Открытая олимпиада по программированию
Осенний тур 2018
20 октября 2018

A. Archeologist's find

Ответ выводится в зависимости от расположения точки относительно осей.

Асимптотика по времени: $O(1)$.

```
#include <iostream>

int main() {
    int x, y, ans;
    std::cin >> x >> y;
    if (y == 0)
        if (x >= 0)
            ans = 0;
        else
            ans = 3;
    else
        if (x > 0)
            ans = 1;
        else
            ans = 2;
    std::cout << ans;
    return 0;
}
```

B. Board rotating

Обозначим $a_{i,j}$ — исходную таблицу, $b_{i,j}$ — итоговую таблицу. При повороте по часовой стрелке $b_{i,j} = a_{15-j,i}$. При повороте против часовой стрелки $b_{i,j} = a_{j,15-i}$.

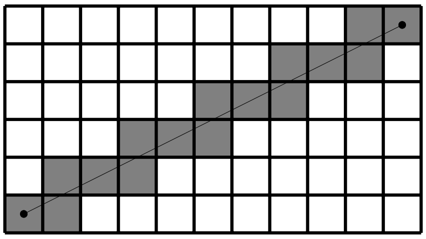
```
#include <iostream>

int main() {
    const int n = 16;
    char a[n][n], b[n][n];
    int right = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            std::cin >> a[i][j];
            if (a[i][j] == 'R')
                right = 1;
        }
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (right)
                b[i][j] = a[n - 1 - j][i];
            else
                b[i][j] = a[j][n - 1 - i];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            std::cout << b[i][j];
        std::cout << std::endl;
    }
    return 0;
}
```

Асимптотика по времени: $O(1)$.

C. Counting pixels

Пусть A — начало отрезка, B — конец отрезка; a – смещение по одной оси, b — по другой оси. Рассмотрим два случая: отрезок не проходит ни через один из углов пикселей и отрезок проходит через какой-либо угол.

В первом случае

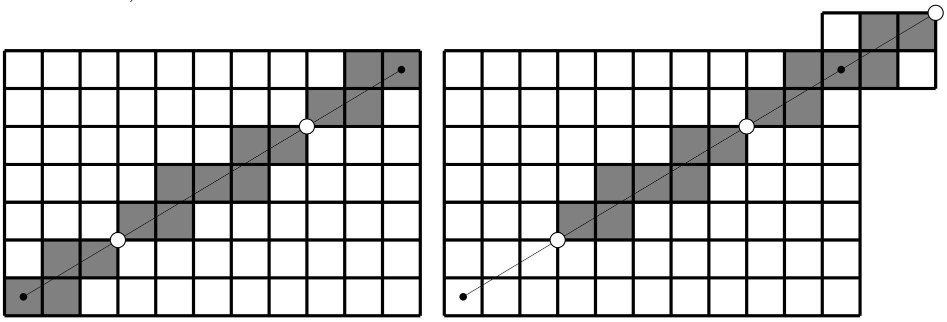


Количество закрашенных пикселей будет

$$a + b + 1$$

так как каждый сдвиг по любой из осей — переход в новый закрашенный пиксель, и еще начальный пиксель.

Во втором случае каждый из узлов уменьшает ответ первого случая на 1 (так как переход происходит сразу по 2 осям). Посчитаем количество узлов. Выберем любой из узлов C, через который проходит отрезок. И перенесем параллельным переносом часть отрезка AC и все сопутствующие ему пиксели так, чтобы A совпало с B.



Получили классическую задачу: количество внутренних узлов на отрезке, которых начинается и заканчивается в узлах. Ответ: $d - 1$, где $d = \text{НОД}(a, b)$. Учитывая сам узел C, получим, что ответ

$$a + b + 1 - d.$$

Наибольший общий делитель можно найти алгоритмом Евклида.

Осталось отметить, как отличать первый и второй случай. Во втором минимальный шаг от узла до узла будет равен $\left(\frac{a}{d}; \frac{b}{d}\right)$. Двигаясь по данному направлению мы попадаем в центр пикселя, только если $k\frac{a}{d}$ и $k\frac{b}{d}$ будут полуцелыми. Ясно, что k тогда само — полуцелое число. Но тогда $\frac{a}{d}$ и $\frac{b}{d}$ не могут быть четными.

Асимптотика по времени: $O(\log \max(i_1, i_2, j_1, j_2))$.

```
#include <iostream>
#include <cstdlib>

long long gcd(long long a, long long b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

int main() {
    long long i1, j1, i2, j2, a, b, d, ans;
    std::cin >> i1 >> j1 >> i2 >> j2;
```

```
    a = llabs(i1 - i2);
    b = llabs(j1 - j2);
    ans = a + b + 1;
    d = gcd(a, b);
    a /= d;
    b /= d;
    if (a % 2 == 1 && b % 2 == 1)
        ans -= d;
    std::cout << ans;
    return 0;
}
```

D. Digits again

Заметим, что в виде $\lambda_1^n + \lambda_2^n$ записываются решения линейного рекуррентного соотношения

$$x_{n+2} - (\lambda_1 + \lambda_2)x_{n+1} + \lambda_1\lambda_2 x_n = 0,$$

у которого характеристический многочлен имеет корни λ_1 и λ_2.

Если обозначить:

$$x_n = \left(a + \sqrt{b}\right)^n + \left(a - \sqrt{b}\right)^n$$

то получим, что

$$x_{n+2} - 2ax_{n+1} + (a^2 - b)x_n = 0$$

Значит:

$$x_{n+1} = 2ax_n + (b - a^2)x_{n-1}$$

Считать такую рекурренту наивным образом слишком долго.

Решение 1. Воспользуемся матричной формой записи:

$$\begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix} = \begin{pmatrix} 2a & b - a^2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_n \\ x_n - 1 \end{pmatrix} = \begin{pmatrix} 2a & b - a^2 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} x_1 \\ x_0 \end{pmatrix}$$

Возвести матрицу в степень можно бинарным возведением. Стоит отметить, что все операции следует выполнять по модулю 1000, не забывая аккуратно работать с разностью остатков (чтобы не получать отрицательные числа).

Асимптотика по времени: $O(\log n)$.

Решение 2. Заметим, что рекуррентное соотношение определяется двумя подряд идущими элементами. Так как мы смотрим рекуррентное соотношение по модулю 1000, то существует не более 1000000 различных пар остатков. Значит, если промоделировать $k > 1000000$ шагов, то там точно найдутся две одинаковые пары остатков.

Найдем максимально $j < k$ такое, что $x_j = x_k$. Тогда $m = k - j$ будет периодом для последовательности x_i. Теперь, чтобы вывести ответ x_n поступим так:

$$x_n = \begin{cases} x_n, & n < k \\ x_{j+((n-k) \bmod m)}, & n \geqslant k \end{cases}$$

Асимптотика по времени: $O(s^2)$, где s — модуль (1000).

```
#include <iostream>
#define mod 1000
typedef int Matrix[2][2];

void matr_mult(Matrix A, Matrix B, Matrix C) {
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++) {
            C[i][j] = 0;
            for (int k = 0; k < 2; k++)
                C[i][j] += A[i][k] * B[k][j];
                C[i][j] %= mod;
        }
}

void matr_power(Matrix A, long long n, Matrix B) {
    if (n == 0) {
        B[0][0] = 1; B[0][1] = 0;
        B[1][0] = 0; B[1][1] = 1;
        return;
    }
    int half[2][2], half2[2][2];
    matr_power(A, n / 2, half);
```

```
    if (n % 2 == 1) {
        matr_mult(half, half, half2);
        matr_mult(half2, A, B);
    } else
        matr_mult(half, half, B);
}

int main() {
    long long a, b, n;
    std::cin >> a >> b >> n;
    a %= mod;
    b %= mod;
    int m11 = 2 * a % mod;
    int m12 = (b - a * a % mod + mod) % mod;
    Matrix A = {{m11, m12}, {1, 0}}, B;
    matr_power(A, n, B);
    int x0 = 2;
    int x1 = 2 * a % mod;
    int ans = (B[1][0] * x1 + B[1][1] * x0) % mod;
    std::cout << ans;
    return 0;
}
```

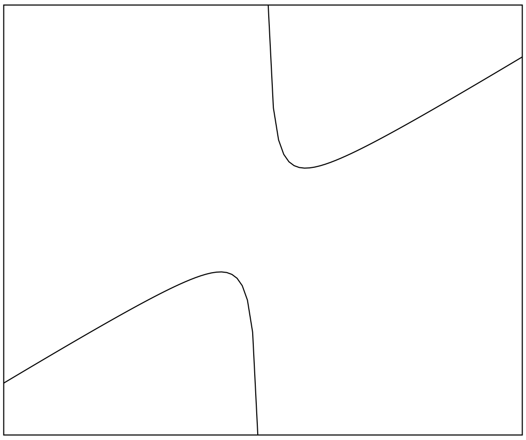
E. Emirates

По внешнем виду можно угадать график функции

$$ax + \frac{b}{x},$$

где $a > 0$, $b > 0$. Коэффициенты подбираются из примеров:

$$f(x) = 2x + \frac{12}{x}$$



А еще день независимости ОАЭ это 2.12 =)

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

int main() {
    double x, ans;
    cin >> x;
    ans = 2.0 * x + 12.0 / x;
    cout << fixed << setprecision(6) << ans;
    return 0;
}
```

F. Finding battleships

Запустим обход в глубину (dfs) на графе, соответствующем таблице (переходить можно на соседнюю по вертикали или горизонтали клетку) для каждой еще не посещенной клетки. При этом будет вычислить 5 параметров: $size$ — количество клеток в компоненте связности, i_{min}, i_{max} — минимальная и максимальная строка, до которой дошел обход в данной компоненте, j_{min}, j_{max} — минимальный и максимальный столбец, до которого дошел обход в данной компоненте. Обозначим $height = i_{max} - i_{min} + 1$, $width = j_{max} - j_{min} + 1$. Область будет прямоугольной тогда, и только тогда, когда:

$$height * width = size$$

Определить тип прямоугольника можно естественным образом.

Асимптотика по времени: $O(n \cdot m)$.

```cpp
#include <iostream>
#include <algorithm>

int n, m, imax, imin, jmax, jmin, size;
bool used[101][101];
char matrix[101][101];
int di[] = {1, -1, 0, 0}, dj[] = {0, 0, 1, -1};

bool check(int i, int j) {
    return i >= 0 && i < n && j >= 0 && j < m &&
           matrix[i][j] == 'X' && not used[i][j];
}

void dfs(int i, int j) {
    used[i][j] = true;
    size++;
    imax = std::max(imax, i);
    imin = std::min(imin, i);
    jmax = std::max(jmax, j);
    jmin = std::min(jmin, j);
    for (int k = 0; k < 4; k++) {
        int nexti = i + di[k];
        int nextj = j + dj[k];
        if (check(nexti, nextj))
            dfs(nexti, nextj);
    }
}

int main() {
    int high = 0, wide = 0, square = 0;
    std::cin >> n >> m;
    for (int i = 0; i < n; i++)
        std::cin >> matrix[i];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++) {
            imax = -1, imin = n;
            jmax = -1, jmin = m;
            size = 0;
            if (check(i, j)) {
                dfs(i, j);
                int height = imax - imin + 1;
                int width = jmax - jmin + 1;
                if (height * width == size) {
                    if (height > width)
                        high++;
                    if (height < width)
                        wide++;
                    if (height == width)
```

```
                        square++;
                }
            }
        }
    std::cout << high << '␣';
    std::cout << wide << '␣';
    std::cout << square;
    return 0;
}
```

G. Geometrying

Уравнение плоскости:

$$\frac{x}{p} + \frac{y}{q} + \frac{z}{r} = 1$$

Точка лежит выше плоскости, если левая часть больше 1. Точка лежит ниже, если левая часть меньше 1. Данные условия можно проверить без перехода к вещественным числам:

$$check(x, y, z) = xqr + ypr + xpq - pqr$$

Таким образом легко проверить, что отрезок строго пересекает плоскость, если

$$\begin{cases} check(x_1, y_1, z_1) < 0 \\ check(x_2, y_2, z_2) > 0 \end{cases}$$

Осталось посчитать сколько из 12 ребер куба строго пересекают плоскость и прибавить к ним количество вершин, которые лежат на плоскости $check(x, y, z) = 0$.

Сгенерировать координаты вершин куба можно с помощью битовых операций. Проверить, какие из вершин образуют ребро можно с помощью условия, что расстояние (вдоль ребер) равно 1.

Асимптотика по времени: $O(1)$.

```
#include <iostream>
#include <cstdlib>

int v[8][3], a, p, q, r;

int check(int i) {
    long long int s = v[i][0] * q * r +
                      p * v[i][1] * r +
                      p * q * v[i][2] -
                      p * q * r;
    if (s < 0)
        return -1;
    if (s > 0)
        return 1;
    return 0;
}

bool is_edge(int i, int j) {
    long long int s = 0;
    for (int k = 0; k < 3; k++)
        s += llabs(v[i][k] - v[j][k]);
    return s == a;
}

int main() {
    int ans = 0;
    std::cin >> a >> p >> q >> r;
    for (int i = 0; i < 8; i++)
        for (int k = 0; k < 3; k++)
            v[i][k] = a * ((i >> k) & 1);
    for (int i = 0; i < 7; i++)
        for (int j = i + 1; j < 8; j++)
            if (is_edge(i, j))
                if (check(i) * check(j) == -1)
                    ans++;
    for (int i = 0; i < 8; i++)
        if (check(i) == 0)
            ans++;
    std::cout << ans;
    return 0;
}
```

H. Highest and greatest only

Обозначим $f(n, k)$ — количество целых чисел от 1 до n, у которых все цифры меньше или равны k. Во-первых, найдем максимальное $m < n$, у которого все цифры не превосходят k. Это можно сделать двигаясь от старших разрядов к младшим пока цифра не больше k. Если встретилась цифра, больше k, то заменяем на k эту цифру и все остальные цифры после неё (в сторону младших цифр). Можно сделать за $\log n$. Во-вторых, получаем ответ как результат перевода числа m из $(k+1)$-ичной системы счисления. Тоже можно сделать за $\log n$. Например, $n = 3251$ и $k = 3$. Найдем $m = 3233$. Сколько чисел меньше 3233 и имеет цифры $0, 1, 2, 3$: 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33, 100, ..., 3230, 3231, 3232, 3233. Если их сопоставить записи в системе счисления по основанию 4, то получим все натуральные числа до $3 \cdot 4^3 + 2 \cdot 4^2 + 3 \cdot 4^1 + 3 \cdot 4^0$.

Обозначим $g(n, k)$ — количество целых чисел от 1 до n, у которых максимальная цифра в точности равна k. Тогда $g(n, k) = f(n, k) - f(n, k-1)$. Ответ на задачу для чисел от 1 до n:

$$s(n) = 1 \cdot g(n, 1) + 2 \cdot g(n, 2) + ... + 9 \cdot g(n, 9)$$

Ответ на задачу для чисел от l до r:

$$s(r) - s(l-1).$$

Асимптотика по времени: $O(\log n)$.

```
#include <iostream>
#define mod 1000000007

long long f(long long n, int k) {
    int d[20];
    int len = 0;
    while (n > 0) {
        d[len++] = n % 10;
        n /= 10;
    }
    for (int i = len - 1; i >= 0; i--)
        if (d[i] > k)
            for ( ; i >= 0; i--)
                d[i] = k;
    long long answer = 0LL;
    for (int i = len - 1; i >= 0; i--)
        answer = answer * (k + 1) + d[i];
    return answer;
}

long long g(long long n, int k) {
    return (f(n, k) - f(n, k - 1) + mod) % mod;
}

long long sum(long long n) {
    long long ans = 0;
    for (int k = 1; k <= 9; k++) {
        ans += k * g(n, k);
        ans %= mod;
    }
    return ans;
}

int main() {
    long long L, R;
    std::cin >> L >> R;
    std::cout << (sum(R) - sum(L - 1) + mod) % mod;
    return 0;
}
```

I. Into the mountains

Посчитаем L_i — длину наибольшей возрастающей подстроки, которая находится левее i-го элемента и заканчивается в i. Это можно сделать за линейный проход слева направо:

$$L_i = \begin{cases} L_{i-1} + 1, & a_{i-1} < a_i \\ 0, & a_{i-1} \geqslant a_i \end{cases}$$

Посчитаем R_i — длину наибольшей убывающей подстроки, которая находится правее i-го элемента и заканчивается в i. Это можно сделать за линейный проход справа налево:

$$R_i = \begin{cases} R_{i+1} + 1, & a_i > a_{i+1} \\ 0, & a_i \leqslant a_{i+1} \end{cases}$$

Полуширина горы с вершиной в точке i определяется как $w_i = \min(L_i, R_i)$. Полная ширина горы с равна $2w + 1$. Найдем максимальную ширину и выведем $i - w_i$, $i + w_i$.

```cpp
#include <iostream>
#include <algorithm>

int a[100000];
int L[100000];
int R[100000];

int main() {
    int n;
    std::cin >> n;
    for (int i = 0; i < n; i++)
        std::cin >> a[i];
    for (int i = 1; i < n; i++)
        if (a[i - 1] < a[i])
            L[i] = L[i - 1] + 1;
    for (int i = n - 2; i >= 0; i--)
        if (a[i] > a[i + 1])
            R[i] = R[i + 1] + 1;
    int ans = 1, start = 0, end = 0;
    for (int i = 1; i < n - 1; i++) {
        int w = std::min(L[i], R[i]);
        int len = 2 * w + 1;
        if (ans < len) {
            ans = len;
            start = i - w;
            end = i + w;
        }
    }
    std::cout << start + 1 << ' ' << end + 1;
    return 0;
}
```