**Лабораторная работа №7**

**Обход графа в глубину**

#include <iostream>

#include <vector>

#include <stack>

#include <windows.h>

using namespace std;

class Graph {

public:

int vertices;

vector<vector<int>> adjacencyList;

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

void addEdge(int v, int w) {

adjacencyList[v].push\_back(w);

}

void DFS(int startVertex) {

vector<bool> visited(vertices, false);

stack<int> stack;

visited[startVertex] = true;

stack.push(startVertex);

while (!stack.empty()) {

int currentVertex = stack.top();

cout << currentVertex << " ";

stack.pop();

for (int adjacentVertex : adjacencyList[currentVertex]) {

if (!visited[adjacentVertex]) {

visited[adjacentVertex] = true;

stack.push(adjacentVertex);

}

}

}

}

};

int main() {

Graph g(6);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(2, 4);

g.addEdge(2, 5);

setlocale(LC\_ALL, "rus");

cout << "DFS начиная с вершины 0: ";

g.DFS(0);

return 0;

}

Этот код создает граф с 6 вершинами и неориентированными ребрами. DFS начинается с определенной вершины (в данном случае, с вершины 0). Вам нужно адаптировать его под ваш конкретный граф и требования.

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием.
3. Реализуйте процедуру обхода в глубину для графа, представленного списками смежности.

Примеры, приведенные ниже, применимы отдельно к каждому элементу задачи

Ответ для задание 1:

1. 1 Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.

Вот пример кода на C++, который генерирует случайную матрицу смежности для неориентированного графа и выводит её на экран:

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <windows.h>

using namespace std;

class RandomGraphGenerator {

public:

vector<vector<int>> generateGraph(int vertices, float density) {

vector<vector<int>> adjacencyMatrix(vertices, vector<int>(vertices, 0));

srand(time(0)); // Инициализация генератора случайных чисел

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

for (int j = i + 1; j < vertices; ++j) {

if (rand() % 100 < density \* 100) { // Вероятность наличия ребра

adjacencyMatrix[i][j] = 1;

adjacencyMatrix[j][i] = 1;

}

}

}

return adjacencyMatrix;

}

};

void printMatrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int val : row) {

cout << val << " ";

}

cout << endl;

}

}

int main() {

int vertices = 5; // Задайте количество вершин

float density = 0.3; // Задайте плотность графа (вероятность наличия ребра)

RandomGraphGenerator generator;

vector<vector<int>> adjacencyMatrix = generator.generateGraph(vertices, density);

setlocale(LC\_ALL, "rus");

cout << "Генерация Матрица смежности:" << endl;

printMatrix(adjacencyMatrix);

return 0;

}

Мы можем изменить значения переменных vertices и density в соответствии с вашими требованиями.

1.2 Для сгенерированного графа осуществите процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием.

Добавим обход в глубину (DFS) к предыдущему коду. Ниже приведен пример, как это можно сделать:

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <stack>

#include <windows.h>

using namespace std;

class RandomGraphGenerator {

public:

vector<vector<int>> generateGraph(int vertices, float density) {

vector<vector<int>> adjacencyMatrix(vertices, vector<int>(vertices, 0));

srand(time(0));

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

for (int j = i + 1; j < vertices; ++j) {

if (rand() % 100 < density \* 100) {

adjacencyMatrix[i][j] = 1;

adjacencyMatrix[j][i] = 1;

}

}

}

return adjacencyMatrix;

}

};

class GraphTraversal {

public:

void DFS(const vector<vector<int>>& adjacencyMatrix, int startVertex) {

int vertices = adjacencyMatrix.size();

vector<bool> visited(vertices, false);

stack<int> stack;

visited[startVertex] = true;

stack.push(startVertex);

setlocale(LC\_ALL, "rus");

cout << "DFS начиная с вершины " << startVertex << ": ";

while (!stack.empty()) {

int currentVertex = stack.top();

stack.pop();

cout << currentVertex << " ";

for (int adjacentVertex = 0; adjacentVertex < vertices; ++adjacentVertex) {

if (adjacencyMatrix[currentVertex][adjacentVertex] == 1 && !visited[adjacentVertex]) {

visited[adjacentVertex] = true;

stack.push(adjacentVertex);

}

}

}

cout << endl;

}

};

void printMatrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int val : row) {

cout << val << " ";

}

cout << endl;

}

}

int main() {

int vertices = 5;

float density = 0.3;

RandomGraphGenerator generator;

vector<vector<int>> adjacencyMatrix = generator.generateGraph(vertices, density);

setlocale(LC\_ALL, "rus");

cout << "Генерация Матрица смежности:" << endl;

printMatrix(adjacencyMatrix);

GraphTraversal traversal;

int startVertex = 0; // Задайте начальную вершину для обхода

traversal.DFS(adjacencyMatrix, startVertex);

return 0;

}

Заменит startVertex на номер вершины, с которой мы хотим начать обход в глубину.

1.3 Реализуете процедуру обхода в глубину для графа , представленного списками смежности

пример реализации обхода в глубину (DFS) для графа, представленного списками смежности на C++:

#include <iostream>

#include <vector>

#include <stack>

#include <windows.h>

using namespace std;

class Graph {

public:

int vertices;

vector<vector<int>> adjacencyList;

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

void addEdge(int v, int w) {

adjacencyList[v].push\_back(w);

}

void DFS(int startVertex) {

setlocale(LC\_ALL, "rus");

vector<bool> visited(vertices, false);

stack<int> stack;

visited[startVertex] = true;

stack.push(startVertex);

cout << "DFS начиная с вершины " << startVertex << ": ";

while (!stack.empty()) {

int currentVertex = stack.top();

stack.pop();

cout << currentVertex << " ";

for (int adjacentVertex : adjacencyList[currentVertex]) {

if (!visited[adjacentVertex]) {

visited[adjacentVertex] = true;

stack.push(adjacentVertex);

}

}

}

cout << endl;

}

};

int main() {

setlocale(LC\_ALL, "rus");

Graph g(6);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(2, 4);

g.addEdge(2, 5);

cout << "Список смежности:" << endl;

for (int i = 0; i < g.vertices; ++i) {

cout << i << ": ";

for (int adjacentVertex : g.adjacencyList[i]) {

cout << adjacentVertex << " ";

}

cout << endl;

}

int startVertex = 0; // Задайте начальную вершину для обхода

g.DFS(startVertex);

return 0;

}

Пример создает граф с 6 вершинами и неориентированными ребрами, представленными списками смежности. Замените значения в addEdge и startVertex в соответствии с вашими требованиями.

В приведенном ниже примере все задачи выполняются вместе.

#include <iostream>

#include <vector>

#include <unordered\_set>

#include <cstdlib>

#include <ctime>

#include <windows.h>

using namespace std;

// Генерация матрицы смежности для неориентированного графа

vector<vector<int>> generateGraph(int vertices, int edges) {

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

srand(time(0));

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

int vertex1 = rand() % vertices;

int vertex2 = rand() % vertices;

// Гарантируем, что не создаем петли и не повторяем ребра

while (vertex1 == vertex2 || graph[vertex1][vertex2] == 1 || graph[vertex2][vertex1] == 1) {

vertex1 = rand() % vertices;

vertex2 = rand() % vertices;

}

// Добавляем ребро между вершинами

graph[vertex1][vertex2] = 1;

graph[vertex2][vertex1] = 1;

}

return graph;

}

// Обход в глубину

void dfs(const vector<vector<int>>& graph, int vertex, unordered\_set<int>& visited) {

if (visited.find(vertex) == visited.end()) {

cout << vertex << ' ';

visited.insert(vertex);

for (int neighbor = 0; neighbor < graph[vertex].size(); ++neighbor) {

if (graph[vertex][neighbor] == 1) {

dfs(graph, neighbor, visited);

}

}

}

}

int main() {

setlocale(LC\_ALL, "rus");

// Генерация графа

int vertices = 5;

int edges = 7;

vector<vector<int>> graph = generateGraph(vertices, edges);

// Вывод матрицы смежности

cout << "Матрица смежности:\n";

for (const auto& row : graph) {

for (int value : row) {

cout << value << ' ';

}

cout << '\n';

}

// Вызов DFS с начальной вершиной 0

cout << "\nОбход в глубину:\n";

unordered\_set<int> visited;

dfs(graph, 0, visited);

return 0;

}

**Задание 2\***

1. Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

Ответ для задание 2:

пример преобразования рекурсивной реализации обхода графа в глубину (DFS) для матричной формы представления графов в не рекурсивную форму с использованием стека на C++:

#include <iostream>

#include <vector>

#include <stack>

#include <windows.h>

using namespace std;

class Graph {

public:

int vertices;

vector<vector<int>> adjacencyMatrix;

Graph(int V) : vertices(V), adjacencyMatrix(V, vector<int>(V, 0)) {}

void addEdge(int v, int w) {

setlocale(LC\_ALL, "rus");

adjacencyMatrix[v][w] = 1;

adjacencyMatrix[w][v] = 1; // для неориентированного графа

}

void DFS(int startVertex) {

setlocale(LC\_ALL, "rus");

vector<bool> visited(vertices, false);

stack<int> stack;

visited[startVertex] = true;

stack.push(startVertex);

cout << "DFS, начиная с вершины " << startVertex << ": ";

while (!stack.empty()) {

int currentVertex = stack.top();

stack.pop();

cout << currentVertex << " ";

for (int adjacentVertex = vertices - 1; adjacentVertex >= 0; --adjacentVertex) {

if (adjacencyMatrix[currentVertex][adjacentVertex] == 1 && !visited[adjacentVertex]) {

visited[adjacentVertex] = true;

stack.push(adjacentVertex);

}

}

}

cout << endl;

}

};

int main() {

setlocale(LC\_ALL, "rus");

Graph g(6);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(2, 4);

g.addEdge(2, 5);

cout << "Матрица смежности:" << endl;

for (const auto& row : g.adjacencyMatrix) {

for (int val : row) {

cout << val << " ";

}

cout << endl;

}

int startVertex = 0; // Задайте начальную вершину для обхода

g.DFS(startVertex);

return 0;

}

Этот код создает граф с 6 вершинами и неориентированными ребрами, представленными матрицей смежности. Замените значения в addEdge и startVertex в соответствии с вашими требованиями.