Университет ИТМО

Факультет программной инженерии и компьютерной техники

Отчет по задачам 1-го блока

по предмету «Алгоритмы и структуры данных»

Выполнил:

Студент группы P3230

Гаврилин Олег Сергеевич

Преподаватели:

Тараканов Денис Сергеевич

Санкт-Петербург

2025

1. **Агроном-любитель**

   Код решения:

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

using namespace std;
void solve() {
    int n;
    cin >> n;
    vector<int> a(n);
    int begin = 0, end = n == 1 ? 0 : 1;
    int begin_max = 0, end_max = end;

    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        if (i > 1) {
            if (a[i] == a[i - 1] && a[i] == a[i - 2]) {
                if (end - begin > end_max - begin_max) {
                    begin_max = begin;
                    end_max = end;
                }
                begin = i - 1;
            } else {
                end = i;
            }
        }
    }

    if (end - begin > end_max - begin_max) cout << ++begin << " " <<
++end;
    else cout << ++begin_max << " " << ++end_max;
}

int main(){
    ios::sync_with_stdio(false); cin.tie(NULL);
    int t = 1;
    while (t--) {
        solve();
    }
}
```

   Описание: Решение основано на сравнении трех подряд идущих элементов массива, при выполнении их равенства сдвигаем отслеживаемые индексы на новый отрезок, и обновляем максимальные значения.

   Сложность по времени: O(n)

   Сложность по памяти: O(n)

2. **Зоопарк Глеба**

Код решения:

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

using namespace std;
void solve() {
    string str;
    cin >> str;
    stack<char> current;
    stack<int> animals;
    stack<int> traps;
    map<int, int> result;
    int animals_count = 0;
    int traps_count = 0;
    for (size_t i = 0; i < str.length(); ++i) {
        if (!current.empty()) {
            char top = current.top();
            if (std::islower(str[i])) {
                animals_count++;
                if (std::toupper(str[i]) == top) {
                    current.pop();
                    result[traps.top()] = animals_count;
                    traps.pop();
                } else {
                    current.push(str[i]);
                    animals.push(animals_count);
                }
            } else {
                traps_count++;
                if (!animals.empty() && tolower(str[i]) == top) {
                    current.pop();
                    result[traps_count] = animals.top();
                    animals.pop();
                } else {
                    current.push(str[i]);
                    traps.push(traps_count);
                }
            }
        } else {
            if (std::islower(str[i])) animals.push(++animals_count);
            else traps.push(++traps_count);
            current.push(str[i]);
        }
    }
    if (!current.empty()) {
        cout << "Impossible";
        return;
```

```
    }
    cout << "Possible\n";
    for (auto item : result) {
        cout << item.second << " ";
    }
}
int main(){
    ios::sync_with_stdio(false); cin.tie(NULL);
    int t = 1;
    while (t--) {
        solve();
    }
}
```

Описание: Алгоритм использует стеки для контроля последовательностей животных и ловушек в строке, в map result хранятся пары индексов соответствующей ловушки и животного к ней.
Сложность по времени: O(n)
Сложность по памяти: O(n)

3. **Конфигурационный файл**
   Код решения:

```
#include <iostream>
#include <queue>
#include <stack>
#include <map>

using namespace std;
void solve() {
    map<string, stack<int>> vars;
    stack<vector<string>> scopes;
    scopes.emplace();

    string line;
    while (getline(cin, line)) {
        if (line == "{") {
            scopes.emplace();
        } else if (line == "}") {
            for (auto item : scopes.top()) vars[item].pop();
            scopes.pop();
        } else {
            string variable = line.substr(0, line.find('='));
            string value = line.substr(line.find('=') + 1);
            int number;
            if (isdigit(value[0]) || value[0] == '-') number =
stoi(value);
```

```
            else {
                if (!vars.contains(value) || vars[value].empty())
number = 0;
                else number = vars[value].top();
                cout << number << endl;
            }
            scopes.top().push_back(variable);
            vars[variable].push(number);
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int t = 1;
    while (t--) {
        solve();
    }
}
```

Описание: Алгоритм использует стек массивов для имитации областей видимости переменных, ограниченных фигурными скобками, при встрече одной из скобок, область видимости добавляется или очищается переменные соответственно. Хранение переменных и истории их значений реализовано при помощи `map<string, stack<int>>` - это позволяет находить значения переменных в разных областях видимости.
Сложность по времени: $O(n^2)$
Сложность по памяти: $O(n)$

4. **Профессор Хаос**
   Код решения:

```
#include <cstdint>
#include <iostream>
int64_t solve() {
    uint16_t b, c;
    int64_t a, d, k;
    std::cin >> a >> b >> c >> d >> k;
    int64_t prev = -1;
    while (k--) {
        a = a * b - c;
        if (a <= 0) return 0;
        if (a > d) return d;
        if (prev == a) return a;
```

```
        prev = a;
    }
    return a;
}
int main() {

std::ios::sync_with_stdio(false);std::cin.tie(nullptr);std::cout.tie(nu
llptr);
    std::cout << solve();
}
```

Описание: Алгоритм вычисляет новое значение бактерий на каждый день и отслеживает соответствие поставленным условиям. Также отслеживается стабилизация показаний количества бактерий – если количество два дня подряд одинаково, то далее можно не смотреть.
Сложность по времени: O(n)
Сложность по памяти: O(1)

5. **Timus 1296**

Код решения:

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

void solve() {
    int n;
    std::cin >> n;
    int sum = 0;
    int result = 0;
    int current;
    for (int i = 0; i < n; i++) {
        std::cin >> current;
        sum = std::max(current, sum + current);
        result = std::max(result, sum);
    }
    std::cout << result << '\n';
}
int main() {

std::ios::sync_with_stdio(false);std::cin.tie(nullptr);std::cout.tie(nu
llptr);
    int t = 1;
    while (t--) {
        solve();
    }
}
```

Описание: Решение основано на алгоритме Кадана для поиска под последовательности с наибольшей суммой элементов.
Сложность по времени: O(n)
Сложность по памяти: O(1)

6. **Timus 1155**

Код решения:

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

void printMoves(const string& moves) {
    cout << moves;
}

int main() {
    int a, b, c, d, e, f, g, h;
    cin >> a >> b >> c >> d >> e >> f >> g >> h;

    if ((f + h + a + c) != (d + g + e + b)) {
        cout << "IMPOSSIBLE" << endl;
        return 0;
    }

    for (int i = 0; i < 100; i++) {
        if (a > 0 && b > 0) { printMoves("AB-\n"); a--; b--; }
        if (a > 0 && d > 0) { printMoves("AD-\n"); a--; d--; }
        if (a > 0 && e > 0) { printMoves("AE-\n"); a--; e--; }
        if (a > 0 && g > 0) { printMoves("FB+\nFG-\nAB-\n"); a--; g--;
}

        if (h > 0 && e > 0) { printMoves("HE-\n"); h--; e--; }
        if (h > 0 && d > 0) { printMoves("HD-\n"); h--; d--; }
        if (h > 0 && g > 0) { printMoves("HG-\n"); h--; g--; }
        if (h > 0 && b > 0) { printMoves("FG+\nHG-\nFB-\n"); h--; b--;
}

        if (f > 0 && b > 0) { printMoves("FB-\n"); f--; b--; }
        if (f > 0 && g > 0) { printMoves("FG-\n"); f--; g--; }
        if (f > 0 && e > 0) { printMoves("FE-\n"); f--; e--; }
        if (f > 0 && d > 0) { printMoves("HG+\nHD-\nFG-\n"); f--; d--;
}

        if (c > 0 && b > 0) { printMoves("CB-\n"); c--; b--; }
        if (c > 0 && d > 0) { printMoves("CD-\n"); c--; d--; }
        if (c > 0 && g > 0) { printMoves("CG-\n"); c--; g--; }
        if (c > 0 && e > 0) { printMoves("HG+\nEH-\nCG-\n"); c--; e--;
}
```

```
    }

    return 0;
}
```

Описание: Основная идея лежит в рассмотрении сумм не смежных точек – опираясь на это можно точно сделать вывод о том является ли задача выполнимой или нет. Далее внутри цикла уничтожаем дуоны, рассматривая все смежно лежащие из них. Также в конце проверки каждой тройки точек я просматриваю точки, лежащие по диагонали куба (a и g в первом случае, например), это позволяет добавлять более сложные последовательности шагов для удаления дуонов на не смежных точках.
Сложность по времени: O(1)
Сложность по памяти: O(1)