

**Федеральное государственное автономное образовательное учреждение высшего
образования «Национальный исследовательский университет ИТМО»**
Факультет программной инженерии и компьютерной техники

Отчет по блоку №3 (timus)
По Алгоритмам и структурам данных

Выполнил:
Ступин Тимур Русланович
Группа № Р3108
Поток № 1.3
Преподаватель:
Тараканов Денис Сергеевич

Санкт-Петербург 2025

Содержание

Задача 1 Монобильярд	3
Задача 2 Белые полосы	5

Задача 1 Монобильярд

Решение

Используем стек, для моделирования состояния лузы. По мере доставания новых шаров ревизором, будем пытаться привести состояние лузы в то состояние, в котором она должна быть к моменту доставания данного шара, при условии что Чичиков не жульничал. Если этого сделать не удалось, то объявляем о жульничестве. Если после обработки всех шаров жульничество не было установлено, выводим сообщение о том что данных не достаточно.

В переменной `cur_max` будет храниться максимальный уже извлечённый шар. Это нужно для корректного обновления состояния лузы.

Обработка очередного шара будет происходить следующим образом. Верхний элемент стека сравнивается с полученным, и если новый шар имеет больший номер то это значит что с момента последнего извлечения Чичиков забил некоторое количество шаров и нужно добавить их в стек. Иначе же, значение вершины стека должно совпадать с извлекаемым шаром, и если это не так, то жульничество обнаружено. В конце итерации происходит обновление значения максимума а также удаление вершины стека, так как там лежит только что рассмотренное значение шара, а он как раз был извлечён.

Сложность алгоритма

- По времени: Работа со стеком занимает $O(1)$, операции выполняются для каждого полученного значения. Итоговая сложность $O(N)$.
- По памяти: В памяти хранится только стек и его размер не превышает N . Сложность $O(N)$

Итого:

- По времени: $O(N)$
- По памяти: $O(N)$

Kod

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main() {
5      int n;
6      cin >> n;
7      stack<int> s;
8      int cur_max = 0;
9      while (n--) {
10         int a;
11         cin >> a;
12         int max_v = s.empty() ? 0 : s.top();
13         if (a > max_v) {
14             for (int i = cur_max + 1; i <= a; i++) {
15                 s.push(i);
16             }
17         } else if (a != max_v) {
18             cout << "Cheater" << endl;
19             return 0;
20         }
21         cur_max = max(cur_max, a);
22         s.pop();
23     }
24     cout << "Not a proof" << endl;
25     return 0;
26 }
```

Задача 2 Белые полосы

Решение

Заметим, что горизонтальная белая полоса имеет длину больше 1 то она гарантированно не входит в какую либо вертикальную белую полосу. Аналогично и для вертикальных белых полос. Таким образом получаем, что особого рассмотрения требуют только белые полосы длинны 1. Причём в общем случае возможно только два исхода:

1. Белая полоса длины 1 входит в белую полосу большей длинны, расположенную перпендикулярно ей. Тогда её считать не нужно
2. Белая полоса длинны 1 является таковой в обоих направлениях. В таком случае её нужно посчитать, но важно сделать это только один раз (например только для горизонтального направления обхода).

Таким образом становится понятно, что для решения задачи нужно научиться по координатам полосы длинны 1 эффективно определять координаты перпендикулярной полосы в которую она включается.

Заведём два двумерных массива: для строк и для столбцов. В i -й ячейке каждого массива будут храниться координаты чёрных клеток, расположенных на данной строке/столбце. Также в этом векторе будут храниться фиктивные элементы, указывающие на элемент перед строкой/столбцом и элемент после него. Каждый из этих векторов будет отсортирован в порядке возрастания.

Так как каждый из векторов упорядочен, то можно используя функцию `upper_bound` за $O(\log N)$ определить первый элемент больший данного, а это именно то что нужно для решения проблемы с полосами единичной длинны. Фактически это позволит находить первый чёрный элемент после данного, а это как раз правая граница искомой полосы. Левая граница при этом ищется тривиально: это просто предыдущий элемент вектора.

В реализации в качестве двумерных массивов выступают векторы a и b , для строк и столбцов соответственно. Фиктивные элементы это -1 и m/n для строки/столбца соответственно.

сам обход представлен в виде одной функции `calc`, во избежании дублирования кода. Первый аргумент это двумерный массив по текущему основному направлению, второй по перпендикулярному. Третий же аргумент `is_one_one_add` служит своего рода флагом, чтобы в случае ситуации 2, описанной ранее, прибавлять или не прибавлять единичный элемент.

Сложность алгоритма

- По времени: С учётом того, что максимальное количество чёрных элементов равно k , получаем что верхняя оценка времени сортировки будет $O(k \log k)$. При этом оценка сложность основного цикла также $O(k \log k)$, так как суммарно будет рассмотрено не более k элементов, а для каждого из них может быть выполнена проверка `upper_bound` за $\log k$. Итого имеет $O(k \log k)$
- По памяти: В памяти суммарно хранится $2k$ элементов, что даёт сложность $O(k)$

Итого:

- По времени: $O(k \log k)$
- По памяти: $O(k)$

Код

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  void init_first(vector<vector<int>>& a) {
5      for (auto& c : a) {
6          c.push_back(-1);
7      }
8  }
9
10 void init_last_and_sort(vector<vector<int>>& a, const int n) {
11     for (auto& c : a) {
12         sort(c.begin(), c.end());
13         c.push_back(n);
14     }
15 }
16
17 int calc(vector<vector<int>>& a, vector<vector<int>>& b, const bool
↪ is_one_one_add) {
18     int sum = 0;
19     for (int i = 0; i < a.size(); i++) {
20         for (int j = 1; j < a[i].size(); j++) {
21             const int l = a[i][j - 1];
22             const int r = a[i][j];
23             if (r - l - 1 == 1) {
24                 auto r_ = upper_bound(b[l + 1].begin(), b[l + 1].end(), i);
25                 auto l_ = r_ - 1;
26                 if (*r_ - *l_ - 1 > 1 || *r_ - *l_ - 1 == 1 && !is_one_one_add) {
27                     continue;
28                 }
29             }
30             if (r - l - 1 != 0) {
31                 sum++;

```

```

32     }
33 }
34 }
35     return sum;
36 }
37
38 int main() {
39     int n, m, k;
40     cin >> n >> m >> k;
41     vector<vector<int>> a(n), b(m);
42
43     init_first(a);
44     init_first(b);
45
46     for (int i = 0; i < k; i++) {
47         int x, y;
48         cin >> x >> y;
49         a[x - 1].push_back(y - 1);
50         b[y - 1].push_back(x - 1);
51     }
52
53     init_last_and_sort(a, m);
54     init_last_and_sort(b, n);
55
56     const int sum = calc(a, b, true) + calc(b, a, false);
57     cout << sum << endl;
58
59     return 0;
60 }

```