



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”

**ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И
КОМПЬЮТЕРНОЙ ТЕХНИКИ**

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине
“АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ”
Базовые задачи.

выполнил:

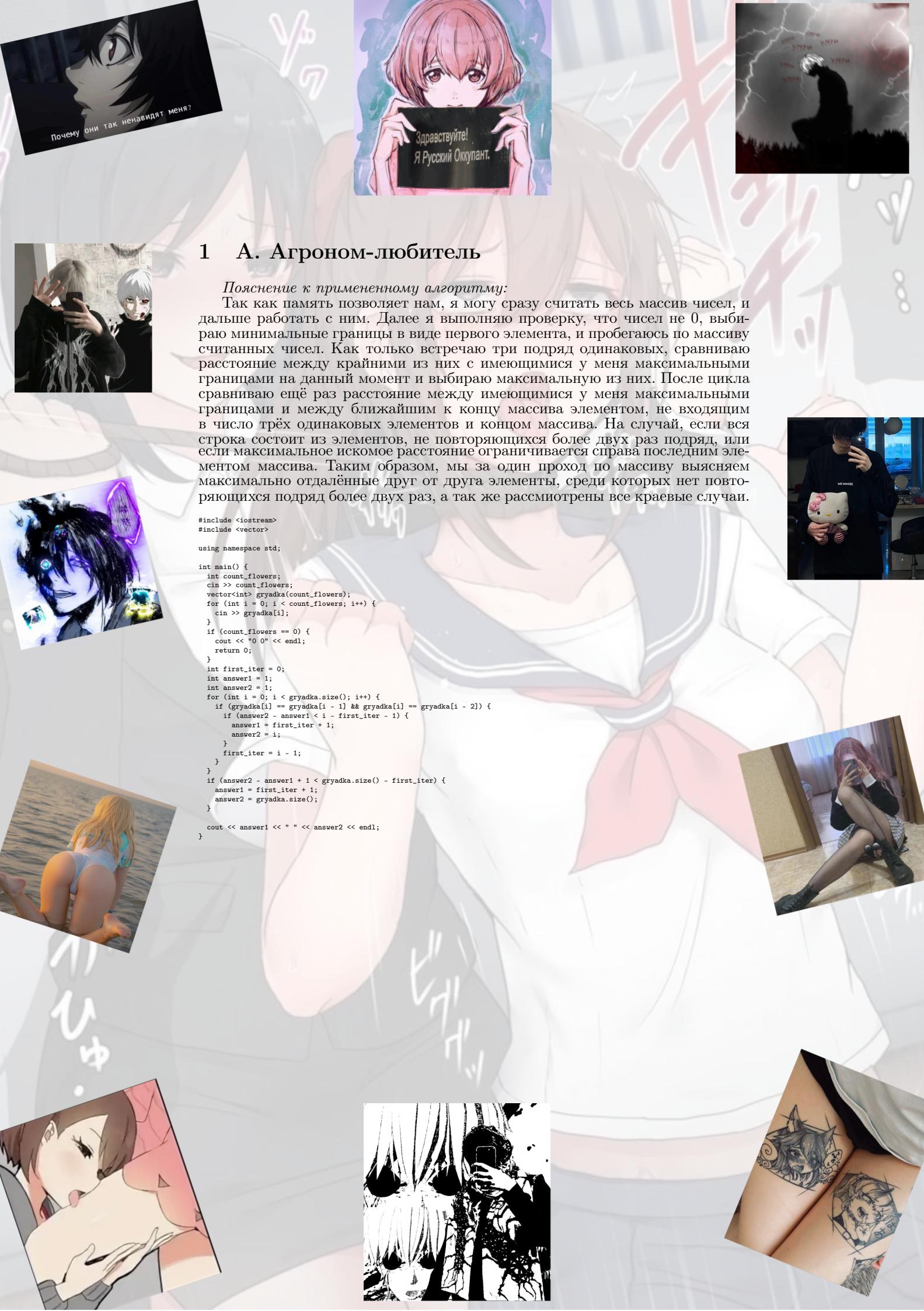
Студент группы Р32311
Птицын Максим Евгеньевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

г. Санкт-Петербург
2022 г.



1 А. Агроном-любитель

Пояснение к примененному алгоритму:

Так как память позволяет нам, я могу сразу считать весь массив чисел, и дальше работать с ним. Далее я выполняю проверку, что чисел не 0, выбираю минимальные границы в виде первого элемента, и пробегаюсь по массиву считанных чисел. Как только встречаю три подряд одинаковых, сравниваю расстояние между крайними из них с имеющимися у меня максимальными границами на данный момент и выбираю максимальную из них. После цикла сравниваю ещё раз расстояние между имеющимися у меня максимальными границами и между ближайшим к концу массива элементом, не входящим в число трёх одинаковых элементов и концом массива. На случай, если вся строка состоит из элементов, не повторяющихся более двух раз подряд, или если максимальное искомое расстояние ограничивается справа последним элементом массива. Таким образом, мы за один проход по массиву выясняем максимально отдалённые друг от друга элементы, среди которых нет повторяющихся подряд более двух раз, а так же рассмотрены все краевые случаи.

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

using namespace std;

int main() {
    int count_flowers;
    cin >> count_flowers;
    vector<int> gryadka(count_flowers);
    for (int i = 0; i < count_flowers; i++) {
        cin >> gryadka[i];
    }
    if (count_flowers == 0) {
        cout << "0" << endl;
        return 0;
    }
    int first_iter = 0;
    int answer1 = 1;
    int answer2 = 1;
    for (int i = 0; i < gryadka.size(); i++) {
        if (gryadka[i] == gryadka[i - 1] && gryadka[i] == gryadka[i - 2]) {
            if (answer2 - answer1 < i - first_iter - 1) {
                answer1 = first_iter + 1;
                answer2 = i;
            }
            first_iter = i - 1;
        }
    }
    if (answer2 - answer1 + 1 < gryadka.size() - first_iter) {
        answer1 = first_iter + 1;
        answer2 = gryadka.size();
    }

    cout << answer1 << " " << answer2 << endl;
}
```



2 В. Зоопарк Глеба

Пояснение к примененному алгоритму:

Читаем заданную строку, создаём несколько массивов:

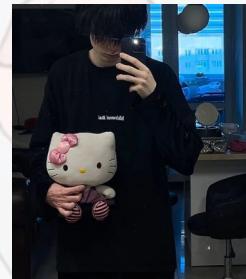
- массив индексов, куда будем кладь на место большой буквы индекс маленькой, ей соответствующей
- стек встречающихся больших букв
- стек, содержащий их индексы в исходной строке
- стек встречающихся маленьких букв
- стек, содержащий их порядковый номер среди маленьких букв
- общий стек для проверки, что между соответствующими друг другу маленькой и большой буквой не было других незакрытых пар.

Далее мы проходим по строке, и, если встречаем букву, проверяем, лежит ли на вершине стеков соответствующая буква:

- Если да, то "закрываем пару достаём нужную нам букву с вершин, а на порядковое место заглавной буквы кладём номер маленькой.
- Если на вершине одного из стека лежит не соответствующая буква, значит, пару закрыть не можем, и отправляем текущую букву на вершину соответствующего ей стека + вершину общего стека, а также записываем её порядковый номер в строке или номер по счёту среди маленьких на вершину соответствующего стека.

Таким образом, после прохода по всей строке, мы имеем :

- ✓ массив чисел, каждое ненулевое число которого находится по индексу "закрытой" заглавной буквы и содержит номер по счёту маленькой буквы, ей соответствующей.
- ✓ пустой стек маленьких чисел (если все пары можно "закрыть")
- ☒ пуст ли стек заглавных букв нам не важно, ибо наша задача поймать животных, а ловушки могут оставаться
- ☒ ровно как и общий стек может быть не пуст (если стек животных пуст, в нём могут содержаться ловушки без пар)
- ☒ пара закрывается только в случае, если между ними ничего нет, или есть пары, закрывающиеся друг с другом.





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

using namespace std;

int main() {
    vector<char> line;
    string str;
    getline(cin, str);
    if (str.size() == 0) {
        cout << "Impossible" << endl;
        return 0;
    }
    for (char i : str) {
        line.push_back(i);
    }
    vector<size_t> indexes(line.size());
    size_t lower_literal_count = 0;
    vector<char> upper_literals;
    vector<size_t> upper_literals_indexes;
    vector<char> lower_literals;
    vector<size_t> lower_literals_counts;
    vector<char> common_stack;
    for (size_t i = 0; i < line.size(); i++) {
        if (islower(line[i])) {
            lower_literal_count++;
            if (!upper_literals.empty() &&
                tolower(upper_literals.back()) == line[i] &&
                tolower(upper_literals.back()) == common_stack.back()) {
                indexes[upper_literals_indexes.back()] = lower_literal_count;
                upper_literals_indexes.pop_back();
                upper_literals.pop_back();
                common_stack.pop_back();
            }
        } else {
            lower_literals_counts.push_back(lower_literal_count);
            lower_literals.push_back(line[i]);
            common_stack.push_back(line[i]);
        }
    }
    else {
        if (!lower_literals.empty() &&
            tolower(lower_literals.back()) == lower_literals.back() &&
            lower_literals.back() == common_stack.back()) {
            lower_literals.pop_back();
            common_stack.pop_back();
            indexes[i] = lower_literals_counts.back();
            lower_literals_counts.pop_back();
        }
        else {
            upper_literals.push_back(line[i]);
            common_stack.push_back(line[i]);
            upper_literals_indexes.push_back(i);
        }
    }
    if (lower_literals.empty()) {
        cout << "Possible" << endl;
        for (size_t index : indexes) {
            if (index != 0)
                cout << index << " ";
        }
        cout << endl;
    } else {
        cout << "Impossible" << endl;
    }
    return 0;
}
```



3 С. Конфигурационный файл

Пояснение к примененному алгоритму:

Считываем инпут в массив построчно, создаём:

- мапу переменная-значение
- мапу, сохраняющую переменную и её старое значение при её изменении.
- мапу, сохраняющую переменную при её создании (ну, кстати, тут можно было не мап использовать)
- стек, содержащий мапы с изменениями (разделены по блокам)
- стек, содержащий мапы с созданиями
- все мапы неупорядоченные для скорости

Далее проходим по порядку по массиву считанных строк. Когда встречаем открытие блока, кидаем текущие мапы изменений и созданий переменных на стек. Когда блок закрывается, возвращаем старые значения переменным из истории изменений, и удаляем из основной мапы переменные, созданные в блоке. После чего достаём со стека историю предыдущего блока и продолжаем её. Если встречаем строку, отличную от скобок, то парсим её (простой алгоритм ветвления), выводим значение переменной при присваивании. Таким образом, в каждый момент времени мы имеем корректную информацию о значениях переменных, а стек историй действует эффективно, что позволяет проходить Memory Limit.





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

using namespace std;

bool check_var_or_not(string s);

int64_t get_delim_pos(string stroka) {
    for (size_t i = 0; i < stroka.size(); i++) {
        char symbol = stroka.at(i);
        if (symbol == 61) {
            return i;
        }
    }
    return 0;
}

int main() {
    vector<string> text;
    unordered_map<string, string> mymap;
    unordered_map<string, string> history_changes;
    unordered_map<string, string> historyCreates;
    stack<unordered_map<string, string>> stackChanges;
    stack<unordered_map<string, string>> stackCreates;
    string line;
    while (cin >> line) {
        text.push_back(line);
    }
    for (string stroka: text) {
        if (stroka == "{") {
            stackChanges.push(std::move(historyChanges));
            stackCreates.push(std::move(historyCreates));
            historyChanges.clear();
            historyCreates.clear();
        } else if (stroka == "}") {
            for (pair<string, string> pair: historyChanges) {
                mymap[pair.first] = pair.second;
            }
            for (pair<string, string> pair: historyCreates) {
                mymap.erase(pair.first);
            }
            historyChanges = (std::move(stackChanges.top()));
            stackChanges.pop();
            historyCreates = (std::move(stackCreates.top()));
            stackCreates.pop();
        } else {
            int64_t delim_pos = get_delim_pos(stroka);
            string left_value = (stroka.substr(0, delim_pos));
            string right_value = (stroka.substr(delim_pos + 1, stroka.size() - 1));
            auto left_value_pos = mymap.find(left_value);
            auto pos_in_history = historyChanges.find(left_value);
            if (left_value_pos != mymap.end() && pos_in_history == historyChanges.end())
                historyChanges.insert({left_value_pos->first, left_value_pos->second});
            if (left_value_pos == mymap.end()) historyCreates.insert({left_value, right_value});
            bool value_is_a_var = check_var_or_not(right_value);
            if (value_is_a_var) {
                auto iter = mymap.find(right_value);
                if (iter == mymap.end()) {
                    mymap[left_value] = "0";
                    cout << "0" << endl;
                } else {
                    mymap[left_value] = iter->second;
                    cout << iter->second << endl;
                }
            } else {
                mymap[left_value] = right_value;
            }
        }
    }
    return 0;
}

bool check_var_or_not(string s) {
    char first_symbol = s.at(0);
    if ((first_symbol == 45) || (first_symbol >= 48 && first_symbol <= 57)) {
        return false;
    }
    return true;
}
```



4 D. Профессор Хаос

Пояснение к примененному алгоритму:

Просто проходим по описанному алгоритму $\min(1000, k)$ раз и выводим ответ. 1000 раз будет достаточно, потому что вместимость $d \leq 1000$, если $a^*b - c$ будет положительным, то количество бактерий будет увеличиваться минимум на единицу, и за 1000 итераций мы дойдём до максимума, а если оно хоть раз обратится в 0, то будет всегда равным 0.

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

int main() {
    int64_t a, result, b, c, d;
    uint64_t k;
    cin >> a >> b >> c >> d >> k;
    result = a;
    for (size_t i = 0; i < 1000 && i < k; i++) {
        a = result;
        result = a * b - c;
        if (result <= 0) {
            result = 0;
            break;
        } else {
            result = min(result, d);
        }
    }
    cout << result << endl;
    return 0;
}
```

