Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Яндекс.Контест

Выполнил:

Студент группы P3211

Кривоносов Е.Д.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2021

Задача №A «Агроном-любитель»

Пояснение к примененному алгоритму:

Конечно, можно сначала запихать в массив наши числа, а потом его пробегать, но тогда программа будет дольше

производить вычисления, т.к. ей 2 раза придётся пробежать n чисел, так еще и память тратить в пустую... Логичнее будет сразу считывать и производить вычисления, я без хранения всего массива чисел.

Мы будем просто через проверку слева направо решать данную задачу.

Для того чтобы решить задачу мы должны проверять каждые 3 числа делая в последствии сдвиг.

Как только мы находим 3 одинаковых числа подряд, то в этот момент мы "сбрасываем" нашу длину до минимального.

В данном случае она будет у на 2 т.к логично, что мы проверяем по 3 числа отбрасывая предыдущее, а 2 < 3 и продолжаем дальше производить проверку.

Каждый раз, когда мы делаем "сброс" мы должны запомнить грядку, с которой должны начать и продолжить двигаться.

С помощью простого прохода слева на право мы сможем узнать максимальную длину фотографии (начальную и последнюю грядку)

В конце решения задачи нужно не забыть проверить текущую длину с прошлой, которую мы запомнили и если она лучше прошлой, то обновить наши параметры.

Сложность алгоритма:
O(n)

Код:

```
#include <iostream>

using namespace std;

void solve(){
    int n;
    int len = 0;
    int start = 1, finish = 2;
    int fisrt = 0, second = 0, third = 0;
    int max[3] = {0, 1, 2}; // 1 - len, 2 - start, 3 - end;
    cin >> n;
    for (int i = 1; i <= n; i++){
        fisrt = second;
        second = third;
        cin >> third;
        if (fisrt == second && second == third){
            if (max[0] < len){
                max[0] = len;
                max[1] = start;
                max[2] = finish;
            }
            start = i - 1;
```

```
            finish = i;
            len = 2;
        } else {
            len++;
            finish = i;
        }
    }
    if (max[0] < len){
        max[1] = start;
        max[2] = finish;
    }
    cout << max[1] << " " << max[2];
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    solve();
    return 0;
}
```

Задача №B «Зоопарк Глеба»

Пояснение к примененному алгоритму:

Когда я изначально решал задачу, мне показалось, суть задачи, что у нас все клетки и животные стоят по краям и они параллельны (так сказать палиндром), поэтому логично было разделить нашу строну на 2 части по n и сравнивать их параллельно друг с другом ну и в конце сравнить с числом n. Если условие выполнялось бы, что все животные попали в клетки, то оно равнялось бы числу 2n/2 и сложность была O(n). Но потом добавили тесты, который проверял, что животные могут несколько стоять в ряд, например: AbCcdDBa, для которого результат 4 2 3 1. Ну в этом случае я решил добавить проверку, что животные и клетки могут стоять рядом и животные могут стоять перед клетками так и после, это тоже нужно было учитывать, из-за этого алгоритм стал работать дольше в 2 раза т.к. нам нужно пробегать весь массив до конца. В итоге логика решения почти сохранилась.

Сложность алгоритма:
O(2n)

Код:

```
#include <iostream>

#include <string.h>
#include <list>

using namespace std;

void solve(){
    string text;
    cin >> text;
    int n = text.size();
    int i, j, count = 0;
    list<int> answer;
```

```
    int start = 1, end = n / 2;
    list<int> answer_help;
    int help = 0;

    for (i = 0; i < n; i++){
        if ((( (int)text[i] == (int)text[n - i - 1] + 32) || ((int)text[i] ==
(int)text[n - i - 1] - 32)) && i < n / 2){
            if ((int)text[i] == (int)text[n - i - 1] - 32){
                answer.push_back(end--);
            }
            if ((int)text[i] == (int)text[n - i - 1] + 32){
                answer_help.push_back(start++);
            }
            count++;
        } else if (((int)text[i] == (int)text[i + 1] - 32 || (int)text[i] ==
(int)text[i + 1] + 32) && text[i] != text[i + 1]){
            if ((int)text[i] == (int)text[i + 1] - 32){
                answer.push_back(start++);
            }
            if ((int)text[i] == (int)text[i + 1] + 32){
                answer.push_back(end--);
            }
            count++;
            help++;
        }
    }

    if (count == n / 2){
        cout << "Possible" << '\n';
        if (help == 0) answer_help.reverse();
        for (auto val : answer_help){
            answer.push_back(val);
        }
        for (auto val : answer){
            cout << val << " ";
        }
    } else {
        cout << "Impossible";
    }
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    solve();
    return 0;
}
```

Задача №C «Конфигурационный файл»

Пояснение к примененному алгоритму:

При решении задачи мы могли натолкнуться на проблему, либо с долгим выполнением т.к. уходим в рекурсии или в большом количестве затраты памяти, когда храним все значения, даже те, которые в последствии нам не нужны. Мое решение учитывает оба этим аспекта, хотя чтобы ускорить программу можно

использовать в место обычной мапы, хэшмапу т.к. наш ключ – это строка, а при поиске в мапе у нас сравнивает полностью всю строку из-за этого дольше происходят вычисления (примерно на 25% будет быстрее, если использовать хэшмапу). В основной мапе я хранил все текущие значения, а вторую мапу я использовал для хранения значений из основной мапы, которые я заменил, войдя в блок т.к. при выходе мы должны вернуть основную мапу в исходное положения до входа в блок (сделать откат). Так же, чтобы не хранить ненужные значения в мапе для блоков я их отчищал тем самым освобождая память для дальнейших вычислений.
Ну, а решение очень простое. Если мы входим в блок “{”, просто увеличиваем переменную block, которая отвечает за номер блока, в который мы вошли. Если мы выходим из блока “}”, мы восстанавливаем основную мапу в исходное состояние, до того, как мы вошли в блок. Дальше считываем для переменной var1 число или переменную. Если мы считываем число, то сохраняем предыдущие значение, если оно есть в основной мапе в мапу для блоков и сохраняем var1 в основную мапу. Если считываем переменную мы делаем аналогично, но ещё её выводим. Тем самым мы линейно пробегаем все строки не храня их все.

Сложность алгоритма:
O(n)

Код:

```
#include <iostream>
#include <string.h>
#include <map>

using namespace std;

map<string, int> stack, stack_blocks[100000]; // stack - основной, stack_blocks
- хранение пременных до входа в блок, чтобы при выходе вернуть их.

void solve(){
    string line;
    int block = 0;
    while (cin >> line){
        if (line[0] == '{'){
            // с каждым входом в блок увеличиваем перменную, чтобы разделять
блоки друг от друга, а при выходе наоборот уменьшаем.
            block++;
        } else if (line[0] == '}'){
            // когда мы выходим из стека, мы в основной стек возвращаем
значения, до входа в блок.
            for (auto p: stack_blocks[block]){
                stack[p.first] = p.second;
                //cout << "key = " << p.first << "; val = " << p.second << '\n';
            }
            //cout << "-------------" << '\n';
            stack_blocks[block].clear();
            block--;
        } else {
            string variable1 = "";
            int i = 0;
            // Считывание переменной (var1)
```

```
            for (i = 0; i < (int)line.size(); i++){
                if ('a' <= line[i] && line[i] <= 'z'){
                    variable1 += line[i];
                } else {
                    break;
                }
            }
            // Когда мы вводим перменную (var1), мы проверяем была ли она до
этого,
            // Если её не было то нужно задать 0 по дефолту.
            if (stack.count(variable1) == 0){
                stack[variable1] = 0;
            }               // Здесь лучше всего наверное сделать проверку на =,
            i++;            // Но по условию наверное не может быть такого,
чтобы на этой позиции было что-то другое
            int num;        // наше число в строке
            int sign = 1;   // знак числа в строке
            if (line[i] == '-'){
                sign = -1;
                i++;
            }
            // Считывание переменной (var2) или присвоение числа
            if ('0' <= line[i] && line[i] <= '9'){
                num = 0;
                for (; i < (int)line.size(); i++){
                    num = 10 * num + (line[i] - '0');
                }
                num *= sign;
            } else {
                string variable2 = "";
                for (; i < (int)line.size(); i++){
                    variable2 += line[i];
                }
                num = stack[variable2];
                cout << num << '\n';
            }
            //cout << variable1 + " = " << stack[variable1] << "  ->  " <<
variable1 << " = " << num << '\n';
            //Если текущей пременной нет в стеке для блоков, то мы его получаем
из основного стека,
            //Чтобы при выходе из блока, вернуть основной стек в порядок.
            if (stack_blocks[block].count(variable1) == 0){
                stack_blocks[block][variable1] = stack[variable1];
            }
            stack[variable1] = num;
        }
    }
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    solve();
    return 0;
}
```

Задача №D «Профессор Хаос»

Пояснение к примененному алгоритму:

Данная задача решается очень просто, выполняя все действия по условию, но тогда мы сталкиваемся с проблемой, что наш код слишком долго работает. Значит для решения этой проблемы мы должны подумать: “А какие частные случаи позволят решить её быстрее?”
1 случай – когда наших клеток, меньшем чем мы должны использовать для эксперимента, значит наш эксперимент может сразу закончиться (по условию)
2 случай – если мы достигли максимального значения в банке, значит дальнейшие вычисления нам не дадут никаких результатов и мы можем завершить выполнение.
3 случай – если значение у нас будет равно значению на предыдущем шаге, например для варианта когда у нас B = любому числу, а C = B – 1, то у нас будем всегда количество бактерий одно и тоже при каждой итерации, значит логично дальнейшие вычисления нам бесполезны.

Сложность алгоритма:

O(k) – в данной задаче, за количество итераций отвечает переменная k

Код:

```cpp
#include <iostream>

using namespace std;

void solve(){
    int a, b, c, d, k, a_last = 0;
    cin >> a >> b >> c >> d >> k;

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

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    solve();
    return 0;
}
```