Министерство образования Новосибирской области
ГАУ ДО НСО «Областной центр развития творчества детей и юношества»
МБОУ «Гимназия № 1» г. Новосибирска
Школа иностранных языков «Юнисити»

ЗОЛОТАЯ СЕРЕДИНА

XV открытая региональная межпредметная олимпиада «Золотая середина» с участием стран СНГ

Разбор заданий

Информатика **Программирование на Pascal/Python/C++**

Максимальное количество баллов: 500
Можно использовать стандартные потоки ввода/вывода (с клавиатуры), либо файлы *input.txt* и *output.txt*
Ограничение по времени: 2 секунды на тест
Ограничение по памяти: 512 МБ
Авторские решения в данном документе приведены на языке C++

Общие слова от авторов задач:

В этом году задачи планировались примерно одинакового уровня сложности с задачами прошлого года. Появилась задача на простую структуру данных *stack*. Результаты оказались чуть ниже ожидаемых, вероятно из-за порядка задач. Участники *обязаны* прочитать условия *всех* задач, перед тем, как начинать работу над одной из них! Это один из первых уроков спортивного (олимпиадного) программирования. Самой сложной была первая задача, а участники не уделили достаточно внимания несравнимо более простой задаче 4. Также в этом году требовалось знание некоторых областей выбранного языка программирования, которые будут упомянуты далее в данном разборе. Ошибки реализации разочаровывают обычно меньше, чем подобные моменты. Для успешного выступления необходимо хорошо разбираться не только в алгоритмизации задач, но и в используемом языке программирования. От участников не требуется знания всего Standard Template Library в C++, но и реализовывать самостоятельно сортировку «пузырёк» — плохая идея, учитывая наличие реализаций куда лучших алгоритмов во всех современных языках.

Приказ №1

Вам передаются подкрепления сил X-CODE. Распределите отряды Кубитной Империи так, чтобы в каждом районе хватало сил для обороны. К сожалению, во избежание путаницы, в один район можно отправить только один отряд. Подходите к выбору внимательно! Командование заинтересовано в победе, ничья в ходе обороны равна поражению. Помните, в X-CODE служат только лучшие из лучших!

Формат входных данных

На вход подаётся натуральное число N — количество районов ($1 \leqslant N \leqslant 10^5$).
Для каждого района известен баланс сил, указанный в следующих N целых числах b_i, по одному в строчке: положительное число обозначает перевес великой Кубитной Империи, отрицательное — мелкие успехи Байтляндии ($-10^{18} \leqslant b_i \leqslant 10^{18}$).
Далее идёт целое число M — количество отрядов ($0 \leqslant M \leqslant 10^5$).
Далее идут M строк, в каждой из которых два натуральных числа: c_i — количество человек в отряде, и s_i — навыки каждого бойца ($1 \leqslant c_i, s_i \leqslant 10^{18}$).
Помните, что суммарная боевая мощь может превосходить все ваши представления!

Формат выходных данных

Контрольный Комитет хочет получить данные о передвижении войск. Вывести M пар чисел, в которых первое указывает на номер подразделения (в порядке ввода), а второе — на номер района. Если участие подразделения не требуется, в качестве номера района выведите 0. Порядок вывода пар чисел неважен.

В самой последней строчке выведите фразу *we lose T districts*, если количество потерянных районов T не равно одному, и *we lose 1 district* в противном случае.

Категоризация

Знание алгоритмов сортировки, особенно сортировки массивов из структур. Навыки алгоритмизации.

Алгоритм решения

Эта задача решается «жадным» алгоритмом после удачной сортировки. Для решения нам требуется устанавливать соответствия «лучший отряд — район с худшим балансом». Соответственно нужно отсортировать массив структур (или двумерный массив с одной из размерностей 2) в противоположных порядках: один по возрастанию, другой по убыванию или наоборот. Для этого нужно было или использовать средства языка программирования (`sorted` в Python, `std::sort` в C++, `Sort` в PascalABC.NET), или реализовывать сортировку самостоятельно.

После сортировки необходимо было рассматривать 4 случая, из них первые 3 в цикле, а 4 – условие останова:

1. Есть отряд и есть район – проверить, что если мы отправим отряд в данный район, мы сможем защитить и удержать данный район. Отправить отряд или отметить район как утраченный соответственно.
2. Есть район, но все отряды уже были использованы – посчитать район как утраченный, если баланс сил в нём неположительный ($\leqslant 0$).
3. Есть отряд, но во все районы уже отправили подкрепления – вывести соответствующее сообщение.
4. Обработаны все отряды и районы – вывести количество утраченных районов и завершить работу.

Примеры входных и выходных данных

Пример входных данных	Верный ответ
2 -10 -5 5 1 5 1 10 1 11 1 1 1 1	3 1 2 2 1 0 4 0 5 0 we lose 0 districts
2 -10 -5 5 1 5 1 2 1 11 1 1 1 1	3 1 1 0 2 0 4 0 5 0 we lose 1 district
1 0 1 1000 1000	1 1 we lose 0 districts

Разбор примеров

Возьмём первый пример и поясним алгоритм на его основе.

Считаем все данные в два динамических массива структур (описание массива на псевдоязыке):

```
vector<map_cell> map = { {1, -10}, {2, -5} };
vector<strike_force> forces = { {1, 5}, {2, 10}, {3, 11}, {4, 1}, {5, 1} };
uint32_t n = 2, m = 5;
```

Отсортируем массивы: районы в неубывающем порядке, отряды в невозрастающем порядке.

```
vector<map_cell> map = { {1, -10}, {2, -5} };
vector<strike_force> forces = { {3, 11}, {2, 10}, {1, 5}, {4, 1}, {5, 1} };
```

Сделаем одну итерацию цикла алгоритма. Есть отряд (3) и район (1), сил хватит, отправим отряд в район.

Сделаем ещё одну итерацию цикла алгоритма. Есть отряд (2) и район (2), сил хватит, отправим отряд в район.

Сделаем ещё одну итерацию цикла алгоритма. Есть отряд (1), но нет района. Случай 3.

Аналогично ещё 2 итерации случая 3: отряд 4 и отряд 5. Это заканчивает обработку всех отрядов и районов. Выводим количество утраченных районов (0).

Во втором примере из условия будет отличаться вторая итерация цикла алгоритма, на которой мощь отряда (номер 1, мощь 5) и района (номер 2, баланс -5) совпадут. Но это значит, что именно на этой итерации мы пометим этот район как утраченный, а отряд останется неиспользованным.

Возможные ошибки

- Самая сложная задача была первой — участники тратили много времени на попытки её решить, вместо того, чтобы найти задачи попроще.
- Участники полагали, что начальный баланс всегда отрицательный либо всегда положительный, т.е. забывали обработать все 3 различных случая.
- Множество ошибок (опечаток) в словах *we lose* и *districts*. Решения тестировались программой, которая такое халатное обращение со словами не прощает.

Авторское решение

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

using namespace std;

// Специализированные типы данных std::pair<K, V> для хранения данных районов и отрядов
using map_cell = pair<uint32_t, int64_t>;
using strike_force = pair<uint32_t, uint64_t>;

// Две функции сравнения std::pair<K, V> по второму аргументу
template <typename T>
bool compareLess(T i1, T i2)
{
    return i1.second < i2.second;
};

template <typename T>
bool compareGreater(T i1, T i2)
{
    return i1.second > i2.second;
};

int main()
{
    vector<map_cell> map;
    vector<strike_force> forces;
    uint32_t n, m;

    cin >> n;
    for (uint32_t i = 0; i < n; i++)
    {
        int64_t tmp;
        cin >> tmp;

        // Сохраняем пару чисел (номер, текущий баланс)
        map.emplace_back(i + 1, tmp);
    }

    cin >> m;
    for (uint32_t i = 0; i < m; i++)
    {
        uint64_t tmp1, tmp2;
        cin >> tmp1 >> tmp2;

        // Сохраняем пару чисел (номер, сила отряда)
```

```
48          forces.emplace_back(i + 1, tmp1 * tmp2);
49      }
50
51      // Сортируем отряды в невозрастающем (убывающем) порядке
52      sort(forces.begin(), forces.end(), compareGreater<strike_force>);
53
54      // Сортируем районы в неубывающем (возрастающем) порядке
55      sort(map.begin(), map.end(), compareLess<map_cell>);
56
57      const uint32_t min_count = min(map.size(), forces.size());
58
59      // Два итератора (текущих индекса) района и отряда
60      uint32_t map_index = 0, force_index = 0;
61      // Счётчик количества утраченных районов
62      uint32_t lost_districts = 0;
63
64      // Обрабатываем ситуации "отряд - район"
65      for (; map_index < min_count; map_index++)
66      {
67          // Аккуратно работаем с модулем баланса для избежания переполнения
68          const uint64_t abs = map[map_index].second >= 0
69              ? static_cast<uint64_t>(map[map_index].second)
70              : static_cast<uint64_t>(-map[map_index].second);
71
72          const bool positive = map[map_index].second > 0;
73
74          if (positive || (forces[force_index].second > abs))
75              // Успешно, отправляем отряд
76              cout << forces[force_index++].first << " " << map[map_index].first << endl;
77          else
78              // Наш лучший оставшийся отряд не может защитить наш худший оставшийся район
79              lost_districts++;
80      }
81
82      // Обрабатываем ситуации " - район"
83      for (; map_index < map.size(); map_index++)
84          // Считаем количество утраченных районов
85          if (map[map_index].second <= 0)
86              lost_districts++;
87
88      // Обрабатываем ситуации "отряд - "
89      for (; force_index < forces.size(); force_index++)
90          // Отряд не используется
91          cout << forces[force_index].first << " " << 0 << endl;
92
93      // Последняя строка вывода
94      cout << "we lose " << lost_districts;
95      if (lost_districts == 1)
96          cout << " district";
97      else
98          cout << " districts";
99  }
```

Приказ №2

Великолепно, коммандер! Пора переходить в наступление! Наши разведчики смогли взломать старые компьютеры Байтляндии и обнаружили документацию на все оборонительные объекты противника. К сожалению, записи об их местоположении зашифрованы интересным кодом. Для расшифровки, в каждой строке необходимо удалить все вхождения наиболее часто встречающейся букры. **Букра** — это цифра или буква латинского алфавита обоих регистров (строчные и прописные). Если разные букры встречаются одинаковое число раз — удалите любую. К сожалению, центр, который работал над этой задачей, 2 стандартных часа назад был уничтожен прямым ударом кружкой с кофе. Они как будто знали, куда лить! Увы, из их наработок осталось лишь описание алгоритма

дешифровки, вашему отделу придётся написать новую программу для этой цели. Нам нужна информация, командер!

Формат входных данных

На вход подаётся набор строк $\{s_i\}$ длин $\{l_i\}$ $(0 \leqslant l_i \leqslant 4 \cdot 10^6)$, их количество заранее неизвестно (считывать необходимо пока есть ещё данные). Некоторые строки могут быть пустыми. Сумма длин всех строк равна $L = \sum_i l_i \leqslant 4 \cdot 10^6$.

Формат выходных данных

Контрольный Комитет хочет получить расшифрованные записи, без изменения порядка следования.

Категоризация

Умение считывать данные до конца потока ввода и знание простейших операций с символами и строками. Одномерные массивы чисел.

Алгоритм решения

Считывать построчно из потока ввода, в каждой строке считать количество вхождений каждого символа из указанного диапазона, выбирать максимум и выводить строку посимвольно, пропуская найденный символ с максимальным числом вхождений.

Возможные ошибки

- Участники не знали, как организовать ввод до конца файла/ввода из консоли. Не умели закрывать поток ввода с клавиатуры (типичные сочетания клавиш в Windows: Ctrl + Z, в Linux: Ctrl + D).
- Допускались ошибки в именах файлов ввода/вывода; вопросы вызывало требование использовать одно и то же имя файлов для всех задач (*input.txt, output.txt*).
- Не обрабатывались некоторые символы или наоборот, лишние символы учитывались в подсчётах максимума.
- Участники считывали весь файл в память, а затем его обрабатывали, что приводило к медленной работе программы и соответствующим вердиктам `Time Limit` (превышение ограничения по времени работы программы).

Авторское решение

```cpp
#include <iostream>
#include <string>
#include <cstdint>
#include <vector>
#include <algorithm>

using namespace std;

// min({ 'A', 'a', '0' }) = min({65, 97, 48}) = 48 = '0'
constexpr char BASE_CODE = '0';

// max({ 'Z', 'z', '9' }) - '0' + 1 = max({90, 122, 57}) - 48 + 1 = 75
constexpr uint32_t LENGTH = max({ 'Z', 'z', '9' }) - BASE_CODE + 1;

pair<bool, char> letter(const string& s)
{
    // Счётчик количества символов в строке ('0'..'z'), со смещением на BASE_CODE
    // Смещение на BASE_CODE необязательно (не приводило к неправильным ответам),
    // но обычно используется в таких задачах
    vector<uint32_t> chars(LENGTH);

    for (auto c : s)
        if ((c ⩾ 'A' && c ⩽ 'Z') || (c ⩾ 'a' && c ⩽ 'z') || (c ⩾ '0' && c ⩽ '9'))
```

```
            chars[c - BASE_CODE]++;

    // После подсчёта находим максимум и соответствующий ему символ
    // (со смещением на BASE_CODE, как заполняли)
    uint32_t max = 0, pos = 0;
    for (uint32_t i = 0; i < LENGTH; i++)
    {
        if (chars[i] > max)
        {
            max = chars[i];
            pos = i;
        }
    }

    // Возвращаем пару (есть символы для удаления, символ для удаления)
    return make_pair(max > 0, pos + BASE_CODE);
}

int main()
{
    string s;
    // Читаем построчно (в строку s) до конца файла
    while (getline(cin, s))
    {
        // Ищем символ для удаления
        const pair<bool, char> let = letter(s);

        if (!let.first)
        {
            // Такого нет (пустая строка), печатаем всю строку
            cout << s << endl;
            continue;
        }

        for (auto c : s)
            // Посимвольно проходимся по строке, печатая все символы кроме найденного
            if (c != let.second)
                cout << c;

        cout << endl;
    }
}
```

Приказ №3

Следующее задание, Коммандер! Как вам известно, Кубитная Империя использует карты, составленные по данным различных источников: аэрофотосъемка, спутниковые фото, зарисовки разведчиков и так далее. В ходе стремительного наступления на всех фронтах, в генеральном штабе армии скопилось огромное количество карт различных местностей. Для удобства, все они пронумерованы и маркированы цветом для идентификации объекта. В штаб постоянно поступают новые карты, действовать нужно без промедления.

Карты лежат на генеральном столе в генеральной стопке. В любой момент может произойти одно из двух событий: либо адъютант принесет ещё одну карту, либо начальник штаба сочтёт хранение верхней карты из стопки нецелесообразным по причине неполноты данных. Штабистам куда удобнее работать не с кусками карт, а с единым целым, поэтому, как только на вершине стопки набирается M карт одного цвета, их объединяют в единую карту. Получившаяся карта сохраняет цветовую маркировку исходных кусочков, а в качестве номера имеет сумму номеров фрагментов. После объединения карта не возвращается в стопку, а передаётся отделу планирования кампании. Переданная карта регистрируется в картотеке, а её номер (каждый, бесспорно, определяющий успех всей кампании!) прибавляется к баллам стратегического перевеса. Если начальник штаба счёл карту неподходящей для использования штабом, из баллов стратегического перевеса вычитается номер карты. Если от начальника поступил приказ изъять верхнюю карту при пустой стопке — увы, ситуация безнадёжная. Без адекватного начальника штаба война уже проиграна, остальное — лишь театральная постановка. Вам остаётся лишь признать поражение.

Ваша задача — написать программу для учёта и обработки всех событий, происходящих во время кампании.

Формат входных данных

В первой строке находятся два натуральных числа: N $(0 \leqslant N \leqslant 3 \cdot 10^6)$ — число событий, и M $(1 \leqslant M \leqslant 10^5)$ — необходимое количество фрагментов информации для склейки в единую карту объекта.
В следующих N строках указано краткое описание события по формату:

1. `+ c n` , где c $(0 \leqslant c \leqslant 10^{18})$ — цвет карты, а n $(0 \leqslant n \leqslant 10^9)$ — её номер;
2. `-`

Формат выходных данных

Контрольный Комитет хочет упростить жизнь своим бойцам. Выполните все команды и выведите балл стратегического перевеса на экран, либо *MISERABLE DEFEAT* в случае неадекватности командного состава.

Категоризация

Знание структуры данных stack (*стэк, LastInFirstOut queue*), аккуратность реализации.

Алгоритм решения

После реализации структуры данных stack (или при выборе её из стандартной библиотеки языка), оставалось лишь реализовать напрямую обе операции, описанные в условии задачи.

Желательно хранить в стэке не только цвет и номер карты, но и количество подряд идущих карт одного цвета для быстрого определения условия удаления верхних M карт одного цвета.

Возможные ошибки

- Попытки переизобрести структуру данных обычно приводили к мелким ошибкам, следовательно, к потери части баллов.
- Почти правильные решения получали вердикт `Time Limit` на последних тестах из-за постоянных операций обращения (*reverse*) массивов.
- Участники считали количество карт одного цвета во всём стэке, а не на его верхушке.

Рекомендуется научиться пользоваться сортировками не только чисел, но и произвольными структурами данных, в том числе объявленными вами. Это сильно упростит ваш код и понизит количество ошибок.

Авторское решение

```cpp
#include <iostream>
#include <cstdint>
#include <stack>

using namespace std;

struct map_data final
{
    uint64_t color, count, number;

    // Конструктор для удобства использования emplace из STL, необязателен
    map_data(const uint64_t Color, const uint64_t Count, const uint64_t Number)
        : color(Color),
          count(Count),
          number(Number)
    {
    }
};

int main()
{
    uint64_t n, m;
    int64_t score = 0;
    cin >> n >> m;
```

```
25
26      // Реализация структуры данных из стандартной библиотеки
27      stack<map_data> stack;
28
29      for (uint64_t i = 0; i < n; i++)
30      {
31          char event;
32          cin >> event;
33          switch (event)
34          {
35              case '+':
36              {
37                  uint64_t c, n;
38                  cin >> c >> n;
39                  // Пустой стэк?
40                  if (stack.empty())
41                  {
42                      // Просто добавим новую карту
43                      stack.emplace(c, 1, n);
44                  }
45                  else
46                  {
47                      // Проверим совпадение верхнего цвета
48                      auto& top = stack.top();
49                      if (top.color == c)
50                      {
51                          // Совпал, будем хранить число идущих подряд карт одного цвета
52                          // С учётом только что добавленной карты
53                          stack.emplace(c, top.count + 1, n);
54                      }
55                      else
56                      {
57                          // Просто добавим новую карту
58                          stack.emplace(c, 1, n);
59                      }
60                  }
61
62                  // Время выкинуть m верхних карт одного цвета?
63                  if (stack.top().count >= m)
64                  {
65                      while (!stack.empty() && stack.top().color == c)
66                      {
67                          // Увеличиваем баланс и выкидываем карту
68                          score += stack.top().number;
69                          stack.pop();
70                      }
71                  }
72                  break;
73              }
74              case '-':
75              {
76                  // Проверяем наличие карт в стэке.
77                  if (stack.empty())
78                  {
79                      cout << "MISERABLE DEFEAT";
80                      return 0;
81                  }
82                  // Карта есть, выкидываем её и уменьшаем баланс
83                  score -= stack.top().number;
84                  stack.pop();
85                  break;
86              }
87          }
```

```
    }

    cout << score;
}
```

Приказ №4

Самое главное в любой войне — обезопасить тылы. Этим вам и предстоит заняться. Необходимо очистить заводы от массы шпионов. Главная проблема в их обнаружении, ибо маскируются они под неопытных бухгалтеров и секретарш. К счастью, их легко опознать по штампованным фразам: «Я что-то нажала и всё сломалось», «Интернет не открывается» и прочей подобной чепухе. Иногда, стараясь казаться незаметными, они начинают путать монитор и процессор, называя всё это системным блоком. Ваша задача: используя разбросанные по городу подразделения Сил Технической Поддержки, ликвидировать угрозу. Вам известно количество ваших бойцов в каждом из районов города, их навыки, а также количество возможных диверсантов. Суммарная мощь отряда бойцов в районе — это навык каждого бойца в отряде в сумме по всему отряду. Ваша задача проста: подсчитайте, в скольких районах не хватит сил для обороны.

Формат входных данных

В первой строке единственное натуральное число N — количество районов на карте ($1 \leqslant N \leqslant 10^7$).
Далее идёт N строк, каждая из которых содержит 4 целых положительных числа s_{num} s_{skill} d_{num} d_{skill}, где:
s_{num} — количество специалистов техподдержки s_{skill} — навыки каждого специалиста
d_{num} — количество диверсантов d_{skill} — навыки каждого диверсанта

$$1 \leqslant s_{num}, d_{num} \leqslant 10^{18}, \quad 1 \leqslant s_{skill}, d_{skill} \leqslant 10^{18}$$

Формат выходных данных

Контрольный Комитет хочет видеть в единственной строке вывода набор чисел (возможно пустой): номера районов q_i ($1 \leqslant q_i \leqslant N$, нумерация в порядке ввода), которые будут потеряны.

Категоризация

Задача на реализацию, базовые знания программирования и умение найти самую простую задачу.

Алгоритм решения

Для каждой четвёрки чисел посчитать и сравнить произведения.

Возможные ошибки

- Неверные типы данных (гарантировалось отсутствие переполнений при использовании *правильного* типа данных). В терминах C: *int*, *integer*, *long* не подходят для хранения таких больших чисел.
- Участники останавливались на первой задаче и не обращали внимание на следующие задачи.
- Забытый вывод разделителя (пробела) сразу приводил к неправильным ответам.

Авторское решение

```cpp
#include <iostream>
#include <stdint.h>

using namespace std;

int main()
{
    uint32_t n;
    cin >> n;
    uint64_t a, b, c, d;
    for (uint32_t i = 0; i < n; i++)
    {
        cin >> a >> b >> c >> d;
        // Сравниваем произведения, выясняем кто сильнее
```

```
        if (a*b < c*d)
            cout << i+1 << " ";
    }
}
```

Приказ №5

Благодарим за сотрудничество, коммандер. Вы успешно провели контрразведку. Но война ещё не закончена. Хотя бригада «Arch» уже штурмует позиции врага, проклятые Висты не хотят умирать! Нам нужно новое оружие! И таким оружием могут стать штурм-черви. Штурм-черви обитают во влажной почве, поэтому достать их бывает проблематично. К несчастью, ситуация усложняется длительным отсутствием дождей.

Мы передаём вам карту поля с пометками об остаточной влажности в каждой клетке. За эти ценные данные мы благодарим отдел №41 Министерства дезинформации и разложения. В каждой клетке поля есть хотя бы один штурм-червь. Влажность в клетке поля — это сумма всех значений остаточной влажности клеток в радиусе r (радиус – сумма модулей разностей координат по вертикали и горизонтали). Чем меньше влажность, тем глубже забираются черви. Уровень невыносимой влажности для червей равен X. Если червям в клетке поля слишком влажно, они находятся на поверхности.

К сожалению, нам некогда пытаться выкопать червей, придётся довольствоваться лёгкой добычей. Контрольный Комитет хочет узнать количество клеток, с которых можно собирать супероружие голыми руками. Напишите программу для подсчёта, а мы пока приготовим троянского коня...

Формат входных данных

Три натуральных и одно вещественное число: n, m, r, X, где:
n — размер поля по вертикали ($1 \leqslant n \leqslant 10^5$) m — размер поля по горизонтали ($1 \leqslant m \leqslant 10^5$)
r — радиус расчета влажности ($1 \leqslant r \leqslant min(n, m)$) X — уровень невыносимой влажности ($0 < X \leqslant 10^{30}$)
Далее идёт n строк по m неотрицательных вещественных чисел — остаточная влажность с последнего дождя в соответствующей клетке поля (в стандартных релах).

$$n \cdot m \cdot (min(n, m))^2 \leqslant 10^5, \text{погрешностью вещественных чисел пренебречь}$$

Формат выходных данных

Неотрицательное целое число — количество клеток поля, на которых штурм-черви готовы для сбора без использования спецоборудования.

Категоризация

Умение работать с вещественными числами в выбранном языке программирования, двумерные массивы.

Алгоритм решения

Задача на реализацию: считать карту поля в двумерный массив, для каждой клетки (два вложенных цикла) посчитать влажность ещё двумя вложенными циклами и вывести количество клеток, влажность которых превышала данное число.

Возможные ошибки

- Отсутствие вещественных чисел в примерах вводило участников в заблуждение. По завершению олимпиады подгруппы были изменены, позволив решениям, которые работали в целых числах, набрать часть баллов.
- Выход за границы массивов при обходе клеток в заданном радиусе.
- Обход клеток только «вправо вниз», игнорирование клеток слева и сверху от текущей при подсчёте влажности.

Авторское решение

```
#include <iostream>
#include <vector>
#include <cstdint>

using namespace std;

```

```cpp
// Вычитание, никогда не возвращающее отрицательные числа
template <typename T>
auto clamp_left(const T& a, const T& b)
{
    return (a > b) ? (a - b) : 0;
}

// Модуль разницы между двумя числами
template <typename T>
T diff(const T& a, const T& b)
{
    return (a > b) ? (a - b) : (b - a);
}

int main()
{
    uint32_t n, m, r;
    float x;
    cin >> n >> m >> r >> x;

    // Двумерный массив (карта поля)
    vector<vector<float>> map(n);

    // Считываем карту поля
    for (uint32_t i = 0; i < n; i ++)
    {
        map[i].reserve(m);
        for (uint32_t j = 0; j < m; j++)
        {
            float tmp;
            cin >> tmp;
            map[i].push_back(tmp);
        }
    }

    uint64_t res = 0;
    for (uint32_t i = 0; i < n; i++)
    {
        for (uint32_t j = 0; j < m; j++)
        {
            // Для каждой клетки считаем влажность
            float wetness = 0.F;

            // Идём по прямоугольнику:
            // строки: [i-r .. i+r]
            // столбцы: [j-r .. j+r]
            for (auto k = clamp_left(i, r); k ⩽ i + r; k++)
            {
                for (auto h = clamp_left(j, r); h ⩽ j + r; h++)
                {
                    // Защищаемся от выхода за границы двумерного массива
                    if (k < 0 || k ⩾ n)
                        continue;
                    if (h < 0 || h ⩾ m)
                        continue;
                    // Проверяем расстояние (радиус)
                    // Функция расстояния Левенштейна (расстояние городских кварталов)
                    if (diff(k, i) + diff(h, j) > r)
                        continue;

                    // Да, эта клетка участвует в подсчёте влажности
                    // Прибавляем её к сумме влажности
                    wetness += map[k][h];
```

```
                }
            }

            // Слишком влажно? Увеличиваем счётчик.
            if (wetness > x)
                res++;
        }
    }

    cout << res;
}
```