|  |
| --- |
| Университет итмо, кафедра вт |
| Лабораторная работа №3 по дисциплине «Алгоритмы и структуры данных» |
| «Структуры данных» |
| Группа Р3202 |
| **Выполнила: Орлова Кристина Александровна** |
| **Преподаватель: Косяков Михаил Сергеевич** |

|  |
| --- |
| *07.06.18* |

**№1494**

Решение данной задачи основано на использовании стека. Как только мы узнаем о первом вытащенном шаре, мы кладем элементы в стек в последовательности от 1 до номера вытащенного шара - 1, а номер вытащенного шара попадает в maxTrashBall (номер последнего максимального вытащенного шара). Как только мы узнаем о следующем вытащенном шаре, мы сравниваем его номер с номером из trash. Если он больше, то дозаполняем стек элементами от maxTrashBall + 1 до номера вытащенного шара - 1 и записываем этот шар в trash. Если к нам пришел шар с номером, меньшим, чем в trash, то сравниваем его с верхним элементом стека: если равен, то делаем pop; если не равен, то выдаем ответ "Сheater". В случае если мы дошли до самого конца без исключений, выдаем ответ "Not a proof".

|  |
| --- |
| #include <iostream> |
|  | #include <stack> |
|  |  |
|  | using namespace std; |
|  |  |
|  | int n; |
|  |  |
|  | void print(int answer) { |
|  | switch (answer) { |
|  | case 0 : |
|  | cout << "Not a proof"; |
|  | break; |
|  | case 1 : |
|  | cout << "Cheater"; |
|  | break; |
|  | default: |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | void getAnswer(stack <int> balls, const int \*array) { |
|  | int answer = 0; |
|  | int maxTrashBall = 0; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | int newBall = array[i]; |
|  |  |
|  | if (newBall > maxTrashBall) { |
|  | for (int ball = maxTrashBall + 1; ball < newBall; ball++) { |
|  | balls.push(ball); |
|  | } |
|  |  |
|  | maxTrashBall = newBall; |
|  | } else { |
|  | if (newBall == balls.top()) { |
|  | balls.pop(); |
|  | } else { |
|  | answer = 1; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | print(answer); |
|  | } |
|  |  |
|  | int main() { |
|  | cin >> n; |
|  | int array[n]; |
|  | stack <int> balls; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | cin >> array[i]; |
|  | } |
|  |  |
|  | getAnswer(balls, array); |
|  |  |
|  | return 0; |
|  | } |

**№1067**

Для решения данной задачи использовалась структура "дерево". Для каждой новой директории сначала совершался поиск директорий на пути к ней. Если их не было, то создавались новые вершины; затем создавалась новая директория. Затем все дети всех вершин были отсортированы, а затем выведены по порядку для каждой вершины с учетом глубины.

|  |
| --- |
| #include <iostream> |
|  | #include <regex> |
|  |  |
|  | using namespace std; |
|  |  |
|  | typedef struct node { |
|  | string name; |
|  | vector <node> children; |
|  | } node; |
|  |  |
|  | int n; |
|  |  |
|  | node \* findPlace(node \*link, const string &name) { |
|  | for (int i = 0; i < link->children.size(); i++) { |
|  | if (link->children[i].name == name) { |
|  | return &link->children[i]; |
|  | } |
|  | } |
|  |  |
|  | node directory; |
|  | directory.name = name; |
|  | link->children.push\_back(directory); |
|  |  |
|  | return &link->children.back(); |
|  | } |
|  |  |
|  | void buildTree(string \*array, node \*root) { |
|  | regex regex("[^\\\\]+"); |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | node \*link = root; |
|  |  |
|  | for (sregex\_token\_iterator it(begin(array[i]), end(array[i]), regex), last; it != last; ++it) { |
|  | string name = it->str(); |
|  | link = findPlace(link, name); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void sortTree(node \*root) { |
|  | node \*link = root; |
|  |  |
|  | if (!link->children.empty()) { |
|  | for (int i = 0; i < link->children.size(); i++) { |
|  | for (int j = i + 1; j < link->children.size(); j++) { |
|  | if (strcmp(link->children[j].name.c\_str(), link->children[i].name.c\_str()) < 0) { |
|  | node timing = link->children[i]; |
|  |  |
|  | link->children[i] = link->children[j]; |
|  | link->children[j] = timing; |
|  | } |
|  | } |
|  |  |
|  | node \*next = &link->children[i]; |
|  | sortTree(next); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | string getSpaces(int depth) { |
|  | string spaces; |
|  |  |
|  | for (int i = 1; i <= depth; i++) { |
|  | spaces += " "; |
|  | } |
|  |  |
|  | return spaces; |
|  | } |
|  |  |
|  | void printTree(node \*root, int depth) { |
|  | node \*link = root; |
|  |  |
|  | for (int i = 0; i < link->children.size(); i++) { |
|  | cout << getSpaces(depth) << link->children[i].name << endl; |
|  |  |
|  | node \*next = &link->children[i]; |
|  | depth++; |
|  | printTree(next, depth); |
|  | depth--; |
|  | } |
|  | } |
|  |  |
|  | void getAnswer(string \*array) { |
|  | node root; |
|  | root.name = "ROOT"; |
|  |  |
|  | buildTree(array, &root); |
|  | sortTree(&root); |
|  | printTree(&root, 0); |
|  | } |
|  |  |
|  | int main() { |
|  | //n = 7; |
|  | cin >> n; |
|  | string array[n]; //{"WINNT\\SYSTEM32\\CONFIG", "GAMES", "WINNT\\DRIVERS", "HOME", "WIN\\SOFT", "GAMES\\DRIVERS", |
|  | //"WINNT\\SYSTEM32\\CERTSRV\\CERTCO~1\\X86"}; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | cin >> array[i]; |
|  | } |
|  |  |
|  | getAnswer(array); |
|  |  |
|  | return 0; |
|  | } |

**№1628**

Для решения данной задачи использовались две карты map<int, vector<int>> rows и map<int, vector<int>> columns. Сначала исходные данные записываются в карту строк: если строки нет, то она создается; для имеющейся строки добавляются только столбцы в соответствующий для нее вектор. Затем данные в векторах сортируются. После чего идет проверка на размер календаря: если в нем всего 1 столбец, значит проверку по строкам осуществлять не будем. Если в нем всего 1 строка, значит, не будем осуществлять проверку по столбцам. Если это клетка размером 1 х 1 и карта строк пуста, значит, эта единственная клетка - белая - инкриментируем переменную и выходим. Если в календаре больше 1 строки и столбца, то делаем проверку и для той карты, и для другой.  
Проверка состоит в поиске расстояний:  
1) между закрашенными клетками, если данная клетка в векторе не одна  
2) от данной клетки для левой границы, если она первая  
3) от данной клетки для правой границы, если она последняя.  
  
Если расстояние больше 1, то инкриментируем переменную, отвечающую за кол-во полос. Если расстояние 1, значит, у нас есть свободная клетка, которая зажата, как минимум, с двух сторон (право/лево или верх/низ). Для такой клетки нужно проверить еще, ограничена ли она с двух других сторон. Если да, то инкриментируем переменную, если нет, то нет.  
Затем трансформируем карту строк в карту столбцов и проделываем для нее все то же самое, кроме проверки ограниченности клетки размером в 1 в том случае, если мы уже сделали это для строк.  
  
Также необходимо засчитать все пустые строки/столбцы в начале, конце и между другими строками за отдельные полосы.

|  |
| --- |
| #include <iostream> |
|  | #include <vector> |
|  | #include <map> |
|  | #include <algorithm> |
|  |  |
|  | using namespace std; |
|  |  |
|  | int countOfBands = 0; |
|  |  |
|  | void incrementCountOfBands(int distance, map<int, vector<int>> :: iterator it, int index, map<int, vector<int>> &array, |
|  | bool key, int range) { |
|  | if (distance > 1) { |
|  | countOfBands++; |
|  | } else if (distance == 1 && key) { |
|  | if ((it == array.begin() && it->first != 0) || (next(it) == array.end() && it->first != range)) { |
|  | return; |
|  | } |
|  |  |
|  | if (it != array.begin()) { |
|  | int current = it->first; |
|  | it--; |
|  |  |
|  | if (it->first != current - 1) { |
|  | return; |
|  | } else { |
|  | auto it\_deep = lower\_bound(it->second.begin(), it->second.end(), index); |
|  |  |
|  | if (it->second[it\_deep - it->second.begin()] != index) { |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | it++; |
|  | } |
|  |  |
|  | if (next(it) != array.end()--) { |
|  | int current = it->first; |
|  | it++; |
|  |  |
|  | if (it->first != current + 1) { |
|  | return; |
|  | } else { |
|  | auto it\_deep = lower\_bound(it->second.begin(), it->second.end(), index); |
|  |  |
|  | if (it->second[it\_deep - it->second.begin()] != index) { |
|  | return; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | countOfBands++; |
|  | } |
|  | } |
|  |  |
|  | void findCountOfBands(map<int, vector<int>> array, int range, int maxExpectedKey, bool key) { |
|  | int expectedKey = 0; |
|  | auto it = array.begin(); |
|  |  |
|  | while (it != array.end()) { |
|  | vector<int> collection = it->second; |
|  |  |
|  | // solve problem with several empty elements before or between |
|  | if (it->first != expectedKey) { |
|  | countOfBands += it->first - expectedKey; |
|  | expectedKey = it->first; |
|  | } |
|  |  |
|  | for (int i = 0; i < collection.size(); i++) { |
|  | int distance; |
|  |  |
|  | if (i + 1 < collection.size()) { |
|  | distance = collection[i + 1] - collection[i] - 1; |
|  | incrementCountOfBands(distance, it, collection[i] + 1, array, key, maxExpectedKey); |
|  | } |
|  |  |
|  | if (i == 0) { |
|  | distance = collection[i]; |
|  | incrementCountOfBands(distance, it, collection[i] - 1, array, key, maxExpectedKey); |
|  | } |
|  |  |
|  | if (i == collection.size() - 1) { |
|  | distance = range - collection[i]; |
|  | incrementCountOfBands(distance, it, collection[i] + 1, array, key, maxExpectedKey); |
|  | } |
|  | } |
|  |  |
|  | it++; |
|  | expectedKey++; |
|  | } |
|  |  |
|  | // solve problem with several empty elements after all |
|  | if (expectedKey <= maxExpectedKey) { |
|  | countOfBands += maxExpectedKey - expectedKey + 1; |
|  | } |
|  | } |
|  |  |
|  | void printAnswer() { |
|  | printf("%d", countOfBands); |
|  | } |
|  |  |
|  | void transformIntoColumns(map<int, vector<int>> rows, map<int, vector<int>> &columns) { |
|  | auto it\_rows = rows.begin(); |
|  |  |
|  | while (it\_rows != rows.end()) { |
|  | vector<int> collection = it\_rows->second; |
|  |  |
|  | for (int column : collection) { |
|  | auto it = columns.find(column); |
|  |  |
|  | if (it == columns.end()) { |
|  | vector<int> array; |
|  |  |
|  | array.push\_back(it\_rows->first); |
|  | columns[column] = array; |
|  | } else { |
|  | columns[column].push\_back(it\_rows->first); |
|  | } |
|  | } |
|  |  |
|  | it\_rows++; |
|  | } |
|  | } |
|  |  |
|  | void sort(map<int, vector<int>> &array) { |
|  | auto it = array.begin(); |
|  |  |
|  | while (it != array.end()) { |
|  | sort(it->second.begin(), it->second.end()); |
|  | it++; |
|  | } |
|  | } |
|  |  |
|  | void getAnswer(map<int, vector<int>> rows, int rowRange, int columnRange) { |
|  | map<int, vector<int>> columns; |
|  |  |
|  | if (columnRange != 0) { |
|  | sort(rows); |
|  | findCountOfBands(rows, columnRange, rowRange, true); |
|  | } |
|  |  |
|  | if (rowRange != 0) { |
|  | transformIntoColumns(rows, columns); |
|  | findCountOfBands(columns, rowRange, columnRange, columnRange == 0); |
|  | } |
|  |  |
|  | if (columnRange == 0 && rowRange == 0 && rows.empty()) { |
|  | countOfBands = 1; |
|  | } |
|  |  |
|  | printAnswer(); |
|  | } |
|  |  |
|  | int main() { |
|  | int n; |
|  | int rowRange; |
|  | int columnRange; |
|  | map<int, vector<int>> rows; |
|  |  |
|  | scanf("%d %d %d", &rowRange, &columnRange, &n); |
|  | rowRange--; |
|  | columnRange--; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | int x; |
|  | int y; |
|  |  |
|  | scanf("%d %d", &x, &y); |
|  | x--; |
|  | y--; |
|  |  |
|  | auto it = rows.find(x); |
|  |  |
|  | if (it == rows.end()) { |
|  | vector<int> array; |
|  |  |
|  | array.push\_back(y); |
|  | rows[x] = array; |
|  | } else { |
|  | rows[x].push\_back(y); |
|  | } |
|  | } |
|  |  |
|  | getAnswer(rows, rowRange, columnRange); |
|  |  |
|  | return 0; |
|  | } |