Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №4**

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P32092

Голиков А.С.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача M «Цивилизация»

Пояснение к примененному алгоритму:

Алгоритм Дейкстры, так как нет отрицательных ребер.

n - число строк

m - число столбцов

вершин nm

ребер (n - 1) \* m + (m - 1) \* n + nm - m + nm - n = 2nm - (n + m)

Асимптотика - O((2nm - (n + m)) \* log nm) = O(mn \* log mn)

Код решения:

#include <iostream>

#include <vector>

#include <queue>

#include <unordered\_map>

using namespace std;

pair<int, string> dijkstra(vector<vector<int>> \*matrix, int start, int finish, int n, int m) {

vector<bool> visited(n \* m, false);

vector<string> dict = **{**"N", "E", "S", "W"**}**;

unordered\_map<string, int> map = **{**{"N", m},

{"E", -1},

{"S", -1\*m},

{"W", 1}**}**;

vector<int> direction = **{**-1\*m, 1, m, -1**}**;

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> q;

vector<pair<int, string>> way(n \* m, make\_pair(1e8, "\_"));

way[start] = make\_pair(0, '\n');

q.emplace(0, start);

while (!q.empty()) {

int v = q.top().second;

q.pop();

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

int len = (\*matrix)[v][i];

if (len != -1) {

if (!visited[v + direction[i]]) {

if (way[v].first + len < way[v + direction[i]].first) {

way[v + direction[i]] = make\_pair(way[v].first + len, dict[i]);

q.emplace(way[v + direction[i]].first, v + direction[i]);

}

}

}

}

visited[v] = true;

}

if (way[finish].first == 1e8) {

return make\_pair(-1, "");

}

string s = "";

int v = finish;

while (v != start) {

s = way[v].second + s;

v += map[way[v].second];

}

return make\_pair(way[finish].first, s);

}

int main() {

int n, m, sx, sy, fx, fy;

cin >> n >> m >> sx >> sy >> fx >> fy;

--sx;

--sy;

--fx;

--fy;

vector<string> field(n);

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

cin >> field[i];

}

vector<vector<int>> matrix(n \* m);

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

vector<int> a(4, -1);

matrix[i] = a;

}

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

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

if (j < m - 1) {

int len = -1;

if (field[i][j + 1] == '.') {

len = 1;

}

if (field[i][j + 1] == 'W') {

len = 2;

}

matrix[i \* m + j][1] = len;

len = -1;

if (field[i][j] == '.') {

len = 1;

}

if (field[i][j] == 'W') {

len = 2;

}

matrix[i \* m + j + 1][3] = len;

}

if (i < n - 1) {

int len = -1;

if (field[i + 1][j] == '.') {

len = 1;

}

if (field[i + 1][j] == 'W') {

len = 2;

}

matrix[i \* m + j][2] = len;

len = -1;

if (field[i][j] == '.') {

len = 1;

}

if (field[i][j] == 'W') {

len = 2;

}

matrix[(i + 1) \* m + j][0] = len;

}

}

}

pair<int, string> p = dijkstra(&matrix, sx \* m + sy, fx \* m + fy, n, m);

if (p.first == -1) {

cout << -1;

} else {

cout << p.first << "\n" << p.second;

}

return 0;

}

Задача N «Свинки-копилки»

Пояснение к примененному алгоритму:

Дан орграф, где у каждой вершины степень входа ровно 1. Нужно разбить на минимальное количество исходящих деревьев.

Утверждаю, что деревьев ровно столько, сколько компонент связности у графа, то есть в каждой компоненте можно выделить свое дерево, покрывающее ее вершины.

Сначала покажу алгоритм выделения дерева в компоненте:

Смотрим на вершину и идем от нее вверх, пока не попадем туда, где уже были. Точно куда-то попадем, так как в графе конечное количество вершин, а у каждой вершины есть, куда идти.

Мы нашли цикл, из которого все стрелки ведут в низ. Открыв любую его вершину, мы откроем весь цикл и вершины, в которые можно из него попасть. Так мы как раз получим это дерево

Предположим, в компоненте осталась вершина, которая не покрылась таким деревом.

Тогда в нее нельзя попасть ни из какой вершины, так как мы открывали все, что можно.

Значит, она является родителем для какой-то вершины, чего тоже не может быть, так как в поиске мы шли вверх и утнулись в саамый верх, путь вверх однозначен в силу степени 1 каждой вершины.

Противоречие - значит в компоненте не может быть вершины, не покрытой деревом, построенной предложенным алгоритмом

Асимптотика - O(n^2)

Код решения:

#include <iostream>

#include <vector>

using namespace std;

void open(int v, vector<vector<int>> \*down, vector<bool> \*opened) {

(\*opened)[v] = true;

for (const auto &i: (\*down)[v]) {

if (!(\*opened)[i]) open(i, down, opened);

}

}

int main() {

int n;

cin >> n;

vector<int> up(n);

vector<vector<int>> down(n);

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

int x;

cin >> x;

up[i] = x - 1;

down[x - 1].push\_back(i);

}

int ans = 0;

vector<bool> opened(n, false);

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

if (!opened[i]) {

vector<bool> visited(n, false);

int v = i;

while (!visited[v]) {

visited[v] = true;

v = up[v];

}

open(v, &down, &opened);

++ans;

}

}

cout << ans;

return 0;

}

Задача O «Долой списывание!»

Пояснение к примененному алгоритму:

Задача - определить двудольность графа

Вершины двудольного графа можно правильно раскрасить в два цвета (цвета и задают доли).

Давайте попробуем это сделать - Если вершина еще не покращена, красим ее в цвет 1, если покрашена - красим всех ее соседей в противоположный цвет.

Если среди соседей есть вершина такого же цвета, мы проиграли и раскраски не существует

Асимптотика - O(n^2)

Код решения:

#include <iostream>

#include <vector>

using namespace std;

bool coloring(int i, int color, vector<int> \*colors, vector<vector<int>> \*e) {

for (auto v: (\*e)[i]) {

if ((\*colors)[v] == 3 - color) return false;

if (!(\*colors)[v]) {

(\*colors)[v] = color;

if (!coloring(v, 3 - color, colors, e)) return false;

}

}

return true;

}

int main() {

int n, k;

cin >> n >> k;

vector<vector<int>> e(n);

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

int a, b;

cin >> a >> b;

e[a - 1].push\_back(b - 1);

e[b - 1].push\_back(a - 1);

}

vector<int> colors(n, 0);

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

if (!colors[i]) {

colors[i] = 1;

if (!coloring(i, 2, &colors, &e)) {

cout << "NO";

return 0;

}

}

}

cout << "YES";

return 0;

}

Задача P «Авиаперелёты»

Пояснение к примененному алгоритму:

Бинпоиск по ответу.

Проверка - удалим все ребра, больше заданного. Если граф все еще связен, такое ребро подходит для размера бака

Для проверки свзяности попробуем дойти от 1 вершины до каждой, а потом от каждой до 1. Если все такие пути есть, то между двумя любыми вершинами есть путь x -> 1 -> y

Путь от 1 до любой будем проверять как всегда - идем куда можем, если встретили - победа)

Асимптотика - O(n^2 log(n^2))

Код решения:

#include <iostream>

#include <vector>

#include <set>

#include <algorithm>

using namespace std;

void visitto(int v, int k, vector<vector<int>> \*matrix, vector<bool> \*visited) {

(\*visited)[v] = true;

int n = (\*visited).size();

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

if ((\*matrix)[v][i] <= k && !(\*visited)[i]) {

visitto(i, k, matrix, visited);

}

}

}

void visitfrom(int v, int k, vector<vector<int>> \*matrix, vector<bool> \*visited) {

(\*visited)[v] = true;

int n = (\*visited).size();

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

if ((\*matrix)[i][v] <= k && !(\*visited)[i]) {

visitfrom(i, k, matrix, visited);

}

}

}

bool is\_ok(int n, int k, vector<vector<int>> \*matrix) {

bool ok = true;

vector<bool> visitedto(n, false);

vector<bool> visitedfrom(n, false);

visitto(0, k, matrix, &visitedto);

visitfrom(0, k, matrix, &visitedfrom);

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

if (!visitedto[i] || !visitedfrom[i]) {

ok = false;

break;

}

}

return ok;

}

int bin\_search(int start, int end, const vector<int> \*edges, int n, vector<vector<int>> \*matrix) {

if (is\_ok(n, (\*edges)[end], matrix)) {

return (\*edges)[end];

}

if (end - start > 1) {

int mid = (start + end) / 2;

if (is\_ok(n, (\*edges)[mid], matrix)) {

return bin\_search(mid, end, edges, n, matrix);

} else {

return bin\_search(start, mid, edges, n, matrix);

}

} else {

return (\*edges)[start];

}

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

cout.tie(nullptr);

int n;

cin >> n;

vector<vector<int>> matrix(n);

set<int> edges\_s;

vector<int> edges;

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

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

int x;

cin >> x;

matrix[i].push\_back(x);

if (edges\_s.find(x) == edges\_s.end()) {

edges.push\_back(x);

edges\_s.insert(x);

}

}

}

sort(edges.begin(), edges.end(), greater<>());

cout << bin\_search(0, edges.size(), &edges, n, &matrix);

return 0;

}