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

Алгоритмам и структурам данных

Работу выполнил:

Гаврилин О.С.

Группа:

P3230

Санкт-Петербург,

2025

1. **M**

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

using namespace std;

void solve() {
    int n, m;
    cin >> n >> m;
    int start_x, start_y, end_x, end_y;
    cin >> start_x >> start_y >> end_x >> end_y;
    --start_x, --start_y, --end_x, --end_y;

    vector<string> grid(n);
    for (auto& row : grid)
        cin >> row;

    const int INF = 1e9;
    vector<vector<int>> cost(n, vector<int>(m, INF));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cost[i][j] = (grid[i][j] == '.') ? 1 :
(grid[i][j] == 'W' ? 2 : INF);

    vector<vector<int>> dist(n, vector<int>(m, INF));
    vector<vector<pair<int, int>>> prev(n, vector<pair<int,
int>>(m, {-1, -1}));

    priority_queue<pair<int, pair<int, int>>,
vector<pair<int, pair<int, int>>>, greater<>> pq;
    dist[start_x][start_y] = 0;
    pq.push({
        0, {start_x, start_y}
    });

    int dx[4] = {-1, 0, 1, 0};
    int dy[4] = {0, 1, 0, -1};
    char move_dir[4] = {'N', 'E', 'S', 'W'};

    while (!pq.empty()) {
        auto [d, pos] = pq.top();
        auto [x, y] = pos;
```

```
        pq.pop();
        if (d > dist[x][y])
            continue;

        for (int i = 0; i < 4; ++i) {
            int nx = x + dx[i], ny = y + dy[i];
            if (nx >= 0 && nx < n && ny >= 0 && ny < m &&
cost[nx][ny] != INF) {
                int nd = d + cost[nx][ny];
                if (nd < dist[nx][ny]) {
                    dist[nx][ny] = nd;
                    prev[nx][ny] = {x, y};
                    pq.push({
                        nd, {nx, ny}
                    });
                }
            }
        }
    }

    if (dist[end_x][end_y] == INF) {
        cout << -1 << '\n';
        return;
    }

    cout << dist[end_x][end_y] << '\n';
    string path;
    int x = end_x, y = end_y;
    while (x != start_x || y != start_y) {
        auto [px, py] = prev[x][y];
        for (int i = 0; i < 4; ++i)
            if (px + dx[i] == x && py + dy[i] == y)
                path += move_dir[i];
        x = px, y = py;
    }
    reverse(path.begin(), path.end());
    cout << path << '\n';
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    solve();
    return 0;
}
```

Описание: Имеем взвешенный граф, точку начала и конца пути – при помощи алгоритма Дейкстры можем оценить наименьшую цену попадания в каждую вершину графа из исходной точки. Для увеличения эффективности алгоритма используем приоритетную очередь.

Сложность по времени: O (nm log(nm))

Сложность по памяти: O (nm)

2. **N**

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

using namespace std;

int n;
vector<int> destination;
vector<bool> explored;

void readData() {
    cin >> n;
    destination.resize(n + 1);
    explored.assign(n + 1, false);
    for (int i = 1; i <= n; ++i) {
        cin >> destination[i];
    }
}

bool traverse(int start) {
    vector<bool> inCurrentPath(n + 1, false);
    int current = start;
    while (!explored[current]) {
        explored[current] = true;
        inCurrentPath[current] = true;
        current = destination[current];
    }
    return inCurrentPath[current];
}

void solve() {
    readData();
    int smashed = 0;
    for (int pig = 1; pig <= n; ++pig) {
```

```
        if (!explored[pig]) {
            if (traverse(pig)) {
                smashed++;
            }
        }
    }
    cout << smashed << '\n';
}

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

Описание: Суть решения заключается в нахождении циклов в ориентированном графе, где каждая вершина (свинка) имеет ровно одно исходящее ребро (прыжок). Итоговый ответ — это количество различных циклов, обнаруженных в этом графе.

Сложность по времени: O (n)

Сложность по памяти: O (n)

3. **O**

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

using namespace std;

int numberOfStudents, numberOfNotes;
vector<vector<int>> communication;
vector<int> teamAssignment;

void readInput() {
    cin >> numberOfStudents >> numberOfNotes;
    communication.assign(numberOfStudents + 1, {});
    teamAssignment.assign(numberOfStudents + 1, -1);

    for (int i = 0; i < numberOfNotes; ++i) {
        int from, to;
        cin >> from >> to;
        communication[from].push_back(to);
        communication[to].push_back(from);
```

```
    }
}

bool assignTeams(int start) {
    stack<int> pending;
    pending.push(start);
    teamAssignment[start] = 0;

    while (!pending.empty()) {
        int current = pending.top();
        pending.pop();

        for (int neighbor : communication[current]) {
            if (teamAssignment[neighbor] == -1) {
                teamAssignment[neighbor] = 1 -
teamAssignment[current];
                pending.push(neighbor);
            } else if (teamAssignment[neighbor] ==
teamAssignment[current]) {
                return false;
            }
        }
    }
    return true;
}

bool validateGrouping() {
    for (int student = 1; student <= numberOfStudents;
++student) {
        if (teamAssignment[student] == -1) {
            if (!assignTeams(student)) {
                return false;
            }
        }
    }
    return true;
}

void solve() {
    readInput();
    if (validateGrouping()) {
        cout << "YES\n";
    } else {
        cout << "NO\n";
    }
```

```
}

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

Описание: Суть решения заключается в проверке, можно ли разбить лкшат на две команды так, чтобы любые два лкшонка, между которыми есть прямая связь (записка), оказались в разных командах.

Для этого задача сводится к проверке, является ли граф двудольным: студенты — вершины, записки — рёбра. Алгоритм осуществляет покраску графа в два цвета при помощи обхода в глубину (реализованного через стек). Если при попытке покраски возникает конфликт (двое связанных студентов оказываются в одной команде), то ответ "NO"; иначе — "YES".

Сложность по времени: O (n + m)

Сложность по памяти: O (n + m)

4. P

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

using namespace std;

int n;
vector<vector<int>> fuel;
vector<bool> visited;

void dfs(int u, int limit, const vector<vector<int>>& graph) {
  visited[u] = true;
  for (int v = 0; v < n; v++) {
    if (!visited[v] && graph[u][v] <= limit) {
      dfs(v, limit, graph);
    }
  }
}

bool isConnected(int limit) {
  visited.assign(n, false);
```

```
  dfs(0, limit, fuel);
  for (bool v : visited) {
    if (!v)
      return false;
  }

  vector<vector<int>> reversed(n, vector<int>(n));
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      reversed[i][j] = fuel[j][i];

  visited.assign(n, false);
  dfs(0, limit, reversed);
  for (bool v : visited) {
    if (!v)
      return false;
  }

  return true;
}

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

  cin >> n;
  fuel.assign(n, vector<int>(n));
  int maxFuel = 0;

  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j) {
      cin >> fuel[i][j];
      maxFuel = max(maxFuel, fuel[i][j]);
    }

  int left = 0, right = maxFuel, answer = maxFuel;

  while (left <= right) {
    int mid = (left + right) / 2;
    if (isConnected(mid)) {
      answer = mid;
      right = mid - 1;
    } else {
      left = mid + 1;
    }
```

```
    }

    cout << answer << '\n';

    return 0;
}
```

Описание: Суть решения заключается в том, чтобы найти минимальное значение ограничения на топливо, при котором можно добраться из любой вершины в любую другую. Вершины графа — это объекты, а fuel[i][j] задаёт, сколько топлива нужно, чтобы попасть из i в j. Решение строится с помощью двоичного поиска по возможным ограничениям на топливо. Для каждого кандидата проводится проверка связности графа с помощью обхода в глубину дважды: сначала по исходным рёбрам, потом по рёбрам в обратном направлении.

Сложность по времени: O (n^2 log(maxFuel))

Сложность по памяти: O (n^2)