Департамент образования администрации города Нижнего Новгорода
Нижегородский государственный университет им. Н. И. Лобачевского
Городской ресурсный центр физико-математического образования

Девятая нижегородская городская олимпиада школьников по информатике

31 января 2013 г.

Нижний Новгород
2013

Результаты, архивы и другие материалы олимпиады
можно найти на сайте `http://olympiads.nnov.ru`

Оригинал-макет подготовлен в системе LaTeX 2$_\varepsilon$
с использованием набора шрифтов LH.

Девятая городская олимпиада по информатике

31 января 2013 г. Городской ресурсный центр физико-математического образования в лице учредителей: Департамента образования администрации города Нижнего Новгорода, Нижегородского государственного университета им. Н. И. Лобачевского, Института прикладной физики РАН и МБОУ Лицей № 40 проводит Девятую городскую олимпиаду по информатике среди учащихся 6–11 классов образовательных учреждений города Нижнего Новгорода. Целью проведения олимпиады является поиск талантливой молодёжи, привлечение её в науку, повышения уровня преподавания предметов физико-математического цикла.

Генеральным спонсором городской олимпиады школьников по информатике является компания «Мера НН».

Девятая городская олимпиада проводится на трех компьютерных площадках:

- Нижегородский государственный университет им. Н. И. Лобачевского (компьютерные классы механико-математического факультета) — 25 мест;
- Институт прикладной физики Российской академии наук (компьютерные классы Научно-образовательного центра) — 23 места;
- Муниципальное бюджетное образовательное учреждение лицей № 40 (компьютерный центр) — 22 места;

Олимпиада проводится в соответствии с Положением о городской олимпиаде по информатике (Приказ № 1453 от 07.11.2008 Департамента образования и СПЗД администрации г. Нижнего Новгорода).

Разрешённые среды программирования — Borland Pascal, Free Pascal, Borland C, Borland C++, GNU C (MinGW), GNU C++ (MinGW), Basic. Ввод и вывод данных в программах осуществляется через файлы.

Регламент проведения олимпиады

9:30—10:00	Регистрация участников (на всех компьютерных площадках одновременно)
10:00—15:00	Решение задач олимпиады
15:00—15:30	Перерыв для сбора всех участников олимпиады в конференц-зале НОЦ ИПФ РАН, переезд в ИПФ РАН, кофе-брейк.
15:30—16:30	Открытое тестирование
16:30—18:00	Приветствие участников олимпиады, выступления учредителей, спонсоров. Подведение итогов олимпиады. Поздравление победителей и призёров.

Состав оргкомитета олимпиады

Швецов В. И., проректор по информатизации ННГУ им. Н. И. Лобачевского, профессор;
Сидоркина С. Л., первый заместитель директора департамента образования администрации города Нижнего Новгорода;
Лелюх В. Д., старший преподаватель ННГУ им. Н. И. Лобачевского;
Цветков М. И., начальник отдела общего среднего образования департамента образования администрации города Нижнего Новгорода;
Бовкун И. Л., главный специалист отдела общего среднего образования департамента образования администрации города Нижнего Новгорода;
Смирнов А. И., директор НОЦ ИПФ РАН, профессор;
Фейгина Т. А., заместитель директора НОЦ ИПФ РАН;
Умнова Н. С., директор муниципального бюджетного образовательного учреждения лицей № 40;
Евстратова Л. П., заместитель директора по учебно-воспитательной работе МБОУ лицей № 40;
Гашпар И. Л., куратор НОЦ ИПФ РАН;
Братчикова Т. А., учитель МБОУ лицей № 40;
Денисов В. В., зав. лабораторией ННГУ им. Н. И. Лобачевского.

Состав предметной комиссии (жюри)

Председатель — Лелюх В. Д., старший преподаватель ННГУ;

Члены комиссии:
Евстратова Л. П., заместитель директора МОУ лицей № 40, ответственный секретарь комиссии;
Вадимов В. Л., студент 4 курса ВШОПФ ННГУ;
Демидов А. Н., инженер, НПП «ПРИМА»;
Епифанов В. Ю., студент 4 курса мехмата ННГУ;
Калинин П. А., старший инженер по разработке программного обеспечения, ЗАО «Интел А/О»;
Кашинская Я. С., студентка 2 курса ФИВТ МФТИ;
Матросов М. В., инженер-программист, НПП «АВИАКОМ»;
Разенштейн И. П., аспирант первого года EECS MIT;
Тимин А. А., студент 2 курса ФИВТ МФТИ;
Тимушев Р. И., старший инженер-программист, Grid Dynamics;
Шмелёв А. С., инженер-программист, НПП «ПРИМА».

Ниже приведены примеры текстов программ, написанных на разрешённых языках программирования. Программы считывают данные (два числа в одной строке) из файла с именем `example.in` и выводят в файл с именем `example.out` их сумму:

Pascal

```
var buf,bufo:text;
    a,b:integer;
begin
  assign(buf,'example.in');
  reset(buf);
  read(buf,a,b);
  assign(bufo,'example.out');
  rewrite(bufo);
  writeln(bufo,a+b);
  close(buf);
  close(bufo);
end.
```

C/C++

```
#include <stdio.h>
int main() {
  FILE *buf,*bufo;
  int a,b;
  buf=fopen("example.in","r");
  fscanf(buf,"%d %d",&a,&b);
  fclose(buf);
  bufo=fopen("example.out","w");
  fprintf(bufo,"%d\n",a+b);
  fclose(bufo);
  return 0;
}
```

Basic

```
OPEN "example.in" FOR INPUT AS #1
OPEN "example.out" FOR OUTPUT AS #2
INPUT #1, A, B
PRINT #2, A + B
CLOSE #2, #1
```

IX Городская олимпиада школьников по информатике 31 января 2013 г.

Задача 1. Расписание маршруток

Входной файл	`schedule.in`
Выходной файл	`schedule.out`
Ограничение по времени	1 с
Ограничение по памяти	256 МиБ
Максимальный балл за задачу	100

Оценив последние успехи вашего друга в экономическом отделе компании, владеющей сетью маршрутных такси, директор повысил его до главного диспетчера. Теперь друг указывает, на какой маршрут должна выходить та или иная маршрутка.

В автопарке компании есть n маршруток, i-ая маршрутка номинально вмещает a_i пассажиров. По договору с департаментом транспорта города компания обязана обслуживать m маршрутов. Накопленная статистика говорит, что оптимальнее всего, если j-ый маршрут обслуживает такси номинальной вместимостью b_j пассажиров. Каждая маршрутка приписывается не более чем к одному маршруту, каждому маршруту приписывается не более одной маршрутки.

Разумеется, каждый уважающий себя диспетчер при назначении маршруток на маршруты старается минимизировать потери, которые бывают следующими:

- если i-ая маршрутка обслуживает j-ый маршрут, то компания теряет $|a_i - b_j|$ у.е., так как чем меньше заполнено такси, тем больше не используются его возможности, а чем больше переполнено такси, тем чаще его приходится ремонтировать;
- от каждой простаивающей маршрутки, то есть такой, которой не назначен ни один маршрут, компания несет убыток p у.е.;
- компании приходится платить штраф q у.е. департаменту транспорта за каждый не обслуживаемый маршрут.

В очередной раз вы хотите помочь другу и написать для него программу, облегчающую ему работу.

Формат входных данных

В первой строке входного файла находятся четыре целых числа — n, m, p, q ($1 \leqslant n, m \leqslant 10^3$, $0 \leqslant p, q \leqslant 10^4$).

Во второй строке через пробел указаны целые числа $a_1, \ldots, a_n$ ($1 \leqslant a_i \leqslant 10^4$).

В третьей строке через пробел указаны целые числа $b_1, \ldots, b_m$ ($1 \leqslant b_j \leqslant 10^4$).

Формат выходных данных

Выведите единственное число — минимально возможные потери компании.

Пример

Входной файл	*Выходной файл*
2 2 100 100 22 12 11 20	3
2 1 100 500 13 13 13	100

Примечание: В примере 1 первая маршрутка назначена на второй маршрут с потерями $|22-20| = 2$ у.е., вторая маршрутка назначена на первый маршрут с потерями $|12-11| = 1$ у.е.. Итого: потери 3 у.е..

В примере 2 одна из маршруток назначается на единственный маршрут с нулевым штрафом, а вторая вынуждена простаивать. Итого: потери 100 у.е.

Решение

Отсортируем по неубыванию массивы a_i и b_j. Теперь можно сделать ценное наблюдение: оптимальное решение можно искать среди тех, у которых нет «перехлестов». Под «перехлестом» будем подразумевать ситуацию, когда маршруткам с номерами i_1 и i_2 назначены маршруты j_1 и j_2 соответственно, причем $i_1 < i_2$, но $j_1 > j_2$.

Действительно, если у нас есть назначение, содержащее «перехлесты», то можно постепенно уменьшать количество «перехлестов», не увеличивая при этом потери. Для этого достаточно брать очередной «перехлест» (i_1, j_1), (i_2, j_2) и переназначать маршрутке i_1 маршрут j_2, а маршрутке i_2 маршрут j_1, оставляя остальные назначения как есть. Нетрудно показать, что от такой замены общее количество «перехлестов» строго уменьшается, а потери изменяются (уменьшаются) на величину

$$|a_{i_1} - b_{j_1}| + |a_{i_2} - b_{j_2}| - |a_{i_1} - b_{j_2}| - |a_{i_2} - b_{j_1}|$$

которая при отсортированных a_i и b_j является величиной неотрицательной.

Теперь все готово, чтобы решить задачу методом динамического программирования. Обозначим через $f[i, j]$ минимальные потери компании, если бы у нее были только первые i маршруток (вместимостью $a_1, \ldots, a_i$) и нужно было бы обслужить только первые j маршрутов $(b_1, \ldots, b_j)$. Минимальные потери для каждой подзадачи дает некоторое оптимальное назначение — совокупность пар (маршрутка, маршрут). Заметим, что для разных подзадач оптимальное назначение может совпадать, однако за счет разного количества простаивающих маршруток и необслуживаемых маршрутов потери могут получиться разными.

Посчитаем f для простейших крайних значений. Так, например, $f[i, 0] = p \cdot i$, так как если бы у компании были только i маршруток, но ни одного маршрута, то каждая маршрутка простаивала бы с потерей p.

Аналогично, $f[0, j] = j \cdot q$, так как при отсутствии маршруток за каждый из j маршрутов пришлось бы заплатить в казну штраф q.

После того, как мы разобрались с крайними случаями, посмотрим, как можно эффективно посчитать $f[i, j]$ при $i > 0$ и $j > 0$, если для всех меньших значений параметров i', j' величины $f[i', j']$ уже посчитаны.

Для оптимального назначения для подзадачи $f[i, j]$ возможны следующие случаи:

- В оптимальное назначение входит как i-ая маршрутка, так и j-ый маршрут. Нетрудно заметить, что тогда i-ая маршрутка назначена именно на j-ый маршрут, иначе был бы «перехлест». Оптимальное назначение для задачи $f[i, j]$ без пары (i, j) будет оптимальным и для подзадачи $f[i-1, j-1]$ (иначе, «склеив» оптимальное назначение для подзадачи $f[i-1, j-1]$ с парой (i, j), можно было бы получить еще более оптимальное решение для $f[i, j]$). Поэтому для данного случая $f[i, j] = |a_i - b_j| + f[i-1, j-1]$.
- В оптимальное назначение не входит i-ая маршрутка или j-ый маршрут (может быть одновременно и то, и другое).
  Если i-ая маршрутка простаивает, то оптимальное назначение для подзадачи $f[i, j]$ совпадает с оптимальным назначением для подзадачи $f[i-1, j]$, а отличие в потерях будет только за счет простоя: $f[i, j] = f[i-1, j] + p$.
  Если j-ый маршрут не обслуживается, то оптимальное назначение для подзадачи $f[i, j]$ совпадает с оптимальным назначением для подзадачи $f[i, j-1]$, а отличие в потерях будет только за счет штрафа государству, т.е. $f[i, j] = f[i, j-1] + q$.

Так как мы заранее не знаем, какой именно случай имеет место на самом деле, то нужно рассмотреть все варианты и взять минимум:

$$f[i, j] = \min(f[i-1, j] + p, \quad f[i, j-1] + q, \quad f[i-1, j-1] + |a_i - b_j|)$$

После того, как все f посчитаны, остается вывести ответ на подзадачу $f[n, m]$, которая, очевидно, совпадает с исходной.

Сложность решения — $O(\max(n, m)^2)$, так как всего подзадач $n \cdot m$, решение каждой из них по приведенной выше формуле занимает фиксированное время. При ограничениях, указанных в условии, этого достаточно. Отметим также, что начальная сортировка, выполненная даже за квадратичное время, а также подсчет крайних вариантов за линейное время не ухудшат асимптотику.

Пример правильной программы

```
{$APPTYPE CONSOLE}
uses SysUtils, Math;
const MAXN = 1000;
type integer = longint;
var    w: text;
    a, b: array[1..MAXN] of integer;
       f: array[0..MAXN, 0..MAXN] of integer;
    n, m: integer;
    p, q: integer;
    i, j: integer;
  i1, i2: integer;

  procedure swap(var x, y: integer);
  var tmp: integer;
  begin
    tmp := x;
    x := y;
    y := tmp;
  end;

begin
  fillchar(a, sizeof(a), $3f);
  fillchar(b, sizeof(b), $3f);
  assign(w, 'schedule.in'); reset(w);
  read(w, n, m, p, q);
  for i := 1 to n do read(w, a[i]);
  for j := 1 to m do read(w, b[j]);
  close(w);
  for i := 1 to max(n, m) do
  begin
    i1 := i;
    i2 := i;
    for j := i + 1 to max(n, m) do
    begin
      if a[j] < a[i1] then i1 := j;
      if b[j] < b[i2] then i2 := j;
```

```
    end;
    swap(a[i], a[i1]);
    swap(b[i], b[i2]);
  end;
  for i := 0 to n do f[i, 0] := i*p;
  for j := 0 to m do f[0, j] := j*q;
  for i := 1 to n do
    for j := 1 to m do
    begin
      f[i, j] := min(f[i-1, j] + p,
                     f[i, j-1] + q);
      f[i, j] := min(f[i, j],
                 f[i-1, j-1] + abs(a[i] - b[j]));
    end;
  assign(w, 'schedule.out'); rewrite(w);
  writeln(w, f[n, m]);
  close(w);
end.
```

Задача 2. Праздничные дни

Входной файл	`holidays.in`
Выходной файл	`holidays.out`
Ограничение по времени	1 с
Ограничение по памяти	256 МиБ
Максимальный балл за задачу	100

В Тридевятом царстве царь был любителем разных заморских традиций. Как прознает, что в другом царстве есть какой-то обычай, сразу думает, как бы его к тридевятым реалиям приспособить.

Вот неделю назад вернулось посольство из Тридесятого царства. И главный посол доложил царю: дескать, придумал Тридесятый царь следующую вещь. Чтобы как-то зарегулировать гуляния народные, повелел он указать определенные дни, и в эти дни устраивать широкие гуляния, а в остальные дни массовые сборища запретить. И с тех пор жизнь в Тридесятом царстве стала прекрасной: гулять так гулять, работать так работать, и все строго по цареву указу.

Понравилась мысль такая царю Тридевятого царства. Подумал он ввести и у себя такие порядки. Собрал царь советников своих, и говорит: подготовьте мне список дней, в которые гулять можно. Только не на год, а на N дней вперед — посмотрим, дескать, что получится; понравится — сделаем круглогодичным.

И вот вчера принесли советники царю список. Но вот незадача: каждый советник свой список приготовил, да еще и обоснование предложил, какой праздник в какой из этих дней надо отмечать. И у всех советников праздники важные, но у всех — разные! Царь думал-думал и решил: а возьмем их все — объединим предложения советников! Если какой-то день есть в списке хотя бы одного советника, то объявим этот день праздничным, и пускай народ гуляет! Глядишь, и не будет недовольных.

Только одна проблема осталась: некоторые дни оказались в списках сразу у нескольких советников. Но царь и тут нашел выход: перенесем некоторые праздники на более поздние дни, так, чтобы в каждый день получался только один праздник, и переносы были бы как можно короче.

Пусть, например, четыре советника сразу предложили сделать некоторый день (пускай день 5) праздничным. Тогда перенесем три из этих четырех праздников на дни 6, 7 и 8 — так, что праздничными будут дни с 5 по 8 включительно. А если оказывается, что, например, день 7 тоже предложен в качестве праздничного кем-нибудь из советников, то перенесем этот праздник еще дальше — на день 9.

Напишите программу, которая, зная предложения советников, определит, какие дни будут праздничными, а какие нет. Не забывайте, что праздники можно переносить только на более поздние дни; на более ранние переносить нельзя.

Формат входных данных

На первой строке входного файла находится одно число N — количество дней, на которые царь хочет произвести планировку праздников.

На второй строке входного файла находятся N неотрицательных целых чисел — для каждого дня указано, сколько советников предложили считать его праздничным.

Гарантируется, что $1 \leqslant N \leqslant 100\,000$, и что сумма всех чисел во второй строке входного файла не превосходит 100 000.

Формат выходных данных

В выходной файл выведите одну строку, состоящую из символов '+' или '-' (без апострофов). Символом '+' обозначайте праздничный день, '-' — непраздничный. Выведите как минимум N символов — по одному для каждого из дней, на которые проводится планирование. Но если праздники приходится переносить на дни после N-ого (что допустимо), то выведите больше символов — до последнего праздничного дня.

Символы разделяйте пробелами.

Пример

Входной файл	*Выходной файл*
5 0 3 0 0 0	- + + + -
10 0 4 0 2 0 0 0 0 1 0	- + + + + + + + - + -
3 0 3 0	- + + +

Решение

Это была одна из самых простых задач олимпиады. Будем обрабатывать дни последовательно. А именно, для каждого дня считаем из файла его «кратность» и сразу же решим и выведем в выходной файл, будет этот день праздничным или нет. При этом в каждый момент времени у нас фактически будет считано некоторое количество p праздников (это количество будет равно сумме всех считанных чисел), и некоторое количество q праздников уже будет выведено в выходной файл (это количество равно количеству выведенных плюсов). Соответственно, у нас в каждый момент будет некоторое количество («баланс») еще «не распределенных» праздников, равное разности величин p и q.

Будем эту разность поддерживать в переменной s. Тогда программа пишется легко. Считав очередную кратность, увеличим баланс s на соответствующую величину. Далее, если теперь баланс положительный, т.е. есть еще праздники, которые не распределены (учитывая те, которые мы только что считали), то, очевидно, текущий день надо назначить праздником. В таком случае выводим плюс и уменьшаем s на единицу. Если же баланс s был равен нулю, то очевидно выводим минус и не меняем s.

После обработки всех дней надо не забыть вывести плюсы, соответствующие праздникам, перенесенным позже N-ого дня — количество таких плюсов будет равно значению переменной s после обработки всех входных данных.

Пример правильной программы

```
var f,g:text;
    n,i:integer;
    a,s:integer;
begin
assign(f,'holidays.in');reset(f);
assign(g,'holidays.out');rewrite(g);
read(f,n);
s:=0;
for i:=1 to n do begin
    read(f,a);
    s:=s+a;
    if s>0 then begin
        write(g,'+ ');
        dec(s);
    end else write(g,'- ');
end;
for s:=s downto 1 do
    write(g,'+ ');
writeln(g);
close(g);
close(f);
end.
```

Задача 3. Выставка ноутбуков

Входной файл	`http.in`
Выходной файл	`http.out`
Ограничение по времени	1 с
Ограничение по памяти	256 МиБ
Максимальный балл за задачу	100

Сеть компьютерных салонов «ХТТП» представлена в городе Н. двумя магазинами. Руководство Н-ского отделения сети решило реорганизовать витрины, на которых представлены ноутбуки. В каждом из двух магазинов на витрине должны быть представлены N моделей ноутбуков, выставленные в ряд от касс вглубь помещения магазина. Маркетологи каждого из магазинов уже определили порядок, в котором на витрине должны быть расположены эти модели (эти порядки в двух магазинах, конечно же, могут быть разными).

На витрины надо выставлять специальные, выставочные, образцы ноутбуков, с соответствующим программным обеспечением, показывающим рекламу, и т.д. В распоряжении администрации компьютерных салонов есть две версии специализированного ПО: работающие под управлением операционных систем Windows и Linux. Соответственно, каждый из выставочных образцов ноутбуков должен иметь предустановленной ровно одну из этих систем.

Для снижения затрат было принято решение закупить по два идентичных экземпляра каждой модели ноутбуков (т.е. с одинаковыми предустановленными операционными системами), отправить по одному экземпляру в каждый из магазинов сети, и расставить их на витрине в соответствии с порядком, определенным маркетологами того магазина.

Но при этом возникла проблема. По требованиям Федеральной антимонопольной службы, компьютерные салоны не должны предоставлять преимущества ни одной из операционных систем. Начальство сети «ХТТП» знает, как проходит проверка на соответствие этой норме законодательства. Инспектор антимонопольной службы идет по магазину начиная от касс вдоль витрины с ноутбуками, считает отдельно количество встреченных ноутбуков с Windows и Linux, а также модуль разности этих чисел (т.е. на сколько ноутбуков с одной системой он встретил больше, чем ноутбуков с другой системой). В некоторый момент он останавливается и

говорит: «Ага!». Это значит: слишком у вас большой дисбаланс между операционными системами, поэтому платите штраф. Размер штрафа пропорционален разнице (взятой по модулю) количества ноутбуков с разными системами, которые увидел инспектор.

Естественно, руководство сети не в состоянии предсказать, в какой из магазинов пойдет инспектор, а также сколько ноутбуков он просмотрит. Тем не менее, они хотят минимизировать штраф, который им будет выписан *в худшем случае*. Помогите им.

Например, пусть $N = 5$: в магазинах должны быть выставлены пять моделей ноутбуков. Будем нумеровать модели ноутбуков от 1 до 5. Пусть в первом магазине маркетологи определили, что оптимальный порядок ноутбуков следующий (от касс вглубь зала):

2 4 1 3 5,

а во втором магазине —

3 1 2 5 4.

Тогда, если закупить ноутбуки моделей 1, 3 и 4 с операционной системой Windows, а 2 и 5 — с Linux, то порядок операционных систем в магазинах будет следующий (от касс вглубь зала):

L W W W L,
W W L L W.

Тогда, если, например, инспектор придет в первый магазин и просмотрит первые четыре ноутбука, то он скажет: «Ага!», и выпишет штраф за то, что он увидел Windows-ноутбуков на два больше, чем Linux. Аналогичный результат будет, если он придет во второй магазин и просмотрит только первые два ноутбука.

А если закупить ноутбуки 2 и 3 с системой Linux, а 1, 4, 5 — с Windows, то порядок операционных систем будет следующий:

L W W L W,
L W L W W,

и в какой бы магазин не пришел инспектор, и сколько бы ноутбуков он не посмотрел, разница Windows- и Linux-ноутбуков не будет превосходить по модулю 1, и это и будет оптимальным вариантом для руководства сети. (Напомним, что инспектор всегда начинает осмотр магазина от касс и идет вглубь магазина вдоль ряда с ноутбуками).

Формат входных данных

В первой строке входного файла записано одно число N ($1 \leqslant N \leqslant 10^5$) — количество моделей ноутбуков, которые должны быть представлены на витрине. Модели ноутбуков нумеруются от 1 до N.

Далее следуют две строки по N чисел в каждой — порядки моделей ноутбуков на витрине, определенные маркетологами первого и второго магазина, от касс вглубь зала. Гарантируется, что порядки корректные, т.е. что в каждой из этих строк все числа находятся в интервале от 1 до N и никакое из чисел не встречается в одной строке дважды.

Формат выходных данных

В выходной файл выведите строку из N символов, описывающих необходимую

конфигурацию ноутбуков. А именно, i-ый символ должен быть 'W' (без кавычек), если i-ую модель ноутбуков надо закупать с предустановленной системой Windows, и 'L' для Linux.

Если есть несколько оптимальных решений, выведите любое.

Пример

Входной файл	*Выходной файл*
5 2 4 1 3 5 3 1 2 5 4	WLLWW
5 1 4 2 3 5 5 1 3 4 2	WLWLL

Решение

Очевидно, что в *любом* решении, если инспектор посмотрит только один ноутбук, то разница количества Windows- и Linux-ноутбуков будет равна единице, и инспектор выпишет соответствующий штраф. Поэтому если мы найдем решение, в котором разница количества Windows- и Linux-ноутбуков не будет превосходить по модулю единицы *независимо* от действий инспектора, то это будет оптимальное решение.

Заметим, что если в первом магазине первый и второй ноутбук будут иметь разные операционные системы, третий и четвертый тоже будут иметь разные ОС, пятый и шестой тоже, и т.д., то сколько бы ноутбуков в первом магазине не посмотрел инспектор, разница количества Windows- и Linux-ноутбуков не превзойдет 1; аналогичное утверждение, конечно, справедливо и для второго магазина. Если обозначить последовательность типов ноутбуков в первом магазине как $a_1, a_2, \ldots, a_N$, а во втором — $b_1, b_2, \ldots, b_N$ (это как раз две последовательности, заданные во входном файле), то описанное выше условие обозначает, что ноутбуки типов a_1 и a_2 должны иметь разные ОС, ноутбуки типов a_3 и a_4 разные ОС, типов a_5 и a_6 — тоже, и т.д.; и аналогично для пар (b_1, b_2); (b_3, b_4) и т.д.

Покажем, как построить решение такого вида. Для этого построим граф, в котором будет N вершин, соответствующих N типам ноутбуков. Соединим в нем ребрами вершины a_1 и a_2, вершины a_3 и a_4, вершины a_5 и a_6, и т.д.; и вершины b_1 и b_2, вершины b_3 и b_4 и т.д. Несложно доказать, что в этом графе не будет циклов нечетной длины. Следовательно, этот граф будет двудольным, т.е. его вершины можно будет раскрасить в два цвета так, чтобы каждое ребро соединяло вершины разных цветов. Несложно видеть, что это и будет требуемое решение задачи.

Пример правильной программы

```
{$m 10000000,1024}
const os:array[1..2] of char='WL';
      maxn=100000;
var f:text;
    n,i:integer;
    gr:array[1..maxn,1..2] of integer;
    c:array[1..maxn] of integer;

procedure readperm(m:integer);
var i:integer;
    a:array[1..maxn] of integer;
    u,v:integer;
begin
for i:=1 to n do
```

```
    read(f,a[i]);
for i:=1 to n div 2 do begin
    u:=a[2*i-1];
    v:=a[2*i];
    gr[u,m]:=v;
    gr[v,m]:=u;
end;
end;

procedure find(i:integer;col:integer);
var j:integer;
begin
if c[i]<>0 then exit;
c[i]:=col;
for j:=1 to 2 do
    if gr[i,j]<>0 then
       find(gr[i,j],3-col);
end;

begin
assign(f,'http.in');reset(f);
read(f,n);
fillchar(gr,sizeof(gr),0);
readperm(1);
readperm(2);
fillchar(c,sizeof(c),0);
for i:=1 to n do
    if c[i]=0 then
       find(i,1);
assign(f,'http.out');rewrite(f);
for i:=1 to n do
    write(f,os[c[i]]);
close(f);
end.
```

Задача 4. Тестирование программы

Входной файл	`testing.in`
Выходной файл	`testing.out`
Ограничение по времени	1 с
Ограничение по памяти	256 МиБ
Максимальный балл за задачу	100

Вчера у Васи был счастливый день: он наконец дописал программу всей своей жизни! И, недолго думая, он тут же запустил ее на Самом Главном Тесте.

Программа Васи очень сложная, и потому работает она долго. Поэтому Вася пошел спать, а наутро, сегодня, обнаружил, что программа, вместо того, чтобы посчитать нужный Ответ, вылетела с непонятной ошибкой.

Вася понял, что напрасно он не тестировал программу. Ведь Самый Главный Тест очень сложный — в нем есть N отдельных подзадач, и каждую из них надо решить. Конечно, надо было бы начать тестирование с меньшего количества подзадач, но ведь Вася — очень умный программист! И он абсолютно уверен, что его программа корректно решила все подзадачи, кроме какой-то одной. Вот только Вася не знает, какой именно.

Вася может изменять Самый Главный Тест, отключая в нем те или иные подзадачи. Он надеется, что, если среди отключенных будет та подзадача, на которой его программа не работает, то получившийся тест его программа спокойно пройдет. Но вот проблема: программа работает долго, а подзадач много, и потому, если отключать задачи по одной, то придется очень долго искать нужную. А Вася очень хочет узнать, где ошибка, уже завтра!

К счастью, у Васи в распоряжении есть много компьютеров. Он может на некоторых из них запустить свою программу на каком-то тесте (т.е. на Самом Главном Тесте с некоторыми отключенными подзадачами), а назавтра посмотреть, какие программы завершились успешно, а какие нет, — и по результатам понять, какая подзадача создавала проблемы. Помогите Васе подобрать тесты для каждого из компьютеров, т.е. объясните ему, какие подзадачи в каком тесте он должен отключить, так, чтобы назавтра, узнав результаты работы программы на этих тестах, он смог бы уверенно определить, с какой подзадачей у него возникают проблемы

(естественно, считая, что такая подзадача только одна, ведь Вася — очень умный программист!)

Учтите, что Васе не хочется делать лишнюю работу по запуску программ и определению их результата, поэтому он хочет минимизировать количество запусков (т.е. фактически количество компьютеров, ведь его программа настолько сложная, что на одном компьютере можно запустить только один экземпляр программы).

Формат входных данных

В первой и единственной строке входного файла находится одно целое число N — количество подзадач в Самом Главном Тесте ($1 \leqslant N \leqslant 10^5$).

Формат выходных данных

В первой строке выходного файла выведите минимальное необходимое количество компьютеров M. В последующих M строках выведите информацию о том, на каком тесте надо запускать программу на каком компьютере. А именно, в i-ую из них выведите последовательность чисел k_i, b_{i1}, b_{i2}, … b_{ik_i}, где k_i — количество подзадач, которые надо отключить в тесте, на котором будет работать программа на i-ом компьютере, а b_{ij} ($1 \leqslant j \leqslant k_i$, $1 \leqslant b_{ij} \leqslant N$) — номера подзадач, которые надо отключать. Числа b_{ij} должны быть различны для каждого фиксированного i.

Подзадачи нумеруются от 1 до N.

В пределах одной строки подзадачи можете выводить в любом порядке. Если есть несколько оптимальных решений, выведите любое.

Пример

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

Примечание: В примере:

- если ошибка возникала на первой подзадаче, то на всех трех компьютерах программа отработает верно, т.к. первая подзадача во всех тестах отключена;
- если ошибка возникала на второй подзадаче, то ошибочно сработает только первый компьютер;
- если ошибка возникала на третьей подзадаче, то ошибочно сработает только второй компьютер;
- если ошибка возникала на четвертой подзадаче, то ошибочно сработают первый и второй компьютеры;
- если ошибка возникала на пятой подзадаче, то ошибочно сработает только третий компьютер.

Поскольку во всех пяти вариантах множества ошибочно сработавших компьютеров различны, то Вася наутро, узнав, на каких компьютерах программа сработала корректно, а на каких — нет, сможет однозначно определить ошибочную подзадачу.

Решение

Докажем, что необходимое минимальное число компьютеров не меньше, чем $\lceil \log_2 N \rceil$. (Запись $\lceil x \rceil$ обозначает округление вверх до ближайшего целого числа.) Неправильная работа на какой-то подзадаче приведет к тому, что некоторые запуски приведут к ошибке (те, на которых эта подзадача не была отключена). Если другая подзадача приводит к ошибке, то набор неудачных запусков может быть другим. Мы можем однозначно определить плохую подзадачу, если для любых двух подзадач наборы упавших запусков различаются. Количество различных наборов неудачных запусков равно 2^M, поскольку каждый запуск может завершиться одним из двух способов. Если $2^M < N$, то однозначно восстановить плохую подзадачу невозможно, в силу принципа Дирихле.

Приведем конструктивное доказательство того, что оптимальное M равно именно $\lceil \log_2 N \rceil$. Пронумеруем подзадачи от 0 до $N-1$. Компьютер номер i будет решать j-ю подзадачу ($0 \leqslant j \leqslant N-1$) тогда и только тогда, когда на i-й младшей позиции двоичной записи числа j находится единица. Очевидно, нам понадобится ровно $\lceil \log_2 N \rceil$ компьютеров, поскольку на более старших позициях все двоичные цифры равны нулю. Если упал запуск на i-м компьютере, то на i-й позиции в двоичной записи номера плохой подзадачи находится единица, в противном случае ноль. Таким образом, мы можем восстановить все цифры из двоичной записи номера проблемной подзадачи, следовательно, и сам номер подзадачи.

Пример правильной программы

```
var fin, fout: text;
    n, ans, i, j, m: longint;
begin
  assign(fin, 'testing.in');
  reset(fin);
  assign(fout, 'testing.out');
  rewrite(fout);
  read(fin, n);
  ans := 0;
  while (1 shl ans) < n do
    inc(ans);
  writeln(fout, ans);
  for i := 0 to ans - 1 do begin
    m := 0;
    for j := 0 to n - 1 do
      if (j and (1 shl i)) = 0 then
        inc(m);
    write(fout, m);
    for j := 0 to n - 1 do
      if (j and (1 shl i)) = 0 then
        write(fout, ' ', j + 1);
    writeln(fout);
  end;
  close(fin);
  close(fout);
end.
```

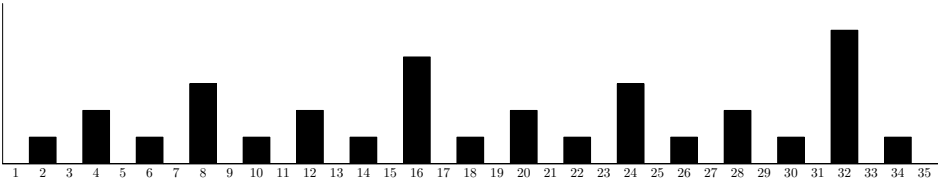
Задача 5. Раскраска автобусов

Входной файл	paint.in
Выходной файл	paint.out
Ограничение по времени	1 с
Ограничение по памяти	256 МиБ
Максимальный балл за задачу	100

Ваш друг по-прежнему работает главным экономистом в компании, владеющей сетью маршруток. В связи с новыми веяниями по введению корпоративной раскраски на транспортных средствах, компания решила окрасить все свои маршрутки в новом стиле. Директор компании своим личным распоряжением приказал принять следующий способ окраски. Сначала все автобусы окрашиваются фоновой красной краской, поверх которой наносится серый узор.

Узор создается следующим образом. Если бы борт маршрутки был бы бесконечным в две стороны: вправо и вверх, — то узор рисовали бы так. Разделим борт на бесконечное количество вертикальных полос одинаковой (единичной) ширины, занумеруем их слева направо, начиная с единицы. Полосы с нечетными номерами красить серой краской вообще не будем. Полосы, номера которых делятся на два, но не делятся на четыре, закрасим снизу до высоты, равной единице длины (т.е. образуется серый квадрат). Полосы, номера которых делятся на четыре, но не делятся на восемь, закрасим снизу до высоты, равной двум единицам длины; полосы, номера которых делятся на восемь, но не делятся на 16 — до высоты, равной 3; номера которых делятся на 16, но не делятся на 32 — до высоты, равной 4, и т.д.

Получим следующий узор:



Естественно, у реальных маршруток ширина борта ограничена (высоту мы для простоты будем считать неограниченной). Можно было бы на каждой маршрутке рисовать начало такого узора, но это не интересно — поэтому было решено для каждой маршрутки выбрать два числа, l и r, и нарисовать на борту фрагмент этого узора с l-ого столбика по r-ый включительно. Определите, сколько на это уйдет серой краски, считая, что единица краски уходит на единичный квадрат узора.

Формат входных данных

Во входном файле находятся два числа — l и r. Гарантируется, что $1 \leqslant l \leqslant r \leqslant 10^{18}$.

Формат выходных данных

Выведите в выходной файл одно число — общую площадь фрагмента узора между l-ым и r-м столбиками включительно.

Пример

Входной файл	*Выходной файл*
5 10	5

Примечание: В примере используются столбики с 5 по 10 включительно. Их площади — соответственно 0, 1, 0, 3, 0, 1 единичных квадратов; соответственно, суммарная их площадь равна 5.

Примечание: Среди тестов будут такие, в которых $r \leqslant 10^5$, общей стоимостью 40 баллов, и тесты с $r > 10^5$, но $r - l \leqslant 10^5$, общей стоимостью еще 20 баллов.

Решение

Сначала научимся делать следующее: по данному n научимся находить количество краски, требующейся на первые n столбиков — с 1 по n-ый включительно. Обозначим это количество краски за $f(n)$. Если мы научимся его быстро находить, то ответ на задачу, очевидно, будет равен $f(r) - f(l-1)$.

Как найти $f(n)$? Во-первых, все столбики с нечетными номерами имеют высоту ноль. Удалим их все, т.е. сместим четные столбики к началу картины так, чтобы между ними не оставалось пустого места — чтобы они были нарисованы в два раза плотнее, чем были сначала. У нас осталось $\lfloor n/2 \rfloor$ столбиков (скобки $\lfloor\ \rfloor$ обозначают округление вниз, т.е. взятие целой части), и у каждого такого столбика высота как минимум 1. Отрежем у каждого столбика нижний квадратик. После этого несложно видеть, что мы получим кусок той же картины столбиков, с которой мы начинали — только теперь у нас будет изображено $\lfloor n/2 \rfloor$ столбиков. Тогда их суммарная площадь равна $f(\lfloor n/2 \rfloor)$. Вспомним, что мы отрезали у $\lfloor n/2 \rfloor$ столбиков нижние квадратики — поэтому окончательно получаем

$$f(n) = \lfloor n/2 \rfloor + f(\lfloor n/2 \rfloor)$$

Данная функция уже легко вычисляется рекурсивно; можно также написать простой цикл `while`, как это сделано в примере решения ниже.

Отметим также, что при маленьких l и r задачу можно было решить, просто перебрав все столбики от l до r и вычислив их высоту; такое решение тоже набирало некоторые баллы.

Пример правильной программы

```
var l,r: int64;
    f: text;

function calc (a: int64): int64;
begin
    result := 0;
    while a > 0 do begin
        a := a div 2;
        result := result + a;
    end;
end;

begin
    assign (f, 'paint.in');
    reset (f);
    read (f, l, r);
    close (f);

    assign (f, 'paint.out');
    rewrite (f);
    writeln (f, calc(r) - calc(l - 1));
    close (f);
end.
```

Задача 6. Руммикуб

Входной файл	`rummikub.in`
Выходной файл	`rummikub.out`
Ограничение по времени	1 с
Ограничение по памяти	256 МиБ
Максимальный балл за задачу	100

В игре «Руммикуб» используются фишки, которые бывают четырех различных цветов, и на каждой из которых написано одно натуральное число от 1 до 13. Для каждого числа и для каждого цвета в наборе фишек есть ровно две соответствующие фишки, т.е. всего в наборе $8 \cdot 13 = 104$ фишки.

Число, написанное на фишке, будем называть ее *достоинством*; цвета будем обозначать латинскими буквами A, B, C и D, и каждую фишку будем обозначать, записывая сначала ее цвет, а потом — ее достоинство. Например, `C12` — это фишка цвета C и достоинством 12.

Комбинацией в игре называется набор из как минимум трех фишек, удовлетворяющий любому из следующих условий:

- Достоинства всех фишек одинаковы, а цвета — попарно различны; или

- Цвета всех фишек одинаковы, а достоинства являются последовательными натуральными числами.

Например, следующие наборы фишек являются комбинациями:

- `C12`, `A12`, `B12`;
- `C12`, `A12`, `B12`, `D12`;
- `C5`, `C6`, `C7`;
- `A3`, `A4`, `A5`, `A6`, `A7`.

При этом следующие наборы *не являются* комбинациями:

- `A3`, `B3` (слишком мало фишек);
- `A3`, `B3`, `C3`, `D3`, `B3` (цвета повторяются);
- `A3`, `A4`, `A4`, `A5`, `A6` (достоинства повторяются);
- `A3`, `A4`, `A6`, `A7`, `A8` (число 5 пропущено).

Одна из основных задач в руммикубе состоит в том, чтобы данный набор фишек распределить на комбинации так, чтобы каждая фишка входила ровно в одну комбинацию. Напишите программу, которая будет это делать.

Формат входных данных

В первой строке входного файла находится одно натуральное число K — количество фишек. Далее следуют K строк, на каждой из которых находится описание одной фишки — цвет и достоинство.

Гарантируется, что эти фишки являются корректным подмножеством фишек из некоторого комплекта для игры в руммикуб; т.е. гарантируется, что каждый цвет — это латинская заглавная буква от A до D, что каждое достоинство — это натуральное число, не превосходящее 13, и что для для каждой пары (цвет, достоинство) в наборе есть не более двух таких фишек.

Формат выходных данных

Если данный набор фишек можно разбить на комбинации так, чтобы каждая фишка входила ровно в одну комбинацию, то в первую строку выходного файла выведите одно число M — количество комбинаций в вашем решении. Далее выведите M строк, в i-ой из которых выведите i-ую комбинацию. А именно, сначала выведите количество фишек в комбинации, а потом сами фишки, разделенные между собой и отделенные от количества фишек пробелами. В пределах каждой комбинации фишки можете выводить в произвольном порядке; комбинации также можете выводить в произвольном порядке.

Если есть несколько решений, выведите любое. Если решений нет, то выведите в выходной файл одно число `-1`.

Пример

Входной файл	*Выходной файл*
3 A2 A3 A5	-1
3 A2 A4 A3	1 3 A2 A4 A3
7 A12 A13 A13 B13 C13 D13 A11	2 3 A11 A12 A13 4 A13 B13 C13 D13

Решение

Будем решать задачу методом динамического программирования. Состоянием будет пара (x, p), обозначающая следующее: пусть мы выкинули все фишки достоинством меньше p. Будем в этой подзадаче называть корректным такое распределение фишек на комбинации, что все комбинации удовлетворяют условию, за исключением, возможно, комбинаций второго типа, включающих фишки достоинства p — для таких комбинаций снимем ограничение «как минимум три фишки» (имея ввиду, что впоследствии к ним можно будет добавить фишки достоинством меньше p). Параметр x будет обозначать как раз конфигурацию таких «оборванных» комбинаций.

Найдем, какие значения может принимать x. Среди фишек достоинства p для каждого из 4 цветов у нас есть 0, 1 или 2 «оборванных» комбинаций второго типа. Значит, для каждого цвета мы можем закодировать состояние двумя числами — длинами этих комбинаций. При этом будем считать, что для каждого цвета первая комбинация не короче второй. Поскольку для того, чтобы комбинация была допустимой, достаточно трех фишек, то мы не будем различать длины комбинаций больших трех (т.е. будем считать, что комбинации длины больше 3 имеет длину 3). Тогда для каждого цвета существует 10 различных состояний «оборванных рядов»: $(0,0)$, $(1,0)$, $(1,1)$, $(2,0)$, $(2,1)$, $(2,2)$, $(3,0)$, $(3,1)$, $(3,2)$, $(3,3)$. Итого, у нас будет 10^4 возможных значений для параметра x, и $10^4 \cdot 13$ различных состояний в динамическом программировании.

В массиве $d[x, p]$ будем хранить, можем ли мы разбить фишки требуемым образом. Для нахождения значения $d[x, p]$ по уже насчитанным ответам для бóльших p необходимо определить, какие из комбинаций x мы будем продолжать. Для каждого цвета имеется 4 варианта использования фишек достоинства p:

- не продолжать ни одну комбинацию;

- продолжить “длинную” комбинацию;
- продолжить “короткую” комбинацию;
- продолжить обе комбинации.

Для каждого варианта мы получаем новое состояние x', которое будет соответствовать рядам, «оборванным» на уровне $p+1$, — и множество фишек z достоинства p, которые мы не включили в комбинации второго типа, т.е. которые надо разбить на комбинации первого типа. Всего для каждого x и p мы имеем $4^4 = 256$ возможных вариантов перехода $(x, p) \to (x', p+1, z)$. Прежде чем использовать ответ для $d[x', p+1]$, необходимо выполнить следующие проверки:

- если в x' есть комбинации длины 1 или 2, к которым мы не подсоединяем фишку достоинства p, то такой переход недопустим;
- если количество продолжаемых комбинаций больше количества имеющихся фишек достоинства p соответствующего цвета, то такой переход недопустим;
- если множество z нельзя разбить на комбинации первого типа, то такой переход недопустим*.

Тогда, если хотя бы один из переходов $x \to x'$ является допустимым и $d[x', p+1] = 1$, то $d[x, p] = 1$, иначе $d[x, p] = 0$. При этом $d[x, 14] = 1$ тогда и только тогда, когда x содержит только комбинации длины 0. Таким образом, мы можем найти значение $d[0, 1]$. Если мы получили, что $d[0, 1] = 0$, то все фишки разбить требуемым образом на комбинации нельзя, иначе искомое разбиение существует.

* Проверку разбиения множества z на комбинации первого типа можно провести следующим образом. Посчитаем общее количество фишек (n) и максимальное количество фишек одного цвета (m) в множестве z. Тогда легко видеть, что z можно разбить на комбинации длины не менее 3 тогда и только тогда, когда $n \geqslant 3m$.

Пример правильной программы

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>

#define mp make_pair
#define sz(x) ((int)(x).size())

using namespace std;

const int M = 13;

int col[M][4];
int table[M + 1][1 << 16], next[M + 1][1<<16],
  used[M + 1][1 << 16], cur[M + 1][1 << 16];
int good[81];

pair<int, int> res[4][16][4];

int getbits(int x, int p, int c) {
  return (x >> p) & ((1 << c) - 1);
}

pair<int, int> getres(int col, int prev,
                      int next) {
  int l[2];
  for (int i = 0; i < 2; i++) {
    l[i] = (prev >> (i * 2)) & 3;
    int c = (next >> i) & 1;
    if (c == 0 && l[i] < 3 && l[i] != 0)
      return mp(-1, -1);
    if (c == 0) l[i] = 0;
    else {
      col--;
      l[i]++;
    }
    if (l[i] > 3) l[i] = 3;
  }
  if (col < 0) return mp(-1, -1);
  if (l[0] < l[1]) swap(l[0], l[1]);
  return mp(col, l[0] + l[1] * 4);
}

int getgood(int x) {
  int sum = 0, m = 0;
  for (int i = 0; i < 4; i++) {
    int c = x % 3;
    m = max(m, c);
    sum += c;
```

```
    x /= 3;
  }
  return sum >= 3 * m;
}

int getans(int p, int prev) {
  int &ans = table[p][prev];
  if (ans != -1) return ans;
  if (p == M) {
    for (int i = 0; i < 8; i++) {
      int c = getbits(prev, 2 * i, 2);
      if (c == 1 || c == 2) {
        ans = 0;
        return ans;
      }
    }
    ans = 1;
    return ans;
  }
  for (int i = 0; i < 256; i++) {
    int f = 1, used = 0, next = 0;
    for (int j = 3; j >= 0; j--) {
      pair<int, int> x  = res[col[p][j]]
          [getbits(prev, j * 4, 4)]
          [getbits(i, j * 2, 2)];
      if (x.first == -1) {
        f = 0;
        break;
      }
      next = next * 16 + x.second;
      used = used * 3 + x.first;
    }
    if (f) {
      if (getans(p+1, next)!=0 && good[used]) {
        ::next[p][prev] = next;
        ::used[p][prev] = used;
        cur[p][prev] = i;
        ans = 1;
        return ans;
      }
    }
  }
  ans = 0;
  return ans;
}

pair<int, int> parse(string s) {
  int x = s[0] - 'A', y = 0;
  for (int i = 1; i < sz(s); i++)
    y = y * 10 + s[i] - '0';
  return make_pair(x, y - 1);
}

string getstring(pair<int, int> x) {
  string ans = "";
  ans += (char) (x.first + 'A');
  if (x.second >= 9)
    ans += (char) ((x.second + 1) / 10 + 48);
  ans += (char) ((x.second + 1) % 10 + 48);
  return ans;
}
```

```
vector<vector<pair<int, int> > > v;
vector<pair<int, int> > column[4][2];

void getrows(int p, int used) {
  int col[4], sum = 0;
  for (int i = 0; i < 4; i++) {
    col[i] = used % 3;
    used /= 3;
    sum += col[i];
  }
  if (sum == 0) return;
  if (sum < 5) {
    vector<pair<int, int> > ans;
    for (int i = 0; i < 4; i++)
      if (col[i]) ans.push_back(mp(i, p));
    v.push_back(ans);
  }
  else {
    vector<pair<int, int> > v1, v2;
    for (int i = 0; i < 4; i++) {
      if (col[i]) v1.push_back(mp(i, p));
      if (col[i] > 1)
        v2.push_back(mp(i, p));
    }
    while (sz(v2) < 3) {
      for (int i = 0; i < sz(v1); i++)
        if (count(v2.begin(), v2.end(),
                          v1[i]) == 0) {
          v2.push_back(v1[i]);
          swap(v1[i], v1[sz(v1) - 1]);
          v1.pop_back();
          break;
        }
    }
    v.push_back(v1);
    v.push_back(v2);
  }
}

void restore(int p, int prev) {
  if (p == M) {
    for (int i = 0; i < 4; i++)
      for (int j = 0; j < 2; j++)
        if (sz(column[i][j]))
          v.push_back(column[i][j]);
    return;
  }
  int x = cur[p][prev], y = used[p][prev];
  getrows(p, y);
  for (int i = 0; i < 4; i++) {
    int cx = x & 3;
    x >>= 2;
    for (int j = 0; j < 2; j++) {
      if ((cx >> j) & 1)
        column[i][j].push_back(mp(i, p));
      else {
        if (sz(column[i][j]))
          v.push_back(column[i][j]);
        column[i][j].clear();
      }
    }
    if (sz(column[i][0]) < sz(column[i][1]))
```

```
      swap(column[i][0], column[i][1]);
  }
  restore(p + 1, next[p][prev]);
}

int main() {
  freopen("rummikub.in", "r", stdin);
  freopen("rummikub.out", "w", stdout);
  for (int i = 0; i < 81; i++)
    good[i] = getgood(i);
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 16; j++)
      for (int k = 0; k < 4; k++)
        res[i][j][k] = getres(i, j, k);
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    pair<int, int> tmp = parse(s);
    col[tmp.second][tmp.first]++;
  }
  memset(table, -1, sizeof(table));
  int ans = getans(0, 0);
  if (ans == 0) {
    cout << -1 << endl;
    return 0;
  }
  restore(0, 0);
  cout << sz(v) << endl;
  for (int i = 0; i < sz(v); i++) {
    cout << sz(v[i]) << ' ';
    for (int j = 0; j < sz(v[i]); j++)
      cout << getstring(v[i][j]) << ' ';
    cout << endl;
  }
  return 0;
}
```

Содержание