

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

**Алгоритмы и структуры данных**

**Отчет по первому блоку задач (Яндекс.Констест)**

Выполнил:

Марьин Григорий Алексеевич

Группа Р3212

Проверил:

Голиков Андрей Сергеевич

Санкт-Петербург

2026

## А. Агроном-любитель

### **Суть задачи:**

Нужно найти наибольшую последовательность чисел, в которой нет 3х подряд идущих одинаковых чисел.

### **Решение:**

Начнем рассматривать цветки и, начиная с третьего, будем смотреть на два предыдущих. Если два предыдущих не совпадают с тем, что мы рассматриваем сейчас, то все хорошо, иначе мы нашли три одинаковых цветка подряд, что недопустимо в искомой последовательности. Встретив три одинаковых цветка подряд, нужно пересчитать ответ. Посчитаем длину найденного отрезка, затем сравним его с максимумом и обозначим левую границу нового отрезка, ей будет правая граница предыдущего. Таким образом мы обрабатываем все цветки, алгоритм отслеживает максимальную удовлетворяющую последовательность на протяжении всего процесса. Значит алгоритм гарантирует, что к окончанию его работы он выдаст границы максимальной последовательности цветков, удовлетворяющей условиям.

### **Алгоритмическая сложность:**

- **Время:** Алгоритм обрабатывает каждый цветок по 1 разу. Линейное время. O(n)
- **Память:** Хранится всегда одинаковое количество переменных. Константная память. O(1)

### **Код:**

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

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> v(n);
    for (int i = 0; i < n; i++) {
        cin >> v[i];
    }
    int max_len = 0;
    int cl = 0;
    int l = 0, r = 0;
    for (int i = 0; i < n; i++) {
        if (i >= 2 && v[i] == v[i - 1] && v[i] == v[i - 2]) {
            int dl = i - cl;
            if (dl > max_len) {
                max_len = dl;
                l = cl;
                r = i - 1;
            }
            cl = i - 1;
        }
    }
    if (n - cl > max_len) {
        max_len = n - cl;
    }
}
```

```

    l = cl;
    r = n - 1;
}
cout << l + 1 << ' ' << r + 1;
return 0;
}

```

## В. Зоопарк Глеба

### **Суть задачи:**

Определить можно ли провести непересекающиеся прямые, соединяющие одинаковые буквы, внутри окружности, так, чтобы они не пересекались.

### **Решение:**

Будем идти по строке и постепенно считать ловушки и животных. Когда мы встречаем очередной символ, то сравниваем его с предыдущим. Если это оказалась не одна буква, то добавляем его в стек. Если совпали, то нужно запомнить индекс. Какая по счету ловушка - такой и номер животного по порядку. Если по итогу стек будет пустой, то можно соединить, если не пустой, то нельзя.

### **Алгоритмическая сложность:**

Время:  $O(n)$ , по каждому символу пробегаем 1 раз.

Память:  $O(n)$ , в худшем случае все символы запишутся в стек.

### **Код:**

```

#include <bits/stdc++.h>

using namespace std;

int main() {
    string s;
    cin >> s;
    stack<int> traps;
    stack<int> animals;
    stack<char> st;
    vector<int> ans = vector<int>(s.length() / 2, 0);
    int trap_count = 0, animal_count = 0;
    for (size_t i = 0; i < s.length(); i++) {
        if (isupper(s[i])) {
            trap_count++;
            traps.push(trap_count);

```

```

} else {
    animal_count++;
    animals.push(animal_count);
}

if (st.empty() || !(toupper(s[i]) == toupper(st.top()) &&
s[i] != st.top())) {
    st.push(s[i]);
} else {
    ans[traps.top() - 1] = animals.top();
    animals.pop();
    traps.pop();
    st.pop();
}
}

if (!st.empty()) {
    cout << "Impossible";
    return 0;
}
cout << "Possible" << '\n';
for (auto x : ans) {
    cout << x << ' ';
}
return 0;
}

```

## С. Конфигурационный файл

### **Суть задачи:**

Присваивать и определять значение переменной на различных уровнях вложенности.

### **Решение:**

В первом решении (которое оказалось медленным) я решил хранить переменные в map, где ключом будет переменная, а значением пара (уровень вложенности и значение). Данное решение оказалось медленным, тк для очистки одного уровня приходилось пробежаться по всем переменным и проверять уровень вложенности. Получалась асимптотика всей программы примерно  $O(n^2)$ .

К своему решению я добавил “хранилище уровня”, в котором видно какие именно переменные были изменены на i-том уровне, это ускоряло очистку уровня до линейной сложности.

Работа алгоритма: При встрече открывающей скобки увеличиваем счетчик уровня. При встрече закрывающей, уменьшаем счетчик. Если мы встречаем оператор присваивания, то запоминаем уровень, на котором оно произошло и выполняем операцию присвоения.

### **Алгоритмическая сложность:**

- **Время:** n-количество строк. Всего алгоритм работает n раз, но в каждой итерации может быть 3 случая:
  1. Открывающая скобка:  $O(1)$ , так просто происходит увеличение счетчика.
  2. Закрывающая скобка:  $O(t)$ , где t - количество переменных, объявленных внутри блока
  3. Присваивание значения:  $O(\log(t))$ , где t - количество элементов map.
- **Память:**  $O(n)$  в худшем случае, если будут храниться значения всех переменных всегда.

### **Код:**

```
#include <bits/stdc++.h>

using namespace std;

pair<string, string> splitEqual(const string& s) {
    size_t pos = s.find('=');
    return {s.substr(0, pos), s.substr(pos + 1)};
}
```

```

}

bool isNumber(const string& s) {
    for (char c : s) {
        if (!(isdigit(c) || (c == '-'))) {
            return false;
        }
    }
    return true;
}

void clearLevel(
    unordered_map<string, stack<pair<int, string>>>& variables,
    vector<string>& current_level
) {
    for (const string& var : current_level) {
        if (!variables[var].empty()) {
            variables[var].pop();
        }
    }
    current_level.clear();
}

void setValue(
    unordered_map<string, stack<pair<int, string>>>& variables,
    const string& var,
    const string& val,
    int lvl,
    vector<vector<string>>& level
) {
    string new_val = "0";
    if (!variables[val].empty()) {
        new_val = variables[val].top().second;
    } else {
        variables[val].emplace(lvl, "0");
        level[lvl].push_back(val);
    }
}

```

```

variables[var].emplace(lvl, new_val);
level[lvl].push_back(var);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    string s;
    unordered_map<string, stack<pair<int, string>>> variables;
    vector<vector<string>> level(1e5 + 1);
    int lvl = 0;
    while (getline(cin, s)) {
        if (s == "{") {
            lvl++;
        } else if (s == "}") {
            clearLevel(variables, level[lvl]);
            lvl--;
        } else {
            pair<string, string> p = splitEqual(s);
            if (!isNumber(p.second)) {
                setValue(variables, p.first, p.second, lvl, level);
                cout << variables[p.first].top().second << endl;
            } else {
                variables[p.first].emplace(lvl, p.second);
                level[lvl].push_back(p.first);
            }
        }
    }
    return 0;
}

```

## D. Профессор Хаос

**Суть задачи:**

Найти количество бактерий в банке после k-того дня.

**Решение:**

Рассмотрим как изменяется количество бактерий в банке:  $a = a * b - c$ . Может быть 3 исхода. Первый самый простой: число бактерий не изменилось. Значит мы зациклены, значит количество бактерий всегда будет одним под конец дня. Если число бактерий стало больше, то значит на следующий день их будет еще больше. Значит мы либо дойдем до ограничения банки d или дни кончатся. Количество бактерий растет как степенная функция, а значит мы достигнем максимума за  $\log(d)$  дней, что укладывается во временные ограничения. Если после окончания дня число бактерий уменьшилось, то мы приближаемся к нулю. А значит либо дни закончатся, либо бактерии кончатся. В любом случае мы придем к ответу.

**Алгоритмическая сложность:**

- **Время:** в худшем случае  $O(d)$
- **Память:** константная  $O(1)$

**Код:**

```
#include <bits/stdc++.h>

using namespace std;
#define ull unsigned long long
#define ll long long
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    ll a, b, c, d;
    ll k;
    cin >> a >> b >> c >> d >> k;
    ll tmp = -9;
    while (k > 0) {
        a *= b;
        a -= c;
        if (a <= 0) {
```

```
a = 0;
break;
}
if (a >= d) {
    a = d;
}
if (a == tmp) {
    break;
}
tmp = a;
k--;
}
cout << a;
return 0;
}
```