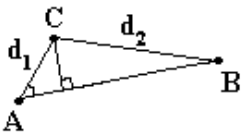


Обозначим центр снаряда — O, мишень — $A_1A_2...A_N$ и рассмотрим ориентированные углы A_1OA_2, A_2OA_3, ..., $A_{N-1}OA_N$, A_NOA_1. Они все имеют одинаковые знаки (поворот между лучами происходит везде по часовой стрелке либо везде — против нее) тогда и только тогда, когда точка лежит внутри многоугольника. На рисунке справа все углы ориентированы против часовой стрелки, а на рисунке слева — угол A_1OA_2 ориентирован по часовой стрелке, а угол A_5OA_1 — против. Ориентацию угла легко определить с помощью векторного произведения векторов OA_i и OA_{i+1}.

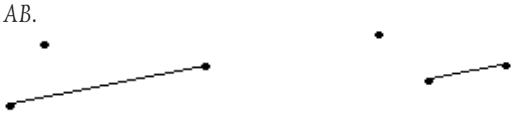
Приведем код функции, определяющей, лежит ли точка внутри многоугольника.

```
function inside: boolean;
var i: integer; ind: boolean;
begin
  ind := true;
  for i := 1 to n do
    if ((x[i] - xx)*(y[i + 1] - yy) -
        (x[i + 1] - xx)*(y[i] - yy))*
       ((x[i + 1] - xx)*(y[i + 2] - yy) -
       (x[i + 2] - xx)*(y[i + 1] - yy)) < 0
      then ind := false;
  inside := ind
end
```

Здесь $x[i]$, $y[i]$ — координаты вершин многоугольника ($x[n+1] = x[1]$, $x[n+2] = x[2]$, $y[n+1] = y[1]$, $y[n+2] = y[2]$), xx, yy — координаты точки.



Итак, пусть мы определили, что точка лежит вне многоугольника, и хотим вычислить расстояния от нее до сторон многоугольника. Рассмотрим произвольный отрезок AB и точку C, заданные координатами. Опустим перпендикуляр из C на прямую AB.



Если он попадает внутрь отрезка (ситуация на рисунке слева), то он и является кратчайшим расстоянием от C до AB. Иначе (см. рисунок справа) расстояние до отрезка равно минимуму из длин AC и BC. Если оба угла CAB и CBA лежат в пределах $[0; \pi/2]$, то налицо первый случай, иначе — второй. Является ли угол тупым, легко узнать по знаку его косинуса, а тот может быть определен через скалярное произведение. Зная косинусы углов CAB и CBA, а также расстояния $AC = d_1$ и $BC = d_2$, вычислим расстояние от C до прямой AB.

Приведем код основной части программы.

```
mind := infinity;
if inside then mind := 0
else
  for i := 1 to n do
    begin
      a := (x[i] - x[i + 1])*(xx - x[i + 1])
       + (y[i] - y[i + 1])*(yy - y[i + 1]);
      b := (x[i + 1] - x[i])*(xx - x[i]) +
           (y[i + 1] - y[i])*(yy - y[i]);
      d1 := sqrt(sqr(xx - x[i + 1]) +
                 sqr(yy - y[i + 1]));
      d2 := sqrt(sqr(xx - x[i]) +
                 sqr(yy - y[i]));
      if a < 0 then d := d1
      else
       if b < 0 then d := d2
       else
        begin
         dt := a/sqrt(sqr(x[i + 1] - x[i]) +
                      sqr(y[i + 1] - y[i]));
         d := sqrt(d1*d1 - dt*dt)
        end;
      if d < mind then mind := d
    end;
writeln(2*mind:0:3);
```

Здесь a и b — скалярные произведения ($\boldsymbol{AC}$, $\boldsymbol{AB}$) и ($\boldsymbol{BC}$, $\boldsymbol{BA}$) соответственно, dt — проекция вектора $\boldsymbol{AC}$ на AB (с помощью нее по теореме Пифагора найдем d — расстояние от C до AB). $x[n + 1] = x[1]$, $y[n + 1] = y[1]$. Константа *infinity* равна 10^8.

Читайте в ближайших номерах

Компьютерные сети

В осенних номерах "Информатики" будут опубликованы материалы по теме "Компьютерные сети" из новой книги А.А. Дуванова из серии "Азы информатики" (книга готовится к печати в издательстве БХВ). Это совершенно новые, оригинальные материалы, в которых на простых и наглядных примерах рассматриваются принципы функционирования современных компьютерных сетей.

Примерные ответы на профильные билеты

В осенних номерах мы начнем публиковать примерные ответы на профильные экзаменационные билеты (см. № 6/2006). Для каждого билета будет полностью раскрыто содержание теоретического вопроса, показаны решения примерных практических заданий. Также будет предложено до пяти авторских вариантов по каждому из двух практических заданий. Материал готовит авторская группа из Перми, хорошо знакомая нашим читателям по публикациям "Примерных ответов на примерные билеты" по базовому курсу информатики.

ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»
главный редактор – А.С. Соловейчик

ГАЗЕТЫ ИЗДАТЕЛЬСКОГО ДОМА

Первое сентября
гл. ред. — Е.В. Бирюкова, индекс подписки — 32024;
Английский язык
гл. ред. — Е.В. Громушкина, индекс подписки — 32025;
Библиотека в школе
гл. ред. — О.К. Громова, индекс подписки — 33376;
Биология
гл. ред. — Н.Г. Иванова, индекс подписки — 32026;
География
гл. ред. — О.Н. Коротова, индекс подписки — 32027;
Дошкольное образование
гл. ред. — М.С. Аромштам, индекс подписки — 33373;
Здоровье детей
гл. ред. — Н.В. Сёмина, индекс подписки — 32033;
Информатика
гл. ред. — С.Л. Островский, индекс подписки — 32291;
Искусство
гл. ред. — М.Н. Сартан, индекс подписки — 32584;
История
гл. ред. — А.Л. Савельев, индекс подписки — 32028;
Литература
отв. сек. — С.Ф. Дмитренко, индекс подписки — 32029;
Математика
и. о. гл. ред. — Л.О. Рослова, индекс подписки — 32030;
Начальная школа
гл. ред. — М.В. Соловейчик, индекс подписки — 32031;
Немецкий язык
гл. ред. — М.Д. Бузоева, индекс подписки — 32292;
Русский язык
гл. ред. — Л.А. Гончар, индекс подписки — 32383;
Спорт в школе
гл. ред. — О.М. Леонтьева, индекс подписки — 32384;
Управление школой
гл. ред. — Я.А. Сартан, индекс подписки — 32652;
Физика
гл. ред. — Н.Д. Козлова, индекс подписки — 32032;
Французский язык
гл. ред. — Г.А. Чесновицкая, индекс подписки — 33371;
Химия
гл. ред. — О.Г. Блохина, индекс подписки — 32034;
Школьный психолог
гл. ред. — И.В. Вачков, индекс подписки — 32898.

Гл. редактор
С.Л. Островский
Редакция
Е.В. Андреева
Д.М. Златопольский (редактор вкладки "В мир информатики")
Л.Н. Картвелишвили
С.Б. Кишкина
Н.П. Медведева
Ю.А. Первин (редактор вкладки "Началка")
Корректор
Е.Л. Володина
Дизайн и верстка
Н.И. Пронская

Адрес редакции и издателя:
Киевская, 24, Москва, 121165
тел. 249-48-96
Отдел рекламы: 249-98-70

Учредитель: ООО "Чистые пруды"

Зарегистрировано в Министерстве РФ по делам печати. ПИ № 77-7230 от 12.04.2001.
Отпечатано в ОИД "Медиа-Пресса", ул. Правды, 24, Москва, ГСП-3, А-40, 125993
Тираж 6000 экз.
Срок подписания в печать по графику 07.07.2006.
Номер подписан 07.07.2006.
Заказ № 615516
Цена свободная

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
комплекта изданий 32744

Тел.: (095) 249-31-38, 249-33-86. Факс (095)249-31-84

Internet: inf@1september.ru
WWW: http://www.1september.ru

ИЗДАТЕЛЬСКАЯ ПОДПИСКА Тел.: (495) 249-47-58 E-mail: podpiska@1september.ru