Всероссийская олимпиада школьников по информатике
Вологодская область, 2022-23 учебный год
II (муниципальный) этап
9 - 11 классы
Методические рекомендации по разбору задач

Задача 1. Поездка на олимпиаду (100 баллов)

Решение выглядит так. В первую очередь билеты на дешёвые места должны брать студенты. Если после этого дешёвые места ещё остались, то дальше билеты на них берут школьники. Если после этого дешёвые места кончились, а участники ещё остались, то они покупают билеты на дорогие места.

Пояснение. Пусть есть одно место стоимости P и одно стоимости Q, где $Q > P$. Если первое место займёт студент, а второе – школьник, то выйдет сумма P + Q/2. А если наоборот – то $P/2 + Q$. Вычтем из второй суммы первую – получим $(Q - P)/2$. Поскольку $Q > P$, то эта разность положительна – то есть, второй вариант хуже. Таким образом, если не удаётся совсем обойтись без дорогих мест, то на них лучше сажать школьников.

Пример решения на языке Python:

```
m = int(input())
n = int(input())
a = int(input())
p = int(input())
b = int(input())
q = int(input())
if p > q:
   p, q = q, p
   a, b = b, a
ans = 0.0

stud_chip_tickets = min(m, a)
ans +=  stud_chip_tickets * p
m -= stud_chip_tickets
a -= stud_chip_tickets

school_chip_tickets = min(n, a)
ans += school_chip_tickets * p * 0.5
a -= school_chip_tickets
n -= school_chip_tickets

ans += m * q + n * q * 0.5
print(ans)
```

Задача 2. Изумительные числа (100 баллов)

Первую подзадачу можно решить «в лоб»: пробегаемся по каждому входному интервалу, и проверяем каждое число на «изумительность» согласно определению.

Полное решение состоит в том, чтобы заранее получить все изумительные числа от 1 до 10^6. Их немного – всего 127 штук. Теперь при обработке очередного интервала достаточно пробежаться по списку изумительных чисел и сосчитать, сколько из них попадают в этот интервал. Пример решения на Python:

```
def amazing(x):
    x0 = x
    s = 0
    while x > 0:
        s += (x % 10) * (x % 10)
        x //= 10
    return x0 % (s * s) == 0

a = []
for i in range(1, 1000000 + 1):
    if amazing(i):
        a.append(i)

for i in range(int(input())):
    m = int(input())
    n = int(input())
    ans = 0
    for x in a:
        if m <= x <= n:
            ans += 1
    print(ans)
```

Для получения полного балла это решение нужно сдавать на PyPy.

При желании решение можно ускорить, если заранее на своём компьютере вычислить список *a* и вставить его в текст программы. Также можно вместо линейного поиска использовать бинарный.

Задача 3. Произведения (100 баллов)

В первой и второй подзадачах все числа неотрицательны. Заметим, что тогда самое маленькое число следует умножать на самое большое. Действительно, если мы умножим минимум не на максимум, то это значит, что максимум умножится на какое-то другое число. Но тогда их произведение будет больше нашего. Аналогично, второй минимум надо умножать на второй максимум, и так далее. Отсюда

получается такое решение. Отсортируем массив. Искомыми парами будут a[1]*[n], a[2]*a[n-1], и так далее. Если все эти произведения равны, то ответ найден, иначе решения нет.

Третью подзадачу можно решить перебором всех перестановок.

Полное решение выглядит как обобщение решения подзадач 1-2. Заметим, что если совпадают произведения, то совпадают и их модули. Тогда, как показано выше, число с минимальным модулем должно умножаться на число с максимальным модулем, и так далее. Однако, произведения могут получиться как со знаком плюс, так и минус. Следует проверить оба варианта.

Сначала предположим, что произведения в ответе отрицательные. Отсортируем массив по модулям чисел, а в случае одинаковых модулей – по значениям. Например, из массива 1 -2 3 -6 2 -3 получится 1 -2 2 -3 3 -6. Если теперь умножать первый элемент на последний, второй – на предпоследний, то как раз будут получаться нужные отрицательные произведения.

Во втором варианте предположим, что произведения в ответе неотрицательные. Сначала отсортируем массив как в предыдущем варианте – например, из массива 2 -2 3 -3 1 6 получится 1 -2 2 -3 3 6. Теперь возьмём правую половину массива и отсортируем её тоже по возрастанию модулей, но в случае одинаковых модулей – по убыванию значений. Получится: 1 -2 2 3 -3 6. В результате числа с одинаковыми модулями в левой половине массива будут идти вначале отрицательные, затем – положительные, а в правой половине – наоборот. Снова пробуем умножать первый элемент на последний, второй – на предпоследний, и так далее. Будут получаться неотрицательные произведения.

Если ни в котором из двух вариантов не совпали произведения всех пар, то ответом будет “Impossible”. Пример решения на Python:

```
n = int(input())
a = [0] * n
for i in range(n):
    a[i] = int(input())

def check():
    res = []
    prod = a[0] * a[-1]
    for i in range(n // 2):
        if a[i] * a[-1 - i] != prod:
            return False
        res.append(a[i])
        res.append(a[-1 - i])
```

```
    print(*res)
    return True

a.sort(key = lambda x : (abs(x), x))
if not check():
    a = a[:n // 2] + sorted(a[n // 2 :], key = lambda x :
(abs(x), -x))
    if not check():
        print("Impossible")
```

Задача 4. Числа с различными цифрами (100 баллов)

Первую задачу можно решить «в лоб»: перебираем все *N*-значные числа, в каждом числе проверяем, нет ли одинаковых цифр, и считаем сумму цифр.

Полное решение можно получить по-разному. Рассмотрим два способа.

В первом способе заметим, что если нам подошло какое-то число без цифры ноль, то любая перестановка его цифр даст число, которое тоже подойдёт. Таким образом, вместо перебора чисел мы можем перебирать всевозможные подмножества из *n* цифр, и для каждого подходящего подмножества прибавлять к ответу количество перестановок. Количество перестановок из *n* элементов равно *n*! (факториал от *n*).

Несложно учесть и случай, когда подмножество содержит цифру 0. Всего перестановок будет *n*!, а недопустимых (начинающихся с нуля) – (*n*–1)! Итого: *n*! – (*n*–1)!

Перебирать подмножества цифр от 0 до 9 можно по-разному. Один из способов – перебирать целые числа от 0 до 2^{10}-1 и смотреть на их младшие 10 битов. Если бит с номером *i* равен 1, то цифра *i* входит в множество, а иначе не входит.

Пример такого решения на языке Python:

```
from math import factorial
n = int(input())
s = int(input())
ans = 0

fact_n = factorial(n)
fact_n1 = factorial(n - 1)

for x in range(1, 1 << 10):
  cnt = 0
  total = 0
  for i in range(0, 10):
```

```
      if ((x >> i) & 1) > 0:
         total += i
         cnt += 1
  if cnt == n and total <= s:
    ans += fact_n
    if (x & 1) == 1 and n > 1:
       ans -= fact_n1

print(ans)
```

Ещё один возможный способ решения – сделать преподсчёт. У нас есть всего 10 возможных вариантов числа *N*, и для каждого n – не более 1 + (19–*N*)**N*/2 вариантов *S* (от 0 до 9+8+7+…+10–*N*). Это менее 500 вариантов. Можно заранее вычислить их все на своём компьютере и вставить ответы прямо в текст программы. Приведём пример программы на языке C++ для получения всех ответов:

```
#include <iostream>
#include <vector>
using namespace std;

long long intpow(long long p, int q) {
   long long ans = 1;
   for (int i = 1; i <= q; i++) {
      ans *= p;
   }
   return ans;
}

int main() {
   cout << "a=(),\n";
   for (int n = 1; n <= 10; n++){
      vector<long long> cnt(1 + (19 - n) * n / 2);
      long long start = intpow(10, n - 1);
      if (start == 1)
         start = 0;
      long long end = intpow(10, n) - 1;
      for (long long i = start; i <= end; i++){
         vector<bool> used(10);
         int c = 0;
         long long t = i;
         bool ok = true;
         while (t > 0){
           c += t % 10;
           if (used[t % 10])
              ok = false;
           used[t % 10] = true;
           t /= 10;
```

```
            }
            if (ok)
               cnt[c] += 1;
         }
         cout << '(';
         cout << cnt[0];
         for (int i = 1; i < cnt.size(); i++) {
            cnt[i] += cnt[i - 1];
            cout << ", " << cnt[i];
         }
         cout << ")," << "\n";
      }
      cout << ")\n";
}
```

Вывод этой программы нужно направить в файл:

```
precalc.exe >precalculated.py
```

Поскольку эта программа ответы вычисляет «в лоб», то работает она медленно – несколько минут. Когда она закончит, останется лишь дописать ввод и вывод, и получаем готовое решение на языке Python:

```
a=((),
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
(0, 1, 2, 5, 8, 13, 18, 25, 32, 41, 49, 57, 63, 69, 73, 77,
79, 81),
(0, 0, 0, 4, 8, 16, 30, 48, 72, 106, 146, 192, 246, 300, 356,
412, 464, 510, 552, 582, 606, 624, 636, 642, 648),
...часть данных опущена
)

n = int(input())
s = min(int(input()), (19 - n) * n // 2)
print(a[n][s])
```

Задача 5. Маршрут ладьи (100 баллов)

Заметим, что ладья обязательно проходит хотя бы раз либо по каждой вертикали (возможно, не целиком), либо по каждой горизонтали. Действительно, если ладья не ходила по какой-то вертикали, то это значит, что все клетки на ней она проходила горизонтально — то есть, она ходила по всем горизонталям. Аналогично, если ладья не ходила по какой-то горизонтали, значит, она ходила по всем вертикалям.

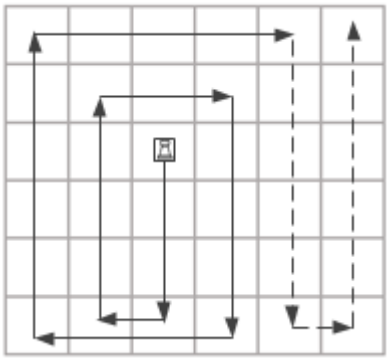
Пусть ладья ходила по всем вертикалям (для горизонталей рассуждения аналогичны). Тогда после прохода по каждой вертикали,

кроме последней, она совершала минимум один горизонтальный ход для перехода на другую вертикаль. Итого получается, что количество ходов не может быть меньше $2n-1$.

Для решения первой подзадачи (ладья стоит в углу поля), достаточно ходить "змейкой" или спиралью. В этом случае как раз получится $2n-1$ ходов.

Для решения второй подзадачи (ладья стоит на границе поля) можно ходить "змейкой". Если ладья стоит на левой границе поля, то ходим влево-вправо, и после каждого горизонтального хода перемещаемся на любую горизонталь, по которой ещё не ходили. Если же ладья стоит на верхней границе поля, то аналогично ходим вверх-вниз.

Если ладья стоит не на границе поля, то $2n-1$ ходов можно получить так. Определим, в которую из четырёх сторон ближе всего идти до границы поля, и первый ход сделаем в противоположном направлении до границы поля. Для примера пусть ближе всего верхняя граница, тогда сначала мы шагнём вниз. Далее будем двигаться по фигуре, похожей на спираль: смещаемся на шаг влево и идём вверх до $y-1$, смещаемся на два шага вправо и идём вниз до границы поля, смещаемся на три шага влево и идём вверх до $y-2$, и так далее, пока не окажемся на верхней границе поля. Затем шагнём вправо до столбца, по которому ещё не ходили, и дальше продолжим идти "змейкой". На рисунке первая часть маршрута ('спираль') показана сплошными линиями, а вторая часть маршрута ('змейка') — пунктирными. Для любой клетки не на границе поля такой маршрут будет оптимальным.



Добавив несколько дополнительных условий в данный алгоритм, можно сделать, чтобы он работал и для клеток на границе поля, в том числе угловых.

Программную реализацию можно упростить, выполнив преобразование координат так, чтобы кратчайшее расстояние от ладьи

до края поля было всегда в конкретную сторону – например, вверх, а расстояние влево было меньше или равно расстоянию справа. При выводе ответа тогда нужно выполнять обратное преобразование координат. Пример решения с таким подходом на языке Python:

```
def printxy(x, y):
    global trans, revX, revY
    if revX:
        x = n + 1 - x
    if revY:
        y = n + 1 - y
    if trans:
        x, y = y, x
    print(x, y)

n = int(input())
x = int(input())
y = int(input())

trans = False
if min(x - 1, n - x) < min(y - 1, n - y):
    x, y = y, x
    trans = True

revX = n - x < x - 1
if revX:
    x = n + 1 - x
revY = n - y < y - 1
if revY:
    y = n + 1 - y

rest_columns = {i for i in range(1, n + 1)}

xc = x
delta = 1
while y > 1:
  yc = n
  printxy(xc, yc)
  rest_columns.remove(xc)
  xc -= delta
  printxy(xc, yc)
  y -= 1
  yc = y
  printxy(xc, yc)
  rest_columns.remove(xc)
  delta += 1
  xc += delta
  printxy(xc, yc)
  delta += 1
assert(xc in rest_columns)
yc = n
```

```
rest_columns.remove(xc)
printxy(xc, yc)
while len(rest_columns) > 0:
  xc = rest_columns.pop()
  printxy(xc, yc)
  if yc == 1:
     yc = n
  else:
     yc = 1
  printxy(xc, yc)
```