Алексеев Александр Сергеевич
Ломов Дмитрий Степанович
Станкевич Андрей Сергеевич

РАЗБОР ЗАДАЧ ФИНАЛЬНЫХ СОРЕВНОВАНИЙ ЧЕМПИОНАТА МИРА ПО ПРОГРАММИРОВАНИЮ

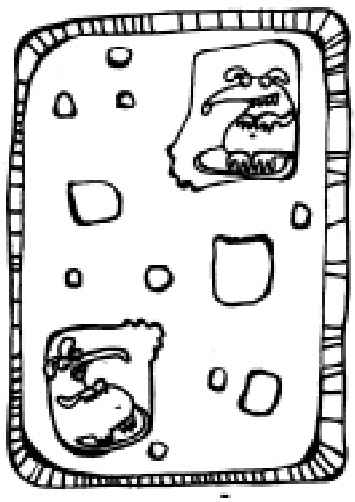
В этом номере журнала мы публикуем анализ шести задач чемпионата мира, решенных командами, получившими золотые медали. Анализ задач подготовлен Андреем Станкевичем, анализ решений и программы к задачам B и F – Александром Алексеевым и Дмитрием Ломовым.

Задача B. Скажите «Сы-ы-ыр»!

Когда-то давным-давно в гигантском куске сыра жила сырная мышка Амелия. Амелия, должно быть, была очень счастлива, потому что ее окружало больше вкуснейшего сыра, чем она могла когда-либо съесть. Тем не менее, она чувствовала, что чего-то не хватает в ее жизни.

Однажды утром ее мечты о сыре были прерваны шумом, который она никогда раньше не слышала. Но она немедленно поняла, что это было, – голос сырного мышонка, вгрызающегося где-то в кусок сыра! (Определить пол сырной мышки только по звуку поедания сыра довольно сложно, но все сырные мышки способны на это. Просто потому, что их научили родители).

Теперь ничего не могло остановить Амелию. Она должна была встретиться с мышонком как можно быстрее! Поэтому она должна была найти кратчайший путь к нему. Амелия прогрызает 1 миллиметр сыра за 10 секунд. Но, оказывается, кратчайший путь к другому мышонку по прямой может оказаться не самым быстрым. Дело в том, что в сыре, где живет Амелия, множество дырок. Эти дырки, которые представляют собой пузырьки воздуха в сыре, вообще говоря, сферические. Но, поскольку некоторые сферические дырки перекрываются, появляются дырки более сложной формы. Проскочить по дырке в сыре Амелия может мгновенно – за 0 секунд из любой точки дырки в любую другую точку этой дырки. Так что может оказаться полезно путешествовать через дырки, чтобы добраться до другой мышки как можно быстрее.



В этой задаче Вам нужно написать программу, которая, получив координаты обоих мышат и дырок в сыре, найдет минимальное время, за которое Амелия доберется до другого мышонка. В этой задаче будем предполагать, что кусок сыра бесконечно большой. Это нужно, чтобы Амелии было не выгодно вылезти из куска сыра, чтобы добраться до второй мышки. (Кроме того, снаружи ее поджидают злобные кошки-поедатели-сырных-мышей!) Вы также можете предполагать, что мышонок с нетерпением ожидает, когда его найдет Амелия, и не перемещается, пока она движется к нему.

Входные данные.

Входной файл содержит описания нескольких тестовых примеров для сырной мышки. Каждый пример начинается со строки, содержащей единственное число – n $(0 \le n \le 100)$, – число дырок в сыре. Затем следуют n строк, содержащих по 4 целых числа, – x_i, y_i, z_i и r_i – координаты центра и ради-

ус i-й дырки. Наконец, следующие две строки содержат по 3 целых числа – x_A, y_A и z_A – координаты Амелии и x_0, y_0 и z_0 – координаты мышонка. Все значения даны в миллиметрах.

Завершает входной файл строка, содержащая единственное число –1.

Выходные данные.

Для каждого тестового примера выведите одну строку, в соответствии с форматом, приведенным в примере. Сначала выведите номер тестового примера (нумерация с 1), а затем минимальное время в секундах, округленное до ближайшего целого числа, за которое Амелия доберется до мышонка (примеры будут подобраны так, что округление будет однозначным).

Пример.

Входной файл.

```
1
20 20 20 1
0 0 0
0 0 10
1
5 0 0 4
0 0 0
10 0 0
-1
```

Вывод.

```
Cheese 1: Travel time = 100 sec
Cheese 2: Travel time = 20 sec
```

Решение.

Это одна из самых простых задач на соревновании. Она легко сводится к одной из базовых задач теории графов. Некоторая сложность приведенных здесь рассуждений связана исключительно с необходимостью формально доказать корректность этого сведения.

Заметим, что расстояние между непересекающимися сферами с центрами в точках (x_i, y_i, z_i) и (x_j, y_j, z_j) и радиусами r_i и r_j, соответственно, вычисляется по формуле $\sqrt{(x_i-x_j)^2+(y_i-y_j)^2+(z_i-z_j)^2}-r_i-r_j$ и равно 0, если сферы пересекаются.

Вычислим значение этого выражения для всех пар сферических дырок в сыре и сохраним их в двухмерном массиве $d[i,j]$, где индексы i, j соответствуют i-ой и j-ой сферам соответственно. Если $d[i,j]$ получилось меньше нуля (сферы пересекаются), то полагаем его равным 0.

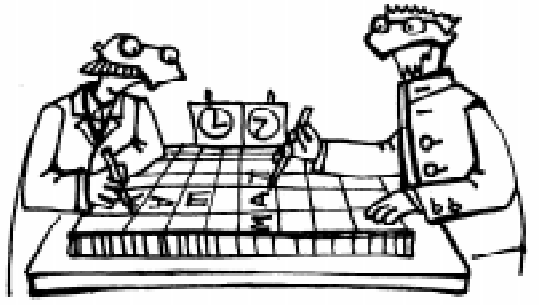
Рассмотрим оптимальный путь Амелии. Взяв последовательность дырок, в которых побывала Амелия, заметим, что прямые, по которым она пробегала между соседними дырками, не могут задевать другие дырки (предлагаем читателю самостоятельно обосновать это утверждение). Задача свелась к поиску кратчайших путей в графе, что можно сделать, применив, например, алгоритм Флойда-Уоршелла.

Будем использовать массив $d[i,j]$ для хранения кратчайших расстояний между любыми сферами. Рассмотрим три произвольных сферы i, j, k. Если $d[i,k]+d[k,j]$ меньше $d[i,j]$, то $d[i,j]$ можно сделать меньше присвоив ему эту сумму. Поскольку центры $n+1$ и $n+2$ сферы – это местоположения мышек, то теперь нам осталось только найти кратчайшее расстояние между этими сферами. Для этого в цикле рассмотрим все тройки сфер и согласно выше приведенному правилу уточним расстояния $d[i,j]$ по всем промежуточным сферам. После выполнения цикла $d[n+1,n+2]$ будет содержать кратчайшее расстояние от Амелии до другого существа. Осталось только умножить его на 10, что бы получить время в секундах.

В приложении 1 приведена программа на Pascal.

Задача С. Головоломка со словами.

Ваш гениальный, но слегка не в своем уме дядя верит, что решил сложней-



шую головоломку со словами, но потерял решение. Теперь ему нужна Ваша помощь, чтобы восстановить его по списку слов, которые он использовал для решения, и одного лишнего слова (он никак не может вспомнить, какое слово лишнее). Ваша программа должна найти решение и указать лишнее слово.

Головоломка, которую решал Ваш дядя, представляет собой квадратную решетку 10x10. Рисунок 1 показывает верхний левый угол одной такой головоломки. У головоломки есть некоторое количество позиций, начиная с которых следует вставить слова. Для каждой позиции указано направление, в котором следует написать слово, начиная с нее («across» – по горизонтали и «down» – по вертикали). А вот длина слова, которое должно начинаться с данной позиции, не указана.

На рисунке 2 показано решение головоломки, приведенной на рисунке 1. В корректном решении с каждой указанной позиции начинается слово из списка. Каждая максимальная горизонтальная и вертикальная последовательность, состоящая из 2 и более букв, должна быть словом, поставленным в головоломку. Никакое слово не должно содержать в себе другое слово, поставленное в головоломку. Любое входное слово может начинаться в любой из указанных позиций, если это не противоречит другим словам. Например, в данном примере все слова-кандидаты использованы, кроме слова «BOY».

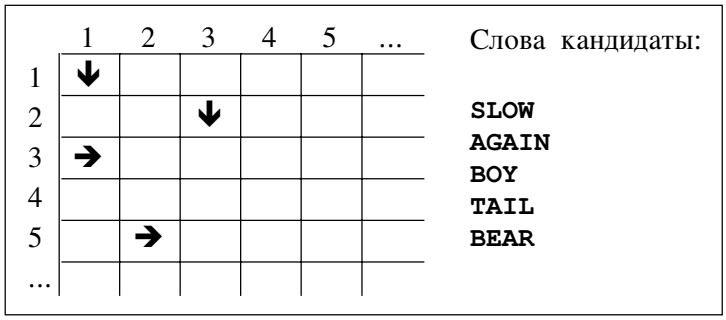


Рисунок 1

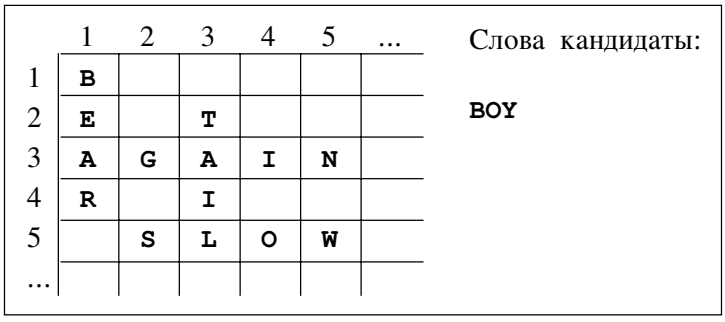


Рисунок 2

Входные данные.

Входной файл содержит описания нескольких тестовых примеров, каждый из которых описывает головоломку. На первой строке каждого примера находится число N, которое означает количество стартовых позиций в головоломке. Следующие N строк описывают эти позиции, каждое описание представляет собой тройку: номер строчки, номер столбца в головоломке, где находится позиция, и букву A или D – в каком направлении следует расположить слово, начинающееся в этой позиции (A – across – по горизонтали, D – down – по вертикали). Следующие N+1 строк содержат слова-кандидаты. Все слова состоят из заглавных латинских букв, все слова разные.

После последнего примера следует единственная строка, содержащая 0.

Выходные данные.

Для каждого тестового примера выведите номер головоломки и, затем, лишнее слово, которое не используется при решении головоломки. Следуйте формату, указанному в примере. Соблюдайте следующие правила:

1) После каждой головоломки выводите пустую строку.

2) Если Ваш дядя ошибся и головоломка не имеет решения, выведите, вместо лишнего слова, слово «Impossible».

3) Если существует более одного способа решить головоломку, следует вывести все слова, которые могут оказать-

ся лишними, в любом порядке, разделяя слова пробелами. При этом, если одно и то же слово может оказаться лишним при более чем одном решении головоломки, его все равно следует вывести только 1 раз. Например, если головоломка № 3 может быть решена тремя способами, в двух из которых лишнее слово CAT, а в одном – DOG, следует вывести

«**Trial 3: CAT DOG**» или
«**Trial 3: DOG CAT**».

Пример.

Входной файл.

```
4
1 1 D
2 3 D
3 1 A
5 2 A
SLOW
AGAIN
BOY
TAIL
BEAR
0
```

Вывод.

```
Trial 1: BOY
```

Решение.

Эта задача представляет собой типичный пример задачи на технику программирования. Для ее решения следует применить перебор с возвратом и отсечениями. Укажем основные идеи реализации ее решения.

Будем рекурсивно перебирать все возможные подстановки слов на позиции. Их, очевидно, $A_{N+1}^{N} = \frac{(N+1)!}{1!} = (N+1)!$. Это довольно много, однако заметим, что ограничения, приведенные в головоломке, серьезно уменьшают количество комбинаций. Действительно, если во время очередной подстановки слова мы нарушаем одно из правил построения головоломки, дальнейший перебор в этой ветви рекурсии, очевидно, бесполезен. Это позволяет отсечь достаточно большое количество случаев.

Рассмотрим теперь реализацию рекурсивной процедуры.

```
type
  ttable = array [0..11, 0..11] of char;
  tdirs = array [0..11, 0..11] of byte;

procedure rec(p: longint; a: ttable;
b: tdirs);
```

Будем передавать *p* – количество уже обработанных позиций. Когда *p* превысит *n*, очевидно, все позиции заполнены, осталось проверить корректность заполнения и, если все хорошо, – пометить, что не использованное нами слово является одним из ответов на задачу. В массиве *a* будем передавать текущее состояние поля. При этом `' '` будет означать пустую клетку, а `'#'` – клетку, в которую нельзя поставить ни одного символа. Сначала заполним «периметр» поля символами `'#'`. Кроме того, поскольку по условию каждая максимальная горизонтальная и вертикальная последовательность, состоящая из 2 и более букв, должна быть словом, поставленным в головоломку, и так как никакое слово не должно содержать в себе другое слово, поставленное в головоломку, очевидно клетка сверху от позиции, где начинается вертикальное слово, и слева от позиции, где начинается горизонтальное слово, не может содержать никакой буквы. Поставим во все такие клетки символ `'#'`.

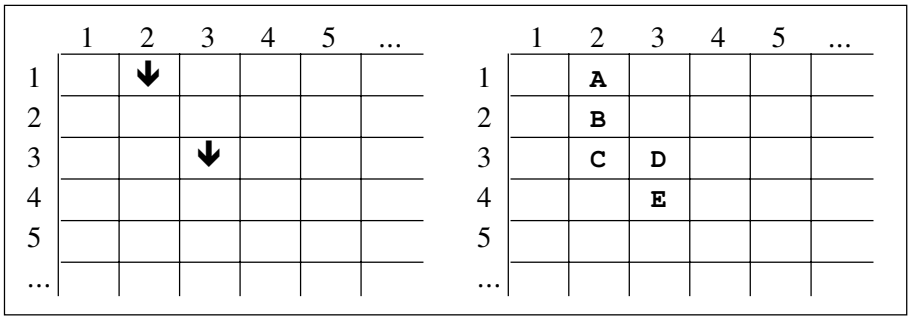


Рисунок 3

Однако этого недостаточно, чтобы обеспечить корректность построения головоломки в процессе перебора, поскольку при этом не отсекается, например, следующая постановка слов, противоречащая условию (рисунок 3).

Здесь «CD» не является словом из списка. Для того, чтобы после построения головоломки осуществить проверку появления таких ситуаций, будем передавать дополнительный массив *b*, в котором в ячейке *b*[*i*][*j*] будем записывать 1, если через эту клетку проходило вертикальное слово, которое, кроме того, не закончилось в этой клетке (таким образом, наличие 1 в *b*[*i*][*j*] оправдывает наличие буквы в *a*[*i*+1][*j*]) и 2, если через эту клетку проходило горизонтальное слово, которое не закончилось в этой клетке (оправдывает наличие буквы в *a*[*i*][*j*+1]). Соответственно, если через эту клетку проходило как вертикальное, так и горизонтальное слово, запишем в *b*[*i*][*j*] число 1 or 2 = 3.

Примечание. В данном случае рассматривается операция «поразрядное или». Она обозначается or (*a* or *b* = *c)* и действует так: если $\overline{a_{n-1}...a_0}$, $\overline{b_{m-1}...b_0}$, $\overline{c_{n-1}...c_0}$ – двоичные представления *a, b* и *c* соответственно, то $c_i = 0$, если $a_i = 0$, $b_i = 0$ и $c_i = 1$ в иных случаях. Так как, 01, 10, 11 - двоичные представления 1, 2, 3 соответственно, то $\overline{01}$ or $\overline{10} = \overline{11}$ или 1 or 2 = 3.

Теперь проверить корректность заполненной головоломки можно так:

```
ok := true;
for i := 1 to 9 do
  for j := 1 to 9 do
  begin
   if not (a[i][j] in [' ', '#']) and
      not (a[i+1][j] in [' ', '#']) and
         (b[i][j] and 1 = 0) then
         ok := false;
  if not (a[i][j] in [' ', '#']) and
     not (a[i][j+1] in [' ', '#']) and
          (b[i][j] and 2 = 0) then
          ok := false;
  end;
```

Проверка корректности заполнения головоломки словами с этой точки зрения представляет собой самую серьезную

проблему в этой задаче. Реализация перебора после решения этой проблемы сложности не представляет.

Задача D. Никак не вырубить лес – деревья мешают.

Когда-то давным давно в одной далекой стране жил был король, и был у него лес, где росли ценные деревья. Однажды, чтобы справиться с проблемой утечки капитала, король решил срубить и продать некоторые из своих деревьев. И попросил своего придворного мудреца найти максимальное количество деревьев, которые можно срубить.

Все королевские деревья были окруженны прямоугольным забором, который защищал их от воров и хулиганов. И рубить деревья было довольно сложно – ведь каждому дереву необходимо место, чтобы упасть. Правда, с дерева можно уда-

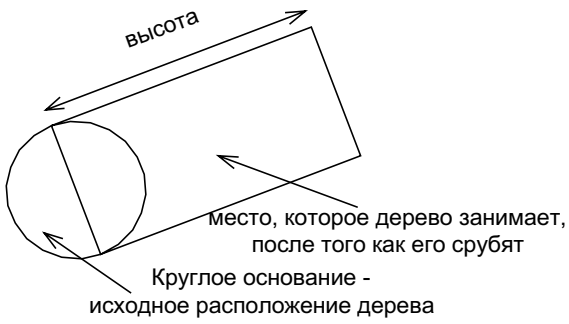


Рисунок 4

лить все ветки перед тем, как рубить его, поэтому для простоты мудрец решил, что каждое дерево, когда его срубят, будет занимать прямоугольный участок земли, как показано на рисунке 4. Одна сторона прямоугольника – это исходный диаметр дерева, а вторая – высота дерева.

Многие королевские деревья располагались рядом с другими деревьями (это одна из особенностей этого леса, которую так любили упоминать в сказках). Мудрец должен был найти максимальное количество деревьев, которые можно срубить одно за другим, чтобы упавшее дерево не задело другое дерево или забор. Как только дерево падает, его разрезают на части и выносят, так что оно уже не мешает рубке последующих деревьев.

Входные данные.

Входной файл содержит несколько тестовых примеров, каждый из которых описывает лес. Первая строка каждого описания содержит пять целых чисел – x_{min}, y_{min}, x_{max}, y_{max} и n. Первые четыре числа представляют собой минимальные и максимальные координаты прямоугольника забора – он расположен так, что его стороны параллельны осям координат. Пятое число – n – это количество деревьев в лесу ($1 \le n \le 100$).

Следующие n строк описывают расположение и размеры деревьев. Каждая линия содержит четыре целых числа – x_i, y_i, d_i и h_i – координаты, диаметр и высоту дерева. Деревья не касаются друг друга и не задевают забор. Все деревья находятся внутри забора.

Завершает файл строка, содержащая 5 нулей.

Выходные данные.

Для каждого тестового примера выведите его номер и на следующей строке максимальное количество деревьев, которое удастся срубить так, что все срубленные деревья после падения не касаются других деревьев и забора. После каждого примера выводите пустую строку.

Пример.

Входной файл.

```
0 0 10 10 3
3 3 2 10
5 5 3 1
2 8 3 9
0 0 0 0 0
```

Вывод.

```
Forest 1
2 tree(s) can be cut
```

Решение.

Это – самая сложная с точки зрения реализации задача на чемпионате. Идея же ее решения довольно проста. Прежде всего заметим, что от того, что мы срубим некоторое дерево (если мы можем это сделать), мы «не испортим ситуацию» – все деревья, которые мы могли до этого срубить, мы по-прежнему можем срубить и, возможно, можем срубить еще некоторые деревья. Отсюда, тело программы следующее:

```
cutcount := 0;
fillchar(cut, sizeof(cut), false);
for i := 1 to n do
  for j := 1 to n do
    if cancut(j) then
    begin
      cut[j] := true;
      inc(cutcount);
    end;
```

Здесь функция

```
function cancut(j: integer): boolean;
```

проверяет, что можно срубить j-е дерево,

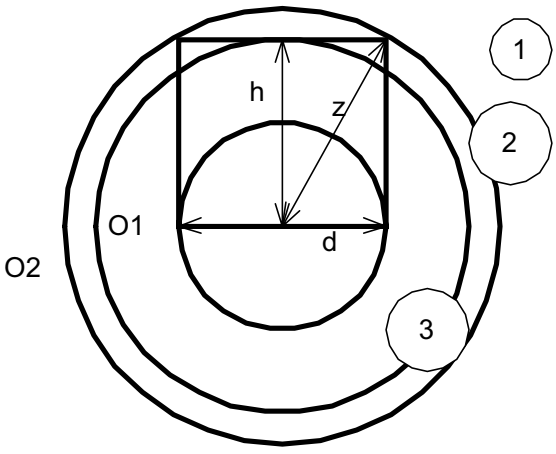


Рисунок 5

если все деревья, для которых **`cut[i] = true`**, уже срублены. Именно написание этой функции и представляет собой основную сложность.

Рассмотрим дерево, обозначим его окружность за D. Просмотрим, каким образом могут другие деревья помешать ему упасть (рисунок 5).

Проведем две окружности, центр которых совпадает с центром нашего дерева и радиусами h и $z = \sqrt{(d/2)^2 + h^2}$ (будем опускать индекс i, когда речь идет о рассматриваемом дереве). Обозначим эти окружности, соответственно, O_1 и O_2, а круги, ими ограниченные, – K_1 и K_2. Тогда все деревья можно разбить на три категории:

1. Деревья, которые полностью лежат вне круга K_2. Эти деревья не могут помешать нам срубить наше дерево.
2. Деревья, которые пересекают круг K_2 (не обязательно окружность O_2 – они могут полностью лежать внутри круга), но не имеют общих точек с кругом K_1.
3. Деревья, которые пересекают круг K_1.

Если рассмотреть возможные значения угла, между какой-нибудь стороной прямоугольника дерева после того, как его срубят, то деревья второй категории вносят по две «запретные зоны» – диапазоны углов, в направлении которых дерево не может упасть, а деревья третьей категории – по одной.

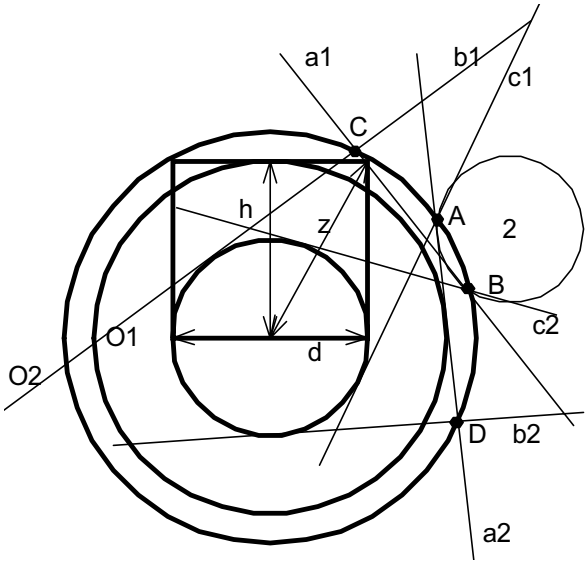


Рисунок 6

Рисунок 6 демонстрирует построение запретных зон для дерева второй категории. Обозначим его окружность за D_2. Сначала проведем касательные к окружности D_1 через точки пересечения окружности D_2 с окружностью O_2, либо внешние касательные к окружностям D_1 и D_2 – в зависимости от того, снаружи или внутри круга K_2 лежат точки касания этих касательных с D_2 – если снаружи, проводим касательные через точки пересечения, если внутри – совместные касательные. На рисунке они обозначены c_1 и c_2. Если бы наше дерево принадлежало третьей категории, то весь угол между ними оказался бы запретным, а так имеется еще одна зона, в которой наше дерево может упасть.

Построим внутренние совместные касательные к окружностям D_2 и O_1 – a_1 и a_2. Точки их пересечения с окружностью O_2 обозначены как C и D. Тепеь проведем через эти точки внешние касательные к окружности D_1, они обозначены b_1 и b_2 на рисунке 6. Зона между b_1 и b_2 не является запретной, в ней дерево может упасть.

Соответственно, получаем, что запретной является зона между c_1 и c_2 за вычетом зоны между b_1 и b_2 – две дуги.

Как уже упоминалось, для дерева третьей категории все упрощается – для него существует единственная запретная зона – между углами, задаваемыми прямыми c_1 и c_2.

Кроме деревьев, помешать падению может забор. Для него запретные зоны строятся аналогично – достаточно разбить забор на 4 отрезка (мы можем для удобства неограниченно продолжить отрезки) и для каждой прямой построить запретную зону. Можно также рассматривать прямые как окружности с бесконечным радиусом.

Теперь, чтобы проверить, что дерево можно повалить, достаточно объединить запретные зоны по отношению к нему

всех еще не срубленных деревьев и забора и проверить, что полученная суммарная зона не покрывает всю окружность. Если это так, то дерево можно повалить.

Задача F. Мажорная задача.

В западной музыке каждая из 12 нот, используемых в музыкальной нотации, обозначается буквой от A до G, за которой может следовать диез # или бемоль b, причем они повторяются по циклу следующим образом:

C/B# C#/Db D D#/Eb E/Fb F/E# F#/Gb G G#/Ab A A#/Bb B/Cb C/B# ...

Две рядом стоящие ноты называют *полутон*. Соответственно, две, между которыми имеется еще ровно одна нота – *тон.* Мажорная гамма состоит из 8 нот, она начинается с одной из вышеуказанных нот и затем соответствует последовательности *тон-тон-полутон-тон-тон-тон-полутон*. Например, мажорные гаммы, начинающиеся с C или с Db, приведены ниже:

C D E F G A B C
Db Eb F Gb Ab Bb C Db

Кроме этого мажорные гаммы должны удовлетворять следующим правилам:
1. Гамма должна содержать все буквы от A до G, причем по одному разу, кроме первой ноты, которая повторяется еще раз в качестве последней.
2. Гамма не может содержать одновременно бемоли и диезы.

Нота, с которой начинается мажорная гамма, называется тоном. Например, выше приведены примеры гамм с мажорной тональностью C и Db, соответственно. Транспонирование ноты из одной гаммы в другую – это просто замена ноты в одной гамме нотой из другой гаммы, находящейся в соответствующей позиции. Например, нота F в мажорной гамме для C заменится на ноту Gb из мажорной гаммы Db.

Вы должны написать программу, которая будет выполнять транспонирование нот из одной мажорной гаммы в другую.

Входные данные.

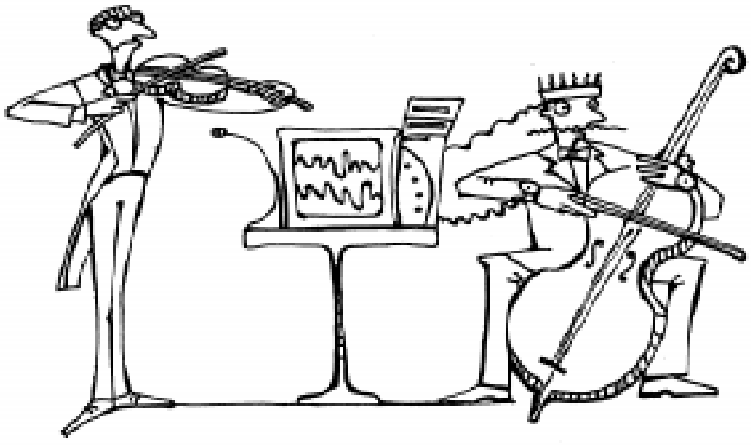
Входной файл содержит несколько тестовых примеров, по одному на строке. Сначала следует тональность первой гаммы, затем тональность второй гаммы и затем список нот, для которых следует произвести транспонирование из второй гаммы в первую. Каждый список заканчивается звездочкой «*», все ноты и звездочка в нем разделены единственным пробелом.

Последняя строка входного файла содержит единственный символ «*» и не должна обрабатываться.

Выходные данные.

Каждому тестовому примеру должна соответствовать одна или несколько строк вывода. Если обе тональности соответствуют некоторым мажорным гаммам, то первая строка вывода для такого примера должна быть «**Transposing from X to Y**», где X – тональность первой гаммы, а Y – второй. Если одна из тональностей не соответствует мажорной гамме, то выведите «**Key of X/Y is not a valid major key**», где X/Y – тональность, которая не соответствует мажорной гамме. Если и первая и вторая тональности не соответствуют мажорной гамме, выведите информацию только о первой тональности. После этого остаток ввода для данного тестового примера следует проигнорировать.

Если же обе тональности соответствуют мажорной гамме, после первой строки вывода должно следовать по



одной строке для каждой ноты, для которой следует произвести транспонирование. Если нота присутствует в мажорной гамме для первой тональности, то следует вывести «**M transposes to N**», где M – нота, указанная во входном файле, а N – соответствующая нота из второй гаммы. Если же нота отсутствует в мажорной гамме, то следует вывести «**M is not a valid note in the X major scale**», где M – по-прежнему нота из входного файла, а X – тональность первой гаммы.

Разделяйте вывод для различных тестовых примеров пустой строкой.

Пример.

Входной файл.

Вывод.

```
C Db F *
Db C Gb
C B# A B *
C D A A# B Bb C *
A# Bb C *
*
Transposing from C to Db:
  F transposes to Gb

Transposing from Db to C:
  Gb transposes to F

Key of B# is not a valid major key

Transposing from C to D:
  A transposes to B
   A# is not a validnote in the C
major scale
  B transposes to C#
  Bb is not a valid note in the
  C major scale   C transposes to D

Key of A# is not a valid major key
```

Решение.

Это – типичная задача на понимание условия. Здесь кажется, что знание музыки должно существенно упростить понимание и решение задачи, ведь для не изучавших музыку терминология является полностью абстрактной, а для изучавших – родной и знакомой. Однако задача содержит хитрую ловушку: у знающих музыку появляется желание просто перечислить все существующие гаммы в исходном тексте программы, определить их в качестве констант. В то же время при перечислении порядка сотни нот возникает высокая вероятность совершить ошибку. Автоматизировав же поиск гамм, мы уберегаем себя от ошибки из-за невнимательности.

Как же автоматизировать поиск гамм? Очень просто. Для каждой ноты существует не более 2^6=64 последовательностей нот (имеются в виду обозначения нот – C и B#, например, подразумеваются разными нотами), начинающихся с данной ноты и удовлетворяющих условию *тон-тон-полутон-тон-тон-тон-полутон*. Проверить для каждой из них выполнимость обоих условий мажорной гаммы не составляет труда. Ясно, что для каждого мажорного ключа существует не более одной гаммы, удовлетворяющей условиям 1 и 2. Остается только найти эти гаммы для каждого ключа и сохранить их в двухмерном массиве. После этого перевод ноты из одной гаммы в другую заключается в нахождении ее индекса в строке массива, хранящей гамму для исходного ключа и чтении из массива по индексу и целевому ключу получившейся ноты. Так же нам

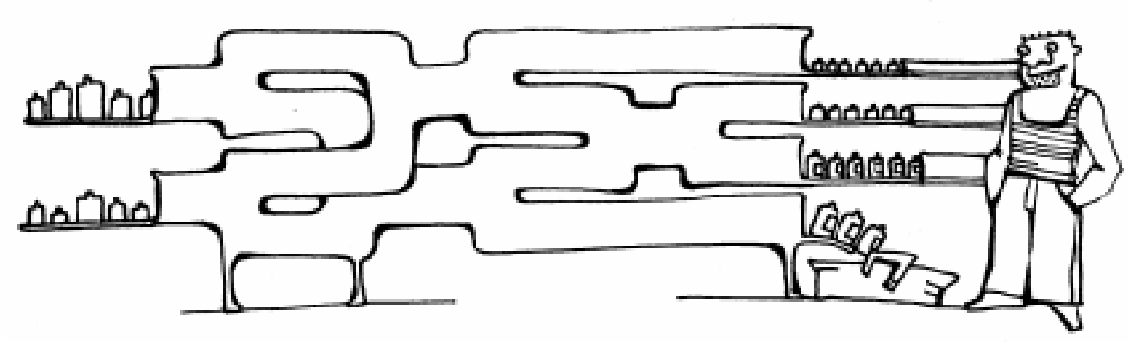


Рисунок 7

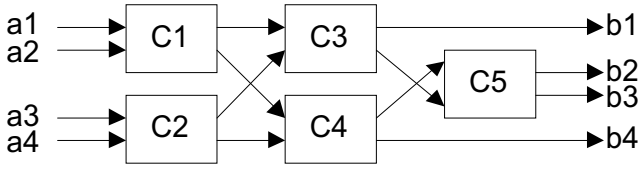


Рисунок 8

понадобится булевый массив, хранящий информацию о корректности ключей. Его легко заполнить во время поиска гамм для каждого ключа.

Для нахождения гаммы по ключу, воспользуемся рекурсией по числу уже найденых нот. Если все 7 нот будут найдены, то ключ – корректен, иначе – нет. Если ключ корректен, то в булевый массив заносится «истина» и в соответсвующую строку массива гамм заносится гамма. Если ключ некорректен, то в булевый массив заносится «ложь», а в массив гамм ничего не заносится.

Детали реализации видны из комментариев в программе (приложение 2).

Задача H. Сети профессора Монотоника.

Профессор Монотоник экспериментирует с сетями компараторов. Компаратор – это устройство с двумя входами и двумя выходами, которое берет значения на входах i_1 и i_2 и выдает их на выходы o_1 и o_2 так, чтобы всегда было $o_1 \le o_2$ (см. рисунок 7).

Сеть компараторов имеет n входов a_1, a_2, ..., a_n и n выходов – b_1, b_2, ..., b_n. Каждый из входов компаратора, входящего в сеть, соединен либо с входом сети, либо с выходом какого-либо другого компаратора. Соответственно, каждый из выходов компаратора соединен либо с выходом сети, либо с входом какого-либо другого компаратора. Причем граф соединений компараторов всегда ациклличен. На рисунке 8 изображена сеть с 4 входами и 4 выходами, состоящая из 5 компараторов.

Во время функционирования сети на входы подаются некоторые значения – целые числа, и компараторы начинают выполнять свою работу. Разумеется, компаратор не может выдать значения на выходы до тех пор, пока оба его входа не получили значение. Будем считать, что срабатывание одного компаратора занимает единицу времени. Тогда, например, приведенная на рисунке сеть работает 3 единицы времени прежде, чем все выходы получат значения. Сначала параллельно сработают компараторы C_1 и C_2, затем C_3 и C_4 и лишь затем C_5.

Профессору Монотонику нужна Ваша помощь – он хочет определить, является ли сеть компараторов сортирующей и сколько она работает. Сортирующая сеть – это такая сеть компараторов, которая вне зависимости от значений, поданых ей на входы, выдает их на выходы в отсортированном порядке. Например, сеть на рисунке 2 – сортирующая.

Входные данные.

Профессор предоставил Вам описание всей сети компараторов, которую Вам предстоит исследовать. Описание начинается со строки, содержащей n – количество входов (и выходов) и k – количество компараторов. Эти значения удовлетворяют условиям $1 \le n \le 12$ и $0 \le k \le 150$. Затем следуют пары чисел, разделенные пробелами и/или переводами строки, задающие входы компараторов. Если вход компаратора помечен i, то это значит, что он либо соединен с входом сети с номером i (a_i), если ни один компаратор еще не соединен с этим входом, либо с выходом одного из предшествующих компараторов, который помечен i, к которому не подключен еще ни один компаратор. Соответствующий выход рассматриваемого

компаратора также помечается i. После срабатывания компаратора выход с меньшим номером будет содержать меньшее значение. В конце неиспользованные входы сети и выходы компараторов соединяются с соответствующими выходами сети.

Первый тестовый пример соответствует сети, приведенной на рисунке 8. Входной файл заканчивает пара нулей.

Выходные данные.

Для каждого тестового примера выведите одну строку, содержащую номер примера, информацию о том, является ли сеть сортирующей и за какое время она работает.

Пример.

Входной файл.

```
4 5
1 2  3 4  1 3
2 4  2 3
8 0
3 3
1 2  2 3  1 2
0 0
```

Вывод.

```
Case 1 is a sorting network and
operates in 3 time units
Case 2 is not a sorting network and
operates in 0 time units
Case 3 is a soreing network and
operates in 3 time units
```

Решение.

Выясним сначала, как определить время срабатывания сети. Ясно, что это время – максимум по всем выходам времени, когда на них появится значение. Будем действовать следующим образом: заведем массив t, в котором будем хранить в ячейке $t[i]$ время, в которое будет доступно значение на очередном проводе, помеченном i. Тогда, очевидно, ответом на задачу будет $\max_{i=1..n} t[i]$ после обработки по очереди всех компараторов. Обработка компаратора заключается в том, что значения t на его входах $t[\text{in}(j,1)]$ и $t[\text{in}(j,2)]$ заменяются на $\max\{t[in(j,1)], t[in(j,2)]\}+1$.

Более сложную проблему представляет собой проверка того, что сеть является сортирующей. Первой идеей, которая возникает, является эмуляция работы сети на всех возможных перестановках чисел от 1 до n. Но таких перестановок $n!$, что при $n = 12$ не позволяет уложиться в разумное время. Однако, оказывается, справедливо следующее предложение:

Если сеть компараторов сортирует все наборы из n чисел, каждое из которых 0 или 1, то она является сортирующей сетью.

Таких наборов 2^n, число гораздо более разумное при данных ограничениях, чем $n!$. Докажем это предложение: пусть это не так, и сеть сортирует все наборы из 0–1, но не сортирует некоторый набор $c_1,c_2,...,c_n$. Тогда найдутся c_i и c_j, такие, что $c_i > c_j$, но c_i оказывается на выходе с меньшим номером, чем c_j. Положим

$$f(x) = \begin{cases} 0, & x \le c_j \\ 1, & x > c_j \end{cases}.$$

Подадим на вход нашей сети $f(c_1)$, $f(c_2)$, ..., $f(c_n)$. Заметим, что если компаратор менял местами значения на входах, когда ему подавали c_k и c_p, то он поменяет их, и когда ему подадут $f(c_k)$ и $f(c_p)$, а если не менял, то не поменяет. Значит, если обозначить как $g(i)$ выход сети, на котором окажется значение c_i, то на том же выходе окажется и значение $f(c_i)$. А значит, $f(c_i)$ окажется на выходе с меньшим номером, чем $f(c_j)$. Но $f(c_i) = 1$, а $f(c_j) = 0$, значит, сеть не сортирует наш набор из 0 и 1, что противоречит предположению. Предложение доказано.

Более подробно о компараторах и сортирующих сетях можно прочитать в Кормен, Лейзерсон, Ривест «Алгоритмы: построение и анализ», МЦНМО, Москва, 2000, стр. 584.

Приложение 1.

```
program cheese;
type    float = extended;
const   MaxN=110;
var    X,Y,Z,R:array[1..MaxN] of integer;
       G:array[1..MaxN,1..MaxN] of float;
var    _,N,i,j,k:integer;
begin
     assign(input,'in.txt'); reset(input);
     assign(output,'out.txt'); rewrite(output);
       _:=0;
      repeat
        readln(N); if N=-1 then break;
        inc(_);
       for i:=1 to N do readln(x[i],y[i],z[i],r[i]);
       readln(x[N+1],y[N+1],z[N+1]); r[N+1]:=0;
       readln(x[N+2],y[N+2],z[N+2]); r[N+2]:=0;
         for i:=1 to N+2 do
         for j:=1 to N+2 do
          begin
       G[i,j]:=sqrt(sqr(x[i]-x[j])+sqr(y[i]-y[j])+sqr(z[i]-z[j]))-r[i]-r[j];
          if G[i,j]<0 then G[i,j]:=0
         end;
         for k:=1 to N+2 do
         for i:=1 to N+2 do
         for j:=1 to N+2 do
        if G[i,k]+G[k,j]<G[i,j] then G[i,j]:=G[i,k]+G[k,j];
         writeln('Cheese ',_,': Travel time = ',G[N+1,N+2]*10:0:0,' sec')
      until false
end.
```

Приложение 2.

```
{$A+,B-,C+,D+,E-,F-,G+,H+,I+,J+,K-,L+,M-,N+,O+,P+,Q+,R+,S+,T-,U-,V+,W-
,X+,Y+,Z1}
{$MINSTACKSIZE $00004000}
{$MAXSTACKSIZE $00100000}
{$IMAGEBASE $00400000}
{$APPTYPE CONSOLE}
program f;

(* Число наименований нот *)
const nnames=21;

(* Число нот в гамме *)
const ngamma=7;

(* Названия нот *)
const names:array [1..nnames] of string [2]=
('C', 'B#', 'C#', 'Db', 'D', 'D#', 'Eb', 'E', 'Fb', 'F', 'E#',
 'F#', 'Gb', 'G', 'G#', 'Ab', 'A', 'A#', 'Bb', 'B', 'Cb');

(* Соответствие нот: название - звук *)
const tone:array [1..nnames] of integer=
(0, 0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 8, 9, 10, 10, 11, 11);
```

```
(* Собственно гамма *)
(* tone-tone-semitone-tone-tone-tone-tone-semitone *)
const scale:array [1..ngamma] of integer=
(2, 2, 1, 2, 2, 2, 1);


(* Существует ли гамма для данной ноты *)
var valid:array [1..nnames] of boolean;
(* Для каждой такой ноты в этом массиве хранится соответствующая гамма *)
var scales:array [1..nnames, 1..ngamma] of integer;



var generatecount:integer; (* уровень вложенности рекурсии для generate*)
    startingnote:integer; (* начальная нота для generate *)
    (* массив использованных нот для generate *)
    used:array ['A'..'G'] of boolean;
(* n - номер ноты (ссылка на имя);
   dstate - что содержит данная гамма: '*' - пока неизвестно
                                       '#' - диезы
                                       'b' - бемоли
   Возвращаемое значение - удалось ли сгенерировать «хвост» гаммы
   с такими параметрами.

   Ясно, что для каждой стартовой ноты может существовать не более одной
   гаммы, удовлетворяющей ограничениям из условия *)
function generate (n:integer;dstate:char):boolean;
var newtone:integer; (* какой звук будет следующим *)
  i:integer;
begin
  generate:=false; (* скорей всего ничего не получится *)
  (* 1. проверка, что данная нота удовлетворяет dstate и f, то есть того,
    что в гамме встречаются только '#' или только 'b', и того, что каждый
    символ используется не более одного раза *)
  if used[names[n][1]] then exit; (* такой символ уже использован *)
  if length (names[n])=2 then
    if dstate='*' then dstate:=names[n][2]
    else if dstate<>names[n][2] then exit; (* нельзя '#' и 'b' одновременно *)
  (* 2. входим на новый уровень вложенности *)
  inc (generatecount);
  scales [startingnote, generatecount]:=n; (* если генерация будет
                           успешной, то scales будет содержать гамму *)
  used[names[n][1]]:=true;
  if generatecount=ngamma then generate:=true (* вся гамма сгенерирована *)
  else
    begin
      newtone:=(tone[n]+scale[generatecount]) mod 12;
      for i:=1 to nnames do if tone [i]=newtone then break;
     {цикл всегда завершается по break - в массиве есть все числа от 0 до 11}
      if generate (i, dstate) then generate:=true (* все получилось *)
      else if tone[i+1]=newtone (* нота имеет два обозначения *) then
      generate:=generate (i+1, dstate); {тогда все зависит от второй
                                                              попытки}
      (* элемент i+1 всегда есть в массиве *)
    end;
```

```
  used[names[n][1]]:=false;
  dec (generatecount); (* вернемся на уровень назад *)
end;


(* читает название ноты из входного потока (0 - '*'); возвращает индекс *)
function readnote:integer;
var c1, c2:char;
    s:string;
    i:integer;
    nfound:integer;
begin
  if ord (seekeoln)=2 then; (* пропустить все пробелы *)
  read (c1);
  if c1='*' then begin readnote:=0;exit;end;
  c1:=upcase (c1);
  read (c2);
  if c2 in [#32, #9] then s:=c1 else s:=c1+c2; (* если c2 - пробельный,
                                                  то нота простая *)
  nfound:=0;
  for i:=1 to nnames do if names [i]=s then begin nfound:=i;break;end;
  if nfound=0 then runerror (239); (* ошибка во входном файле *)
  readnote:=nfound;
end;


(* переменные цикла для основной программы *)
var c:char;
    i:integer;
    (* ноты *)
    s, d, r, t:integer;

begin
  generatecount:=0; (* процедура generate начинает с уровня 0 *)
  for c:='A' to 'G' do used[c]:=false; (* ни одной ноты не использовано *)
  for i:=1 to nnames do begin startingnote:=i;(* i - первая нота для
                                                 generate *)
                             valid[i]:=generate (i, '*');end;

  assign (input,  'major.in');  reset (input);

  repeat
    if ord (seekeof)=2 then exit; (* пропустить пробелы и переводы строки *)
    s:=readnote;
    if s=0 then break; (* конец файла *)
    if not valid [s] then
      begin
        writeln ('Key of ', names [s], ' is not a valid major key');
        repeat until readnote=0; (* пропустить остаток до '*' *)
      end
    else
      begin
        d:=readnote;
        if not valid [d] then
          begin
```

```
          writeln ('Key of ', names [d], ' is not a valid major key');
          repeat until readnote=0; (* пропустить остаток до '*' *)
        end
        else
          begin
           writeln ('Transposing from ', names [s], ' to ', names [d], ':');
           repeat
             r:=readnote;
             if r=0 then break;
             t:=0;
             for i:=1 to 7 do
              if scales[s][i]=r then begin t:=i;break;end;
             if t=0 then writeln ('  ', names[r], ' is not a valid note
                                  in the ', names[s], ' major scale')
              else
              writeln (' ', names[r], 'transposes to', names[scales[d, t]]);
            until false;
           end;
      end;
    writeln;
  until false;
end.
```

Алексеев Александр Сергеевич, студент 5-го курса математико-механического факультета СПбГУ.

Ломов Дмитрий Степанович, студент 5-го курса математико-механического факультета СПбГУ.

Станкевич Андрей Сергеевич, студент 3-го курса СПбГИТМО (ТУ).