Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №6**

По дисциплине «Математические основы интеллектуальных систем»

Тема: «Определение компонент сильной связности в орграфе. Определение компонент двусвязности для неориентированного графа»

**Выполнил:**

Студент 2 курса

Группы ИИ-21

Литвинюк Т. В.

**Проверил:**

Козинский А. А.

Брест 2022

**Цель:** научиться находить компонент сильной связности и компонент двусвязности.

**Ход работы:**

**Вариант 7**

#include <iostream>

#include <vector>

#include "..\graphs.h"

using namespace std;

// транспонировать матрицу

// входные данные: матрица смежности, количество вершин

// выходные данные: транспонированная матрица

vector<vector<int>> transposeMatrix(vector<vector<int>> &matrix, int n) {

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

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

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

transposedMatrix[i][j] = matrix[j][i];

}

}

return transposedMatrix;

}

// поиск в глубину

// входные данные: номер вершины, матрица смежности, вектор посещенных вершин, вектор порядка обхода

// cout

void dfs1(int v, vector<vector<int>> &matrix, vector<int> &visited, vector<int> &order) {

visited[v] = 1;

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

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

dfs1(i, matrix, visited, order);

}

}

order.push\_back(v);

}

// поиск в глубину

// входные данные: номер вершины, матрица смежности, вектор посещенных вершин, номер компоненты

// cout

void dfs2(int v, vector<vector<int>> &matrix, vector<int> &visited, int componentNumber) {

visited[v] = componentNumber + 1;

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

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

dfs2(i, matrix, visited, componentNumber);

}

}

}

// найти компонент сильной связности орграфа

// входные данные: матрица смежности, количество вершин

// cout

void findStronglyConnectedComponents(vector<vector<int>> &matrix, int n) {

vector<int> visited(n, 0);

vector<int> order;

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

if (!visited[i]) {

dfs1(i, matrix, visited, order);

}

}

vector<vector<int>> transposedMatrix = transposeMatrix(matrix, n);

vector<int> component(n, 0);

int componentNumber = 0;

for (int i = n - 1; i >= 0; i--) {

if (!component[order[i]]) {

dfs2(order[i], transposedMatrix, component, componentNumber);

componentNumber++;

}

}

cout << "Strongly connected component: " << componentNumber << endl;

cout << "Components: " << endl;

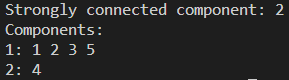
for (int i = 0; i < componentNumber; i++) {

cout << i + 1 << ": ";

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

if (component[j] == i + 1) {

cout << j + 1 << " ";

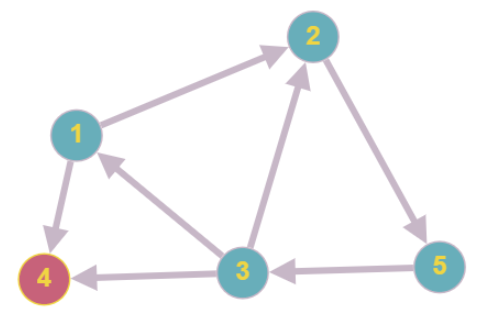
 }

}

cout << endl;

}

}



int main(){

char path[] = "cons";

const int TOPS = getTopsCount(path), EDGES = getEdgesCount(path);

int \*\*adjMatrix = adjacencyMatrixFromConnectionsORGRAPH(path);

vector<vector<int>> graph(TOPS, vector<int>(TOPS, 0));

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

for (int j = 0; j < TOPS; j++)

graph[i][j] = adjMatrix[i][j];

}

findStronglyConnectedComponents(graph, TOPS);

}

#include <iostream>

#include <vector>

#include "..\graphs.h"

using namespace std;

// транспонировать матрицу

// входные данные: матрица смежности, количество вершин

// выходные данные: транспонированная матрица

vector<vector<int>> transposeMatrix(vector<vector<int>> &matrix, int n) {

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

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

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

transposedMatrix[i][j] = matrix[j][i];

}

}

return transposedMatrix;

}

// поиск в глубину

// входные данные: номер вершины, матрица смежности, вектор посещенных вершин, вектор порядка обхода

// cout

void dfs1(int v, vector<vector<int>> &matrix, vector<int> &visited, vector<int> &order) {

visited[v] = 1;

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

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

dfs1(i, matrix, visited, order);

}

}

order.push\_back(v);

}

// поиск в глубину

// входные данные: номер вершины, матрица смежности, вектор посещенных вершин, номер компоненты

// cout

void dfs2(int v, vector<vector<int>> &matrix, vector<int> &visited, int componentNumber) {

visited[v] = componentNumber + 1;

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

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

dfs2(i, matrix, visited, componentNumber);

}

}

}

// найти компонент двусвязности неориентированного графа

// входные данные: матрица смежности, количество вершин

// cout

void findBiconnectedComponents(vector<vector<int>> &matrix, int n) {

vector<int> visited(n, 0);

vector<int> order;

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

if (!visited[i]) {

dfs1(i, matrix, visited, order);

}

}

vector<vector<int>> transposedMatrix = transposeMatrix(matrix, n);

vector<int> component(n, 0);

int componentNumber = 0;

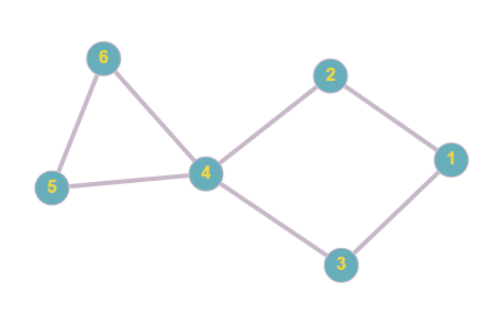
for (int i = n - 1; i >= 0; i--) {

if (!component[order[i]]) {

dfs2(order[i], transposedMatrix, component, componentNumber);

componentNumber++;

}

 }

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

cout << "Component " << i + 1 << ": ";

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

if (component[j] == i + 1) {

cout << j + 1 << " ";

}

}

cout << endl;

}

}

int main(){

char path[] = "cons2";

const int TOPS = getTopsCount(path), EDGES = getEdgesCount(path);

int \*\*adjMatrix = adjacencyMatrixFromConnections(path);

vector<vector<int>> graph(TOPS, vector<int>(TOPS, 0));

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

for (int j = 0; j < TOPS; j++)

graph[i][j] = adjMatrix[i][j];

 }

findBiconnectedComponents(graph, TOPS);

}

**Вывод:** в ходе лабораторной работы я научился находить кратчайшие пути в графе.