Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

**по курсу «Логика и основы алгоритмизации в инженерных задачах»**

на тему «Обход графа в глубину»

Выполнили:

студент группы 20ВВ2

Александров В.С.

Кирюткин И.А.

Сафронов Д.В.

Принял:

к.т.н., доцент

Митрохин М.А.

**Пенза 2021**

**Цель работы:**   Составить программу, которая будет осуществлять процедуру обхода графа в глубину и оценить сложность работы.

**Лабораторные задания:**

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.

**Листинг :**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <ctime>

#include <cmath>

#include <stack>

#include <queue>

using namespace std;

struct node

{

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph

{

int \*visited;

int numVertices;

struct node\*\* adjLists;

};

struct Graph\* createGraph(int vertices);

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

void printGraph(struct Graph\* graph);

void search(struct Graph\* graph);

//void DFS2(struct Graph\* graph, int vertex);

int main1()

{

setlocale(LC\_ALL, "Rus");

int versh, \* a, i, ver, conect;

printf("Введите количество вершин: ");

scanf("%d", &versh);

a = (int\*)malloc(versh \* sizeof(int));

struct Graph\* graph = createGraph(versh);

for (i = 0; i < versh - 1; i++)

{

conect = i+1;

ver = conect-1;

addEdge(graph, ver, conect);

}

printGraph(graph);

//DFS2(graph, 2);

//search(graph);

return 0;

}

struct node\* createNode(int v)

{

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

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

graph->numVertices = vertices;

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

graph->visited = 0;

int i;

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

graph->adjLists[i] = NULL;

return graph;

}

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

{

struct node\* newNode = createNode(dest);

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

graph->adjLists[src] = newNode;

newNode = createNode(src);

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

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph)

{

int v;

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

{

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

printf("\n Список смежности вершины %d\n ", v);

while (temp)

{

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

/\*void DFS2(struct Graph\* graph, int vertex) {

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

struct node\* temp = adjList;

int visited[2];

visited[vertex] = 1;

printf(" вершина %d \n", vertex);

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (visited[connectedVertex] == 0) {

DFS2(graph, connectedVertex);

}

temp = temp->next;

}

}\*/

void search(struct Graph\* graph)

{

int v;

printf("\n Список смежности вершины");

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

{

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

while (temp)

{

printf("% d", temp->vertex);

temp = temp->next;

}

}

}

//////////////////////////////

//поиск в глубину

void DFS(int st, int\*\* graph, int size, bool\* visited)

{

int r;

cout << st + 1 << " ";

visited[st] = true;

for (r = 0; r <= size; r++)

if ((graph[st][r] != 0) && (!visited[r]))

DFS(r, graph, size, visited);

}

void dfs1(int p, int\*\* G, int N)

{

queue<int> S;

int \*v;

int t;

v = (int\*)malloc(N \* sizeof(int));

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

v[i] = 0;

}

S.push(p);

v[p]+=1;

while (!S.empty())

{

t = S.front();

S.pop();

cout << t + 1 << " ";

for (int i = 0; i < N ; i++)

if (G[t][i] && !v[i])

{

S.push(i);

v[i]+=1;

}

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

int start, i, j;

int size, \*\* graph, n;

cout << "Введите размерность: " << endl;

cin >> size;

srand(time(NULL));

graph = (int\*\*)malloc(size \* sizeof(int));

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

{

graph[i] = (int\*)malloc(size \* sizeof(int));

}

bool\* visited = new bool[size];

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

for (int j = 0; j < i + 1; j++) {

int num = rand() % 2;

graph[i][j] = num;

graph[j][i] = num;

}

}

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

for (i = 0; i < size; i++)

{

visited[i] = false;

for (j = 0; j < size; j++)

cout << " " << graph[i][j];

cout << endl;

}

cout << "Стартовая вершина "; cin >> start;

//массив посещенных вершин

bool\* vis = new bool[size];

cout << "Порядок обхода рекурсивный метод: ";

int start1 = start, size1 = size;

DFS(start - 1, graph, size, visited);

cout << "\n " << endl;

cout << "Порядок обхода не рекурсивный метод: ";

dfs1(start-1, graph, size);

delete[]visited;

cout << "\n " << endl;

cout << "Порядок обхода списки \n";

main1();

return(0);

}

* **Псевдокод:**

АЛГ node

int vertex

struct node \*next

create node(int)

graph

int numVertex

node \*\* abjlist

Прототип структуры Graph\*   
Прототип функции addEdge  
Прототип функции printGraph

АЛГ main1

int versh, \* a, i, ver, conect;  
 Выводим "Введите количество вершин: “  
 Считываем число в versh

Создаём структуру на указаное число элементов  
 ДЛЯ i от 0 до versh делаем

conect = рандомному числу  
 ver = conect+1;  
 Вызов addEdge(graph, ver, conect);

Вызов функции printGraph

Конец

Структура createNode  
 newNode->vertex = v;  
 newNode->next = NULL;  
 конец newNode;

Структ createGraph  
 graph->numVertices = vertices;  
  
 int i;   
 Для i от 0 до vertices  
 graph->adjLists[i] = NULL;  
  
Конец graph;  
  
  
АЛГ addEdge  
 Структ newNode = createNode(dest);  
 newNode->next = graph->adjLists[src];  
 graph->adjLists[src] = newNode;  
  
 newNode = createNode(src);  
 newNode->next = graph->adjLists[dest];  
 graph->adjLists[dest] = newNode;  
Конец addEdge  
  
АЛГ printGraph  
  
 int v;  
 ДЛЯ v от 0 до graph  
  
 Структ temp = graph->adjLists[v];  
 Вывод "\n Список смежности вершины %d\n ", v  
 Пока (temp!=0)  
 Конец  
Вывод temp  
temp =Адрес( temp) + 1  
  
ПРОЦ DFS  
  
 int r;  
 Ввод st + 1   
 visited[st] = true;  
 Для r от 0 до size  
 Если graph[st][r] != 0 И !visited[r]  
 DFS(r, graph, size, visited);  
 Конец  
  
АЛГ dfs1  
  
 stac<int> S  
 Проверяем, является ли v() вектором  
 int t  
 S.push(p)  
 v[p] = v[p] + 1   
 ПОКА (!S.empty())  
  
t = Результату функции [S.top](https://vk.com/away.php?to=http%3A%2F%2FS.top&cc_key=)()  
Вызов S.pop()  
Ввод t + 1   
ДЛЯ i от N - 1 до 0  
 Если G[t][i] И !v[i]  
  
 Вызов S.push(i)  
 v[i] = v[i]+1  
Конец  
  
АЛГ main()  
 int start, i, j;  
  
 int size, \*\* graph, n;  
  
  
 Вывод "Введите размерность: "   
 Ввод size  
  
  
Для i от 0 до size  
 visited = new bool[size];  
  
 Для i от 0 до size  
 Для j от 0 до i + 1  
 num = рандомное число  
 graph[i][j] = num  
 graph[j][i] = num  
  
  
Вывод "Матрица смежности графа: "   
Для i от 0 до size  
  
 visited[i] = false  
 for (j = 0; j < size; j++)  
 cout « " " « graph[i][j]  
 cout « endl  
}  
 cout « "Стартовая вершина "; cin » start

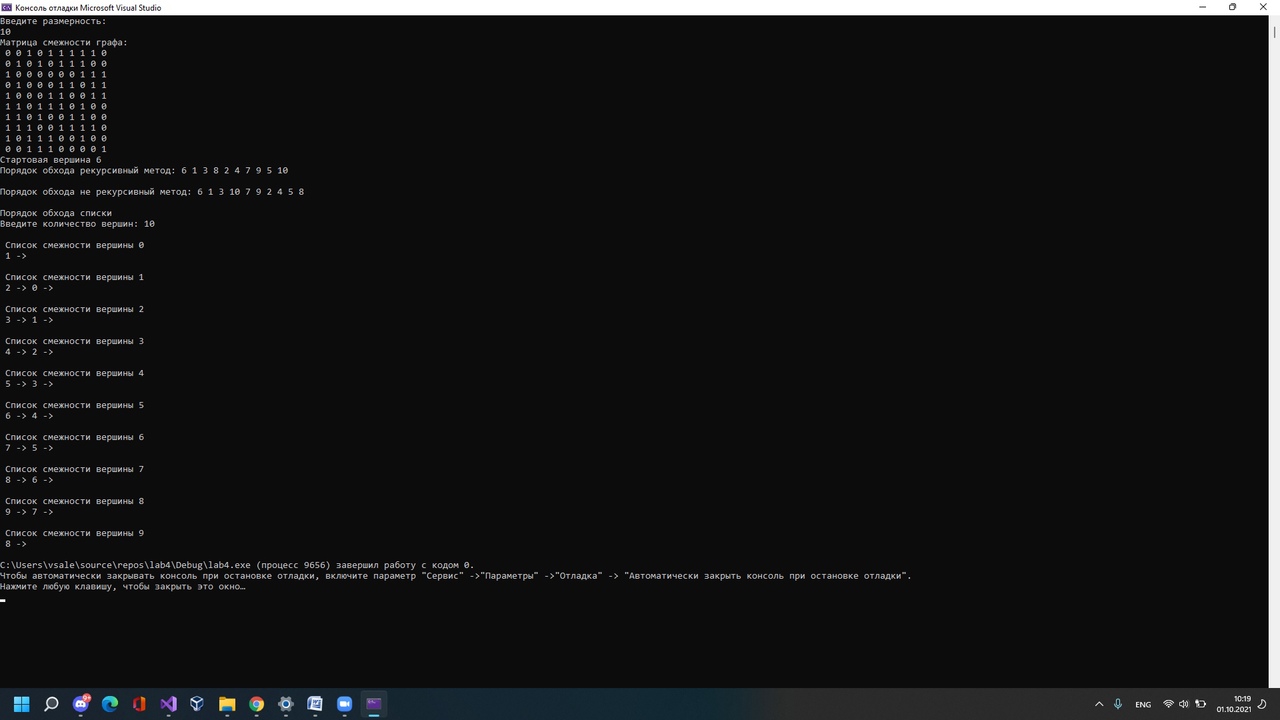
vis = new bool[size]  
Вывод "Порядок обхода рекурсивный метод: "  
  
int start1 = start, size1 = size  
  
DFS(start - 1, graph, size, visited)  
   
 Вывод "Порядок обхода не рекурсивный метод: "  
 Проц dfs1(start-1, graph, size);  
 delete[]visited;

Вывод "Порядок обхода списки ";  
 Вызов функции main1

Конец



**Результаты программы :**



**Вывод:** Составили программу, осуществляющую обход графа в глубину.