Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3233
Богатов Александр Сергеевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2022

Задача A «Агроном-любитель»

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

int main() {
    int n;
    cin >> n;
    int array[n];
    int sameFlowersAmount = 1;
    int longestStart = 0;
    int longestEnd = 0;
    int currentStart = 0;
    int currentEnd = 0;

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

    for (int i = 1; i < n; i++) {
        if (array[i] == array[i-1]) {
            sameFlowersAmount++;
            if (sameFlowersAmount >= 3) {
                if (currentEnd - currentStart > longestEnd - longestStart) {
                    longestStart = currentStart;
                    longestEnd = currentEnd;
                }
                currentStart = i - 1;
                sameFlowersAmount--;
            }
        } else sameFlowersAmount = 1;
        currentEnd = i;
        if (i == n - 1 && (currentEnd - currentStart > longestEnd - longestStart)) {
             longestStart = currentStart;
             longestEnd = currentEnd;
        }
    }

    cout << longestStart + 1 << " " << longestEnd + 1;
    return 0;

}
```

Пояснение к примененному алгоритму:

В цикле проходим по введенной последовательности, если текущий элемент одинаков с предыдущим, можно увеличить счетчик одинаковых цветов подряд.

Если значение счетчика достигло трех, значит потенциально выгодный участок закончился, сравниваем длину закончившегося участка с длиной ранее найденного. Если она больше, то присваиваем новые значения переменным, хранящим начало и конец лучшего отрезка. Далее начало текущего участка отодвигаем на цветок назад и соответственно уменьшаем счетчик одинаковых цветков.

Если текущий и предыдущий элемент не равны, счетчик одинаковых цветов подряд устанавливается в единицу (т.е. текущий цветок).

Далее конец текущего участка устанавливается в i, отслеживаем ход по последовательности.

Если мы достигли последней итерации, необходимо провести дополнительную проверку на нахождение наилучшей последовательности.

При выводе найденных чисел прибавляем единицы, т.к. до этого мы работали с индексами массива, задача требует найти порядковые номера цветов.

Задача B «Зоопарк Глеба»

```cpp
#include <iostream>
#include <stack>
#include <list>

using namespace std;

int main() {
    stack <pair<char, int>> zoo;
    int animals = 0;
    int traps = 0;
    string sequence;
    cin >> sequence;
    int n = sequence.size() / 2;
    int solution[n];
    for (char current : sequence) {
        if (!zoo.empty() && tolower(current) == tolower(zoo.top().first) &&
        ( isupper(current) && islower(zoo.top().first) || isupper(zoo.top().first) && islower(current))) {
            if (isupper(current))
                solution[traps++] = zoo.top().second;
            else solution[zoo.top().second] = animals++;
            zoo.pop();
        } else {
            if (isupper(current))
                zoo.push(pair<char,int>(current, traps++));
            else zoo.push(pair<char,int>(current, animals++));
        }
    }
    if (zoo.empty()) {
        cout << "Possible" << endl;
        for (int i = 0; i < n; i++)
            cout << solution[i] + 1 << " ";
    } else cout << "Impossible" << endl;
    return 0 ;
}
```

Пояснение к примененному алгоритму:

Животных можно поймать, если между животным и соответствующей ловушкой нет других животных или их ловушек по отдельности. Например, AaBbCc или ABCcba – возможно, AaBcbC – невозможно. Однако, если другое животное и его ловушка находятся рядом, пересечения не будет и последовательность можно разрешить. Например, AaBcCb – здесь cC отделяет животное b от его ловушки, однако животное c может пойти в свою ловушку не пересекая пути других животных.

Будем хранить пары символов и их индексов на стеке. Для каждого символа в последовательности – цикл. Если текущая и предыдущие буквы в последовательности различны, то заполняем стек: если текущий символ означает ловушку, в качестве пары к символу записываем счетчик ловушек (далее счетчик ловушек инкрементируется), иначе пара – счетчик животных (тоже с постинкрементом).

При каждом новом символе проверка на то, что текущая и предыдущая буквы одинаковы, но разных регистров, в случае если да – можно довести животного в ловушку. Счетчики ловушек и животных используются для определения их индексов

при пуше на стек и для распределения животных по ловушкам, если это возможно. Если мы ищем решение при встрече с ловушкой, то индексом решения будет текущее число ловушек, иначе индексом решения будет индекс последнего животного на стеке.

Задача C «Конфигурационный Файл»

```cpp
#include <fstream>
#include <vector>
#include <map>
#include <string>
#include <algorithm>

using namespace std;

int main () {

    ifstream fin("input.txt");
    ofstream fout("output.txt");

    vector<vector<string>> scopeVariables = vector<vector<string>>();
    map <string, vector<int>> storage = map<string, vector<int>>();

    string current;

    scopeVariables.push_back(vector<string>());
    while (true) {
        fin >> current;
        if(fin.eof())
            break;
        if (current == "{") {
                scopeVariables.push_back(vector<string>());
        } else if (current == "}") {
            for (auto &key : scopeVariables.back()) {
                storage[key].pop_back();
            }
            scopeVariables.pop_back();
        } else {
            int signPos = current.find("=");
            string var1 = current.substr(0, signPos);
            string var2 = current.substr(signPos+1);
            if(var2.find_first_of("0123456789") != string::npos) {
                if(storage.find(var1) == storage.end()) {
                    storage.insert(pair<string,vector<int>>(var1, vector<int>()));
                }
                scopeVariables.back().push_back(var1);
                storage[var1].push_back(stoi(var2));
            } else {
                 if(storage.find(var1) == storage.end()) {
                    storage.insert(pair<string,vector<int>>(var1, vector<int>()));
                    storage[var1].push_back(0);
                }
                 if(storage.find(var2) == storage.end()) {
                    storage.insert(pair<string,vector<int>>(var2, vector<int>()));
                    storage[var2].push_back(0);
                }
                scopeVariables.back().push_back(var1);
                storage[var1].push_back(storage[var2].back());
                fout << storage[var2].back() << endl;

            }

        }
    }

    return 0;
```

Пояснение к примененному алгоритму:

Используем коллекцию коллекций строк для отображения конкретных переменных введенных/измененных внутри блока. Используем мапу строка->vector<int>, чтобы хранить к каждой переменной все её введенные значения. Когда скоуп закрывается, происходит итерация по строка последнего вектора в векторе scopeVariables, число появлений переменной в этом векторе = числу изменений значения данной переменной в текущем скоупе => для каждой переменной необходимо провести именно такое число pop_back'ов с соответствующего вектора в мапе.

Например:
a=3 { a=1 a=b a=5 } a=a
Перед выходом из внутреннего блока мапа будет выглядеть так:
a: 3 1 0 5
b: 0

После выхода из блока:

a: 3
b: 0

Если на вход поступает строка с переменной(ыми) она разделяется по знаку =. Если это строка типа var=<number>, сначала ищем есть ли эта переменная в мапе и создаем для нее пару, если ее в мапе нет. Далее помещаем переменную в scopeVariables для отслеживания изменений и кладем в вектор переменной в мапе её новое значение.

Для строки типа var1 = var2 аналогичные действия, но с двумя переменными.

Задача D «Профессор Хаос»

```cpp
#include <iostream>

using namespace std;

int main() {

    int a;
    int b;
    int c;
    int d;
    int k;

    cin >> a >> b >> c >> d >> k;

    int left = a;
    int infCycleCheck = 0;

    for (int i = 0; i < k; i++) {
        left = left * b - c;
        if (left <= 0) {
            left = 0;
            break;
        }
        if (left >= d) {
            left = d;
        }
        if (left == infCycleCheck)
            break;
        infCycleCheck = left;
    }
    cout << left;

    return 0;
}
```

Пояснение к примененному алгоритму:

Всё просто: применяем все изначальные вычисления к оставшемуся числу образцов, далее проверяем условия по заданию и выполняем соответствующие действия. На случай данных, при которых число остаточных образцов никогда не меняется, выходим из цикла заранее.