Всероссийская олимпиада школьников по информатике
Вологодская область
II (муниципальный) этап
2020-2021 учебный год
9 - 11 классы
Методические рекомендации по разбору задач

Задача 1. Дискриминант (100 баллов)

Первую подзадачу можно решить перебором всех троек a, b, c в некотором небольшом диапазоне. При $|D| \le 100$ всегда либо есть ответ, где либо все коэффициенты по модулю не превышают 25, либо нет решения.

Вторую подзадачу можно решить так. $D = b^2 - 4ac$. Сразу заметим, что можно сделать a=1. Если решение есть, то оно будет и в таком случае. Выразим b^2: $b^2 = D + 4c$.

Нам нужно подобрать такое c, чтобы $D+4c$ являлось полным квадратом. Можно просто перебрать все возможные значения коэффициента c в некотором диапазоне, проверяя, не является ли $D+4c$ полным квадратом. При ограничениях второй подзадачи $|D| \le 10^6$ достаточно поставить границы перебора от -250000 до 250000 (проверено авторами с помощью перебора). Конечно, участники заранее не знают границ – поэтому можно просто взять их с запасом, чтобы решение укладывалось в одну секунду. Пример такого решения на C++:

```
#include <bits/stdc++.h>

int main() {
  int d;
  std::cin >> d;
  for (int c = -10000000; c <= 10000000; c++) {
    int s = d + 4 * c;
    if (s < 0) continue;
    int b = std::sqrt((double) s);
    if (b * b > s) b--;
    else if ((b + 1) * (b + 1) <= s) b++;
    if (b * b == s) {
      std::cout << "1 " << b << " " << c;
      return 0;
    }
  }
  std::cout << -1;
}
```

Заметим, что функция *sqrt* вычисляется с некоторой погрешностью. Поэтому после обрезания дробной части ответ может на единицу отличаться от верного – этот момент учтён в решении. Правда, при наших не столь больших границах перебора эта ошибка не возникает, поэтому эту проверку можно было бы и не делать.

Полное решение задачи выглядит так. Заметим, что при любом c выражение $D+4c$ имеет одинаковый остаток от деления на 4, равный $D \bmod 4$. Рассмотрим все его варианты.

1). $D \bmod 4 = 0$. Возьмём $c = -D/4$. Тогда $D + 4c = 0$, то есть $b=0$. Ответ: $a=1$, $b=0$, $c=-D/4$.

2). $D \bmod 4 = 1$. Возьмём такое c, чтобы $b^2=9$. $D + 4c=9$, то есть, $c = (9-D)/4$. Ответ: $a=1$, $b = 3$, $c = (9-D)/4$.

3). $D \bmod 4 = 2$ или $D \bmod 4 = 3$. Известно, что квадрат любого натурального числа либо делится на 4, либо при делении на 8 даёт остаток 1. В данном случае не выполняется ни то, ни другое. Ответ: решений нет (нужно вывести -1).

Пример решения на языке Python:

```python
d = int(input())
if d % 4 == 0:
  print("1 0", -d // 4)
elif d % 4 == 1:
  print("1 3", (9 - d) // 4)
else:
  print(-1)
```

При решении на большинстве других языков надо учитывать, что в них остаток для отрицательных чисел считается не так, как принято в математике. Например, на языке C++ выражение -7 % 4 даст не 1, а -3. Пример решения на языке C++:

```cpp
#include <iostream>

int main() {
  int d;
  std::cin >> d;
  if (d % 4 == 0)
   std::cout << "1 0 " << -d / 4;
  else if (d % 4 == 1 || d % 4 == -3)
    std::cout << "1 3 " << (9 - d) / 4;
  else
   std::cout << -1;
}
```

Задача 2. Проверка паролей (100 баллов)

Пусть p – правильный пароль, t – введённый. Сравним их первые два символа. Если они не совпали, то пароль введён неверно. Иначе посчитаем количество одинаковых символов, с которых начинается строка p и строка t. Пусть эти количества равны c_1 и c_2. Если пароль введён правильно, то должно выполняться условие $c_1 \leq c_2 \leq 3c_1$. Если да, то повторим те же действия для оставшихся суффиксов строк p и t. Если строка t закончилась досрочно, либо p уже закончилась, а t – ещё нет, то пароль неправильный.

Пример решения на C++:

```
#include <bits/stdc++.h>

using namespace std;

bool isLongPressedPassword(string pass, string typed) {
  int n = pass.length();
  int i = 0, j = 0;
  while (pass[i] != '#') {
    if (pass[i] != typed[j]) return false;
    int cnt_pass = 1;
    for(i++; pass[i] == pass[i - 1]; i++, cnt_pass++);
    int cnt_typed = 1;
    for(j++; typed[j] == typed[j - 1]; j++, cnt_typed++);
    if (cnt_typed < cnt_pass || cnt_typed > cnt_pass * 3)
return false;
  }
  return pass[i] == typed[j];
}

int main() {
  string pass;
  cin >> pass;
  pass.push_back('#');
  int n;
  cin >> n;
  for (; n; --n) {
    string typed;
    cin >> typed;
    typed.push_back('#');
    if (isLongPressedPassword(pass, typed)) {
      cout << "Yes\n";
    } else {
      cout << "No\n";
    }
  }
}
```

В этом примере для удобства к концу каждой строки дописывается терминальный символ ‘#’, но можно было вместо этого сравнивать текущие позиции с длинами строк.

Первая подзадача в данной задаче рассчитана на решения, работающие в целом правильно, но недостаточно быстро – например, когда из строк каждый раз удаляются совпавшие части (что даст квадратичное время работы), и тому подобное.

Задача 3. Сумма цифр (100 баллов)

Чтобы решить первую подзадачу, можно просто перебрать все целые числа в интервале от A до B, сосчитать для каждого сумму цифр и выбрать число с максимальной суммой.

Для решения второй подзадачи удобнее работать с числами как со строками. Рассмотрим два случая.

1). Второе число длиннее первого. Если оно имеет вид d999…9, где d – какая-то цифра, то оно и будет ответом. В противном случае первую цифру уменьшаем на 1, а все последующие заменяем на девятки. Например, 4835 заменяем на 3999 – это и будет ответ.

Ещё по условию задачи в ответе не должно быть ведущих нулей – поэтому, если первая цифра была 1, то ведущий ноль не пишем.

2). Числа имеют одинаковую длину. Тогда находим первую слева цифру, где они различаются. Во втором числе эта цифра больше, чем в первом. Если дальше во втором числе идут все девятки, то это и будет ответ. Например, для A=1234, B=1299 ответом будет 1299. В противном случае эту цифру уменьшаем на 1, а все последующие заменяем на девятки. Например, для A=1234, B=1257 ответом будет 1249.

Пример решения на языке Python:

```
a = input()
b = input()
if a == b:
  print(b)
elif len(a) < len(b):
  if b[1:] == '9' * (len(b) - 1):
    print(b)
  else:
    d = int(b[0]) - 1
    if d > 0: print(d, end='')
    print('9' * (len(b) - 1))
else:
```

```
  for i in range(len(a)):
    if a[i] != b[i]:
      if b[i+1:] == '9' * (len(b) - i - 1):
        print(b)
      else:
        d = int(b[i]) - 1
        print(b[0:i], d, '9' * (len(b) - i - 1), sep='')
      break
```

При желании можно решить данную задачу, и работая с данными как с числами, а не строками. Пример на C++:

```
#include <cstdio>
#include <cassert>

int main() {
    long long a, b;
    scanf("%lld %lld", &a, &b);
    b++;
    assert(a < b);
    long long mod = 1;
    while (b / mod > 9) {
        mod *= 10;
    }
    while (a / mod == b / mod) {
        mod /= 10;
        assert(mod >= 1);
    }
    printf("%lld", (b / mod) * mod - 1);
    return 0;
}
```

Задача 4. Неделящиеся числа (100 баллов)

Решение 1.Чтобы никакое число не делилось ни на какое другое, будем использовать только простые числа.

Обязательно включим в ответ число 2. Поскольку сумма любых двух других чисел в ответе будет чётной, она будет делиться на 2. Осталось только добиться, чтобы сумма любого числа и двойки делилась хотя бы на одно из оставшихся чисел.

Это можно сделать следующим образом. Взяв очередное простое число, добавим к нему двойку и попробуем поделить эту сумму на несколько маленьких простых чисел – скажем, 3, 5 и 7. Если

не поделилось ни на одно из них, то просто пропустим это число – не будем его добавлять в ответ.

Чтобы эффективно получать простые числа, можно либо использовать стандартный приём проверки числа на простоту с помощью цикла до квадратного корня, либо реализовать решето Эратосфена. При решении на Питоне первый способ проходит по времени, только если при отправке решения выбирать компилятор Pypy. Второй способ проходит и на интерпретаторе Python.

Пример решения с решетом Эратосфена на Python:

```
n = int(input())
if n == 3:
    print("-1")
    quit()

# sieve of Eratosthenes
maxp = 2000000
p = [True] * (maxp + 1)
i = 2
while i * i <= maxp:
    if p[i]:
        j = i * i
        while j <= maxp:
            p[j] = False
            j += i
    i += 1

a = [2, 3, 5, 7]
cur = 11
for i in range(5, n + 1):
    while not p[cur] or (cur+2) % 3 != 0 and (cur+2) % 5 !=
0 and (cur+2) % 7 != 0:
        cur += 1
    a.append(cur)
    cur += 1
print(*a, sep='\n')
```

Решение 2. В качестве первых четырёх чисел возьмём 2, 3, 5, 7. А дальше будем брать 13, 19 и так далее – с шагом 6. Никакое из них не делится на 2, 3, 5 или 7, сумма любых двух из них делится на 2, сумма любого из них и 2 делится на 3, сумма любого из них и 3, 5 или 7 делится на 2.

Проблема только в том, что некоторые из таких чисел могут делиться на одно из предыдущих – но это можно проверять в цикле, и такие числа не брать. При ограничении N до 10000 это с запасом проходит по времени. Пример решения на C++:

```cpp
#include <cstdio>
#include <vector>

int main() {
    int n;
    scanf("%d", &n);
    if (n == 3) {
        printf("-1\n");
        return 0;
    }
    std::vector<int> a(4);
    a[0] = 2;
    a[1] = 3;
    a[2] = 5;
    a[3] = 7;
    for (int i = 5; (int)a.size() < n; i++) {
        int nv = 7 + (i - 4) * 6;
        bool good = true;
        for (int v : a) {
            if (nv % v == 0) {
                good = false;
                break;
            }
        }
        if (good) {
            a.push_back(nv);
        }
    }
    for (int v : a) {
        printf("%d\n", v);
    }
    return 0;
}
```

Рассмотрим ещё возможные решения на частичные баллы.

Для решения первой подзадачи можно было попробовать найти ответы вручную (хотя бы для части тестов) или реализовать какой-либо вариант перебора для поиска ответов. Сам перебор может выполняться на машине участника, а в систему отправляться только найденные ответы.

Вторая подзадача рассчитана на решения, работающие в целом правильно, но недостаточно быстро – например, проверка простоты числа циклом до самого числа, а не до корня, и др.

Задача 5. Обход прямоугольника (100 баллов)

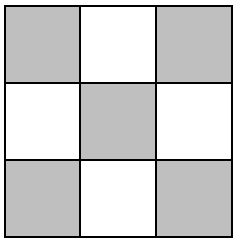
Для решения первой подзадачи можно сделать заполнение, например, «змейкой» (или спиралью).

Для решения второй подзадачи можно написать перебор с возвратом. При ограничении до 6x6 он легко укладывается в одну секунду. Более того, первую подзадачу такое решение тоже пройдёт, поскольку в переборе закрутится спираль, и практически сходу найдётся ответ. Пример такого решения на языке C++:

```
#include <bits/stdc++.h>

using namespace std;

int n, m, y, x;
int a[101][101];

void bt(int i, int j, int cur) {
  a[i][j] = cur;
  if (cur == n * m) {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
          std::cout << a[i][j] << " ";
        }
        std::cout << "\n";
    }
    exit(0);
  }
  if (i > 1 && a[i - 1][j] == 0) bt(i - 1, j, cur + 1);
  if (i < n && a[i + 1][j] == 0) bt(i + 1, j, cur + 1);
  if (j > 1 && a[i][j - 1] == 0) bt(i, j - 1, cur + 1);
  if (j < m && a[i][j + 1] == 0) bt(i, j + 1, cur + 1);
  a[i][j] = 0;
}

int main() {
  cin >> n >> m >> y >> x;
  bt(y, x, 1);
  std::cout << -1;
}
```

Для решения третьей подзадачи вначале разберёмся, когда решений нет. Это возможно в двух случаях. Во-первых, когда полоска имеет ширину или высоту 1, и начальная клетка – не крайняя. Во-вторых, если высота и ширина прямоугольника нечётны, и сумма координат начальной клетки тоже нечётна. Доказать это можно так.

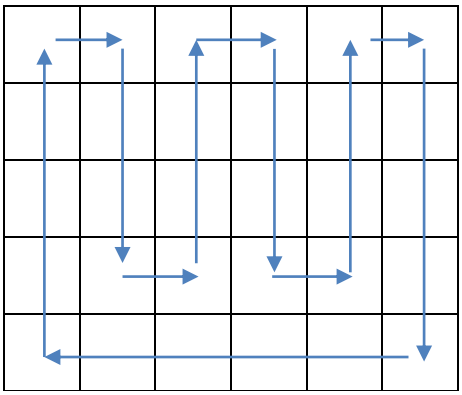
Выполним шахматную раскраску прямоугольника, чтобы в углах получились черные клетки:



Допустим, что искомый путь начинается в белой клетке. Тогда клетки в нём чередуются так: БЧБЧБЧБ… На каждом шаге чёрных клеток пройдено либо на одну меньше, чем белых, либо столько же. Но в прямоугольнике чёрных клеток на одну больше, чем белых. Получается, что, начав в белой клетке, всё поле обойти невозможно.

Во всех остальных случаях решение существует. Рассмотрим, как его можно построить, для этого выделим несколько случаев.

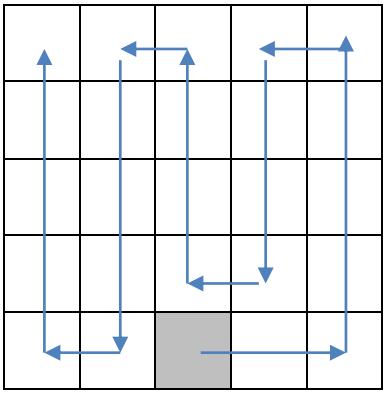
1). Хотя бы одна сторона прямоугольника чётна. Тогда можно построить не просто путь, а замкнутый путь. Например, если чётна ширина, то путь может выглядеть так (для чётной длины – аналогично):



Мы можем начать с любой клетке и, идя по этому пути, заполнить всё поле.

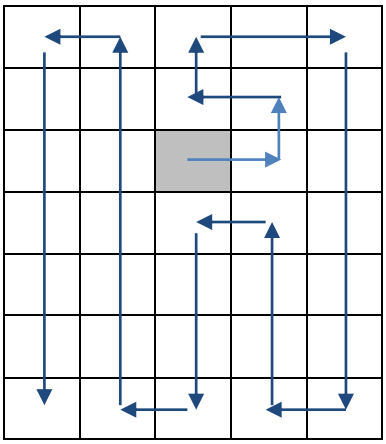
2). Обе стороны нечётны, и начальная клетка стоит в углу – просто заполним змейкой (или спиралью) – как в подзадаче 1.

3). Обе стороны нечётны, начальная клетка стоит в первой или последней строке либо столбце. Возможная схема заполнения для последней строки (остальные варианты – аналогично):



Заметим, что в первой или последней строке или столбце начальной может быть только клетка с нечётным номером, иначе решения нет (частный случай утверждения, доказанного выше).

4). Обе стороны нечётны, начальная клетка стоит не в крайнем столбце и не крайней строке. Возможная схема для случая, когда координата *y* чётна, *x* нечётна (для противоположной чётности делается аналогично):



Рассмотрим теперь особенности программной реализации. Каждый вариант заполнения можно свести к нескольким горизонтальным и/или вертикальным змейкам. Поэтому стоит написать две отдельные функции для заполнения змейкой.

Для удобства можно сделать эти функции универсальными – в качестве аргументов передаются координаты заполняемого прямоугольника и угловой клетки в нём, а функция сама определяет, в которую сторону пойдёт заполнение. При таком подходе размер решения оказывается относительно небольшим, хотя требует аккуратной реализации.

Пример полного решения на языке C++:

```
#include <bits/stdc++.h>

int a[21][21], cur = 0;

void vert_snake(int i, int j, int imin, int jmin, int imax,
int jmax) {
  int di = (i == imin) ? 1 : -1;
  for (int dj = (j == jmin) ? 1 : -1; j >= jmin && j <=
jmax; j += dj) {
    for (; i >= imin && i <= imax; i += di) {
      a[i][j] = ++cur;
    }
    di = -di;
    i += di;
  }
}

void hor_snake(int i, int j, int imin, int jmin, int imax,
int jmax) {
  int dj = (j == jmin) ? 1 : -1;
  for (int di = (i == imin) ? 1 : -1; i >= imin && i <=
imax; i += di) {
    for (; j >= jmin && j <= jmax; j += dj) {
      a[i][j] = ++cur;
    }
    dj = -dj;
    j += dj;
  }
}

int main() {
  int n, m, y, x;
  scanf("%d %d %d %d", &n, &m, &y, &x);
  if (m == 1) {
    if (y == 1 || y == n) {
      vert_snake(y, 1, 1, 1, n, m);
    }
    else {
      std::cout << -1;
      return 0;
    }
  } else if (n == 1) {
    if (x == 1 || x == m) {
      hor_snake(1, x, 1, 1, n, m);
    }
    else {
```

```
      std::cout << -1;
      return 0;
    }
  } else if ((n * m) % 2 == 0) {
    if (m % 2 == 0) {
      vert_snake(n - 1, 1, 1, 1, n - 1, m);
      hor_snake(n, m, n, 1, n, m);
    } else {
      hor_snake(1, m - 1, 1, 1, n, m - 1);
      vert_snake(n, m, 1, m, n, m);
    }
    int delta = a[y][x] - 1;
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= m; j++) {
        a[i][j] = (a[i][j] - 1 - delta + n*m) % (n*m) + 1;
      }
    }
  } else if ((y + x) % 2 != 0) {
      std::cout << -1;
      return 0;
  } else if (y % 2 == 0) {
    hor_snake(y, x, 1, x, y, m);
    vert_snake(1, x - 1, 1, 1, y, x - 1);
    vert_snake(y + 1, 1, y + 1, 1, n, m);
  } else if (y < n && x < m) {
    hor_snake(y, x, 1, x, y, m - 1);
    vert_snake(1, m, 1, m, y, m);
    vert_snake(y + 1, m, y + 1, x, n, m);
    if (x > 1) vert_snake(n, x - 1, 1, 1, n, x - 1);
  } else if (y == n && x == m) {
    hor_snake(y, x, 1, 1, n, m);
  } else if (y == n) {
    hor_snake(y, x, n, x, n, m);
    vert_snake(n - 1, m, 1, x, n - 1, m);
    if (x > 1) vert_snake(1, x - 1, 1, 1, n, x - 1);
  } else { // x == m
    vert_snake(y, x, y, m, n, m);
    hor_snake(n, m - 1, y, 1, n, m - 1);
    if (y > 1) hor_snake(y - 1, 1, 1, 1, y - 1, m);
  }
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      printf("%4d", a[i][j]);
    }
    printf("\n");
  }
}
```