Университет ИТМО

Факультет ПИиКТ

Алгоритмы и структуры данных

Лабораторная работа №3

Выполнила: Наумова Надежда Александровна

Группа P3201

Преподаватель: Тропченко Андрей Александрович

Санкт-Петербург

2020 г.

## **1067. Disk Tree**

## Постановка проблемы

Hacker Bill has accidentally lost all the information from his workstation's hard drive and he has no backup copies of its contents. He does not regret for the loss of the files themselves, but for the very nice and convenient directory structure that he had created and cherished during years of work.

Fortunately, Bill has several copies of directory listings from his hard drive. Using those listings he was able to recover full paths (like "WINNT\SYSTEM32\CERTSRV\CERTCO~1\X86") for some directories. He put all of them in a file by writing each path he has found on a separate line.

Your task is to write a program that will help Bill to restore his state of the art directory structure by providing nicely formatted directory tree.

### Исходные данные

The first line of the input contains single integer number N (1 ≤ N ≤ 500) that denotes a total number of distinct directory paths. Then N lines with directory paths follow. Each directory path occupies a single line and does not contain any spaces, including leading or trailing ones. No path exceeds 80 characters. Each path is listed once and consists of a number of directory names separated by a back slash ("**\**").

Each directory name consists of 1 to 8 uppercase letters, numbers, or the special characters from the following list: exclamation mark, number sign, dollar sign, percent sign, ampersand, apostrophe, opening and closing parenthesis, hyphen sign, commercial at, circumflex accent, underscore, grave accent, opening and closing curly bracket, and tilde ("**!#$%&'()-@^\_`{}~**").

### Результат

Write to the output the formatted directory tree. Each directory name shall be listed on its own line preceded by a number of spaces that indicate its depth in the directory hierarchy. The subdirectories shall be listed in lexicographic order immediately after their parent directories preceded by one more space than their parent directory. Top level directories shall have no spaces printed before their names and shall be listed in lexicographic order. See sample below for clarification of the output format.

## Описание решения

Для решения задачи представим данные в следующем формате:

* Массив, в каждой ячейке которого хранится map
* Map хранит информацию о дочерних каталогах в виде пары ключ - значение, где ключ - имя дочерней директории для текущего каталога, а значение - ссылка на описание дочернего каталога (такую же структуру).

### Визуализация примера:

WINNT\SYSTEM32\CONFIG

GAMES

WINNT\DRIVERS

HOME

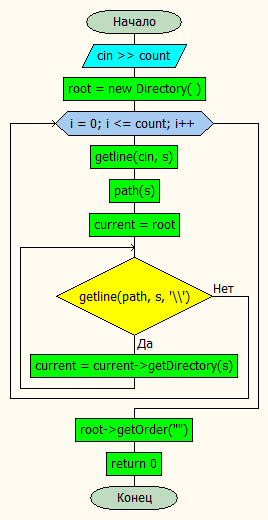
WIN\SOFT

GAMES\DRIVERS

WINNT\SYSTEM32\CERTSRV\CERTCO~1\X86

Сначала анализируем и разбираем таким образом вводимые данные, затем генерируем ответ.

## Схема решения



## Исходный код

#include <iostream>  
#include <bits/stdc++.h>  
  
using namespace std;  
  
class Directory;  
  
// создаем класс, представляющий один из уровней  
class Directory {  
public:  
 Directory() {}  
  
 //метод получения директории  
 Directory \*getDirectory(string const name) {  
 if (directory\_content.find(name) != directory\_content.end()) {  
 //нашел - возвращает найденный  
 return directory\_content[name];  
 } else {  
 // не нашел - создает  
 directory\_content[name] = new Directory();  
 return directory\_content[name];  
 }  
 }  
  
 void getOrder(string tab\_current) {  
 //увеличиваем кол-во пробелов при переходе на  
 // следующий уровень вложенности  
 string tab = " ";  
 tab += tab\_current;  
  
 //сортировка содержимого по алфавиту  
 map<string, Directory \*> sorted\_content(directory\_content.begin(), directory\_content.end());  
 //объявление итератора для прохода по словарю  
 map<string, Directory \*>::iterator it;  
  
 for (it = sorted\_content.begin(); it != sorted\_content.end(); it++) {  
 // выводим ключ - название данной директории  
 cout << tab\_current << it->first << endl;  
 //рекурсия к содержимому  
 it->second->getOrder(tab);  
 }  
 }  
  
private:  
 // словарь для хранения внутренних каталогов одной директории  
 map<string, Directory \*> directory\_content;  
  
};  
  
int main() {  
 //объявление переменной с кол-вом строк ввода  
 // и ее считывание  
 int count;  
 cin >> count;  
  
 // создаем корневую директорию  
 Directory \*root = new Directory();  
 // объявляем строку для считывания  
 // в нее одного из путей  
 string s;  
  
 for (int i = 0; i <= count; i++) {  
 // считывание одного пути  
 getline(cin, s);  
 stringstream path(s);  
 //объявляем текущую директорию корневой - для   
 // удобства дальнейшего разбиения  
 Directory \*current = root;  
  
 while (getline(path, s, '\\')) {  
 // пока строка разбивается, извлекаем из нее   
 // уже имеющиеся или новые директории  
 current = current->getDirectory(s);  
 }  
 }  
 // запускаем создание дерева ответа  
 root->getOrder("");  
  
 return 0;  
}

## Вывод:

в данной задаче я реализовала класс для более удобного решения задачи.

## **1494. Монобильярд**

## Постановка проблемы

Стол для монобильярда, установленный в игровом доме уездного города N, оказался очень прибыльным вложением. До того, как в городе появился небезызвестный господин Чичиков. Раз за разом он выигрывал, и хозяин, подсчитывая убытки, понимал, что дело тут нечисто. Однако уличить подлеца в жульничестве не удавалось до прибытия в город N ревизора из Петербурга.

Правила игры в монобильярд очень просты: нужно последовательно закатить в единственную лузу шары с номерами 1, 2, …, *N* (именно в этом порядке). Пока господин Чичиков играл, ревизор несколько раз подходил к столу и забирал из лузы последний закатившийся туда шар. В конце концов, оказалось, что Чичиков закатил в лузу все шары, а ревизор все шары достал и обследовал. Аферист утверждал, что закатил шары в правильном порядке. Хозяин понял, что это его шанс: ревизор должен помнить, в каком порядке он доставал шары. Однако так ли легко будет доказать жульничество?

### Исходные данные

В первой строке записано целое число *N* — количество бильярдных шаров (1 ≤ *N* ≤ 100000). В следующих *N* строках даны номера этих шаров в том порядке, в котором ревизор забирал их из лузы.

### Результат

Выведите слово «Cheater», если Чичиков не мог закатить все *N* шаров в правильном порядке. Иначе выведите «Not a proof».

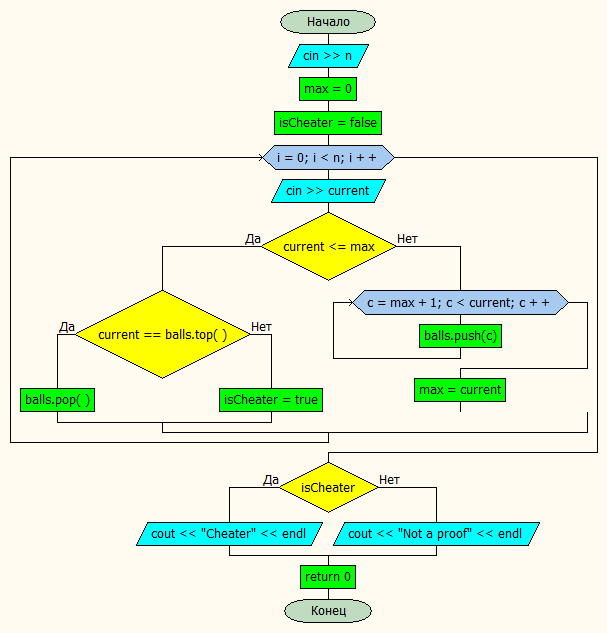
## Описание решения

Заведем стэк для моделирования ситуации. Каждый раз при считывании номера шара будем проверять, было ли нарушение порядка – если мы его зафиксировали, то возможны две ситуации:

* текущий вынутый шар совпал с верхним на стэке – удалим его
* текущий вынутый шар не совпал с верхним на стэке – Чичиков жульничал.

Если нарушения порядка зафиксировано не было, то заполняем стэк элементами, которые больше максимума, но меньше текущего элемента и переприсваиваем максимальный.

## Схема решения



## Исходный код

//подключение необходимых библиотек

#include <iostream>  
#include <bits/stdc++.h>

// использование пространства имен std  
using namespace std;  
  
int main () {  
 // объявление переменной и считывание  
 // в нее кол-ва шаров  
 int n;  
 cin >> n;  
 // объявление переменных для хранения номера  
 // текущего шара и максимального  
 int current;  
 int max = 0;  
 // стэк с номерами шаров  
 stack <int> balls;  
 // объявление логической переменной для хранения результата  
 bool isCheater = false;  
 for (int i = 0; i < n; i ++) {  
 // считываем номер шара  
 cin >> current;  
 //произошло нарушение порядка  
 if (current <= max) {  
 //если текущий совпал с верхним на стэке  
 if (current == balls.top())  
 // удаляем его  
 balls.pop();  
 else  
 // иначе - Чичиков жульничает  
 isCheater = true;  
 } else {  
 //заполняем стек элементами, которые больше имеющегося на данный момент

//максимума, но меньше текущего  
 for (int c = max + 1; c < current; c ++)  
 balls.push(c);  
 // переприсваивание максимального элемента  
 max = current;  
 }  
 }  
 // вывод ответа  
 if (isCheater) {  
 cout << "Cheater" << endl;  
 } else {  
 cout << "Not a proof" << endl;  
  
 }  
 return 0;  
}

## Вывод:

в данной задаче я потренировалась в использовании стэка для решения задачи.

## **1521. Военные учения 2**

## Постановка проблемы

### Вступление

В ходе недавних военных учений (более подробно эта история рассказана в задаче [«Военные учения»](https://acm.timus.ru/problem.aspx?space=1&num=1458)) министр обороны Советской Федерации товарищ Иванов имел возможность лично убедиться в блестящей боевой готовности солдат вверенной ему Советской Армии. Но одна вещь всё же продолжала беспокоить выдающегося военачальника. Прославленный генерал понимал, что была продемонстрирована лишь физическая подготовка солдат. Теперь настало время организовать очередные учения и проверить интеллектуальные способности личного состава.

Генерал Шульман, вновь назначенный ответственным за проведение учений, пожертвовал все выделенные деньги бедным и с чистой совестью лёг спать. Во сне генералу явился учебник по тактике и изложил схему, руководствуясь которой можно провести учения совершенно бесплатно.

### Задача

В соответствии с этой схемой учения делятся на **N** раундов, в течение которых **N** солдат, последовательно пронумерованных от 1 до **N**, маршируют друг за другом по кругу, т.е. первый следует за вторым, второй за третьим, ..., (**N**-1)-й за **N**-м, а **N**-й за первым. В каждом раунде очередной солдат выбывает из круга и идёт чистить унитазы, а оставшиеся продолжают маршировать. В очередном раунде выбывает солдат, марширующий на **K** позиций впереди выбывшего на предыдущем раунде. В первом раунде выбывает солдат с номером **K**.

Разумеется, г-н Шульман не питал никаких надежд на то, что солдаты в состоянии сами определить очерёдность выбывания из круга. «Эти неучи даже траву не могут ровно покрасить», – фыркнул он и отправился за помощью к прапорщику Шкурко.

### Исходные данные

Единственная строка содержит целые числа **N** (1 ≤ **N** ≤ 100000) и **K** (1 ≤ **K** ≤ **N**).

### Результат

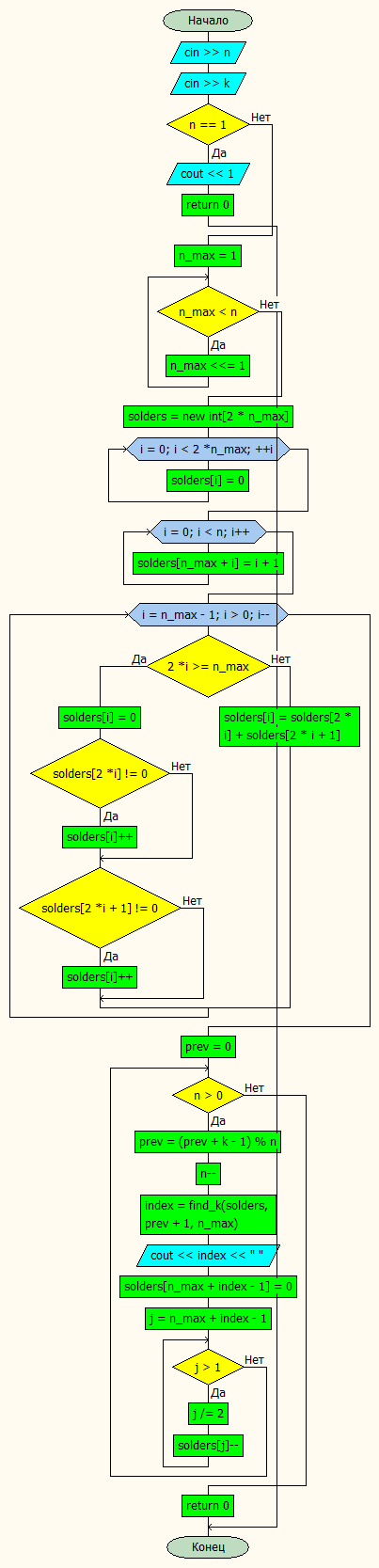
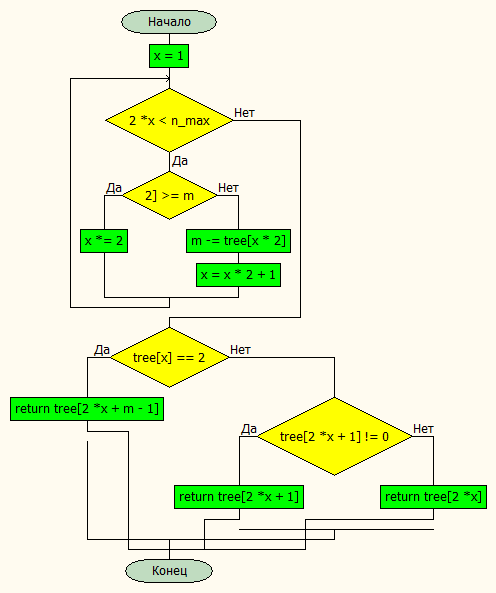
## Вывести через пробел номера солдат в порядке их выбывания из круга.

## Описание решения

Для решения задачи используем дерево отрезков (так как связный список совсем не проходит по времени).

С его помощью будем хранить, сколько солдат осталось на конкретном отрезке. Поиск и удаление в дереве происходит за O(log n), т.к. необходимо дойти до корня.

## Схема решения



## Исходный код

// импорт необходимых библиотек  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
// функция поиска к-ого солдата в дереве  
int find\_k(int \*tree, int m, int n\_max) {  
 int x = 1;  
 while (2 \* x < n\_max) {  
 if (tree[x \* 2] >= m) {  
 x \*= 2;  
 }  
 else {  
 m -= tree[x \* 2];  
 x = x \* 2 + 1;  
 }  
 }  
 if (tree[x] == 2)  
 return tree[2 \* x + m - 1];  
 else  
 if (tree[2 \* x + 1] != 0)  
 return tree[2 \* x + 1];  
 else  
 return tree[2 \* x];  
}  
  
int main() {  
 // объявление переменных и считывание кол-ва   
 // солдат и числа k

int n, k;  
 cin >> n;  
 cin >> k;  
 if (n == 1) {  
 cout << 1;  
 return 0;  
 }  
  
 // считаем количество листьев для полного дерева  
 // для n солдат  
 int n\_max = 1;  
 while (n\_max < n)  
 n\_max <<= 1;  
  
 // массив для хранения дерева отрезков  
 int\* solders = new int[2 \* n\_max];  
  
 // чистка массива  
 for (int i = 0; i < 2 \* n\_max; ++i) {  
 solders[i] = 0;  
 }  
  
 // запись всех солдатов по порядку в листья  
 for (int i = 0; i < n; i++)  
 solders[n\_max + i] = i + 1;  
  
 // заполнение внутренних узлов и корня значениями,  
 // равными количеству солдат в поддереве  
 for (int i = n\_max - 1; i > 0; i--) {  
 if (2 \* i >= n\_max) {  
 solders[i] = 0;  
  
 if (solders[2 \* i] != 0) {  
 solders[i]++;  
 }  
  
 if (solders[2 \* i + 1] != 0)  
 solders[i]++;  
 }  
 else  
 solders[i] = solders[2 \* i] + solders[2 \* i + 1];  
 }  
  
 int prev = 0;  
 while (n > 0) {  
 // вычисляем номер солдата, идущего перед тем,  
 // которого сейчас надо удалить  
 prev = (prev + k - 1) % n;  
  
 // уменьшаем кол-во солдат  
 n--;  
  
 // поиск номера солдата, которого сейчас надо удалять  
 int index = find\_k(solders, prev + 1, n\_max);  
 cout << index << " ";  
  
 // обнуляем значение листа с этим солдатом  
 solders[n\_max + index - 1] = 0;  
  
 // достаем индекс этого листа  
 int j = n\_max + index - 1;  
  
 // уменьшаем на 1 значения тех поддеревьев,  
 // где был этот лист  
 while (j > 1) {  
 j /= 2;  
 solders[j]--;  
 }  
 }  
 return 0;  
}

## Вывод:

в данной задаче я сделала реализацию дерева отрезков с помощью массива, поскольку связный список падал с TLE

## **1628. Белые полосы**

## Постановка проблемы

У каждого неудачника в жизни бывают не только чёрные, но и белые полосы. Марсианин Вась-Вась отмечает в календаре, представляющем собой таблицу *m* × *n*, те дни, когда ему ужасно не повезло. Если Вась-Васю не повезло в *j*-й день *i*-й недели, то он закрашивает ячейку таблицы (*i*, *j*) в чёрный цвет. Все незакрашенные ячейки в таблице имеют белый цвет.

Будем называть отрезками жизни прямоугольники размером 1 × *l* либо *l* × 1. Белыми полосами Вась-Вась считает все максимальные по включению белые отрезки таблицы. А сможете ли Вы определить, сколько всего белых полос было в жизни Вась-Вася?

### Исходные данные

Первая строка содержит целые числа *m*, *n*, *k* — размеры календаря и количество неудачных дней в жизни Вась-Вася (1 ≤ *m*, *n* ≤ 30000; 0 ≤ *k* ≤ 60000). В следующих *k* строках перечислены неудачные дни в виде пар (*xi*, *yi*), где *xi* — номер недели, к которой относится неудачный день, а *yi* — номер дня в этой неделе (1 ≤ *xi* ≤ *m*; 1 ≤ *yi* ≤ *n*). Описание каждого неудачного дня встречается только один раз.

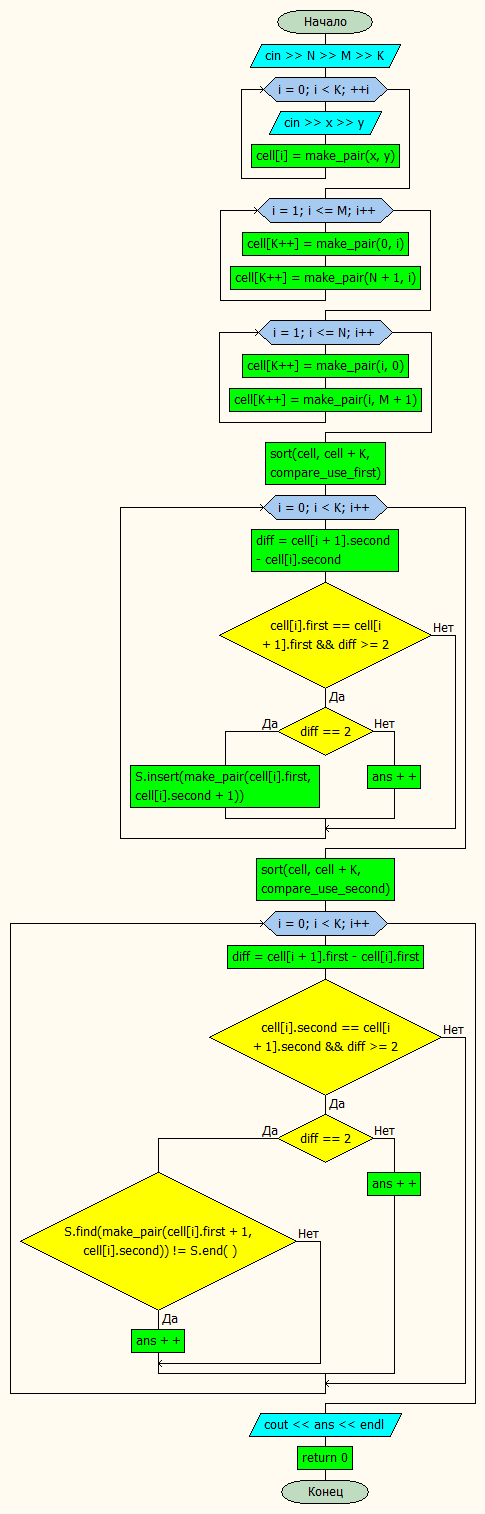
### Результат

Выведите число белых полос в жизни Вась-Вася.

## Описание решения

* Добавляем контур для исходной матрицы (чтобы при разворачивании матрицы строки и столбцы были разделены)
* Записываем все строки в один ряд (по строкам и столбцам отдельно (см. пример ниже))
* В разложении матрицы по строкам находим все разрывы, длина которых больше 1
* Все единичные разрывы складываем в Set.
* Аналогичные действия проводим с разложением по столбцам.
* При нахождении единичных разрывов учитываем их только в том случае, если они уже присутствуют в Set.

## Схема решения



## Исходный код

//подключение необходимых библиотек

#include <algorithm>

#include <set>

#include <iostream>

// использование пространства имен std

using namespace std;

// объявление необходимых переменных для хранения размеров календаря, количества

// неудачных дней и ответа

long N, M, K, ans;

// компараторы для сравнения

bool compare\_use\_first(pair<int, int> a, pair<int, int> b) {

if (a.first != b.first) return a.first<b.first;

return a.second<b.second;

}

bool compare\_use\_second(pair<int, int> a, pair<int, int> b) {

if (a.second != b.second) return a.second<b.second;

return a.first<b.first;

}

// массив пар для хранения календаря после разворачивания

pair<int, int> cell[180000];

// множество для хранения единичных разрывов

set<pair<int, int> > S;

int main() {

// считывание значений переменных для хранения размеров календаря, количества

// неудачных дней и ответа

cin >> N >> M >> K;

// объявление переменных для хранения

// недели, к которой относится неудачный день и номера дня в этой неделе

int x, y;

for (long i = 0; i < K; ++i) {

// считывание переменных для хранения недели, к которой относится неудачный

// день и номера дня в этой неделе и запись в массив пар

cin >> x >> y;

cell[i] = make\_pair(x, y);

}

// добавляем границы матрицы

for (long i = 1; i <= M; i++) {

// левая вертикальная граница

cell[K++] = make\_pair(0, i);

// правая вертикальная граница

cell[K++] = make\_pair(N + 1, i);

}

for (long i = 1; i <= N; i++) {

//верхняя горизонтальная граница

cell[K++] = make\_pair(i, 0);

// нижняя горизонтальная граница

cell[K++] = make\_pair(i, M + 1);

}

// разворотка с сортировкой по Х

sort(cell, cell + K, compare\_use\_first);

for (int i = 0; i < K ; i++) {

// длина полоски по координате X

int diff = cell[i + 1].second - cell[i].second;

// если в одном столбце и не соседние

if (cell[i].first == cell[i + 1].first && diff >= 2) {

// если нашлась одна белая клетка

if (diff == 2)

// кладем в множество эту белую клетку, т.к. i - номер идущей перед

// ней черной клетки

S.insert(make\_pair(cell[i].first, cell[i].second + 1));

else ans ++;

}

}

// разворотка с сортировкой по Y

sort(cell, cell + K, compare\_use\_second);

for (int i = 0; i < K; i++) {

// длина полоски по координате X

int diff = cell[i + 1].first - cell[i].first;

// если на одной строке и клетки не рядом

if (cell[i].second == cell[i + 1].second && diff >= 2) {

// одна белая клетка посередине

if (diff == 2) {

if (S.find(make\_pair(cell[i].first + 1, cell[i].second)) != S.end())

ans ++;

}

else ans ++;

}

}

// вывод ответа

cout << ans << endl;

return 0;

}

## Вывод:

в данной задаче я потренировалась в использовании таких структур данных, как множество и пара для решения задачи.

## **1650. Миллиардеры**

## Постановка проблемы

Возможно, вы знаете, что из всех городов мира больше всего миллиардеров живёт в Москве. Но, поскольку работа миллиардера подразумевает частые перемещения по всему свету, в определённые дни какой-то другой город может занимать первую строчку в таком рейтинге. Ваши приятели из ФСБ, ФБР, MI5 и Шин Бет скинули вам списки перемещений всех миллиардеров за последнее время. Ваш работодатель просит посчитать, сколько дней в течение этого периода каждый из городов мира был первым по общей сумме денег миллиардеров, находящихся в нём.

### Исходные данные

В первой строке записано число *n* — количество миллиардеров (1 ≤ *n* ≤ 10000). Каждая из следующих *n* строк содержит данные на определённого человека: его имя, название города, где он находился в первый день данного периода, и размер состояния. В следующей строке записаны два числа: *m* — количество дней, о которых есть данные (1 ≤ *m* ≤ 50000), *k* — количество зарегистрированных перемещений миллиардеров (0 ≤ *k* ≤ 50000). Следующие *k* строк содержат список перемещений в формате: номер дня (от 1 до *m* − 1), имя человека, название города назначения. Вы можете считать, что миллиардеры путешествуют не чаще одного раза в день, и что они отбывают поздно вечером и прибывают в город назначения рано утром следующего дня. Список упорядочен по возрастанию номера дня. Все имена и названия городов состоят не более чем из 20 латинских букв, регистр букв имеет значение. Состояния миллиардеров лежат в пределах от 1 до 100 миллиардов.

### Результат

В каждой строке должно содержаться название города и, через пробел, количество дней, в течение которых этот город лидировал по общему состоянию миллиардеров, находящихся в нём. Если таких дней не было, пропустите этот город. Города должны быть отсортированы по алфавиту (используйте обычный порядок символов: ABC...Zabc...z).

## Описание решения

Для решения задачи нам понадобится 2 структуры.

Отсортированное по возрастанию множество городов, в котором храним актуальное на данный день состояние городов. При каждом перелете миллиардера изменяем общую сумму денег в городе и обновляем информацию для этого множества.

Во второй структуре данных будем хранить данные об имени миллиардера и городе, в котором он сейчас находится – для этого используем map, где ключом будет имя миллиардера, а значением – специальная структура, содержащая количество денег, которые находятся в городе и количество дней, которое они там находятся.

Все миллиардеры и города хранятся в map для того, чтобы изменения какой-либо информации совершалось за логарифм.

# 

## Схема решения

## Исходный код

// импорт необходимых библиотек  
#include <iostream>  
#include <bits/stdc++.h>  
  
// использование пространства имен std  
using namespace std;  
  
// структура для хранения кол-ва денег и кол-ва дней,  
// в течение которых было столько денег  
struct City {  
 long long money;  
 int days;  
} city[10000 + 50000];  
  
// структура для описания человека - его денег и местонахождения  
struct Person {  
 long long money;  
 City \*location;  
} person[10000];  
  
// объявление множеств для миллиардеров, городов  
map<string, Person \*> human;  
map<string, City \*> world;  
// множество, для хранения актуальное отсортированное множество городов  
set<pair<long long, City \*>, greater<>> score;  
  
int main() {  
  
 // объявление переменной для хранения и считывание кол-ва миллиардеров  
 int n;  
 cin >> n;  
 // объявление переменной-счетчика  
 int counter = 0;  
 // считывание и заполнение начальными данными о человеке - имя, название города,  
 // где он находился в первый день данного периода, и размер состояния, кроме  
 // того, здесь же сразу записываем начальное состояние для всех заданных городов  
 for (int i = 0; i < n; ++i) {  
 string nameOfPerson, nameOfCity;  
 long long money;  
 cin >> nameOfPerson >> nameOfCity >> money;  
 if (world[nameOfCity] != nullptr)  
 world[nameOfCity]->money += money;  
 else {  
 city[counter].money = money;  
 world[nameOfCity] = &city[counter];  
 counter++;  
 }  
 person[i].money = money;  
 person[i].location = world[nameOfCity];  
 human[nameOfPerson] = &person[i];  
 }  
  
 // записываем актуальное состояние городов в множество  
 for (auto &item : world) {  
 score.insert({item.second->money, item.second});  
 }  
  
 // объявление переменных для хранения количества дней и количества  
 // зарегистрированных перемещений миллиардеров, счетчика дней, считывание  
 int m, k, today = 0;  
 cin >> m >> k;

// при каждом перелете изменяем общую сумму денег в городах, между которыми

// совершается перелет миллиардера, текущее положение миллиардера и

// обновляем актуальное состояние  
 for (int i = 0; i < k; ++i) {  
 int day;  
 string name\_p, name\_c;  
 cin >> day >> name\_p >> name\_c;  
 int count = day - today;  
 today = day;  
 auto it2 = score.begin();  
 auto it = it2++;  
 if (it2 == score.end() || it->first > it2->first) {  
 it->second->days += count;  
 }  
 City \*to\_city = world[name\_c];  
 Person \*which\_pers = human[name\_p];  
 if (to\_city == nullptr) {  
 world[name\_c] = &city[counter];  
 counter++;  
 to\_city = world[name\_c];  
 }  
 score.erase({which\_pers->location->money, which\_pers->location});  
 score.erase({to\_city->money, to\_city});  
 which\_pers->location->money -= which\_pers->money;  
 score.insert({which\_pers->location->money, which\_pers->location});  
 which\_pers->location = to\_city;  
 to\_city->money += human[name\_p]->money;  
 score.insert({to\_city->money, to\_city});  
 }  
  
 // производим подсчет дней  
 int count = m - today;  
 auto it2 = score.begin();  
 auto it = it2++;  
 if (it2 == score.end() || it->first > it2->first) {  
 it->second->days += count;  
 }  
  
 // вывод ответа  
 for (auto &item : world) {  
 if (item.second->days > 0)  
 cout << item.first << " " << item.second->days << "\n";  
 }  
 return 0;  
}

## Вывод:

в данной задаче я потренировалась в использовании комбинации собственных структур в сочетании с set, map для решения.