

Разбор задач муниципального этапа
Всероссийской олимпиады школьников по информатике
2023–2024 учебного года

Задачи для вас готовили и тестировали выпускники Ульяновского государственного технического университета, многократные участники международных соревнований по программированию:

Даниил Александрович Горшков, Андрей Дмитриевич Дуванов, Максим Олегович Кашичкин,
Евгений Валерьевич Кондратьев, Максим Валерьевич Крутов, Владимир Александрович Фолунин.

Будем рады видеть вас среди участников чемпионата ИТ-сферы Ульяновской области,
а также регионального этапа Всероссийской командной олимпиады школьников по программированию!

Подробнее на сайтах <http://ulivt.ru> и http://vk.com/op_ulstu

Задача А (7–8 классы). Макс и длительность рабочего дня

Прочитаем значения a , m и b . Умножим a и b на 60, чтобы перевести из часов в минуты. Сложим все получившиеся значения, чтобы узнать общую длительность рабочего дня в минутах.

Это была простейшая задача, не содержащая «подводных камней» и не обладающая большой вариативностью решений.

Решение на Python

```
a = int(input())
m = int(input())
b = int(input())

print(a * 60 + m + b * 60)
```

Решение на C++

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

int main() {
    int a, m, b;
    cin >> a >> m >> b;

    cout << a * 60 + m + b * 60;
}
```

Задача В (7–8 классы), А (9–11 классы). Макс и вахта

Прочитаем продолжительность периода в переменную *period*, а продолжительность контракта — в переменную *totalDays*.

Сохраним в переменной *cycleCount* количество полных циклов, состоящих из двух периодов (один из них будет периодом труда, второй — периодом отдыха), происходящих за время контракта. Для этого целочисленно разделим *totalDays* на $(2 \cdot period)$. Также сохраним в переменную *remainingDays* остаток от этого деления — это количество дней, не входящих в полный цикл из двух периодов.

Чтобы найти минимальное количество рабочих дней, предположим, что самый первый день контракта Макса был выходным. В каждом из *cycleCount* полных циклов первые *period* дней будут выходными, а вторые *period* дней — рабочими. Из оставшихся *remainingDays* дней первые *period* дней будут выходными, а оставшиеся — рабочими. Таким образом, всего в этом варианте будет $cycleCount \cdot period + \max(remainingDays - period, 0)$ рабочих дней.

Аналогичным образом вычисляем максимальное количество рабочих дней. Для этого предположим, что первый день контракта был рабочим. Полные циклы так же содержат $cycleCount \cdot period$ рабочих дней, а из оставшихся первые *period* дней будут рабочими. Общее количество — $cycleCount \cdot period + \min(remainingDays, period)$.

Для получения частичного балла за первую группу тестов достаточно было выводить число $cycleCount \cdot period$.

Определённые дополнительные баллы (но не полный балл) также можно было получить за решения, основанные на переборе дней контракта в цикле и определении вида каждого из них.

Решение на Python

```
period = int(input())
total_days = int(input())

cycle_count = total_days // (2 * period)
remaining_days = total_days % (2 * period)

min_working_days = cycle_count * period +
    max(remaining_days - period, 0)
max_working_days = cycle_count * period +
    min(remaining_days, period)

print(min_working_days)
print(max_working_days)
```

Решение на C++

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

int main() {
    int period, totalDays;
    cin >> period >> totalDays;

    int cycleCount = totalDays / (2 * period);
    int remainingDays = totalDays % (2 * period);

    int minWorkingDays = cycleCount * period +
        max(remainingDays - period, 0);
    int maxWorkingDays = cycleCount * period +
        min(remainingDays, period);

    cout << minWorkingDays << " " << maxWorkingDays;
}
```

Задача С (7–8 классы), В (9–11 классы). Макс и слон

Пусть первая ладья расположена в строке y_1 и в столбце x_1 , а вторая — в строке y_2 и в столбце x_2 . Для простоты будем считать, что $y_1 \leq y_2$ (если это не так, обменяем значения y_1, x_1 с y_2, x_2).

Если две фигуры находятся на одной диагонали, то модуль разности номеров их строк будет равен модулю разности номеров их столбцов. В начале можно рассмотреть частный случай, когда обе ладьи стоят на одной диагонали, и в этом случае вывести координаты любого поля между ними на той же диагонали. Так, если $x_1 < x_2$, то ответом может быть поле в строке $y_1 + 1$ и в столбце $x_1 + 1$. Если же $x_1 > x_2$, то в качестве ответа можно взять поле в строке $y_1 + 1$ и в столбце $x_1 - 1$.

Перейдём к случаю, когда ладьи не находятся на одной диагонали. Можно перебрать двумя вложенными циклами все поля доски и для каждого из них проверить, лежит ли оно на одной диагонали с обеими ладьями (и не совпадает ли оно с ними) и вывести первый подходящий ответ. Сложность этого решения — $O(N^2)$, что не позволит набрать полный балл.

Заметим, что места расположения слонов обязаны быть точками пересечения прямых, проведённых через позиции ладей под углами 45 и 135 градусов относительно положительного направления оси Ox .

Проведём через первую ладью прямую под углом 45 градусов, а через вторую под углом 135 градусов к положительному направлению оси Ox . Уравнение первой прямой имеет вид $y = x + (y_1 - x_1)$, второй — $y = -x + (y_2 + x_2)$.

Пусть точка пересечения этих прямых имеет координаты $(x_3; y_3)$. Тогда $y_3 = x_3 + (y_1 - x_1) = -x_3 + (y_2 + x_2)$, откуда $x_3 = \frac{(y_2 + x_2) - (y_1 - x_1)}{2}$, $y_3 = \frac{(y_2 + x_2) + (y_1 - x_1)}{2}$. Таким образом находим первую позицию, куда возможно поставить слона.

Аналогично, проведя через первую ладью прямую под углом 135 градусов, а через вторую под углом 45 градусов к положительному направлению оси Ox , находим вторую возможную позицию слона. Из двух полученных позиций можно выбрать любую, не выходящую за границы доски.

Решение на Python

```
n = int(input())
row1 = int(input())
col1 = int(input())
row2 = int(input())
col2 = int(input())

if row1 > row2:
    row1, row2 = row2, row1
    col1, col2 = col2, col1

if row2 - row1 == col2 - col1:
    print(row1 + 1)
    print(col1 + 1)
elif row2 - row1 == col1 - col2:
    print(row1 + 1)
    print(col1 - 1)
else:
    col3 = (row2 + col2 - row1 + col1) // 2
    row3 = col3 + row1 - col1

    row1, row2 = row2, row1
    col1, col2 = col2, col1
    col4 = (row2 + col2 - row1 + col1) // 2
    row4 = col4 + row1 - col1

    if 1 <= row3 <= n and 1 <= col3 <= n:
        print(row3)
        print(col3)
    else:
        print(row4)
        print(col4)
```

Решение на C++

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

int main() {
    long long n, row1, col1, row2, col2;
    cin >> n >> row1 >> col1 >> row2 >> col2;

    if (row1 > row2) {
        swap(row1, row2);
        swap(col1, col2);
    }

    if (row2 - row1 == col2 - col1)
        cout << row1 + 1 << endl << col1 + 1;
    else if (row2 - row1 == col1 - col2)
        cout << row1 + 1 << endl << col1 - 1;
    else {
        long long col3 = (row2 + col2 - row1 + col1) / 2;
        long long row3 = col3 + row1 - col1;

        swap(row1, row2);
        swap(col1, col2);
        long long col4 = (row2 + col2 - row1 + col1) / 2;
        long long row4 = col4 + row1 - col1;

        if (1 <= row3 && row3 <= n &&
            1 <= col3 && col3 <= n)
            cout << row3 << endl << col3;
        else
            cout << row4 << endl << col4;
    }
}
```

Задача D (7–8 классы), C (9–11 классы). Макс и боулинг

В задаче требуется найти максимальную сумму двух непересекающихся подмассивов с заданными размерами (назовём эти подмассивы a и b , а их размеры — $aSize$ и $bSize$). Решения, основанные на наивном переборе всех положений a и b и наивном расчёте сумм, имеют сложность $O(N^3)$ и могут пройти лишь первую группу тестов.

Попробуем вычислять суммы эффективнее. Для этого используем вспомогательный массив p : элемент $p[i]$ будет хранить сумму элементов исходного массива с индексами от 0 до i включительно. Тогда сумма элементов исходного массива с индексами от l до r включительно может быть вычислена как $p[r] - p[l - 1]$ (либо просто как $p[r]$, если $l = 0$).

Такой вспомогательный массив префиксных сумм даёт возможным даже при полном переборе вариантов расположения a и b получить ответ со сложностью $O(N^2)$, что позволяет пройти вторую группу тестов. В качестве альтернативы можно было бы заранее сохранить все суммы участков нужной длины, начинающихся во всех возможных позициях.

Перейдём к полному решению. Предположим, что a располагается левее b (симметричный случай рассматривается аналогично). Пусть b занимает индексы от $bFrom$ до $bFrom + bSize - 1$, тогда его сумма равна $p[bFrom + bSize - 1] - p[bFrom - 1]$. Подмассив a должен заканчиваться где-то до $bFrom$, причём из всех возможных сумм a нам интересна максимальная. Мы можем для каждого возможного индекса сохранить максимальную сумму a , заканчивающегося левее этого индекса (у нас получится массив префиксных максимумов), это позволит найти ответ за один проход (для всех возможных позиций $bFrom$ складываем сумму b и максимальную сумму a левее $bFrom$, из всех таких значений выбираем наибольшее). На самом деле, массив префиксных максимумов избыточен, достаточно одной переменной.

Решение на Python

```
size = int(input())
a_size, b_size = [int(_) for _ in input().split()]

p = [int(_) for _ in input().split()]
for i in range(1, size):
    p[i] += p[i - 1]

max_points = 0

max_a_points = p[a_size - 1]
for b_from in range(a_size, size - b_size + 1):
    max_points = max(max_points, max_a_points + p[b_from + b_size - 1] - p[b_from - 1])
    max_a_points = max(max_a_points, p[b_from] - p[b_from - a_size])

max_b_points = p[b_size - 1]
for a_from in range(b_size, size - a_size + 1):
    max_points = max(max_points, p[a_from + a_size - 1] - p[a_from - 1] + max_b_points)
    max_b_points = max(max_b_points, p[a_from] - p[a_from - b_size])

print(max_points)
```

Решение на C++

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

int main() {
    int size, aSize, bSize;
    cin >> size >> aSize >> bSize;

    vector<long long> p(size);
    for (long long &value : p)
        cin >> value;
    partial_sum(p.begin(), p.end(), p.begin());

    long long maxPoints = 0;

    long long maxAPoints = p[aSize - 1];
    for (int bFrom = aSize; bFrom + bSize <= p.size(); bFrom++) {
        maxPoints = max(maxPoints, maxAPoints + p[bFrom + bSize - 1] - p[bFrom - 1]);
        maxAPoints = max(maxAPoints, p[bFrom] - p[bFrom - aSize]);
    }

    long long maxBPoints = p[bSize - 1];
    for (int aFrom = bSize; aFrom + aSize <= p.size(); aFrom++) {
        maxPoints = max(maxPoints, p[aFrom + aSize - 1] - p[aFrom - 1] + maxBPoints);
        maxBPoints = max(maxBPoints, p[aFrom] - p[aFrom - bSize]);
    }

    cout << maxPoints;
}
```

Задача Е (7–8 классы), D (9–11 классы). Макс и команда

Для прохождения первой группы тестов достаточно сложить в массив все возможные произведения, отсортировать его по неубыванию и вывести K -й элемент.

Во второй группе тестов значение K довольно мало. Здесь помогает наблюдение, что ответ обязательно должен быть среди произведений всех элементов первого массива и K наибольших элементов второго массива. Произведения можно, например, складывать в set или в приоритетную очередь, чтобы оставлять не более K максимальных.

Полное решение можно получить при помощи двоичного поиска по ответу. Научимся определять количество произведений, больших заданного числа P . Отсортируем оба массива. Переберём элемент первого массива, назовём его A . Тогда, чтобы произведение оказалось больше P , элемент второго массива должен быть больше $\left\lfloor \frac{P}{A} \right\rfloor$ (скобки обозначают округление вниз). Найти количество таких элементов во втором массиве можно либо при помощи второго двоичного поиска (функция `upper_bound` в C++, функция `bisect_right` в Python), либо при помощи метода двух указателей.

Теперь, когда мы умеем определять число произведений, больших P , осталось подобрать максимальное P , при котором таких произведений не меньше K . Это делается двоичным поиском: задаём заведомо подходящие начальные границы интервала (например, 0 и 10^{18}), используем в качестве P середину интервала. Если произведений оказалось достаточно, сдвигаем левую границу интервала на P , иначе сдвигаем правую границу интервала на P . Продолжаем до тех пор, пока границы интервала поиска не окажутся на соседних числах, тогда правая граница будет ответом.

Решение на Python

```
def greater_count(a, b, product):
    bi, count = len(b) - 1, 0
    for ai in range(len(a)):
        while bi >= 0 and a[ai] * b[bi] > product:
            bi -= 1
        count += len(b) - 1 - bi
    return count

size, place = [int(_) for _ in input().split()]
place -= 1

a = [int(_) for _ in input().split()]
a.sort()

b = [int(_) for _ in input().split()]
b.sort()

l, r = 0, 10 ** 18
while l + 1 < r:
    m = (l + r) // 2
    if greater_count(a, b, m) > place:
        l = m
    else:
        r = m

print(r)
```

Решение на C++

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

long long greaterCount(vector<long long> &a,
                       vector<long long> &b,
                       long long product) {
    long long count = 0;
    for (int ai = 0, bi = b.size() - 1; ai < a.size(); ai++) {
        long long minB = product / a[ai];
        auto it = upper_bound(b.begin(), b.end(), minB);
        count += b.end() - it;
    }
    return count;
}

int main() {
    int size;
    long long place;
    cin >> size >> place;
    place--;

    vector<long long> a(size);
    for (long long &value : a)
        cin >> value;
    sort(a.begin(), a.end());

    vector<long long> b(size);
    for (long long &value : b)
        cin >> value;
    sort(b.begin(), b.end());

    long long l = 0, r = 1e18;
    while (l + 1 < r) {
        long long m = l + (r - l) / 2;
        if (greaterCount(a, b, m) > place)
            l = m;
        else
            r = m;
    }

    cout << r;
}
```

Задача Е (9–11 классы). Макс и сбалансированные числа

Для первой группы тестов достаточно перебрать все числа от L до R и каждое проверить на соответствие условию.

Во второй группе тестов диапазон от L до R покрывает все числа с определённым количеством цифр. Чтобы эффективно определять количество сбалансированных чисел, имеющих N цифр, нужны определённые знания из комбинаторики.

Пусть, например, надо найти количество сбалансированных чисел из 6 цифр. Из этих цифр 3 — чётные и 3 — нечётные. Если самой левой была чётная цифра, то это может быть 2, 4, 6 или 8. Для любой из остальных цифр есть ровно 5 вариантов. Кроме того, из оставшихся 5 цифр нужно выбрать 2 позиции для чётных цифр. Это можно сделать $C_5^2 = 10$ способами (C_5^2 — число сочетаний из 5 по 2). Таким образом, общее количество сбалансированных шестизначных чисел, начинающихся с чётной цифры, — $4 \cdot 5^5 \cdot C_5^2 = 125000$.

Если самой левой была нечётная цифра, то для неё также есть 5 вариантов. Тогда общее количество сбалансированных шестизначных чисел, начинающихся с нечётной цифры, — $5^6 \cdot C_5^2 = 156250$, а всех сбалансированных шестизначных чисел — 281250. Для N -значных чисел ответ определяется по формуле $(4 + 5) \cdot 5^{N-1} \cdot C_{N-1}^{N/2}$.

Полное решение использует довольно нечасто встречающийся вид динамического программирования — динамическое программирование по цифрам (Digit DP). Заметим, что задачу «найти количество сбалансированных чисел от L до R включительно» можно свести к задаче «найти количество сбалансированных чисел от 1 до X включительно»: если ответ на вторую задачу обозначить как $f(X)$, то ответ на первую определяется как $f(R) - f(L - 1)$.

Будем рассматривать числа как строки: пусть верхняя граница задана строкой s , и мы хотим подсчитать количество сбалансированных чисел, не превышающих s . Будем подбирать цифры числа слева направо, пусть текущая позиция равна i . Нам нужно знать текущую разность между количеством нечётных и количеством чётных цифр, будем поддерживать её в переменной *balance*. Если на i -ю позицию мы ставим нечётную цифру, то *balance* увеличивается на 1, иначе — уменьшается на 1. Набранное число является сбалансированным, если $i = |s|$ и *balance* = 0.

Какие цифры мы можем поставить на i -ю позицию? Если цифры, выставленные ранее, совпадают с началом строки s , то нам нельзя использовать цифры, большие чем $s[i]$ (так как тогда всё набираемое число будет больше s). Однако, если ранее была использована хотя бы одна цифра, меньшая соответствующей цифры в s , то всё набираемое число уже заведомо меньше s , и на i -й позиции мы можем использовать любую цифру.

Чтобы разграничить эти две ситуации, будем поддерживать параметр *isPrefixOfS* — является ли набираемое число префиксом числа s . Если это так, то на i -й позиции мы можем использовать только цифры, не превышающие $s[i]$, а если нет, то мы можем использовать любые цифры.

Кроме того, нужно обратить внимание на ведущие нули: они не должны учитываться как чётные цифры при подсчёте *balance*. Поэтому нам понадобится ещё один параметр, *hasLeadingZero* — являлись ли все предыдущие выставленные цифры нулями. Если это так, то при выставлении нуля на i -ю позицию мы не должны уменьшать *balance*.

Соберём всё вместе: пусть $f(s, i, balance, isPrefixOfS, hasLeadingZero)$ — это количество сбалансированных чисел, не превышающих числа, заданного строкой s , в которых мы уже выставили первые i цифр и получили разность нечётных и чётных, равную *balance*, при этом *isPrefixOfS* = *true*, если все выставленные цифры совпадают с началом числа s , а *hasLeadingZero* = *true*, если все выставленные цифры были нулями.

Последовательно вычислим значения f , при этом будем сохранять уже рассчитанные ответы в массив, чтобы не вычислять их дважды (основная идея динамического программирования). В Python вместо явно определённого массива также можно использовать декоратор `lru_cache`.

Внутри функции перебираем цифру, которую мы поставим на i -е место (верхняя граница определяется числом s и параметром *isPrefixOfS*). На основе этой цифры определяем новые значения:

- *balance'* получает значение *balance* + 1, если новая цифра нечётная, *balance*, если новая цифра — ноль и *hasLeadingZero* = *true*, либо *balance* - 1, если новая цифра чётная и не равна нулю либо *hasLeadingZero* = *false*;
- *prefixOfS'* получает значение *true*, если *isPrefixOfS* = *true* и новая цифра равна $s[i]$, иначе *false*;
- *hasLeadingZero'* получает значение *true*, если *hasLeadingZero* = *true* и новая цифра — 0, иначе *false*.

Затем для каждой новой цифры выполняем рекурсивный вызов $f(s, i + 1, balance', isPrefixOfS', hasLeadingZero')$ и суммируем ответы.

Итоговый ответ на задачу равен $f(R, 0, 0, true, true) - f(L - 1, 0, 0, true, true)$.

Решение на Python

```
from functools import lru_cache

@lru_cache(maxsize=None)
def rec(s, i, balance, is_prefix_of_s, has_leading_zero):
    if i == len(s):
        return balance == 0 and not has_leading_zero

    res = 0
    max_digit = int(s[i]) if is_prefix_of_s else 9

    for digit in range(max_digit + 1):
        next_balance = balance
        if digit % 2:
            next_balance += 1
        elif digit or not has_leading_zero:
            next_balance -= 1

        next_is_prefix_of_s = is_prefix_of_s and digit == max_digit
        next_has_leading_zero = has_leading_zero and digit == 0

        res += rec(s, i + 1, next_balance, next_is_prefix_of_s, next_has_leading_zero)

    return res

l = input()
r = input()
r_count = rec(r, 0, 0, 1, 1)
l_count = rec(str(int(l) - 1), 0, 0, 1, 1)
print(r_count - l_count)
```

Решение на C++

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

long long rec(const string &s, int i, int balance, bool isPrefixOfS, bool hasLeadingZero,
              vector<vector<vector<vector<long long>>>> &memo) {
    long long &res = memo[i][balance + 20][isPrefixOfS][hasLeadingZero];
    if (res != -1)
        return res;

    if (i == s.size())
        return res = !balance && !hasLeadingZero;

    res = 0;
    int maxDigit = isPrefixOfS ? s[i] - '0' : 9;

    for (int digit = 0; digit <= maxDigit; digit++) {
        int nextBalance = balance;
        if (digit % 2)
            nextBalance++;
        else if (!hasLeadingZero || digit)
            nextBalance--;

        int nextIsPrefixOfS = isPrefixOfS && digit == maxDigit;
        int nextHasLeadingZero = hasLeadingZero && !digit;

        res += rec(s, i + 1, nextBalance, nextIsPrefixOfS, nextHasLeadingZero, memo);
    }

    return res;
}

int main() {
    long long l, r;
    cin >> l >> r;

    vector<vector<vector<vector<long long>>>> memo(20, vector<vector<vector<long long>>>>(41,
        vector<vector<long long>>(2, vector<long long>(2, -1))));
    long long rCount = rec(to_string(r), 0, 0, 1, 1, memo);

    memo.assign(20, vector<vector<vector<long long>>>>(41,
        vector<vector<long long>>(2, vector<long long>(2, -1))));
    long long lCount = rec(to_string(l - 1), 0, 0, 1, 1, memo);

    cout << rCount - lCount;
}
```