Университет ИТМО
Кафедра ВТ

Задачи 3 (1 - 5)
Алгоритмы и Структуры Данных

Выполнил: Федоров Сергей
Группа: P3212

Санкт-Петербург
2020 г.

- # Задача 1 - Disk Tree - 1067

У нас задача - построить дерево. Хм, хм, хм 🧐. Давайте построим дерево.

O(n * log(n))

Код:

```
//
// Created by Sergey Fedorov on 07/05/2020.
//
#include <iostream>
#include <algorithm>
#include <map>
#include <cstring>

using namespace::std;

template<typename T>
class Tree {
public:
        T value;
        map<T, Tree<T>> children;

        explicit Tree(T single_value){
            this->value = single_value;
            this->children = {};
        }

        Tree() = default;

        void add_sub_tree(Tree<T> tree){
            children.insert({tree.value, tree});
        }

        void print_tree(int offset = 0){
            if (!value.empty()) cout << string(offset++, ' ') + value << endl;
            for(pair<T, Tree<T>> child : children){
                child.second.print_tree(offset);
            }
        }
};

int main(){

    int n;
    cin >> n;

    Tree<string> root = Tree<string>();

    const char delim[] = "\\";

    for(int i = 0; i < n; ++i) {
        string next_path;
        cin >> next_path;
        char to_split[next_path.size()];
        strcpy(to_split, next_path.c_str());
        char* split = strtok(to_split, delim);

        Tree<string>* workTree = &root;
```

```
        while(split != nullptr){
            auto f = workTree->children.find(string(split));
            if (f == workTree->children.end()){
                Tree<string> new_tree = Tree<string>(string(split));
                workTree->add_sub_tree(new_tree);
                workTree = &workTree->children.find(string(split))->second;
            } else {
                workTree = &f->second;
            }
            split = strtok(nullptr, delim);
        }
    }

    root.print_tree(0);
}
```

- **Задача 2 - Монобильярд - 1494**

Аналогично первой задаче, попытаемся смоделировать ситуацию забивания шаров в лузу. Шары будем хранить в stack'е. Каждый раз читая следующий шар с ввода, сравниваем его с максимальным шаров о котором мы знаем:

- Если новый шар больше, чем максимальный, то все хорошо, потому что порядок сохраняется. Но для проверки следующих шаров, положим в стек все шары в прорежутке **(max_ball, current_ball)**.
- Если новый шар меньше чем максимальный, то в таком случае Ревизор взял шар перед тем как Чичиков забил еще один (следующий за максимальным шар). Тогда такой шар должен быть меньше на 1 чем предыдущий. А следующий за ним еще на 1, и тд. Если в какой-то момент это не так, то вердикт - Mr. Chichikov = "Cheater"

Почему stack: могли бы просто использовать число следующего маленького шара, но тогда мы не сможем рассматривать ситуацию, когда после меньшего шара, Чичиков забивает больший и мы достаем снова максимальный. Для решения нам нужно хранить историю меньших шаров -> используем stack.

O(n)

Код:

```
//
// Created by Sergey Fedorov on 07/05/2020.
//
#include <iostream>
#include <stack>

using namespace::std;

int main() {
    int n;
    cin >> n;
    stack<int> balls;
    int max_ball = -1;

    for (int i = 0; i < n; i++) {
        int current;
        cin >> current;

        if (current > max_ball) {
            for (int j = max_ball + 1; j < current; j++) {
```

```
                balls.push(j); // Так как ревизор, в конце концов, достал все шары, то
имеем заполнить стек one-by-one
            }
            max_ball = current;
        } else {
            if (current == balls.top()) {
                balls.pop();
            } else { // Так как в начале был максимальный шар, то затем должны быть
только шары меньше и в установленном порядке
                cout << "Cheater" << endl;
                return 0;
            }
        }
    }

    cout << "Not a proof" << endl;
    return 0;
}
```

- ## Задача 3 - Военные учения 2 - 1521

Можем использовать **дерево отрезков** для данной задачи, но так же можем использовать **бинароное индексированное дерево (дерево Фенвика)**, так как нам нужно уметь обновлять элемента, а наша функция - суммирование.
Почему так? Могли бы просто каждый раз уменьшать шаг на 1 (потому что стало меньше людей), но мы в таком случае мы не уверены на каком отрезке у нас ушли люди.

O(n * log(n))

Код:

```
//
// Created by Sergey Fedorov on 07/05/2020.
//

#include <vector>
#include <iostream>

using namespace std;

class BinaryIndexedTree{
private:
    int max_n;
    int *tree;

public:
    BinaryIndexedTree(){
        max_n = 1 << 17;
        tree = (int*) calloc(max_n * 4, sizeof(int));
    }

    void inline sum_update(int x, int amount) {
        for (x += max_n; x > 0; x /= 2)
            tree[x] += amount; // Øynene mine blør på grunn av den koden
    }

    int inline find_k_th_soldier(int start_pos, int k_th) {
```

```
        while (start_pos < max_n){
            start_pos *= 2;
            if (k_th > tree[start_pos]) {
                k_th -= tree[start_pos];
                start_pos++;
            }
        }
        return start_pos - max_n;
    }
};

int main() {
    int n, k;
    std::cin >> n >> k;

    BinaryIndexedTree tree = BinaryIndexedTree();

    // Fill tree with soldiers
    // Ugh, bruh.
    for (int i = 0; i < n; ++i) {
        tree.sum_update(i, 1);
    }

    int curr_pos = k - 1;
    for (int i = 0; i < n; ++i) {
        // Find position of next soldier
        int pos = tree.find_k_th_soldier(1, curr_pos + 1);
        cout << pos + 1 << " ";

        // Reduce distance sum
        tree.sum_update(pos, -1);

        int remaining = n - i - 1;
        int next_pos = curr_pos - 1 + k;
        if (i < n - 1) curr_pos = next_pos % remaining;
    }
}
```

- ## Задача 4 - Белые полосы - 1628

Нам нужно посчитать белые полосы. Мы можем выделить полосы на три категории:
1. Ширина 1, длинна L
2. Ширина L, длина 1
3. В силу того как будем их искать, нам важно направление полосы, поэтому полосы с шириной 1, длинной 1, выделим в отдельную категорию. Нету направления, считаем отдельно.

Искать горизонтальные и вертикальные полосы, можно было бы используя map с ключом по дневной и недельной, соответсвенно. Но это не обязательно, а нашем случаем можем просто отсортировать по этим координатам и потом просто пройтись по всем черным точкам.
Также стоит сделать ограничения по краям календаря в виде тех же самый черных точек (для подсчета длин, высот полос).
При нахождении квадратов, скидываем их в отдельный вектор, затем посчитаем их количество.

Все черные точки и квадраты храним в линейной структуре, например в векторе.

O(k * log(k))

Код:

```
//
// Created by Sergey Fedorov on 07/05/2020.
//

#include <iostream>
#include <algorithm>
#include <vector>

using namespace::std;

bool comp_weeks(pair<int, int> a, pair<int, int> b) {
    if (a.first != b.first) {
        return a.first < b.first;
    } else {
        return a.second < b.second;
    }
}

bool comp_days(pair<int, int> a, pair<int, int> b) {
    if (a.second != b.second) {
        return a.second < b.second;
    } else {
        return a.first < b.first;
    }
}

int main() {
    int m, n, k, result = 0;
    cin >> m >> n >> k;

    if (m == 0 || n == 0) {
        return 0;
    }

    vector<pair<int, int>> points;
    vector<pair<int, int>> squares;

    // Given dots

    for (int i = 0; i < k; ++i) {
        pair<int, int> p;
        cin >> p.first >> p.second;
        p.first--;
        p.second--;
        points.push_back(p);
    }

    // Vertical and horizontal borders

    for (int i = 0; i < m; ++i) {
        pair<int, int> p = {i, -1};
        pair<int, int> p2 = {i, n};
        points.push_back(p2);
        points.push_back(p);
    }
```

```
    for (int i = 0; i < n; ++i) {
        pair<int, int> p = {-1, i};
        pair<int, int> p2 = {m, i};
        points.push_back(p2);
        points.push_back(p);
    }

    // Counting long boxes

    // Horizontal
    sort(points.begin(), points.end(), comp_days);
    for (int i = 0; i < points.size() - 1; ++i) {
        if (points[i].second == points[i + 1].second) {
            int length = points[i + 1].first - points[i].first - 1;
            if (length == 1){
                squares.push_back((pair<int, int>) {points[i].first + 1,
points[i].second});
            } else if (length > 1) result++;

        }
    }

    // Vertical
    sort(points.begin(), points.end(), comp_weeks);
    for (int i = 0; i < points.size() - 1; ++i) {
        if (points[i].first == points[i + 1].first) {
            int height = points[i + 1].second - points[i].second - 1;
            if (height == 1){
                squares.push_back((pair<int, int>) {points[i].first, points[i].second +
1});
            } else if (height > 1) result++;
        }
    }

    // Counting short boxes (just boxes)

    sort(squares.begin(), squares.end(), comp_days);
    sort(squares.begin(), squares.end(), comp_weeks);

    switch (squares.size()) {
        case 1: result++; break;
        case 0: break;
        default:
            for (int i = 0; i < squares.size() - 1; i++) {
                if (squares[i] == squares[i + 1]) {
                    result++;
                    i++;
                }
            }
    }

    cout << result << endl;
}
```

- **Задача 5 - Миллиардеры - 1650**

Смоделируем ситуацию (по старой традиции):

- Отсортированное по возрастанию множество городов (**set<A, greater<>>**). Храним актуальное состояние города на данный момент. При перелетах меняем суммы. Каждый день записываем +1 к очкам первому городу в множестве.

- Данные об имени миллиардера и городе, в котором он сейчас находится храним в **map**.

- Данные об имени миллиардеров и их самих тоже храним в **map**,

Таким образом изменение информации о городах или о миллиардерах будет выполнятся за логарифм.

(Так же храним все города и людей в массиве, чтобы было куда ссылаться, и чтобы память в лишний раз не расходовать)

O(k * log(k))

Код:

```
//
// Created by Sergey Fedorov on 07/05/2020.
//

#include <iostream>
#include <functional>
#include <set>
#include <map>

using namespace std;

typedef struct {
    string name;
    long long money;
    int days;
} City;

typedef struct {
    long long money;
    City *location;
} SuccessfulPerson;


int main() {

    int n;
    cin >> n;

    SuccessfulPerson person[n];
    City city[n + 50000]; // Can use dynamic coll, but I'm a bit tired already.

    set<pair<long long, City *>, greater<>> ranking;
    map<string, SuccessfulPerson *> names_to_billionaires;
    map<string, City *> names_to_city;

    // Reading billionaires

    int city_index = 0;
    for (int i = 0; i < n; i++) {
        string person_name;
```

```
    string city_name;
    long long money;
    cin >> person_name >> city_name >> money;

    if (!names_to_city[city_name]) {
        city[city_index].name = city_name;
        city[city_index].money = money;
        names_to_city[city_name] = &city[city_index];
        city_index++;
    } else names_to_city[city_name]->money += money;

    person[i].money = money;
    person[i].location = names_to_city[city_name];
    names_to_billionaires[person_name] = &person[i];
}

for (auto &item : names_to_city) ranking.insert({item.second->money, item.second});

// Reading days

int m, k, cur_day = 0;
cin >> m >> k;

for (int i = 0; i < k; i++) {
    int day;
    string person_name;
    string city_name;
    cin >> day >> person_name >> city_name;

    int offset = day - cur_day;
    cur_day = day;

    auto it2 = ranking.begin();
    auto it = it2++;

    if (it2 == ranking.end() || it->first > it2->first) {
        it->second->days += offset;
    }

    City *to_city = names_to_city[city_name];
    SuccessfulPerson *cur_person = names_to_billionaires[person_name];

    // in case got city not listed at the start
    if (to_city == nullptr) {
        city[city_index].name = city_name;
        names_to_city[city_name] = &city[city_index];
        city_index++;

        to_city = names_to_city[city_name];
    }

    ranking.erase({cur_person->location->money, cur_person->location});
    ranking.erase({to_city->money, to_city});

    cur_person->location->money -= cur_person->money;
    ranking.insert({cur_person->location->money, cur_person->location});

    cur_person->location = to_city;
    to_city->money += cur_person->money;
    ranking.insert({to_city->money, to_city});
```

```
    }

    int days_left = m - cur_day; // m = all days
    auto it2 = ranking.begin();
    auto it = it2++;
    if (it2 == ranking.end() || it->first > it2->first) {
        it->second->days += days_left;
    }

    for (auto &item : names_to_city) {
        if (item.second->days > 0) {
            cout << item.first << " " << item.second->days << endl;
        }
    }
}
```