Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

# Решения задач блока 4 тимус

# Алгоритмам и структурам данных

Работу выполнил:

Гаврилин О.С.

Группа:

P3230

Санкт-Петербург,

2025

1. **1806**

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
#include <string>
#include <algorithm>

using namespace std;

constexpr long long INFINITE = 1e18;

int totalDevices;
vector<int> prefixTime(10);
vector<string> deviceNumbers;
unordered_map<string, int> numberToIndex;
vector<long long> fastestPath;
vector<int> previousDevice;

void readInput() {
    cin >> totalDevices;
    for (int i = 0; i < 10; ++i) {
        cin >> prefixTime[i];
    }
    deviceNumbers.resize(totalDevices);
    for (int i = 0; i < totalDevices; ++i) {
        cin >> deviceNumbers[i];
        numberToIndex[deviceNumbers[i]] = i;
    }
}

int commonPrefixLength(const string& a, const string& b) {
    int match = 0;
    while (match < 10 && a[match] == b[match]) {
        ++match;
    }
    return match;
}

void tryUpdate(int from, const string& candidateNumber,
priority_queue<pair<long long, int>, vector<pair<long long,
int>>, greater<>>& pq) {
    auto it = numberToIndex.find(candidateNumber);
    if (it == numberToIndex.end()) return;
    int to = it->second;
```

```
    if (to == from) return;

    int prefix = commonPrefixLength(deviceNumbers[from],
deviceNumbers[to]);
    long long cost = prefixTime[prefix];

    if (fastestPath[to] > fastestPath[from] + cost) {
        fastestPath[to] = fastestPath[from] + cost;
        previousDevice[to] = from;
        pq.emplace(fastestPath[to], to);
    }
}

void searchPaths() {
    fastestPath.assign(totalDevices, INFINITE);
    previousDevice.assign(totalDevices, -1);

    priority_queue<pair<long long, int>, vector<pair<long
long, int>>, greater<>> pq;
    fastestPath[0] = 0;
    pq.emplace(0, 0);

    while (!pq.empty()) {
        auto [timeSpent, current] = pq.top();
        pq.pop();

        if (timeSpent > fastestPath[current]) continue;

        string original = deviceNumbers[current];

        for (int pos = 0; pos < 10; ++pos) {
            char saved = original[pos];
            for (char newChar = '0'; newChar <= '9';
++newChar) {
                if (newChar == saved) continue;
                original[pos] = newChar;
                tryUpdate(current, original, pq);
            }
            original[pos] = saved;
        }

        for (int i = 0; i < 10; ++i) {
            for (int j = i + 1; j < 10; ++j) {
                if (original[i] == original[j]) continue;
                swap(original[i], original[j]);
```

```
                tryUpdate(current, original, pq);
                swap(original[i], original[j]);
            }
        }
    }
}

void outputResult() {
    if (fastestPath[totalDevices - 1] == INFINITE) {
        cout << "-1\n";
        return;
    }

    cout << fastestPath[totalDevices - 1] << '\n';

    vector<int> path;
    int position = totalDevices - 1;
    while (position != -1) {
        path.push_back(position + 1);
        position = previousDevice[position];
    }
    reverse(path.begin(), path.end());

    cout << path.size() << '\n';
    for (int fighter : path) {
        cout << fighter << ' ';
    }
    cout << '\n';
}

void solve() {
    readInput();
    searchPaths();
    outputResult();
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    solve();
}
```

**Описание:** У каждого устройства есть 10-значный номер. Можно менять одну цифру или переставлять две разные цифры.

Если из одного номера можно получить другой одной такой операцией, между ними считается переход.

Время перехода зависит от того, сколько цифр у номеров совпадает в начале — для этого есть таблица prefixTime.

При помощи Дейкстры: стартуем с устройства 0, пробуем все допустимые изменения номера и обновляем кратчайшие пути.

**Сложность по времени: O (n log(n))**

**Сложность по памяти: O (n)**

2. **1329**

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

using namespace std;

int numberOfEvents, numberOfQueries;
unordered_map<int, vector<int>> treeStructure;
unordered_map<int, int> timeIn, timeOut;
int globalTimer = 0;

void addEdge(int parent, int child) {
    treeStructure[parent].push_back(child);
}

void dfsTraversal(int startingNode) {
    stack<pair<int, int>> dfsStack;
    dfsStack.push({startingNode, 0});

    while (!dfsStack.empty()) {
        auto [currentNode, phase] = dfsStack.top();
        dfsStack.pop();

        if (phase == 0) {
            timeIn[currentNode] = globalTimer++;
            dfsStack.push({currentNode, 1});
            for (auto it =
treeStructure[currentNode].rbegin(); it !=
treeStructure[currentNode].rend(); ++it) {
                dfsStack.push({*it, 0});
            }
        } else {
```

```
            timeOut[currentNode] = globalTimer++;
        }
    }
}

bool isAncestor(int ancestor, int descendant) {
    return timeIn[ancestor] <= timeIn[descendant] &&
timeOut[ancestor] >= timeOut[descendant];
}

int evaluateRelation(int a, int b) {
    if (isAncestor(a, b)) return 1;
    if (isAncestor(b, a)) return 2;
    return 0;
}

void readTree() {
    vector<int> rootNodes;
    cin >> numberOfEvents;
    for (int i = 0; i < numberOfEvents; ++i) {
        int nodeId, parentId;
        cin >> nodeId >> parentId;
        if (parentId == -1) {
            rootNodes.push_back(nodeId);
        } else {
            addEdge(parentId, nodeId);
        }
    }
    for (int root : rootNodes) {
        dfsTraversal(root);
    }
}

void processQueries() {
    cin >> numberOfQueries;
    while (numberOfQueries--) {
        int eventA, eventB;
        cin >> eventA >> eventB;
        cout << evaluateRelation(eventA, eventB) << '\n';
    }
}

void solve() {
    readTree();
    processQueries();
```

```
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    solve();
}
```

**Описание:** Дано дерево событий (у каждого события может быть родитель). Нужно быстро отвечать на вопросы: является ли одно событие предком другого.
Для этого мы запускаем обход дерева и для каждого события сохраняем время входа и выхода.
Тогда событие a — предок события b, если timeIn[a] <= timeIn[b] и timeOut[a] >= timeOut[b].
На каждый запрос просто проверяем через сохранённые времена.

**Сложность по времени: O (n + q)**

**Сложность по памяти: O (n)**