Университет ИТМО

Факультет программной инженерии и компьютерной техники

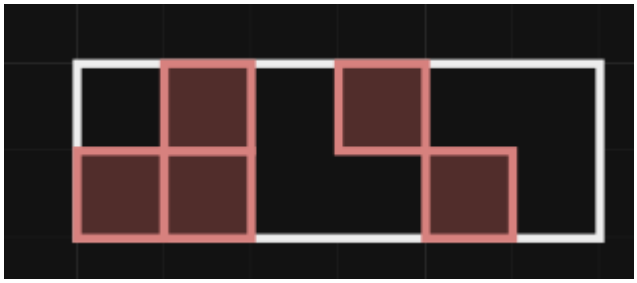
Лабораторная работа №3
по “Алгоритмам и структурам данных”
Тимус

Выполнил:
Студент группы P3206
Сорокин А.Н.
Преподаватели:
Косяков М.С.
Тараканов Д.С.

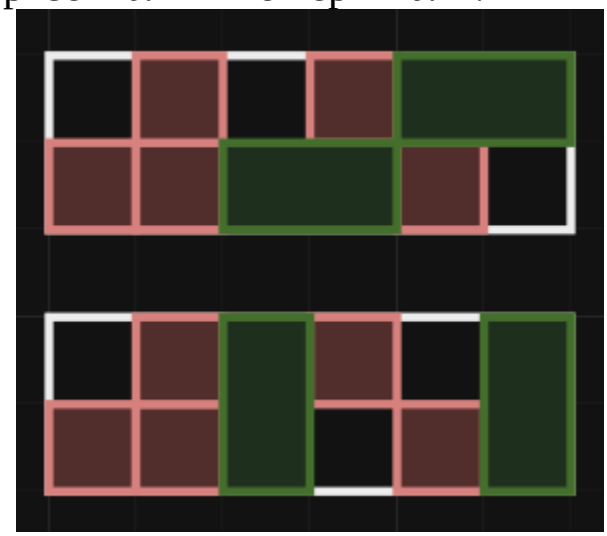
Санкт-Петербург
2024

Задача №5 “Белые полосы”
Решение за O(n log n).

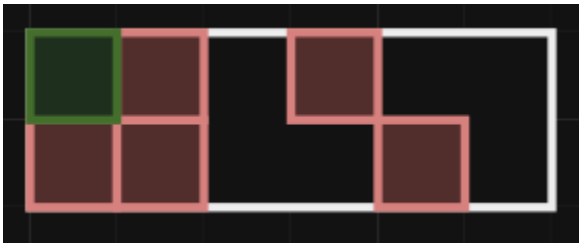
Дальнейшие объяснения буду производить с помощью примера ниже.
Пример входных данных:
6 2 5
1 2
2 1
2 2
4 1
5 2



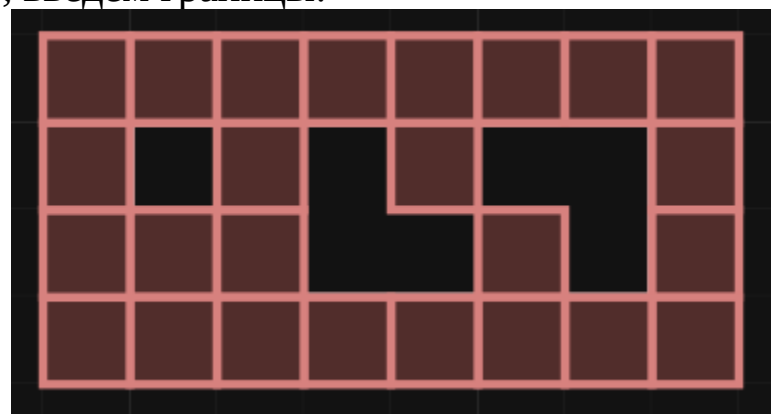
Красные клетки – это плохие дни.
Итак, требуется посчитать количество белых полос
Посчитаем их по горизонтали и по вертикали:



Обнаружим интересную вещь.
При вертикальном/горизонтальном поиске появляются 3 единичные точки. Эти точки могут потенциально являться белыми полосами.
Однако, белой полосой является только эта точка:



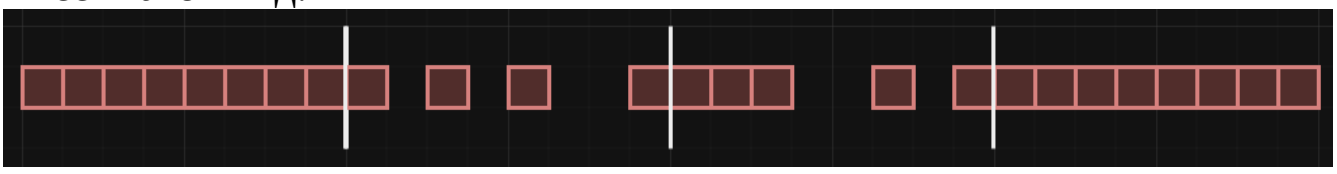
Дело в том, что 2 другие точки входят в белые полосы большей длины либо по вертикали, либо по горизонтали, а эта конкретная точка 1 1 является окруженной черными точками со всех сторон.
Чтобы проверять, что единичная точка является белой полосой необходимо удостоверяться в том, что точки слева, справа, сверху и снизу являются черными. Чтобы иметь возможность делать такие проверки везде, введем границы:



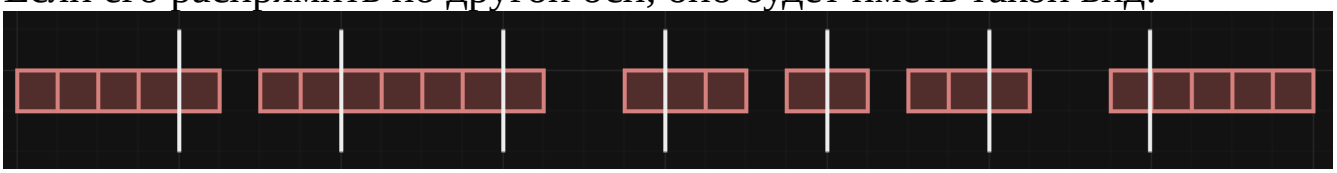
Изначальная идея создавать матрицу $m \times n$ и заполнять ее точками, очевидно, была очень дорогой как по памяти, так и по времени. Необходим более эффективный подход.

Пусть в структуре **vector<pair<int, int>>** будут храниться координаты черных дней. Дополнительно занесем в этот массив граничные точки.

Календарь представляет из себя m недель, каждая из которых имеет n дней. Чтобы массив представлял из себя одну линию длины mn, отсортируем его по значениям y . Теперь наше распрямленное поле имеет такой вид:



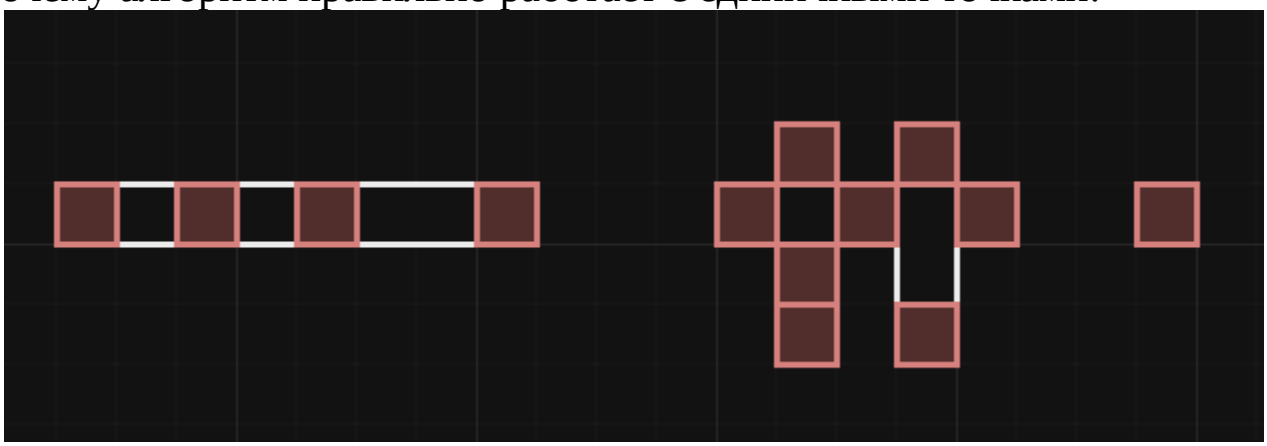
Если его распрямить по другой оси, оно будет иметь такой вид:



Алгоритм следующий:

1) Сортируем точки по значению y . Считаем расстояние между соседними точками. Если точки находятся на одном y и разница между их значениями x больше или равно 2, то это – белая полоса. Особый случай: если разница между значениями x равна 2, то между этими точками находится единичная точка, и, предположительно, белая полоса – ее необходимо сохранить в **set<pair<int, int>>**.
2) Сортируем точки по значению x и делаем то же самое с одним изменением: если найдена единичная точка проверяем, есть ли она в сете – если да, увеличиваем счетчик белых полос на 1. Поиск осуществляется функцией find(), которая в сете работает за log n.

Почему алгоритм правильно работает с единичными точками:



Левая часть картинки: мы сортировали точки по значению y и при поиске обнаружили 2 единичные точки: (2 1) и (4 1) – обе записали в сет.
Правая часть картинки: мы сортировали точки по значению x и обнаружили только одну единичную точку: (2 1) – вторая точка (4 1) обнаружена не была, поскольку при такой сортировке она не является единичной.

Справедливо и обратное: если при втором проходе мы обнаружили единичную точку, которой нет в сете, значит, при первом проходе было установлено, что эта точка является частью белой полосы большей длины, а, следовательно, единичной точкой не является.

Код решения:

```
#include <algorithm>
#include <iostream>
#include <set>
#include <utility>
#include <vector>
#define XY pair<int, int>
using namespace std;

int main() {
  int m, n, k;
  int res = 0;
  cin >> m >> n >> k;
  vector<pair<int, int>> blackDays;
  set<pair<int, int>> singularDay;
  for (int i = 0; i < k; ++i) {
    int x, y;
    cin >> x >> y;
    blackDays.push_back({x, y});
  }

  for (int i = 0; i <= m + 1; ++i) {
    blackDays.push_back({i, 0});
    blackDays.push_back({i, n + 1});
  }

  for (int j = 0; j <= n + 1; ++j) {
    blackDays.push_back({0, j});
    blackDays.push_back({m + 1, j});
  }

  sort(blackDays.begin(), blackDays.end(),
       [](const pair<int, int> a, const pair<int, int> b) {
         return (a.second == b.second) ? a.first < b.first
                                       : a.second < b.second;
       });
```

```
  for (int i = 1; i < blackDays.size() - 1; ++i) {
    int dif = blackDays[i + 1].first - blackDays[i].first;
    if (blackDays[i + 1].second == blackDays[i].second && dif >= 2) {
      if (dif == 2) {
        singularDay.insert({blackDays[i].first + 1, blackDays[i].second});
      } else {
        ++res;
      }
    }
  }

  sort(blackDays.begin(), blackDays.end(), [](const XY a, const XY b) {
    return (a.first == b.first) ? a.second < b.second : a.first < b.first;
  });

  for (int i = 1; i < blackDays.size() - 1; ++i) {
    int dif = blackDays[i + 1].second - blackDays[i].second;
    if (blackDays[i + 1].first == blackDays[i].first && dif >= 2) {
      if (dif == 2) {
        if (singularDay.find({blackDays[i].first, blackDays[i].second + 1}) !=
            singularDay.end()) {
          ++res;
        }
      } else {
        ++res;
      }
    }
  }
  cout << res;
  return 0;
}
```

Задача №6 “Миллиардеры”
Решение за O(n log n).

Необходимо сделать то, что требуется в задаче.

Когда миллиардер перелетает в другой город, требуется изменить как данные о его местоположении, так и капитал городов, задействованных в операции.

Нам необходим быстрый доступ ко всей необходимой информации, касаемо имени миллиардера, его финансов, его местоположения. Следовательно, воспользуемся структурой **unordered_map<string, pair<long, string>>** где ключ – имя миллиардера, а значение – информация о нем (финансы, местоположение).

Далее необходимо по названию города быстро определять его капитал. Для этого воспользуемся структурой **unordered_map<string, long>** где ключ – название города, значение – его капитал.

Очевидно, что проходиться по всей мапе городов и искать среди них город с максимальным капиталом – бесполезная задача. Необходимо иметь рейтинг городов по их капиталу, где на вершине будут находиться города с максимальным капиталом, т.е. структура должа иметь сортировку. Если несколько городов имеют один и тот же капитал, их порядок не важен. Воспользуемся структурой **map<long, unordered_set<string>>**, где ключ – капитал города, значение – набор городов с данным капиталом. *Заметим, что если самый большой капитал имеют как минимум 2 города одновременно, то ни один из них не является лидером.*

Далее понадобятся 2 структуры: **queue**, например, для отслеживания передвижений миллиардеров, и **map**, для сохранения лидирующих городов и количества дней.

Алгоритм достаточно простой. Первым циклом считываем миллиардеров и заполняем необходимую информацию. В конце работы

у нас будет список миллиардеров, список городов и топ городов по капиталу.

Далее вторым циклом заполняем очередь перемещений. Третьим циклом обрабатываем перемещения по порядку: поскольку перемещение происходит в конце дня мы запоминаем лидера среди городов (если есть), далее обрабатываем перелет. Изменяем всю необходимую информацию: сначала удаляем старый город из топа городов, затем уменьшаем его капитал, затем вставляем обратно в топ на другую позицию, затем удаляем новый город из топа, увеличиваем его капитал и вставляем обратно, затем изменяем местоположение миллиардера. Утром следующего дня проверяем топ городов.

Структура, используемая для хранения конкретно лидеров и количества дней лексикографически сравнивает строки, поэтому вывести лидеров можно просто пройдясь по мапе.

Код решения:

```
#include <iostream>
#include <map>
#include <queue>
#include <unordered_map>
#include <unordered_set>

using namespace std;

int main() {
  int n, m, k;
  cin >> n;
  string man, city;
  long long money;

  unordered_map<string, pair<long long, string>> manInfo;
  map<long long, unordered_set<string>> leaderboard;
  unordered_map<string, long long> cityMoney;
```

```
for (int i = 0; i < n; ++i) {
  cin >> man >> city >> money;
  manInfo[man] = {money, city};
  if (cityMoney.find(city) == cityMoney.end()) {
    cityMoney[city] = money;
    leaderboard[money].insert(city);
  } else {
    leaderboard[cityMoney[city]].erase(city);
    if (leaderboard[cityMoney[city]].empty())
      leaderboard.erase(cityMoney[city]);
    cityMoney[city] += money;
    leaderboard[cityMoney[city]].insert(city);
  }
}

cin >> m >> k;
queue<pair<int, pair<string, string>>> movement;
int day;
for (int i = 0; i < k; ++i) {
  cin >> day >> man >> city;
  movement.push({day, {man, city}});
}
day = 0;
int prev = 0, cur = 0;
map<string, long long> cityLeadCount;
for (int i = 0; i <= k; ++i) {
  prev = day;
  if (i == k) {
    day = m;
  } else {
    day = movement.front().first;
    man = movement.front().second.first;
    city = movement.front().second.second;
    movement.pop();
  }
  cur = day;
```

```
    if (cur != prev && leaderboard.rbegin()->second.size() == 1) {
      string cityToAdd = *(leaderboard.rbegin()->second.begin());
      cityLeadCount[cityToAdd] += cur - prev;
    }
    if (i < k) {
      string oldCity = manInfo[man].second;
      string newCity = city;
      money = manInfo[man].first;

      leaderboard[cityMoney[oldCity]].erase(oldCity);
      if (leaderboard[cityMoney[oldCity]].empty()) {
        leaderboard.erase(cityMoney[oldCity]);
      }
      cityMoney[oldCity] -= money;
      leaderboard[cityMoney[oldCity]].insert(oldCity);

      leaderboard[cityMoney[newCity]].erase(newCity);
      if (leaderboard[cityMoney[newCity]].empty()) {
        leaderboard.erase(cityMoney[newCity]);
      }
      cityMoney[newCity] += money;
      leaderboard[cityMoney[newCity]].insert(newCity);

      manInfo[man].second = newCity;
    }
  }
  for (const auto it : cityLeadCount) {
    cout << it.first << " " << it.second << "\n";
  }
  return 0;
}
```