Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»
Кафедра Вычислительной Техники

Алгоритмы и структуры данных
Задание №3

Выполнила: Калугина Марина
Группа: P3202

г. Санкт-Петербург
2019 г.

1067.cpp

Все до безобразия просто -- строим дерево. У каждого элемента есть свое имя - название каталога и указатели на дочернии каталоги.

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1067.cpp

```
#include
<iostream>
                    #include <map>


                    /**
                    * все до безобразия просто -- строим дерево
                    */


                    struct tree {
                       tree() {
                           this->name = "";
                           this->child = {};
                       }


                       tree(std::string name) {
                           this->name = name;
                           this->child = {};
                       }


                       tree(std::string name, std::map< std::string, tree *>
                    child) {
                           this->name = name;
                           this->child = child;
                       }


                       std::string name;
```

```
    std::map< std::string, tree *> child;
};

void print_tr(tree *tr, int n) {
    for (int i = 0; i < n - 1; ++i) {
        std::cout << " ";
    }
    if (tr->name != "") {
        std::cout << tr->name << std::endl;
    }
    ++n;
    for (auto &j : tr->child) {
        print_tr(j.second, n);
    }
}

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

    tree *tr_root = new tree();

    for (int i = 0; i < n; ++i) {
        std::string patch;
        std::cin >> patch;
        std::string dir = "";

        tree *tr = tr_root;
        for (int j = 0; j <= patch.size(); ++j) {
            if (patch[j] == '\\' || patch[j] == '\0') {
```

```
                auto dir_tr = tr->child.find(dir);
                if (dir_tr == tr->child.end()){
                    tree *new_tree = new tree(dir);
                    tr->child[dir] = new_tree;
                    tr = tr->child.find(dir)->second;
                } else {
                    tr = dir_tr->second;
                }
                dir = "";
            } else {
                dir += patch[j];
            }
        }


    }
    print_tr(tr_root, 0);
    return 0;
}
```

```
#include
<iostream>
                    #include <map>

                    struct tree {
                       tree() {
                           this->name = "";
                           this->child = {};
                       }

                       tree(std::string name) {
                           this->name = name;
                           this->child = {};
                       }

                       tree(std::string name, std::map< std::string, tree *>
                    child) {
                           this->name = name;
                           this->child = child;
                       }

                       std::string name;
                       std::map< std::string, tree *> child;
                    };

                    void print_tr(tree *tr, int n) {
                       for (int i = 0; i < n - 1; ++i) {
                           std::cout << " ";
                       }
                       if (tr->name != "") {
                           std::cout << tr->name << std::endl;
```

```
    }
    ++n;
    for (auto &j : tr->child) {
        print_tr(j.second, n);
    }
}


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


    tree *tr_root = new tree();


    for (int i = 0; i < n; ++i) {
        std::string patch;
        std::cin >> patch;
        std::string dir = "";


        tree *tr = tr_root;
        for (int j = 0; j <= patch.size(); ++j) {
            if (patch[j] == '\\' || patch[j] == '\0') {
                auto dir_tr = tr->child.find(dir);
                if (dir_tr == tr->child.end()){
                    tree *new_tree = new tree(dir);
                    tr->child[dir] = new_tree;
                    tr = tr->child.find(dir)->second;
                } else {
                    tr = dir_tr->second;
                }
                dir = "";
            } else {
```

```
                dir += patch[j];
            }
        }


    }
    print_tr(tr_root, 0);
    return 0;
}
```

1494.cpp

Попробуем положить шары в стек так же, как их забивал бы Чичиков.
Если текущий шар больше, чем максимальный на данный момент, то считаем, что все в порядке, потому что именно в этом порядке шары попадают в лунку. (Если же он больше на несколько позиций, то, возможно, ревизор не подходил за шарами в момент, пока Чичиков забивал шары с номерами между максимальным и текущим, там собрался еще один стак шаров. Чтобы это проверить, сохраним все шары, которые должны были попасть в лунку в этот промежуток)

Если текущий шар меньше, то надо проверить, соответствует ли этот шар ожидаемому стеку, который мы сохранили выше.

Если не соответствует, то Чичиков жульничает.

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1494.cpp

```
#include
<iostream>
#include            <stack>


int main()
{
                int n;
                std::cin >> n;
                std::stack<int> balls;
                int tmp_max = 0;
                for (int i = 0; i < n; ++i) {
                    int cur;
                    std::cin >> cur;
                    if (cur > tmp_max) {
                        for (int j = tmp_max + 1; j <= cur - 1;
            j++) {
                            balls.push(j);
                        }
                        tmp_max = cur;
                    } else {
                        if (cur == balls.top()){
                            balls.pop();
                        } else {
```

```
                std::cout << "Cheater";
                return 0;
            }
        }
    }
    std::cout << "Not a proof";
    return 0;
}
```

1521.cpp

Создаем дерево Фенвика,позволяющее находить порядковый номер следующего человека в очереди. Это дерево помогает быстро (за O(log n)) изменять значения в массиве.

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1521.cpp

```
 #include <iostream>
int maxn = 1 <<
17;
                    int t[2 << 18];


                    void update(int x, int amount) {
                       for (x += maxn; x > 0; x /= 2)
                           t[x] += amount;
                    }


                    int find_kth(int x, int kth) {
                       while (x < maxn){
                           x *= 2;
                           if (kth > t[x]) {
                               kth -= t[x];
                               x++;
                           }
                       }
                       return x - maxn;
                    }


                    int main() {


                       int n, k;
                       std::cin >> n >> k;
                       for (int i = 0; i < n; ++i) {
                           update(i, 1);
```

```
//        for(int &item : t){
//            std::cout << item << " ";
//        }
//        std::cout << "\n";
    }
    int currPos = k - 1;
//    std::cout << "====================\n";


    for (int i = 0; i < n; ++i) {
        int pos = find_kth(1, currPos + 1);
        update(pos, -1);
//        for(int &item : t){
//            std::cout << item << " ";
//        }
//        std::cout << "\n";
        std::cout << pos + 1 << " ";
        if (i < n - 1)
            currPos = (currPos - 1 + k) % (n - i -
1);
    }
    return 0;
}
```

1628.cpp

Отсортируем все точки с левого верхнего угла до правого нижнего: сначала строки, потом столбцы.Каждую полосу, длиной больше 1 добавим в результат, каждую полосу, длиной 1 сохраним на будущее в вектор sq.

Аналогично отсортируем все точки снова, только теперь сначала столбцы, потом строки.

Теперь у нас есть количество вертикальных и горизонтальных полос. Осталось проверить только полосы 1x1.

Если полоса 1x1 встречается в векторе sq 2 раза, значит - квадрат 1x1 максимален и по вертикали и по горизонтали => его следует добавить в результат

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1628.cpp

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

bool comp1(std::pair<int, int> a,
std::pair<int, int> b) {
                                        if (a.first != b.first) {
                                            return a.first < b.first;
                                        } else {
                                            return a.second < b.second;
                                        }
                                    }


                                    bool comp2(std::pair<int, int> a,
                                    std::pair<int, int> b) {
                                        if (a.second != b.second) {
                                            return a.second < b.second;
                                        } else {
                                            return a.first < b.first;
                                        }
                                    }
```

```cpp
int main() {
    int m, n, k, res = 0;

    std::cin >> m >> n >> k;

    std::vector<std::pair<int, int>> point;
    std::vector<std::pair<int, int>> sq;

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

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

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

```
    sort(point.begin(), point.end(), comp1);


    for (int i = 0; i < point.size() - 1;
++i) {
        if (point[i].first == point[i +
1].first) {
            if (point[i + 1].second -
point[i].second - 1 > 1) {
                res++;
            } else if (point[i + 1].second -
point[i].second - 1 == 1) {
                std::pair<int, int>
p(point[i].first, point[i].second + 1);
                sq.push_back(p);
            }
        }
    }


    sort(point.begin(), point.end(), comp2);
    sort(sq.begin(), sq.end(), comp2);


    for (int i = 0; i < point.size() - 1;
++i) {
        if (point[i].second == point[i +
1].second) {
            if (point[i + 1].first -
point[i].first - 1 > 1) {
                res++;
            } else if (point[i + 1].first -
point[i].first - 1 == 1) {
                std::pair<int, int>
p(point[i].first + 1, point[i].second);
                sq.push_back(p);
            }
        }
    }
```

```
    sort(sq.begin(), sq.end(), comp1);


    if (sq.size() > 1) {
        int i = 0;
        while (i < sq.size() - 1) {
            if (sq[i] == sq[i + 1]) {
                res++;
                i++;
            }
            i++;
        }
    }


    std::cout << "\n\n" << res;


    return 0;
}
```

1650.cpp

Смоделируем ситуацию представленную в задаче.

В score_board хранится актуальное отсортированное множество городов. Из него мы можем узнавать город, находящийся в топе за константу.

При каждом перелете изменяем общую сумму денег в городах между которыми совершается перелет миллиардера, текущее положение миллиардера и обновляем score_board (обновление совершается за log).

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

Итоговый перфоманс: O(k log k)

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1650.cpp

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

struct
city {
                std::string name;
                long long money;
                int days;
              } ct[10000 + 50000];


              struct pers {
                long long money;
                city *location;
              } pr[10000];


              std::map<std::string, pers *> hum;
              std::map<std::string, city *> world;
              std::set<std::pair<long long, city *>, std::greater<>>
              score_board;
```

```
int main() {
    std::ios_base::sync_with_stdio(false);

    int n;
    std::cin >> n;

    int c = 0;
    for (int i = 0; i < n; ++i) {
        std::string name_p;
        std::string name_c;
        long long mon;
        std::cin >> name_p >> name_c >> mon;
        if (!world[name_c]) {
            ct[c].name = name_c;
            ct[c].money = mon;
            world[name_c] = &ct[c];
            c++;
        } else {
            world[name_c]->money += mon;
        }
        pr[i].money = mon;
        pr[i].location = world[name_c];
        hum[name_p] = &pr[i];
    }

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

    int m, k, today = 0;
```

```
    std::cin >> m >> k;


    for (int i = 0; i < k; ++i) {
        int day;
        std::string name_p, name_c;
        std::cin >> day >> name_p >> name_c;
        int count = day - today;
        today = day;
        auto it2 = score_board.begin();
        auto it = it2++;
        if (it2 == score_board.end() || it->first > it2->first) {
            it->second->days += count;
        }
        city *to_city = world[name_c];
        pers *which_pers = hum[name_p];
        if (to_city == nullptr) {
            ct[c].name = name_c;
            world[name_c] = &ct[c];
            c++;
            to_city = world[name_c];
        }
        score_board.erase({which_pers->location->money,
which_pers->location});
        score_board.erase({to_city->money, to_city});
        which_pers->location->money -= which_pers->money;
        score_board.insert({which_pers->location->money,
which_pers->location});
        which_pers->location = to_city;
        to_city->money += hum[name_p]->money;
        score_board.insert({to_city->money, to_city});
    }


    int count = m - today;
```

```
    auto it2 = score_board.begin();
    auto it = it2++;
    if (it2 == score_board.end() || it->first > it2->first) {
        it->second->days += count;
    }


    for (auto &item : world) {
        if (item.second->days > 0)
            std::cout << item.first << " " << item.second->days <<
"\n";
    }
    return 0;
}
```