Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по «Алгоритмам и структурам данных»

Timus

Выполнил:

Студент группы P3233
Богатов Александр Сергеевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2022

Задача 1067: Структура папок

```cpp
#include <iostream>
#include <map>
#include <algorithm>
#include <vector>
#include <sstream>

using namespace std;

class Directory {

    public:

    map<string, Directory*> children;

    Directory() {

    }

    Directory* make_dir(string dir_name) {
        if (children.find(dir_name) == children.end()) {
            children[dir_name] = new Directory();
        }
        return children[dir_name];
    }

    void display_hierarchy(string indentation, bool is_root) {
        if(!is_root)
            indentation += " ";
        for (auto& child : children) {
            cout << indentation << child.first << endl;
            child.second->display_hierarchy(indentation, false);
        }

    }

};

int main()
{
    int n;

    cin >> n;

    Directory* root_dir = new Directory();
    char separator = '\\';
    Directory* current_dir;

    for (int i = 0; i < n; i++) {
        string current;
        vector<string> parsed_path;

        current_dir = root_dir;
        cin >> current;
        replace(current.begin(), current.end(), separator, ' ');
        stringstream ss(current);
        string tmp;
```

```
        while (ss >> tmp)
            parsed_path.push_back(tmp);

        for (string parsed_dir : parsed_path) {
            current_dir = current_dir->make_dir(parsed_dir);
        }
    }
    root_dir->display_hierarchy("", true);
    delete current_dir;
    delete root_dir;
    return 0;
}
```

Пояснение к примененному алгоритму:

Для удобного парсинга создадим класс директории, содержащий одно поле – мапу с дочерними директориями. В главном методе заведем объект корневой директории, для каждой строки создаем массив встретившихся в пути папок. Используя переменную current_dir мы погружаемся вниз по иерархии, заполняя список дочерних директорий (для каждого имени папки проверяется, проходили ли мы через неё уже, в таком случае нового объекта в мапе не создается). После того, как мы заполнили для всех директорий список их дочерних директорий, рекурсивно выведем их, на каждом шаге увеличивая отступ для верного отображения. Предположительная алгоритмическая сложность: $mx_i\ log(n) + \sum m_i$, где xi – число папок в текущем пути для каждой m строки, mi – число предков каталога нижнего уровня для каждого корневого каталога.

Задача 1494: Монобильярд

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    int n;

    cin >> n;

    int pulled[n];

    vector<int> rolled;

    for (int i = 0; i < n; i++) {
        cin >> pulled[i];
    }

    int max_pulled = 0;

    for (int i = 0; i < n; i++) {
        if (pulled[i] > max_pulled) {
            for (int j = max_pulled + 1; j < pulled[i]; j++)
                rolled.push_back(j);
            max_pulled = pulled[i];
        } else {
            if (pulled[i] == rolled.back()) {
                rolled.pop_back();
```

```
                continue;
            }

            else {
                cout << "Cheater" << endl;
                return 0;
            }
        }
    }

    cout << "Not a proof" << endl;

    return 0;
}
```

Пояснение к примененному алгоритму:

Ревизор подходит к столу периодически, между его отходами может быть забит новый шар. То есть, порядок доставания шаров 3, 4, 2, 5, 1 – допустим: ревизор достал 3й шар, пока его не было был забит 4й, далее во время отхода шаров забито не было – он достает второй и т.д. Однако порядок 3, 4, 1, 5 – доказательство жульничества, т.к. шар 1 никак не мог быть выше шара 2 по правилам игры. Создадим массив вытянутых шаров и вектор закаченных в лузу шаров и будем запоминать вытянутый шар с максимальным числом. Каждый раз, когда достается шар с новым максимумом, предполагаем, что все шары с числами меньше уже в лунке и заполняем вектор соответствующими числами. Если в будущем мы встретили это самое число max-1 – оно должно быть на вершине вектора и мы его снимаем. В ином же случае, мы нашли доказательство жульничества. Алгоритмическая сложность в худшем случае – O (N*C). C – некая константа, число шаров забитых между подходами ревизора. В худшем случае – N*(N-1), но это не точно.