Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №4**

по «Алгоритмам и структурам данных»

Алгоритмы на графах

Выполнил:

Студент группы P32081

Кочнев Роман

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

# Yandex

#### Задача M «Цивилизация»

Пояснение к примененному алгоритму:

Мы используем алгоритм Дейкстры для поиска кратчайшего пути в взвешенном графе, который в данном случае представлен как двумерная карта, и ищем кратчайшие пути до каждой вершины в графе.

Дейкстра работает за O(N\*log(N)), где N – количество ребер в графе.

Код работает за O(n\*m\*(log(n)+log(m))

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

#### using namespace std;

#### const int MAXN = 1000;

#### const int INF = 1e9 + 7;

#### int dx[] = {0, 0, 1, -1};

#### int dy[] = {1, -1, 0, 0};

#### char dir[] = {'E', 'W', 'S', 'N'};

#### int dist[MAXN][MAXN];

#### int prv[MAXN][MAXN];

#### int x\_end, y\_end;

#### string world\_map[MAXN];

#### int n, m;

#### bool valid(int x, int y){

#### return (x >= 0 && x < n && y >= 0 && y < m && world\_map[x][y] != '#');

#### }

#### void dijkstra(int x\_start, int y\_start){

#### priority\_queue<pair<int, pair<int, int>>> q;

#### for(int i = 0; i < n; ++i)

#### for(int j = 0; j < m; ++j)

#### dist[i][j] = INF;

#### 

#### dist[x\_start][y\_start] = 0;

#### q.push({0, {x\_start, y\_start}});

#### 

#### while(!q.empty()){

#### int d = -q.top().first;

#### int x = q.top().second.first;

#### int y = q.top().second.second;

#### q.pop();

#### if(d != dist[x][y]) continue;

#### 

#### for(int i = 0; i < 4; ++i){

#### int nx = x + dx[i];

#### int ny = y + dy[i];

#### if(valid(nx, ny)){

#### int cost = d + (world\_map[nx][ny] == 'W' ? 2 : 1);

#### if(cost < dist[nx][ny]){

#### dist[nx][ny] = cost;

#### prv[nx][ny] = i;

#### q.push({-cost, {nx, ny}});

#### }

#### }

#### }

#### }

#### }

#### int main(){

#### cin >> n >> m;

#### int x\_start, y\_start;

#### cin >> x\_start >> y\_start;

#### --x\_start, --y\_start;

#### cin >> x\_end >> y\_end;

#### --x\_end, --y\_end;

#### for(int i = 0; i < n; ++i)

#### cin >> world\_map[i];

#### 

#### dijkstra(x\_start, y\_start);

#### if(dist[x\_end][y\_end] == INF){

#### cout << -1 << "\n";

#### } else {

#### cout << dist[x\_end][y\_end] << "\n";

#### string path;

#### for(int x = x\_end, y = y\_end; x != x\_start || y != y\_start;){

#### int i = prv[x][y];

#### path += dir[i];

#### x -= dx[i];

#### y -= dy[i];

#### }

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

#### cout << path << "\n";

#### }

#### return 0;

#### }

#### }Задача N «Свинки-копилки»

Пояснение к примененному алгоритму:

Задача сводится к поиску количества циклов в графе. Мы просто для каждой вершины ищем родителя для родителя – родителя и т.д. После используем поиск в глубину от вершины. Поиск родителя занимает O(n), dsf тоже за O(n)

Код работает за O(N)

#include <iostream>

#include <vector>

using namespace std;

int findParent(vector<int>& parent, int x) {

if (parent[x] != x) {

parent[x] = findParent(parent, parent[x]);

}

return parent[x];

}

void unite(vector<int>& parent, int x, int y) {

int rootX = findParent(parent, x);

int rootY = findParent(parent, y);

if (rootX != rootY) {

parent[rootY] = rootX;

}

}

int main() {

int n;

cin >> n;

vector<int> parent(n + 1);

for (int i = 1; i <= n; ++i) {

parent[i] = i;

}

for (int i = 1; i <= n; ++i) {

int x;

cin >> x;

unite(parent, i, x);

}

int res = 0;

for (int i = 1; i <= n; ++i) {

if (i == parent[i]) {

++res;

}

}

cout << res << endl;

return 0;

}

#### Задача O «Долой списывание »

Пояснение к примененному алгоритму:

Данная задача является примером задачи в 2-расраски, что две соседние вершины раскрашены в разные цвета. Мы просто для каждой вершины, ищем пару, где окрашена только одна вершина окрашиваем вторую в другой цвет.

Код работает за O(N^2)

#include <iostream>

#include <vector>

using namespace std;

bool dfs(vector<int> &group,vector<vector<int>> &g, int v, int c) {

group[v] = c;

for (int u : g[v]) {

if (!group[u]) {

if (!dfs(group, g, u, 3 - c)) {

return false;

}

} else if (group[u] == c) {

return false;

}

}

return true;

}

int main() {

int n, m;

cin >> n >> m;

vector<int> group(n+1);

vector<vector<int>> g(n+1);

for (int i = 1; i <= m; ++i) {

int u, v;

cin >> u >> v;

g[u].push\_back(v);

g[v].push\_back(u);

}

for (int i = 1; i <= n; ++i) {

if (!group[i] && !dfs(group, g, i, 1)) {

cout << "NO" << endl;

return 0;

}

}

cout << "YES" << endl;

return 0;

}

#### Задача P «Авиаперелеты»

В данной задаче нужно определить является граф сильно связанным, если учитывать все ребра, меньше k. Для этого мы парсим граф, создавая матрицу смежности для каждого значения. Для быстрого поиска значения, используем бинарный поиск. Проверка графа на сильную связанность выполняется за O(n^2), поиск за log(N)

Код работает за O(N^2\*log(N))

#include <vector>

#include <iostream>

using namespace std;

vector<vector<int>> parse\_graph(int n, const vector<vector<int>>& graph){

vector<vector<int>> new\_graph(graph.size(), vector<int>(graph.size(), 0));

for (int i = 0; i < graph.size(); i++) {

for (int j = 0; j < graph.size(); j++) {

if(graph[i][j]<=n & graph[i][j]!=0) {new\_graph[i][j]=1;

}

}

}

return new\_graph;

}

void DFS(const vector<vector<int>>& graph, int v, vector<bool>& visited) {

visited[v] = true;

for (int i = 0; i < graph[v].size(); ++i) {

if (graph[v][i] && !visited[i]) {

DFS(graph, i, visited);

}

}

}

bool isStronglyConnected(const vector<vector<int>>& graph) {

int n = graph.size();

vector<bool> visited(n, false);

DFS(graph, 0, visited);

for (int i = 0; i < n; ++i) {

if (!visited[i]) {

return false;

}

}

visited = vector<bool>(n, false);

vector<vector<int>> transposed\_graph(n, vector<int>(n, 0));

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

transposed\_graph[i][j] = graph[j][i];

}

}

DFS(transposed\_graph, 0, visited);

for (int i = 0; i < n; ++i) {

if (!visited[i]) {

return false;

}

}

return true;

}

int main() {

int maxFuel=0;

int n;

cin >> n;

vector<vector<int>> graph1(n, vector<int>(n, 0));

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

cin >> graph1[i][j];

maxFuel = max(maxFuel, graph1[i][j]);

}

}

int left =0;

int right=maxFuel;

while (left < right) {

int mid = left + (right - left) / 2;

vector<vector<int>> parsed\_graph=parse\_graph(mid, graph1);

if (isStronglyConnected(parsed\_graph)) {

right = mid;

} else {

left = mid + 1;

}

}

cout << left << endl;

return 0;

}