Университет ИТМО

Факультет программной инженерии и компьютерной техники

# Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P32302

Терновский И.Е.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача №1 «Агроном-любитель»

```
#include <iostream>
#include <vector>
using namespace std;


int main() {
    int n;
    cin >> n;
    vector<int> v(n);
    for (auto& it : v)
        cin >> it;
    int max1 = 0;
    int count = 0;
    int index = 0;

    for (int i = 1; i< n + 1; i++){
        if ((v[i - 1] == v[i]) && (v[i]== v[i + 1])){
            count++;
            if (count > max1){
                max1 = count + 1;
                index = i + 1;
            }
            count = 0;
        }
        else{
            count++;
            if (count > max1){
                max1 = count;
                index = i;
            }
        }
    }
    cout<<index-max1+1<<" "<<index;
}
```

Пояснение к примененному алгоритму:

Алгоритм очевиден и прост =).

Мы проходимся по всем цветкам и если 3 подряд идущих цветка не одинаковы мы увеличиваем длину самого длинного участка при этом записывая индекс последнего такого цветка( что бы вывести этот индекс - длина самого длинного участка). Если же 3 подряд идущих цветка одинаковы мы увеличиваем длину на 2 и записываем ее, при этом обнуляя длину участка.

Сложность, очевидно, O(n) так как мы один раз проходим по всем цветкам и выполняем простые операции на каждом цветке.

Задача №2 «Зоопарк Глеба»

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

using namespace std;

int main() {

    stack<char> mainStack;

    stack<int> animalCounterStack;
    stack<int> trapCounterStack;

    int animalCounter = 0;
    int trapCounter = -1;

    string data;
    cin >> data;

    vector<int> v(data.size()/2);

    mainStack.push(0);

    for (char letter : data) {

        if (islower(letter)) {
            animalCounterStack.push(++animalCounter);
        } else
        {
            trapCounterStack.push(++trapCounter);
```

```
        }
        if (( (islower(mainStack.top()) && isupper(letter)) || (isupper(mainStack.top()) &&
islower(letter)))
            && tolower(mainStack.top()) == tolower(letter)) {

                v[trapCounterStack.top()] = animalCounterStack.top();
                animalCounterStack.pop();
                trapCounterStack.pop();
                mainStack.pop();

            } else
            {
                mainStack.push(letter);
            }
    }

    if (mainStack.size() == 1) {
        cout << "Possible" << endl;
        for (int i : v) {
            cout << i << " ";
        }
    } else
    {
        cout << "Impossible" << endl;
    }

}
```

Пояснение к примененному алгоритму:

Тут алгоритм чуть сложнее, если очень сильно упростить, то я проверяю что каждое животное стоит “по соседству” со своей ловушкой. То есть я прохожусь по всей строке, записываю в стек количество всех ловушек и всех животных (что бы потом правильно вывести какая ловушка поймает какое животное). Если в строке попадается животное я проверяю является ли предыдущий элемент ловушкой для него, соответственно для ловушки я проверяю является ли предыдущий проверенный символ животным для этой ловушки. Если нет то просто записываю этот символ в стек. В конце проверяю если стек пустой, то есть если для каждого элемента была найдена пара, то вывожу что такое возможно и вывожу список, который формируется как [номер ловушки] = [номер животного]

Сложность O(n) так как я один раз прохожусь по строке и все операции внутри выполняются за константу.

Задача №3 «Конфигурационный файл»

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <cstring>
#include <stack>
using namespace std;


int main() {
    unordered_map<string, stack<int>> hackmap;
    string currentString;
    vector<unordered_map<string , int>> callStack;
    callStack.emplace_back();
    string str="-123456789";
    std::ios::sync_with_stdio(false);

    while(cin >> currentString){

        if (currentString == "{"){
            callStack.emplace_back();
            continue;
        }

        if (currentString == "}"){
            for (const auto & [ key, value ] : callStack.back()) {
                hackmap[key].pop();
                if (hackmap[key].empty()){
                    hackmap.erase(key);
                }
            }
            callStack.pop_back();
            continue;
        }

        size_t EndPart1 = currentString.find_first_of('=');
        string var1 = currentString.substr(0, EndPart1);
        string var2 = currentString.substr(EndPart1 + 1);



        if (str.find(var2[0]) == std::string::npos){ // if right side is a variable

            int new_val;
            try {
                if (hackmap.at(var2).empty()){
                    new_val = 0;
                }else{
                    new_val = callStack[hackmap.at(var2).top() - 1].at(var2) ;
                }
```

```
            } catch (...){
                new_val = 0;
            }
            callStack.back()[var1] = new_val;
            cout << new_val<<"\n";
        }else{
            callStack.back()[var1] = atoi(var2.c_str());
        }
        if (hackmap.count(var1) == 0 ){
            hackmap[var1].push(callStack.size());
        }else{
            if (hackmap[var1].empty() || hackmap[var1].top() != callStack.size()){
                hackmap[var1].push(callStack.size());
            }
        }
    }
}
```

Пояснение к примененному алгоритму:

Задача заключается в том что бы хранить последнее значение переменной, я это решал через костыль, который заключается в том что бы хранить номер уровня вложенности в стеке ( то есть у меня есть vector в котором хранятся map<” переменная ”:” значение переменной на этом уровне вложенности ”>

И есть map <” переменная ”:” стек уровней вложенности с этой переменной ”>.

Таким образом, когда мне нужна какая-то переменная я смотрю на каком последнем уровне она встречалась и обращаюсь к этому уровню вложенности что бы получить значение.

Сложность оценить сложно, так как есть и поиск по строке который зависит от длины строки и преобразование строки к числу для которой я вообще не смог найти сложность. Предположу, что наихудший вариант это $O(n^2)$, но было бы интересно узнать, как можно найти сложность для такого сложного кода.

Задача №4 «Профессор Хаос»

```
#include <iostream>

using namespace std;

int main() {

    int a, b, c, d, k;
    cin >> a;
    cin >> b;
    cin >> c;
    cin >> d;
    cin >> k;
```

```
    int currentDay;

    if (d == 1 && b>c){
        cout << 1;
    }
    else {
        for (int i = 0; i < k; i++) {

            currentDay = a * b - c;

            if (currentDay >= d){
                currentDay = d;
            }

            if (a == currentDay){
                break;
            }

            if (currentDay <= 0) {
                currentDay = 0;
                break;
            }
            a = currentDay;
        }

        cout << currentDay;
    }
}
```

Пояснение к примененному алгоритму:

В данной задаче я просто выполняю симуляцию которая требуется по заданию и проверяю результат каждый день, если количество бактерий в новый день = количеству бактерий в прошлый день, то можно заявить что эксперимент закончится с таким же количеством бактерий, так как количество на которое мы умножаем и которое вычитаем константы, то при одном и том же количестве бактерий в начале дня результат будет один и тот же. Так же если после эксперимента осталось 0 бактерий, то очевидно эксперимент заканчивается, так как 0*b-c <= 0 при любых b и c.

Сложность в худшем случае O(k). Так же если контейнер вмещает 1 бактерию и b > c, то сложность O(1)