Окружной этап всероссийской олимпиады школьников по информатике
Москва, 1 декабря 2013 г.

Решения заданий для 9–11 классов

Ограничение по времени работы программы во всех задачах: 1 секунда.
Каждая задача оценивается в 100 баллов.

# Задача 1. Управляющий совет

В управляющий совет школы входят родители, учителя и учащиеся школы, причём родителей должно быть не менее одной трети от общего числа членов совета. В настоящий момент в совет входит $N$ человек, из них $K$ родителей. Определите, сколько родителей нужно дополнительно ввести в совет, чтобы их число стало составлять не менее трети от числа членов совета.

Программа получает на вход два целых числа $N$ и $K$ ($N > 0$, $0 \le K \le N$), записанные в отдельных строках, — текущее число членов совета и число родителей в совете.

Программа должна вывести единственное число — минимальное число родителей, которое необходимо ввести в совет.

## Пример

| **Ввод** | **Вывод** |
|---|---|
| 27<br>7 | 3 |

В примере совет состоит из 27 человек, из которых родители составляют 7 человек. Если в совет ввести ещё 3 родителей, то в совете станет 30 человек, из которых родителей будет 10.

## Ограничения и система оценивания

Решение, правильно работающее в случае, когда числа $N$ и $K$ не превосходят 100, будет оцениваться в 60 баллов.

Решение, правильно работающее в случае, когда числа $N$ и $K$ не превосходят $2 \cdot 10^9$, будет оцениваться в 100 баллов.

## Решение

Посчитаем число не-родителей в совете: их N - K. Чтобы родителей в совете было не меньше одной трети, отношение числа родителей к числу не-родителей должно быть не менее 1/2, то есть родителей в совете должно быть не менее половины от числа не-родителей. Необходимо поделить значение N - K на 2 с округлением вверх — именно это и есть минимальное число не-родителей в совете. Если это число меньше, чем K, то родителей уже больше необходимого числа, поэтому ответом будет 0, иначе нужно вывести разность между данным числом и N.

Пример правильного решения на языке Python:

```
N = int(input())
K = int(input())
M = (N - K + 1) // 2
if M < K:
    print(0)
else:
    print(M - K)
```

Решение, в котором в цикле добавляется по одному родителю в совет (то есть значения N и K увеличиваются на 1) до тех пор, пока отношение K/N меньше 1/3, набирают 70 баллов, так как не будут укладываться в ограничения по времени при больших входных данных.

Пример решения на 70 баллов:

```
N = int(input())
K = int(input())
ans = 0
while K / N < 1 / 3:
    N += 1
    K += 1
    ans += 1
print(ans)
```

# Задача 2. Подготовка к олимпиаде

Юра решил подготовиться к региональной олимпиаде по информатике. Он выбрал на сайте informatics.mccme.ru $N$ задач, чтобы решить их на каникулах. В первый день Юра решил $K$ задач, а в каждый следующий день Юра решал на одну задачу больше, чем в предыдущий день. Определите, сколько дней уйдёт у Юры на подготовку к олимпиаде.

Программа получает на вход два целых положительных числа $N$ и $K$, записанных в отдельных строках — количество задач, которые намерен решить Юра, и количество задач, которые он решил в первый день подготовки.

Программа должна вывести единственное число — количество дней, которое потребовалось Юре для решения задач.

## Пример

| **Ввод** | **Вывод** |
|---|---|
| 10<br>3 | 3 |

В примере в первый день Юра решит 3 задачи, во второй день — 4, в третий день — 5, итого на решение 10 задач у Юры уйдёт 3 дня.

## Ограничения и система оценивания

Решение, правильно работающее в случае, когда числа $N$ и $K$ не превосходят 100, будет оцениваться в 80 баллов.

Решение, правильно работающее в случае, когда числа $N$ и $K$ не превосходят $2 \cdot 10^9$, будет оцениваться в 100 баллов.

## Решение

В этой задаче будем в цикле считать, сколько задач решил Юра с начала каникул. Для этого заведем переменную solved (число решенных задач с начала каникул) и переменную day (количество прошедших дней). Далее в цикле пока solved < N значение solved увеличивается на K, а значение K увеличивается на 1 (на следующий день Юра решит на одну задачу больше).

Пример правильного решения на языке Python:

```
N = int(input())
K = int(input())
solved = 0
```

```
day = 0
while solved < N:
    solved += K
    K += 1
    day += 1
print(day)
```

Решения на языках Pascal, C++ и других, использующих 32-битные целые числа могли набирать только 80 баллов, так как при больших входных данных при прибавлении к переменной solved значения K происходит переполнение 32-битного целого числа (максимальное значение, которое можно записать в 32-битной целочисленной знаковой переменной равно $2^{31}$-1=2.147.483.647, то есть входные данные N и K могут быть сохранены в 32-битной переменной, а сумма значений K и K + 1 — уже нет. В правильном решении необходимо использовать беззнаковый 32-битный тип или 64-битный тип данных (в решении на Python это не требуется, т. к. в языке Python встроенная длинная целочисленная арифметика).

Пример решения на языке Pascal с переполнением целочисленной переменной:

```
var n, k, solved, day: integer;
begin
    readln(n);
    readln(k);
    solved := 0;
    day := 0;
    while solved < n do
    begin
        solved := solved + k + day;
        inc(day);
    end;
    writeln(day);
end.
```

# Задача 3. Следующий палиндром

Натуральное число называется палиндромом, если его запись в десятичной системе счисления одинаково читается как слева направо, так и справа налево. По данному натуральному числу *N* определите следующее за ним натуральное число (то есть наименьшее число, которое превосходит *N*), являющееся палиндромом.

Программа получает на вход одно натуральное число *N*.

Программа должна вывести наименьшее натуральное число, которое больше *N* и является палиндромом.

## Пример

| **Ввод** | **Вывод** |
|---|---|
| 4321 | 4334 |

## Ограничения и система оценивания

Решение, правильно работающее в случае, когда число *N* содержит не более 4 цифр, будет оцениваться в 30 баллов.

Решение, правильно работающее в случае, когда число *N* содержит не более 9 цифр, будет оцениваться в 60 баллов.

Решение, правильно работающее в случае, когда число *N* содержит не более 100 цифр,

будет оцениваться в 100 баллов.

## Решение

Поскольку длина входных данных может быть до 100 символов, необходимо для хранения данных использовать строковый тип данных, а не числовой.

Пусть строка S содержит запись данного числа. Прежде всего заметим, что если строка является палиндромом, то вторая половина строки является симметричной первой половине строки. Разделим строку на две части и сделаем правую часть строки симметричной левой части. В приведенном ниже решении результат этого действия записан в переменную S1. В зависимости от реализации, может понадобится отдельно рассмотреть случай четной и нечетной длины строки S. После этого сравним строки S и S1. Если S1 больше S в лексикографическом порядке, то S1 и является ответом.

Иначе необходимо увеличить левую половину строки на 1, после чего сделаем правую половину строки симметричной левой половине, это и будет ответом. В приведенном ниже решении результат этого действия записан в строку S2. В этом решении для увеличения числа, записанного в левой половине строки S1 используется длинная целочисленная арифметика языка Python, но можно это реализовать исключительно при помощи строковых операций: идем с конца, заменяя символы «9» на «0», пока не будет встречен символ, отличный от символа «9», который нужно увеличить на 1.

Возможно понадобиться отдельно рассмотреть случай, когда строка состоит из одни символов «9», в этом случае алгоритм может работать некорректно. В приведенном решении неверно обрабатывается случай, когда число состоит из одной цифры «9», этот случай разобран отдельно (если число состоит более чем из одной цифры «9» решение работает корректно).

Пример правильного решения на языке Python:

```
import sys
S = str(input())
if S == '9':
    print('11')
    sys.exit(0)

n = len(S)
S1 = S[:-(n // 2)] + S[:n // 2][::-1]
if S1 > S:
    print(S1)
    sys.exit(0)

n1  = (n + 1) // 2
n2 = n - n1
N = str(int(S[:n1]) + 1)
S2 = N + N[:n2][::-1]
print(S2)
```

Более простое решение может циклом перебирать все числа, увеличивая текущее число на 1, пока его запись не станет палиндромом. Такое решение не будет укладываться в ограничение по времени на больших тестах и может набирать до 60 баллов.

Пример неэффективного решения на 60 баллов на языке Python:

```
n = int(input())
n = n + 1
while str(n) != str(n)[::-1]:
    n = n + 1
print(n)
```

# Задача 4. Рассадка участников

На олимпиаду по информатике пришло *N* участников. Известно, в каких школах учатся участники олимпиады. В компьютерном классе имеется *N* компьютеров, стоящих в линию вдоль стены. Вам необходимо рассадить участников олимпиады так, чтобы никакие два участника из одной школы не сидели рядом.

Программа получает на вход целое положительное число участников олимпиады *N*. Далее в *N* строках записаны номера школ, в которых учатся участники олимпиады. Номера школ — целые числа от 1 до 3000.

Программа должна вывести *N* чисел — номера школ участников олимпиады в том порядке, в котором их необходимо рассадить в компьютерном классе. Выведенная последовательность номеров школ должна быть перестановкой данных номеров школ. В выведенном ответе не должно быть двух одинаковых номеров школ, идущих подряд.

Если задача не имеет решения, необходимо вывести одно число 0.

Числа можно выводить как в отдельных строках, так и в одной строке через пробел. Если есть несколько вариантов рассадки, то необходимо вывести любой из них (но только один).

## Примеры

| **Ввод** | **Вывод** |
|---|---|
| 4<br>1005<br>1005<br>5<br>2005 | 2005<br>1005<br>5<br>1005 |
| 4<br>1005<br>1005<br>2005<br>1005 | 0 |

## Ограничения и система оценивания

Решение, правильно работающее в случае, когда число *N* не превосходит 10, будет оцениваться в 30 баллов.

Решение, правильно работающее в случае, когда число *N* не превосходит 100, будет оцениваться в 100 баллов.

## Решение

Есть несколько способов решить эту задачу.

Можно подсчитать для каждого номера школы число участников олимпиады из этой школе. Затем начнем подряд заполнять элементы массива, в котором будет храниться результат решения задачи. Необходимо сначала размещать участников из школ, у которых число участников максимально. Поэтому на первое место посадим учащегося из такой школы (с наибольшим числом участников олимпиады). На второе место посадим учащегося из школы с наибольшим числом участников, но кроме школы, участник из которой уже находится на первом месте. На третье место посадим участника из школы, с наибольшим числом участников олимпиады, но за исключением школы, чей участник сидит на втором месте и т. д. При этом размещая участника из какой-то школы необходимо уменьшать на 1 число участников от этой школы.

Например, пусть из школы №1 было 4 участника, из школы №2 — 3 участника, из

школы №3 — 2 участника, из школы №1 — 3 участника. Тогда этот алгоритм разместит участников следующим образом:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |

Другое возможное решение сначала сортирует номера школ всех участников, в результате получится следующий массив (для уже рассмотренного примера):

1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4.

Потом начнем размещать участников в указанном порядке (то есть последовательно по номерам школ) с начала массива с шагом 2, до тех пор пока не дойдем до конца массива. Получится следующее заполнение:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 1 | | 1 | | 1 | | 2 | | 2 | |

Когда дойдем до конца массива, начнем далее заполнять свободные места с начала массива (тоже с шагом 2), тем самым участники, которые в отсортированном массивы были соседними, не будут сидеть рядом. Получим следующий результат:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 3 | 1 | 3 | 1 | 4 | 2 | 4 | 2 | 4 |

Это решение работает в большинстве случаев кроме того случая, когда половина участников олимпиады из одной школы. Например, пусть из школы №1 будет 2 участника, из школы номер 2 — 6 участников, из школы номер 3 — 4 участника. Тогда такое заполнение приведет к следующему результату:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |

То есть в результате того, что участники из школы №2 занимают конец массива, а затем начало массива, то первый посаженный участник из школы №2 окажется соседним с последним посаженным. Чтобы избежать этой проблемы достаточно определить школу с наибольшим числом участников и начать рассаживание с участников из этой школы, а затем продолжить в порядке сортировки:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |

После окончания рассадки проверим, что в полученном массиве нет двух равных соседних элементов, то есть что полученная рассадка удовлетворяет условию задачи. В этом случае выведем полученную рассадку, иначе выведем число 0.

Пример правильного решения на языке Python:

```
n = int(input())
A = [str(input()) for i in range(n)]
maxcount = 0
maxval = 0
for i in range(n):
    if A.count(A[i]) > maxcount:
        maxcount = A.count(A[i])
        maxval = A[i]

A = [maxval] * maxcount + sorted(elem for elem in A
                                 if elem != maxval)
```

```
B = [0] * n

j = 0
for i in range(0, n, 2):
    B[i] = A[j]
    j += 1

for i in range(1, n, 2):
    B[i] = A[j]
    j += 1

for j in range(1, n):
    if B[j] == B[j - 1]:
        print(0)
        break
else:
    print(" ".join(B))
```

## Задача 5. Остров

На клетчатой бумаге нарисована карта острова (клетки острова закрашены). При этом остров является *клетчато-выпуклой* фигурой, то есть каждая горизонтальная или вертикальная линия на карте либо не пересекает остров, либо пересекает его по отрезку (линия пересечения не содержит разрывов). Также остров является связной фигурой, то есть любые две клетки острова соединены путём, каждые две соседние клетки которого имеют общую сторону.

Клетка считается соседней с островом, если она не принадлежит острову, но имеет общую сторону или угол с одной из клеток острова. Например, на следующей карте клетки острова закрашены, а соседние с островом клетки отмечены звёздочками.

Самолёт должен облететь вокруг острова по соседним с ним клеткам, не вторгаясь на территорию острова. Программа должна составить маршрут полёта самолёта. Самолёт начинает облёт острова в одной из соседних клеток с островом и должен побывать во всех клетках, соседних с островом, ровно один раз. При этом самолёт может перемещаться из одной клетки в другую клетку, только если эти клетки имеют общую сторону.

Программа получает на вход два числа $N$ и $M$, записанные в отдельных строках, — количество строк и столбцов карты острова ($3 \le N \le 100$, $3 \le M \le 100$). Далее записана карта острова — $N$ строк, каждая содержащая $M$ символов. Каждый символ карты может быть либо символом «`.`», что означает клетку, не принадлежащую острову, либо символом «`#`», что означает клетку острова. При этом остров не касается края карты.

Введём на карте систему координат. Первая координата является номером строки, строки нумеруются сверху вниз числами от 1 до $N$. Вторая координата — номер столбца, столбцы нумеруются слева направо числами от 1 до $M$.

Программа должна вывести координаты клеток карты в порядке их облёта самолётом. Каждая строка вывода должна содержать два числа $x$ и $y$ — координаты самолёта,

записанные через пробел ($1 \le x \le N$, $1 \le y \le M$). Самолёт должен побывать в каждой соседней с островом клетке ровно один раз. Каждые две клетки, идущие подряд в выводе, должны иметь общую сторону. Можно вывести любой возможный маршрут облёта острова.

## Пример

| Ввод | Вывод |
|---|---|
| 6<br>7<br>.......<br>.......<br>.......<br>.###...<br>.###...<br>.......<br> | 3 5<br>4 5<br>5 5<br>6 5<br>6 4<br>6 3<br>6 2<br>6 1<br>5 1<br>4 1<br>3 1<br>3 2<br>3 3<br>3 4 |

В данном примере карта острова является прямоугольником, и соседние с островом клетки — это «рамка» с углами в клетках (3, 1), (3, 5), (6, 5), (6, 1). Самолёт начинает облёт из правого верхнего угла «рамки» (3, 5), потом движется вниз, влево, вверх и вправо.

## Ограничения и система оценивания

Решение, правильно работающее в случае, когда остров является прямоугольником, будет оцениваться в 30 баллов.

Решение, правильно работающее в случае, когда остров является произвольной связной клетчато-выпуклой областью, будет оцениваться в 100 баллов.

## Решение

Будем хранить карту в виде двумерного массива, каждый элемент которого является либо символов «.», либо символом «#». Прежде всего найдем «границу» острова, то есть те клетки, в которых должен побывать самолет. Это клетки, которые не принадлежат острову, то есть помеченные символом «.» на карте, одна из соседних клеток с которыми по углу или стороне помечена «#». Пометим эти клетки символом «*». Для приведенного

Данная «граница» будет некоторой замкнутой линией, соседние две клетки границы имеют общую сторону. Например, для теста из условия граница будет такой:

```
.......
.......
*****..
*###*..
*###*..
*****..
```

Самолет должен облететь все клетки, отмеченные «*». Найдем какую-нибудь клетку, отмеченную «*», выведем ее координаты. После этого программа будет перемещать самолет в соседнюю по стороне клетку, также помеченную «*» и выводить координаты этой клетки. Чтобы алгоритм не зациклился, будем «стирать» каждую клетку после посещения ее самолетом, то есть будем символ карты, соответствующий позиции самолета, менять на «.». Тогда самолет обойдет все клетки границы и вернется в клетку, соседнюю с той, откуда был начат облет.

Пример правильного решения на языке Python:

```
import sys
N = int(sys.stdin.readline())
M = int(sys.stdin.readline())
Map = [["."] * (M + 2)]
for i in range(N):
  Map.append(list("." + sys.stdin.readline().rstrip() + '.'))
Map.append(['.'] * (M + 2))

for y in range(1, N + 1):
  for x in range(1, M + 1):
    if Map[y][x] == '.' and (
      Map[y - 1][x] == '#' or Map[y + 1][x] == '#' or
      Map[y][x - 1] == '#' or Map[y][x + 1] == '#' or
      Map[y - 1][x - 1] == '#' or Map[y - 1][x + 1] == '#' or
      Map[y + 1][x - 1] == '#' or Map[y + 1][x + 1] == '#'):
      Map[y][x] = '*'
        cx = x
        cy = y
while True:
  sys.stdout.write(str(cy) + ' ' + str(cx) + '\n')
  Map[cy][cx] = '.'
  for dx, dy in ((-1, 0), (1, 0), (0, -1), (0, 1)):
    if Map[cy + dy][cx + dx] == '*':
      cx += dx
      cy += dy
      break
  else:
    break
```

В ограничениях на 30 баллов граница является «рамкой» прямоугольника, поэтому можно найти левый верхний угол границы, затем вывести подряд координаты клеток на верхней стороне, правой стороне, нижней стороне и левой стороне прямоугольной рамки.