Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3232

Терновский И.Е.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Задача 13 «M. Цивилизация»

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

using namespace std;

const int MAXN = 1000;
const int INF = 1e9;
const vector<pair<int, int>> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
const vector<char> dirChar = {'N', 'S', 'W', 'E'};

int n, m, xStart, yStart, xEnd, yEnd;
char world[MAXN][MAXN];
int cost[MAXN][MAXN];
pair<int, int> parent[MAXN][MAXN];

void bfs() {
    priority_queue<pair<int, pair<int, int>>, vector<pair<int, pair<int, int>>>, greater<pair<int,
pair<int, int>>>> q;
    q.push({0, {xStart, yStart}});
    cost[xStart][yStart] = 0;

    while (!q.empty()) {
        auto [c, cell] = q.top();
        auto [x, y] = cell;
        q.pop();

        if (c > cost[x][y]) continue;

        for (int i = 0; i < 4; i++) {
            int newX = x + directions[i].first;
            int newY = y + directions[i].second;

            if (newX >= 0 && newX < n && newY >= 0 && newY < m && world[newX][newY] !=
'#') {
                int newCost = cost[x][y] + (world[newX][newY] == '.' ? 1 : 2);
                if (newCost < cost[newX][newY]) {
                    cost[newX][newY] = newCost;
                    parent[newX][newY] = {x, y};
                    q.push({newCost, {newX, newY}});
                }
            }
```

```
        }
    }
}

int main() {
    cin >> n >> m >> xStart >> yStart >> xEnd >> yEnd;
    xStart--; yStart--; xEnd--; yEnd--;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> world[i][j];
            cost[i][j] = INF;
        }
    }

    bfs();

    if (cost[xEnd][yEnd] == INF) {
        cout << -1 << endl;
        return 0;
    }

    vector<char> path;
    for (pair<int, int> p = {xEnd, yEnd}; p != make_pair(xStart, yStart); p =
parent[p.first][p.second]) {
        pair<int, int> diff = {p.first - parent[p.first][p.second].first, p.second -
parent[p.first][p.second].second};
        path.push_back(dirChar[find(directions.begin(), directions.end(), diff) -
directions.begin()]);
    }

    reverse(path.begin(), path.end());

    cout << cost[xEnd][yEnd] << endl;
    for (char c : path) {
        cout << c;
    }
    cout << endl;

    return 0;
}
```

Пояснение к примененному алгоритму:

Основной алгоритм заключатся в применении алгоритма поиска в ширину (bfs), он пройдется по всем клеткам начиная с начальной и обновляя стоимость прохождения по каждой из клеток найдет самый дешевый маршрут от начала до конца. Так же для оптимизации применяется приоритетная очередь, складывая клетки, в которую, мы делаем так, чтобы наш алгоритм поиска в ширину для начала искал путь по самым дешевым клеткам, то есть по полю, тем самым наш алгоритм для начала пройдет по полям, а затем будет проверять менее оптимальные маршруты через леса.

Сложность: O(n log(n). Так как алгоритм поиска в ширину в худшем случае пройдет по всем клеткам, а так как мы используем приоритетную очередь, вставка в которую стоит log(n), то сложность умножается на log(n).

Задача 14 «N. Свинки-копилки»

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

using namespace std;

const int MAXN = 105;
vector<int> piggyBanks[MAXN];
int status[MAXN];
int breakCount = 0;

void DFS(int currentNode) {
  status[currentNode] = 1;
  for (int nextNode : piggyBanks[currentNode]) {
    if (status[nextNode] == 0) {
      DFS(nextNode);
    } else if (status[nextNode] == 1){
      breakCount++;
    }
  }
  status[currentNode] = 2;
}
```

```
int main() {
  int n, key;
  cin >> n;
  for (int i = 0; i < n; ++i) {
    cin >> key;
    piggyBanks[--key].push_back(i);
  }

  for (int i = 0; i < n; ++i) {
    if (status[i] == 0) {
      DFS(i);
    }
  }

  cout << breakCount << endl;

  return 0;
}
```

Пояснение к примененному алгоритму:

В данной задаче мы представляем копилки и копилки с ключами от них как граф, в таком случае, если на графе есть цикл, значит открыть копилку мы никак не сможем, поэтому ее придется разбить. Для поиска циклов в графе используется алгоритм поиска в глубину (dfs). Мы начинаем с первой копилки, помечаем ее, затем проверяем всех ее соседей, если сосед указывает на нее же, то мы получили цикл, в таком случае нужно увеличить ответ на 1.

Сложность: O(n). Так как в данной задаче сосед у копилок всего 1, поэтому поиск в глубину в худшем случае можно оценить как o(n).

Задача 15 «O. Долой списывание!»

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

using namespace std;
```

```
vector<vector<int>> adj;
vector<int> color;

bool dfs(int v, int c) {
    color[v] = c;
    for (int u : adj[v]) {
        if (color[u] == -1) {
            if (!dfs(u, 1 - c)) {
                return false;
            }
        } else if (color[u] == color[v]) {
            return false;
        }
    }
    return true;
}

int main() {
    int n, m;
    cin >> n >> m;
    adj.resize(n + 1);
    color.assign(n + 1, -1);
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    for (int i = 1; i <= n; i++) {
        if (color[i] == -1) {
            if (!dfs(i, 0)) {
                cout << "NO" << endl;
```

```
                return 0;
            }
        }
    }
    cout << "YES" << endl;
    return 0;
}
```

Пояснение к примененному алгоритму:

Тут мы проверяем является ли граф построенный из лкшат двудольным, для этого мы проверяем каждого лкшата алгоритмом поиска в глубину(dfs), каждому лкшату присваивается группа и если при проверке нового лкшата мы получаем то, что сосед изначального лкшата принадлежит к той же группе и уже был проверен, то тогда группу разделить нельзя и выводится соответственный ответ, если же таких случаев не было, тогда группу можно разделить и выводится ответ.

Сложность: O(n^2). Но эта сложность практически недостижима, так как она будет только если каждый узел графа связан со всеми остальными узлами, в нашем же случае сложность скорее будет O(n).