Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Блок 1

Выполнил:
Студент группы P3216
Билошийцкий Михаил Владимирович

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

Санкт-Петербург
2024

A. Агроном-любитель

Решение:
Я использую два счетчика: один для подсчета количества цветков, которые стоят подряд, и второй для хранения самого длинного участка без трех одинаковых цветков. Как только находится подходящий участок, я сохраняю его начало и длину, проверяю с максимальным для результата и сохраняю, если он больше.
Алгоритм эффективен, так как выполняет один проход по массиву, не требует сортировки и работает за линейное время O(n), где n - количество цветков на грядке. Это достигается за счет использования одного цикла и проверки на равенство элементов внутри него. Также, алгоритм использует минимальное количество дополнительной памяти, только для хранения массива с цветами.
Дополнительные пояснения в коде:

```
#include <iostream>

// Функция проверки массива на равенство элементов
bool is_equal(int32_t* nums, size_t size) {
    for (size_t i = 0; i < size; i++) {
        if (nums[0] != nums[i]) {
            return false;
        }
    }
    return true;
}

int main(int argc, char** argv) {
    using namespace std;
    // Ввод данных
    size_t n;
    cin >> n;
    int32_t* row = new int32_t[n];
    for (size_t i = 0; i < n; i++) {
        cin >> row[i];
    }
    // Сколько цветков может стоять подряд
    const int32_t offset = 2;
    // Частный случай
    if (n <= offset) {
        cout << 1 << ' ' << n;
        return 0;
    }
```

```
    // Общий случай
    int32_t counter = offset, pos = 0, res_counter = offset, res_pos = 0;
    for (size_t i = offset; i < n; i++) {
        if (is_equal(&row[i - offset], offset + 1)) {
            counter = offset;
            pos = i - (offset - 1);
        } else {
            counter++;
        }
        if (res_counter < counter) {
            res_counter = counter;
            res_pos = pos;
        }
    }
    cout << (res_pos + 1) << ' ' << (res_pos + res_counter) << endl;
    delete[] row;
    return 0;
}
```

B. Зоопарк Глеба

Решение:

Я прохожу по каждому символу строки. Если символ маленькая буква (животное), я проверяю, есть ли в стеке ловушка для этого животного. Если есть, то я сохраняю порядковый номер животного в ловушке. Если нет, то я помещаю животное в стек. Если символ большая буква (ловушка), я проверяю, есть ли в стеке животное для этой ловушки. Если есть, то я сохраняю порядковый номер животного. Если нет, то я помещаю ловушку в стек. Использование общего стека для ловушек и животных как раз реализовывает механизм передвижения по кругу, чтобы никто не попался. Также алгоритм задачи очень похож на “правильную скобочную последовательность”.

После обработки всех символов, если стек пустой, значит все животные попали в свои ловушки, и я вывожу "Possible" и порядковые номера животных для каждой ловушки. Иначе я вывожу "Impossible".

Алгоритм эффективен, так как выполняет один проход по строке, использует стек для хранения символов, и обработка каждого символа выполняется за константное время. Также алгоритм использует линейное количество памяти относительно размера входной строки.

Дополнительные пояснения в коде:

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


// Либо животное, либо ловушка
// type - код символа
```

```
// number - порядковый номер в строке
struct zoo_object {
    int32_t type;
    size_t number;
};

int main(int argc, char** argv) {
    using namespace std;
    string line;
    cin >> line;
    // Стек для ловушек и животных
    stack<zoo_object> zoo_objects;
    // Счётчики животных и ловушек
    size_t animals_counter = -1, traps_counter = -1;
    // Массив-результат
    size_t* trapped_animals = new size_t[line.size() / 2];
    for (size_t i  = 0; i < line.size(); i++) {
        int32_t s = line[i];
        // В случае, если животное
        if (islower(s)) {
            animals_counter++;
            int32_t trap = toupper(s);
            if (!zoo_objects.empty() && zoo_objects.top().type == trap) {
                trapped_animals[zoo_objects.top().number] = animals_counter;
                zoo_objects.pop();
            } else {
                zoo_object a = { .type = s, .number = animals_counter };
                zoo_objects.push(a);
            }
        // В случае, если ловушка
        } else if (isupper(s)) {
            traps_counter++;
            int32_t animal = tolower(s);
            if (!zoo_objects.empty() && zoo_objects.top().type == animal) {
                trapped_animals[traps_counter] = zoo_objects.top().number;
                zoo_objects.pop();
            } else {
                zoo_object t = { .type = s, .number = traps_counter };
                zoo_objects.push(t);
```

```
            }
        }
    }
    // Все попались
    if (zoo_objects.empty()) {
        cout << "Possible" << endl;
        size_t n = line.size() / 2;
        for (size_t i = 0; i < n; i++) {
            cout << trapped_animals[i] + 1;
            if (i != n - 1) cout << ' ';
            else cout << endl;
        }
    } else {
        cout << "Impossible" << endl;
    }
    delete[] trapped_animals;
    return 0;
}
```

C. Конфигурационный файл

Решение:

Если строка является началом нового блока, уровень вложенности увеличивается и создается новый стек для хранения переменных в этом блоке. Если строка обозначает конец блока, уровень вложенности уменьшается, и происходит очистка стека переменных для этого блока.

Если строка содержит присваивание переменной числового значения, оно записывается в стек переменных текущего блока. Если строка содержит присваивание переменной значения другой переменной, выводится значение, которое было присвоено переменной в момент присваивания.

Этот алгоритм эффективен, так как обрабатывает строки по одной и использует стеки для хранения значений переменных, а также очищает их, что позволяет эффективно использовать память и нет ML на 15 тестах))).

Дополнительные пояснения в коде:

```
#include <iostream>
#include <stack>
#include <unordered_map>

bool is_number(std::string s);

int main(int argc, char** argv) {
    using namespace std;
    // Стек для хранения значений переменных в текущем блоке
```

```
    unordered_map< string, stack<string> > variable_stack;
    // Счётчик присваиваний на каждом слое
    stack<int32_t> values_counter_by_layer;
    // Имена переменных, присвоенных на каждом слое
    stack<string> variables_names_by_layer;

    // Начальный блок
    values_counter_by_layer.push(0);

    string line;
    while (getline(cin, line)) {
        // Начало нового блока
        if (line == "{") {
            values_counter_by_layer.push(0);
        // Конец блока
        } else if (line == "}") {
            int32_t values_counter = values_counter_by_layer.top();
            values_counter_by_layer.pop();
            // Очистка стека после завершения блока
            for (int32_t i = 0; i < values_counter; i++) {
                variable_stack[variables_names_by_layer.top()].pop();
                variables_names_by_layer.pop();
            }
        } else {
            // Новое присваивание значения
            values_counter_by_layer.top()++;
            size_t pos = line.find('=');
            string variable = line.substr(0, pos);
            // Новая переменная
            variables_names_by_layer.push(variable);
            string value = line.substr(pos + 1);
            if (is_number(value)) {
                variable_stack[variable].push(value);
            } else {
                // Получение значения
                string out_value = variable_stack[value].empty() ? "0" :
variable_stack[value].top();
                // Запись значенияs
                cout << out_value << endl;
```

```
                variable_stack[variable].push(out_value);
            }
        }
    }
    return 0;
}

// Проверка на число
bool is_number(std::string s) {
    return isdigit(s[0]) || (s[0] == '-' && isdigit(s[1]));
}
```

D. Профессор Хаос

Решение:

В цикле происходит выполнение одного дня эксперимента до достижения k-го дня, либо до тех пор, пока количество бактерий не станет равным 0 или не перестанет изменяться (это позволяет обработать случай, когда наступает стационарное состояние). На каждом шаге вызывается функция `do_experiment`, которая рассчитывает количество бактерий после одного дня эксперимента.

Функция `do_experiment` принимает количество бактерий, количество новых бактерий, количество бактерий для опытов и максимальное количество бактерий в контейнере. Она вычисляет количество бактерий после дня эксперимента, учитывая все условия.
Алгоритм эффективен, так как выполняет только один цикл с фиксированным количеством итераций (k дней эксперимента), и на каждой итерации выполняет простые операции вычисления количества бактерий. Также он не требует дополнительной памяти, кроме переменных для хранения входных данных и промежуточных результатов.

Дополнительные пояснения в коде:

```
#include <iostream>

// Функция одного дня эксперимента
int64_t do_experiment(int64_t a, int64_t b, int64_t c, int64_t d) {
    int64_t bac_counter = a * b - c;
    if (bac_counter <= 0) return 0;
    if (bac_counter > d) return d;
    return bac_counter;
}

int main(int argc, char** argv) {
    using namespace std;
    int64_t a, b, c, d, k;
    cin >> a >> b >> c >> d >> k;
```

```
    int64_t res = a, last_res = -1;
    // Прогон k дней эксперимента
    for (int64_t i = 0; i < k && res != 0 && res != last_res; i++) {
        res = do_experiment(last_res = res, b, c, d);
    }
    cout << res << endl;
    return 0;
}
```