Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по дисциплине “Алгоритмы и структуры данных”

Базовые задачи

Выполнила:
Шевченко Д.П.,
группа P3230

Преподаватели:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2024

Задача A
Агроном-любитель

```cpp
#include <iostream>

int main() {
    int count = 0;
    std::cin >> count;
    int number[count];//считываем количество цветов
    for (int i = 0; i < count; i++) {
        std::cin >> number[i]; //считываем каждый цветок
    }
    int maxLength = 1; // максимальная допустимая длина для трех цветков
    int start = 0; //начало
    int length = 1; // текущая длина
    int end = 1; //конец
    int maxCurLen = 2; //максимальная длина из списка
    int minNumber = 0; //начало текущего участка
    for (int i = 1; i < count; i++) {
        if (number[i - 1] == number[i]) {
            maxLength += 1;
        }
        if (maxLength == 3) {
            if (length > maxCurLen) {
                maxCurLen = length;
                start = minNumber;
                end = i - 1;
            }
            maxLength = 2;
            length = 2;
            minNumber = i - 1;
        } else if (number[i - 1] != number[i]) {
            maxLength = 1;
            length++;
        } else {
            length++;
        }
        if (i == count - 1 and length > maxCurLen) {
            start = minNumber;
            end = count - 1;
            length++;

        }
    }
    if (count == 1) {
        start = end = 0;
    }
    if (count == 0) {
        start = end = -1;
    }

    std::cout << start + 1 << " " << end + 1;

    return 0;


}
```

Пояснение к примененному алгоритму:

number[count] - массив, который вводит пользователь с номерами цветков
maxLength - счетчик, который показывает текущее количество одинаковых цветков подряд
start - номер первого цветка в итоговом ответе
length - количество цветков в текущей последовательности
end - номер последнего цветка в итоговом ответе
maxCurLen - максимальная длина подходящей последовательности(без трех цветков)
minNumber - минимальный номер первого цветка подходящей последовательности

1) Программа сравнивает предыдущий цветок с текущим, если они одинаковые, то счетчик maxLength увеличиваем. При достижении 3-х одинаковых цветков подряд, сравниваем текущую длину участка с максимальной, которая была. Если текущий участок больше, то забываем про предыдущий. Цикл продолжаем с maxLength = 2, так как два одинаковых цветка в последовательности уже есть.
2) Если два подряд идущих цветка не равны, то сбрасываем счетчик maxLength
3) Увеличиваем длину текущего участка
4) Также нужно проверить крайний участок. Если мы дошли до последнего цветка и текущая длина больше максимальной, которая была, то текущий участок записываем в ответ.

Отдельно проверяем случаи с 1 и 0 цветками.
В ответ записываем start+1 и end+1
Сложность:
O(n)
n = count – общее количество цветков

Задача B
Зоопарк Глеба

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

int main() {
    std::string str;
    std::cin >> str;

    std::stack<std::pair<char, int>> stack;
    int small = 0;
    int big = 0;
    int array[str.length() / 2];
    for (char current: str) {
        if (!stack.empty() and isupper(current) and
islower(stack.top().first) and
            tolower(current) == stack.top().first) {
            array[big] = stack.top().second + 1;
            big += 1;
        } else if (!stack.empty() and isupper(stack.top().first) and
islower(current) and
                   current == tolower(stack.top().first)) {
            array[stack.top().second] = small + 1;
            small += 1;
        } else if (stack.empty() or tolower(current) !=
tolower(stack.top().first) ||
                   ((islower(stack.top().first) and islower(current)) ||
                    isupper(stack.top().first) and isupper(current))) {
            if (isupper(current)) {
                stack.emplace(current, big);
                big += 1;
            } else {
                stack.emplace(current, small);
                small += 1;
            }
            continue;
        }
        stack.pop();
    }
    if (stack.empty()) {
        std::cout << "Possible" << std::endl;
        for (int i = 0; i < str.length() / 2; i++) {
            std::cout << array[i] << " ";
        }
    } else {
        std::cout << "Impossible" << std::endl;
    }
    return 0;
}
```

Пояснение к примененному алгоритму:

stack – стек, в котором хранятся пары животных и ловушек с их индексами
small – порядковый номер животного

big – порядковый номер ловушки
array – массив с итоговыми значениямиё
Пути животных не будут пересекаться, если получится найти пару одинаковых букв разного регистра, стоящих рядом, после этого вычеркнуть их, и так пока все буквы не будут вычеркнуты.

То есть если дана строка aBbA , вычеркиваем две стоящие одинаковые Bb, остается aA ,которые тоже можно вычеркнуть. Получается, что пути не пересекаются.

Проходимся по каждому элементу.

1) Если стек не пустой и текущая буква заглавная, не совпадает с регистром предыдущей буквы и буквы одинаковы, то в массив под порядковый номер заглавной буквы записываем порядковый номер строчной + 1. Увеличиваем индекс заглавной буквы. Из стека удаляем строчную предыдущую букву.
2) Если же буквы равны, но текущая является строчной, а предыдущая заглавной, то в массив под порядковый номер заглавной буквы(то есть предыдущей) записывается индекс строчной(текущей). Из стека удаляем заглавную предыдущую букву.
3) Если букв в стеке нет или две подряд идущие буквы не равны или одинакового регистра, то в стек записываем букву и ее индекс. Начинаем заново.

В результате смотрим, если в стеке остались буквы, то результат “Impossible”, иначе выводим индексы.
Сложность:
O(n)
n – количество символов в строке, так как проходимся по каждому символу

Задача C
Конфигурационный файл

```
#include <iostream>
#include <string>
#include <fstream>
#include <map>
#include <regex>
#include <unordered_map>

bool isNumber(const std::string &str) {
    for (char ch: str) {
        if (std::isdigit(ch)) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<std::string> lines;
    std::string line;
    std::smatch match;
    std::map<std::string, std::vector<int>> mapOld;

    // Регулярное выражение для поиска символов перед знаком равенства
    std::regex pattern2("(.+?)=");
    std::vector<std::string> keys;

    // Считываем строки с ввода
    while (std::getline(std::cin, line)) {
        if (line == "\0") {
            break;
        }
        lines.push_back(line);
    }
    int countBlocks = 0;
    int countCloseBlocks = 0;
    std::string before;
    std::string after;
    std::unordered_map<int, std::vector<std::string>> newArr;
    for (const auto &str: lines) {
        if (str == "{") { //если встретили {
            countBlocks++;
        } else if (str == "}") {
            for (const std::string &key: newArr[countBlocks]) {
                keys.push_back(key);
            }

            for (const std::string &key: keys) {
                if (!mapOld[key].empty()) {
                    mapOld[key].pop_back();
                } else {
                    mapOld[key].push_back(0);
                }
            }
            newArr.erase(countBlocks);
            keys.clear();
            countCloseBlocks -= 1;
            countBlocks -= 1;
        } else {
            std::regex_search(str, match, pattern2);
```

```
            before = match.str(1); // символ до равно
            before.erase(remove_if(before.begin(), before.end(), ::isspace),
before.end());
            size_t found = str.find("=");
            if (found != std::string::npos) { // символ после равно
                after = str.substr(found + 1);
                after.erase(remove_if(after.begin(), after.end(), ::isspace),
after.end());
                if (isNumber(after)) { // если после равно стоит число
                    int a = std::stoi(after);
                    mapOld[before].push_back(a);

                } else { // если числа нет после равно
                    if (mapOld.count(after) > 0) { // если значение уже есть
                        int a = mapOld[after].back();
                        mapOld[before].push_back(a);
                        std::cout << mapOld[after].back();
                        std::cout << std::endl;
                    } else { // если значения нет
                        mapOld[before].push_back(0);
                        std::cout << "0";
                        std::cout << std::endl;
                    }

                }
            }
            if (countBlocks > 0) {
                newArr[countBlocks].push_back(before);
            }
        }


    }

    return 0;


}
```

Пояснение к примененному алгоритму:

lines – вектор, содержащий все строки
mapOld – map, которая содержит все переменные и все их значения
keys – вектор, содержащий переменные, которые менялись в текущем блоке
countBlocks – счетчик {
countCloseBlocks - счетчик}
before – элемент до знака равно
after – элемент после знака равно
newArr – unordered map хранящий номер блока и названия переменых которые в нем определены

Записываем каждую строку в вектор, после чего проходимся по каждому элементу вектора, то есть по каждой строке.

Если встретилось начало блока, то счетчик блоков увеличиваем на один.
Если встретился конец блока, то из unordered map берем все переменные, которые были в текущем блоке и записываем их в вектор. Проходимся по каждому элементу вектора, если у переменной было значение, то убираем последнее(то есть если было b = 10 { b = 20 } после этого b станет равно 10) Иначе переменной присваиваем 0. Удаляем все переменные этого блока и сам блок из newArr и keys.
Если встретилась переменная, то возможно два варианта:

1) После равно стоит число, в таком случае просто записываем переменную и ее значение в mapOld
2) После равно стоит переменная, проверяем, есть ли у этой переменной значение, если да, то присваиваем его, если нет, то присваиваем 0.Выводим либо 0, либо значение переменной после знака равно.

А также если находимся внутри блока, то записываем переменную и номер блокат в newArr.

Сложность:

Считывание строк O(n) n-количество строк
Проход по каждой строке O(n)
O(n^2)

Задача D
Профессор Хаос

```cpp
#include <iostream>

int main() {
    int a, b, c, d, k;
    std::cin >> a >> b >> c >> d >> k;
    for (int i = 0; i < k; i++) {
        if (a * b <= c) {
            a = 0;
            break;
        }
        if (a * b - c == a) {
            break;
        }
        a = a * b - c;
        if (a >= d) {
            a = d;
            break;
        }
    }
    std::cout << a;
    return 0;
}
```

Пояснение к примененному алгоритму:

a – количество особо опасных бактерий в начале первого дня
b – новых бактерий, которые образуются вместо каждой одной особо опасной
c – кол-во бактерий для опытов, которые потом уничтожаются
d – кол-во бактерий, которые могут поместиться в контейнер
k – кол-во дней

Циклом проходимся по каждому дню эксперимента. Проверяем, хватает ли новых бактерий (a*b) для проведения опыта (c), если не хватает, то прерываем цикл, итоговое значение равно 0, так как считать дальше нет смысла a*b-c будет давать 0, либо отрицательное значение.
Если количество бактерий после эксперимента не изменилось, то завершаем цикл, так как меняться дальше не будет. Иначе присваиваем итоговому значению новое количество бактерий.
Если итоговое значение превышает(или равно) возможное количество бактерий в контейнере, то в ответ выводим максимум бактерий в контейнере.

Сложность:
Общая сложность O(n)

Так как считывание данных и итерация цикла O(1), а сложность цикла O(n), потому что цикл проходит n раз.