Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в глубину»

Выполнили:

студенты группы 21ВВ1.2

Брянцев Артём  
Сущёв Максим  
Тюрин Владислав

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Название**

Обход графа в глубину.

**Лабораторное задание.**

**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.

**Задание 2\*.**

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

**Листинг**

laba7.1.cpp

#include <iostream>

using namespace std;

struct node

{

int vertex; //номер вершины

struct node\* next; //ссылка на след. вершину

};

struct node\* createNode(int);

struct Graph

{

int numVertices; //кол-во вершин

int\* visited; //массив посещёных вершин

struct node\*\* adjLists; //массив на указатели, которые хранят адрес вершин

};

struct Graph\* createGraph(int vertices);

void addEdge(struct Graph\* graph, int src, int dest);

void printGraph(struct Graph\* graph);

void fillGraph(struct Graph\* graph, int\*\* g, int n);

void freeMatrix(struct Graph\* graph);

void dfs(struct Graph\* graph, int vertex);

int\*\* createMatrix(int\*\* g, int n);

void printMatrix(int\*\* g, int n);

void freeMatrix(int\*\* g, int n);

void dfs(int\*\* g, int n, int v, int\* visited);

int main()

{

setlocale(LC\_ALL, "");

srand(time(NULL));

int\*\* g = NULL, n, \* visited = NULL, v;

struct Graph\* graph;

cout << "Введите количество вершин для графа: ";

cin >> n;

g = createMatrix(g, n);

printMatrix(g, n);

cout << "Введите вершину для поиска: ";

cin >> v;

visited = (int\*)malloc(n \* sizeof(int));

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

visited[i] = false;

}

dfs(g, n, v, visited);

cout << endl;

graph = createGraph(n);

fillGraph(graph, g, graph->numVertices);

printGraph(graph);

dfs(graph, v);

cout << endl;

freeMatrix(g, n);

freeMatrix(graph);

free(visited);

}

void dfs(int\*\* g, int n, int v, int\* visited) {

visited[v] = true;

cout << v << " ";

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

if (g[v][i] == 1 && !visited[i]) {

dfs(g, n, i, visited);

}

}

}

int\*\* createMatrix(int\*\* g, int n) {

g = (int\*\*)malloc(n \* sizeof(int\*));

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

\*(g + i) = (int\*)malloc(n \* sizeof(int)); //выделяем память под значения столбцов

}

for (int i = 0; i < n; i++) // цикл по строкам

{

for (int j = i; j < n; j++) // цикл по столбцам строки

{

if (i == j) {

\*(\*(g + i) + j) = 0;

}

else {

\*(\*(g + i) + j) = rand() % 2;

\*(\*(g + j) + i) = \*(\*(g + i) + j);

}

}

}

return g;

}

void printMatrix(int\*\* g, int n) {

cout << " ";

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

cout.width(3);

cout << i;

}

cout << endl << endl;

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

cout << i << " ";

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

{

cout.width(3);

cout << \*(\*(g + i) + j);

}

cout << endl;

}

cout << endl << endl;

}

void freeMatrix(int\*\* g, int n) {

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

free(\*(g + i)); //освобождаем память

}

free(g); //освобождаем память

}

// Алгоритм

void dfs(struct Graph\* graph, int vertex) {

struct node\* adjList = graph->adjLists[vertex];

struct node\* temp = adjList;

graph->visited[vertex] = 1;

cout << vertex << " ";

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (graph->visited[connectedVertex] == 0) {

dfs(graph, connectedVertex);

}

temp = temp->next;

}

}

// Создание вершины

struct node\* createNode(int v) {

struct node\* newNode = (node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Создание пустого графа

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = (Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = (node\*\*)malloc(vertices \* sizeof(struct node\*));

graph->visited = (int\*)malloc(vertices \* sizeof(int));

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

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

// Добавление ребра

void addEdge(struct Graph\* graph, int src, int dest) {

// Проводим ребро от начальной вершины ребра графа к конечной вершине ребра графа

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Проводим ребро из конечной вершины ребра графа в начальную вершину ребра графа

//newNode = createNode(src);

//newNode->next = graph->adjLists[dest];

//graph->adjLists[dest] = newNode;

}

// Выводим граф

void printGraph(struct Graph\* graph) {

for (int v = 0; v < graph->numVertices; v++) {

struct node\* temp = graph->adjLists[v];

cout << endl << "Список смежности вершин " << v << endl;

while (temp) {

if (temp->next == NULL) {

cout << temp->vertex;

}

else {

cout << temp->vertex << ", ";

}

temp = temp->next;

}

cout << endl;

}

cout << endl;

}

//Заполнение графа

void fillGraph(struct Graph\* graph, int\*\* g, int n) {

for (int i = 0; i < n; i++) // цикл по строкам

{

for (int j = n - 1; j >= 0; j--) // цикл по столбцам строки

{

if (\*(\*(g + i) + j) == 1) {

addEdge(graph, i, j);

}

}

}

}

//Освобождение памяти

void freeMatrix(struct Graph\* graph) {

struct node\* temp1, \* temp2;

for (int i = 0; i < graph->numVertices; i++) {

temp1 = graph->adjLists[i];

temp2 = graph->adjLists[i];

while (temp2)

{

temp1 = temp1->next;

free(temp2);

temp2 = temp1;

}

}

free(graph); //освобождаем память

}

laba7.2.cpp

#include <iostream>

using namespace std;

struct node

{

int inf;

struct node\* next; // ссылка на следующий элемент

};

int\*\* createMatrix(int\*\* g, int n);

void printMatrix(int\*\* g, int n);

void freeMatrix(int\*\* g, int n);

void dfs(int\*\* g, int n, int v, int\* visited);

node\* push(int num, node\* vertex);

node\* pop(node\* vertex);

int main()

{

setlocale(LC\_ALL, "");

srand(time(NULL));

int\*\* g = NULL, n, \* visited = NULL, v;

node\* vertex = NULL;

cout << "Введите количество вершин для графа: ";

cin >> n;

g = createMatrix(g, n);

printMatrix(g, n);

cout << "Введите вершину для поиска: ";

cin >> v;

visited = (int\*)malloc(n \* sizeof(int));

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

visited[i] = false;

}

dfs(g, n, v, visited);

cout << endl;

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

visited[i] = false;

}

do {

while (visited[v]) {

v = vertex->inf;

vertex = pop(vertex);

if (!vertex) break;

}

if (!vertex && visited[v]) break;

cout << v << " ";

visited[v] = true;

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

if (g[v][i] == 1) vertex = push(i, vertex);

}

} while (vertex);

freeMatrix(g, n);

free(visited);

}

void dfs(int\*\* g, int n, int v, int\* visited) {

visited[v] = true;

cout << v << " ";

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

if (g[v][i] == 1 && !visited[i]) {

dfs(g, n, i, visited);

}

}

}

int\*\* createMatrix(int\*\* g, int n) {

g = (int\*\*)malloc(n \* sizeof(int\*));

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

\*(g + i) = (int\*)malloc(n \* sizeof(int)); //выделяем память под значения столбцов

}

for (int i = 0; i < n; i++) // цикл по строкам

{

for (int j = i; j < n; j++) // цикл по столбцам строки

{

if (i == j) {

\*(\*(g + i) + j) = 0;

}

else {

\*(\*(g + i) + j) = rand() % 2;

\*(\*(g + j) + i) = \*(\*(g + i) + j);

}

}

}

return g;

}

void printMatrix(int\*\* g, int n) {

cout << " ";

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

cout.width(3);

cout << i;

}

cout << endl << endl;

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

cout << i << " ";

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

{

cout.width(3);

cout << \*(\*(g + i) + j);

}

cout << endl;

}

cout << endl << endl;

}

void freeMatrix(int\*\* g, int n) {

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

free(\*(g + i)); //освобождаем память

}

free(g); //освобождаем память

}

node\* push(int num, node\* vertex) {

node\* tmp = (node\*)malloc(sizeof(node));

if (vertex == NULL) {

vertex = tmp;

vertex->inf = num;

vertex->next = NULL;

}

else {

tmp->next = vertex;

tmp->inf = num;

vertex = tmp;

}

return vertex;

}

node\* pop(node\* vertex) {

node\* tmp = vertex;

vertex = vertex->next;

free(tmp);

return vertex;

}

**Результат работы программы**

Результаты работы программы (задание 1 (пункты 1, 2, 3)) показаны на рисунке 1.

**Рисунок 1 — Результаты работы программы**

Результаты работы программы (задание 2) показаны на рисунке 2.

Изображение выглядит как текст

Автоматически созданное описание

**Рисунок 2 — Результаты работы программы**

### Выводы

Обход графа – одна из наиболее распространенных операций с графами. Задачей обхода является прохождение всех вершин в графе. Обходы применяются для поиска информации, хранящейся в узлах графа, нахождения связей между вершинами или группами вершин и т. д.

Функция обхода графа реализована через рекурсию при помощи массива, который хранит данные о посещённых вершинах.

Функция обхода графа без рекурсии реализована при помощи массива, который хранит данные о посещённых вершинах, и при помощи стека.