Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №10-11

По дисциплине «МОИС»

Тема: «Теория графов»

Выполнил:

Студент 2 курса

Группы ИИ-23

Вышинский А. С.

Проверил:

Козинский А.А.

Брест 2024

Вариант 5

Задание 1: Решить задачу коммивояжера.

3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | ∞ | 3 | 9 | 8 | 8 | 5 |
| **2** | 8 | ∞ | 1 | 9 | 6 | 2 |
| **3** | 5 | 4 | ∞ | 2 | 3 | 3 |
| **4** | 4 | 7 | 9 | ∞ | 0 | 7 |
| **5** | 0 | 0 | 9 | 7 | ∞ | 0 |
| **6** | 5 | 1 | 0 | 9 | 8 | ∞ |

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <algorithm>

#include <fstream>

using namespace std;

#define INF 9999999

bool gen(vector<int>& a, int n) {

int j = a.size() - 2;

while (j != -1 && a[j] > a[j + 1]) j--;

if (j == -1) return false;

int k = n - 1;

while (a[k] < a[j]) k--;

swap(a[k], a[j]);

int l = j + 1;

int r = n - 1;

while (l < r) {

swap(a[l], a[r]);

l++;

r--;

}

return true;

}

void print(vector<int> a) {

//static int num = 1;

cout.width(3);

//cout << num++ << ": ";

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

cout << a[i] << " ";

cout << endl;

}

int calculate(vector<int> a, vector<vector<int>> matrix) {

int way = matrix[a[a.size()-1]][a[0]];

for (int i = 0; i < a.size()-1; i++) {

way += matrix[a[i]][a[i + 1]];

}

return way;

}

int main() {

string filename = "task1.txt";

vector<vector<int>> matrix;

ifstream file(filename);

if (file.is\_open()) {

string line;

while (getline(file, line)) {

vector<int> row;

istringstream iss(line);

string num;

while (iss >> num) {

if (num > "99") { row.push\_back(INF); }

else { row.push\_back(stod(num)); }

}matrix.push\_back(row);

}

file.close();

for (auto row : matrix) {

for (auto num : row) { cout << num << " "; }

cout << endl;

}

}

else { cout << "Error opening file: " << filename << endl; }

vector<int> a;

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

a.push\_back(i);

}

int minway = calculate(a, matrix);

vector<int> way = a;

while (gen(a, matrix.size())) {

if (minway > calculate(a, matrix)) {

minway = calculate(a, matrix);

way = a;

}

}

cout << "Mit path is: ";

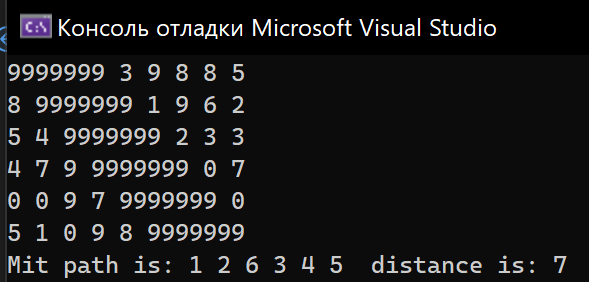
for (int i = 0; i < way.size(); i++) {

cout << way[i] +1 << " ";

}

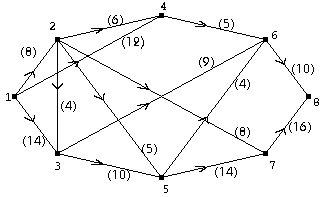
cout << " distance is: " << minway;

}



Задание 2: найти максимальный поток в заданной транспортной сети

**3.**



#include <iostream>

#include <limits.h>

#include <list>

#include <queue>

#include <algorithm>

#include <fstream>

using namespace std;

// Структура для представления ребра с весом

struct Edge {

int destination;

int capacity;

Edge(int dest, int cap) : destination(dest), capacity(cap) {}

};

class Graph {

int V; // Количество вершин графа

list<Edge>\* adj; // Список смежности

public:

Graph(int V); // Конструктор

// Добавление ребра к графу

void addEdge(int u, int v, int capacity);

// Поиск максимального потока из истока в сток

int fordFulkerson(int source, int sink);

int getV() { return V; }

};

Graph::Graph(int V) {

this->V = V;

adj = new list<Edge>[V];

}

void Graph::addEdge(int u, int v, int capacity) {

adj[u].push\_back(Edge(v, capacity));

adj[v].push\_back(Edge(u, 0));

}

bool bfs(list<Edge>\* residualGraph, int source, int sink, int parent[]) {

vector<bool> visited(100, false);

queue<int> q;

visited[source] = true;

q.push(source);

parent[source] = -1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (const auto& edge : residualGraph[u]) {

int v = edge.destination;

int cap = edge.capacity;

if (!visited[v] && cap > 0) {

q.push(v);

parent[v] = u;

visited[v] = true;

}

}

}

return visited[sink];

}

int Graph::fordFulkerson(int source, int sink) {

int\* parent = new int[V];

int max\_flow = 0;

list<Edge>\* residualGraph = new list<Edge>[V];

// Копируем значения из исходного графа в резервный граф

for (int u = 0; u < V; u++) {

for (const auto& edge : adj[u]) {

int v = edge.destination;

int capacity = edge.capacity;

residualGraph[u].push\_back(Edge(v, capacity));

}

}

while (bfs(residualGraph, source, sink, parent)) {

int path\_flow = INT\_MAX;

for (int v = sink; v != source; v = parent[v]) {

int u = parent[v];

auto it = find\_if(residualGraph[u].begin(), residualGraph[u].end(), [v](const Edge& edge) { return edge.destination == v; });

path\_flow = min(path\_flow, it->capacity);

}

for (int v = sink; v != source; v = parent[v]) {

int u = parent[v];

auto it\_uv = find\_if(residualGraph[u].begin(), residualGraph[u].end(), [v](const Edge& edge) { return edge.destination == v; });

it\_uv->capacity -= path\_flow;

auto it\_vu = find\_if(residualGraph[v].begin(), residualGraph[v].end(), [u](const Edge& edge) { return edge.destination == u; });

it\_vu->capacity += path\_flow;

}

max\_flow += path\_flow;

}

delete[] parent;

delete[] residualGraph;

return max\_flow;

}

int main() {

ifstream in("input.txt");

int V;

in >> V;

Graph graph(V);

int u, v, w;

while (in >> u >> v >> w) {

graph.addEdge(u, v, w);

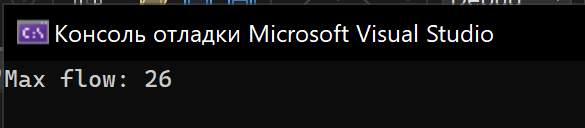
}

int source = 0, sink = V-1;

cout << "Max flow: " << graph.fordFulkerson(source, sink) << endl;

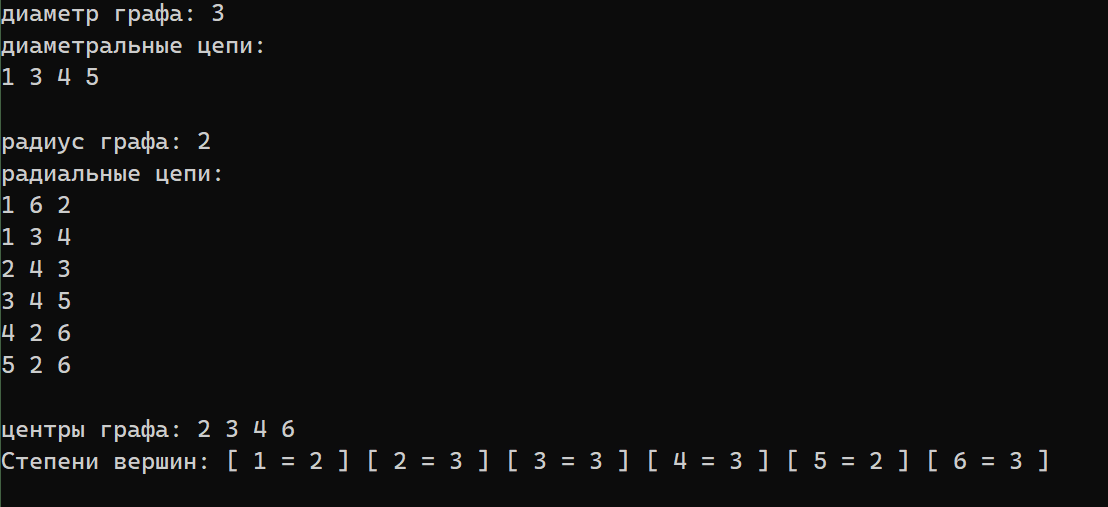
return 0;

}



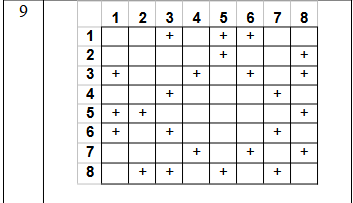
Задание 3: 1)Определите, из какого минимального числа кусков проволоки можно спаять данный каркас (толщина всех ребер каркаса должна быть одинаковой). Ответ обоснуйте. 2)Изобразите все реберно-непересекающиеся цепи, на которые можно разбить ребра графа, соответствующего данному каркасу (т.е. покажите, как спаять такие каркасы из минимального числа кусков проволоки). 3)Построить неориентированный граф G = <V,R> (множества V и R указаны для каждого варианта). Для графа G найти:

его диаметр и все диаметральные цепи; его радиус и все радиальные цепи; все центры графа; степень каждой его вершины; все разделяющие вершины.



Задание 4:

Найдите минимальное число валов, на которые можно поместить шестерни.



#include <iostream>

#include <vector>

#include <set>

#include <fstream>

using namespace std;

class Graph {

public:

int V; // количество вершин

vector<set<int>> adj; // список смежности

Graph(int vertices) : V(vertices), adj(vertices) {}

void addEdge(int v, int w) {

adj[v].insert(w);

adj[w].insert(v);

}

void greedyColoring() {

vector<int> result(V, -1); // массив для хранения цветов вершин

// Цвет первой вершины

result[0] = 0;

// Доступные цвета (индексы в массиве)

vector<bool> available(V, false);

// Раскрашиваем остальные вершины

for (int u = 1; u < V; u++) {

// Метим все доступные цвета

for (auto neighbor : adj[u]) {

if (result[neighbor] != -1) {

available[result[neighbor]] = true;

}

}

// Находим первый доступный цвет

int color;

for (color = 0; color < V; color++) {

if (!available[color]) {

break;

}

}

result[u] = color; // Присваиваем вершине найденный цвет

// Сбрасываем доступность цветов для следующей итерации

for (auto neighbor : adj[u]) {

if (result[neighbor] != -1) {

available[result[neighbor]] = false;

}

}

}

// Выводим раскраску

cout << "minimal number of shafts: " << \*max\_element(result.begin(), result.end()) + 1 << endl;

}

};

int main() {

ifstream in("task4.txt");

if (in.is\_open()) {

int NumV;

in >> NumV;

Graph g(NumV);

int V, U;

while (in >> V >> U) {

g.addEdge(V-1, U-1);

}

g.greedyColoring();

}

return 0;

}

