Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Л и ОА в ИЗ»

на тему «Обход графа в глубину»

**Выполнили:**

студентом группы 19ВВ3

Земляков В.Д.

**Принял:**

Митрохин М.А.

Пенза 2020

**Цель работы:**

Изучить алгоритмы обхода графа в глубину.

**Ход работы:**

Поиск в **глубину** (англ. Depth-first search, DFS) — один из методов **обхода графа**. Стратегия поиска в **глубину**, как и следует из названия, состоит в том, чтобы идти «вглубь» **графа**, насколько это возможно. Алгоритм поиска описывается рекурсивно и не рекурсивно. Их часто используют для проверки связности, поиска цикла и компонент сильной связности и для топологической сортировки.

Графы можно представить в виде матриц и списков смежности.

**Листинг:**

**Header.h:**

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <conio.h>

#include<locale.h>

#include <String.h>

#include <time.h>

#include <stack>

**Lab4.cpp:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Header.h"

typedef struct NODE{

int v;

NODE\* Next;

};

typedef struct LIST {

int M;

struct NODE\*\* HEAD;

};

NODE\* CreateNODE(int v){

NODE\* NewNODE = (NODE\*)malloc(sizeof(NODE));

NewNODE->v = v;

NewNODE->Next = NULL;

return NewNODE;

}

LIST\* CreateLIST(int M) {

LIST\* list=(LIST\*)malloc(sizeof(LIST));

list->M=M;

list->HEAD=(NODE \*\*)malloc(sizeof(struct NODE\*)\*M);

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

list->HEAD[i] = NULL;

}

return list;

}

void AddFirstElement(NODE\*\* HEAD, NODE\* NewNODE) {

NewNODE->Next = \*HEAD;

\*HEAD = NewNODE;

}

void after(NODE\* AfterNODE, NODE\* NewNODE) {

NewNODE->Next = AfterNODE->Next;

AfterNODE->Next = NewNODE;

}

void before(NODE\*\* HEAD,NODE\* BeforeNODE,NODE\* NewNODE) {

NODE\* active = \*HEAD;

if (\*HEAD == BeforeNODE) {

AddFirstElement(HEAD,NewNODE);

return;

}

while (active&&active->Next!=BeforeNODE)

active = active->Next;

if (active)

after(active,NewNODE);

}

void AddLastElement(NODE\*\* HEAD, NODE\* NewNODE) {

NODE\* active = \*HEAD;

if (\*HEAD == NULL) {

AddFirstElement(HEAD,NewNODE);

return;

}

while (active->Next)

active = active->Next;

after(active,NewNODE);

}

void outputLIST(LIST\* list) {

NODE\* active;

for (int i = 0; i < list->M;i++) {

printf("%d",i+1);

active = list->HEAD[i];

while (active!=NULL) {

printf("->%d",active->v+1);

active = active->Next;

}

printf("\n");

}

}

void transfer(int\*\* Array,int N, LIST\* list) {

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

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

if(Array[i][j]==1){

NODE\* NewNODE = CreateNODE(j);

AddLastElement(&list->HEAD[i],NewNODE);

}

}

}

}

void output(int\*\* Array, int N) {

printf("\n");

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

{

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

{

printf("%d ", Array[i][j]);

}

printf("\n");

}

}

void DFSLIST(int v,LIST \*list, int\* visitedArray, int N) {

printf("%d ", v + 1);

visitedArray[v] = 1;

NODE\* active = list->HEAD[v];

while (active) {

if (visitedArray[active->v] == 0) {

DFSLIST(active->v,list,visitedArray,N);

}

active = active->Next;

}

}

void Gen(int\*\* Array, int N) {

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

{

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

{

Array[i][j] = rand() % 2;

Array[j][i] = Array[i][j];

if (i == j) {

Array[i][j] = 0;

}

}

}

output(Array, N);

}

void DFS(int u, int\*\* Array, int\* visitedArray, int N) {

printf("%d ", u+1);

visitedArray[u] = 1;

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

if (Array[u][uNext] && visitedArray[uNext] != 1) {

DFS(uNext, Array, visitedArray,N);

}

}

}

void DFSN(int v, int \*\*Array,int\* visitedArray, int N)

{

std::stack<int>S;

S.push(v);

while (!S.empty()) {

v = S.top();

S.pop();

if (visitedArray[v]==0) {

visitedArray[v] = 1;

printf("%d ", v + 1);

int i = N - 1;

while (i >= 0) {

if (Array[v][i] == 1 && visitedArray[i] == 0) {

S.push(i);

}

i--;

}

}

}

}

int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

int M = 0;

int N = 0;

int u= 0;

int\*\* ArrayM1 = NULL;

int\* Visitedarray = NULL;

printf("Введите размерность матрицы:");

scanf("%d", &N);

ArrayM1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

ArrayM1[i] = (int\*)malloc(N \* sizeof(int));

}

Gen(ArrayM1, N);

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

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

Visitedarray[i] = 0;

}

printf(" Рекурсивный\n");

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

if ( Visitedarray[i] == 0) {

DFS(i, ArrayM1, Visitedarray,N);

}

}

printf("\n");

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

Visitedarray[i] = 0;

}

printf("Не рекурсивный\n");

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

if (Visitedarray[i] == 0) {

DFSN(i, ArrayM1, Visitedarray, N);

}

}

LIST\* list = CreateLIST(N);

transfer(ArrayM1, N,list);

printf("\n");

outputLIST(list);

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

Visitedarray[i] = 0;

}

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

if (Visitedarray[i] == 0) {

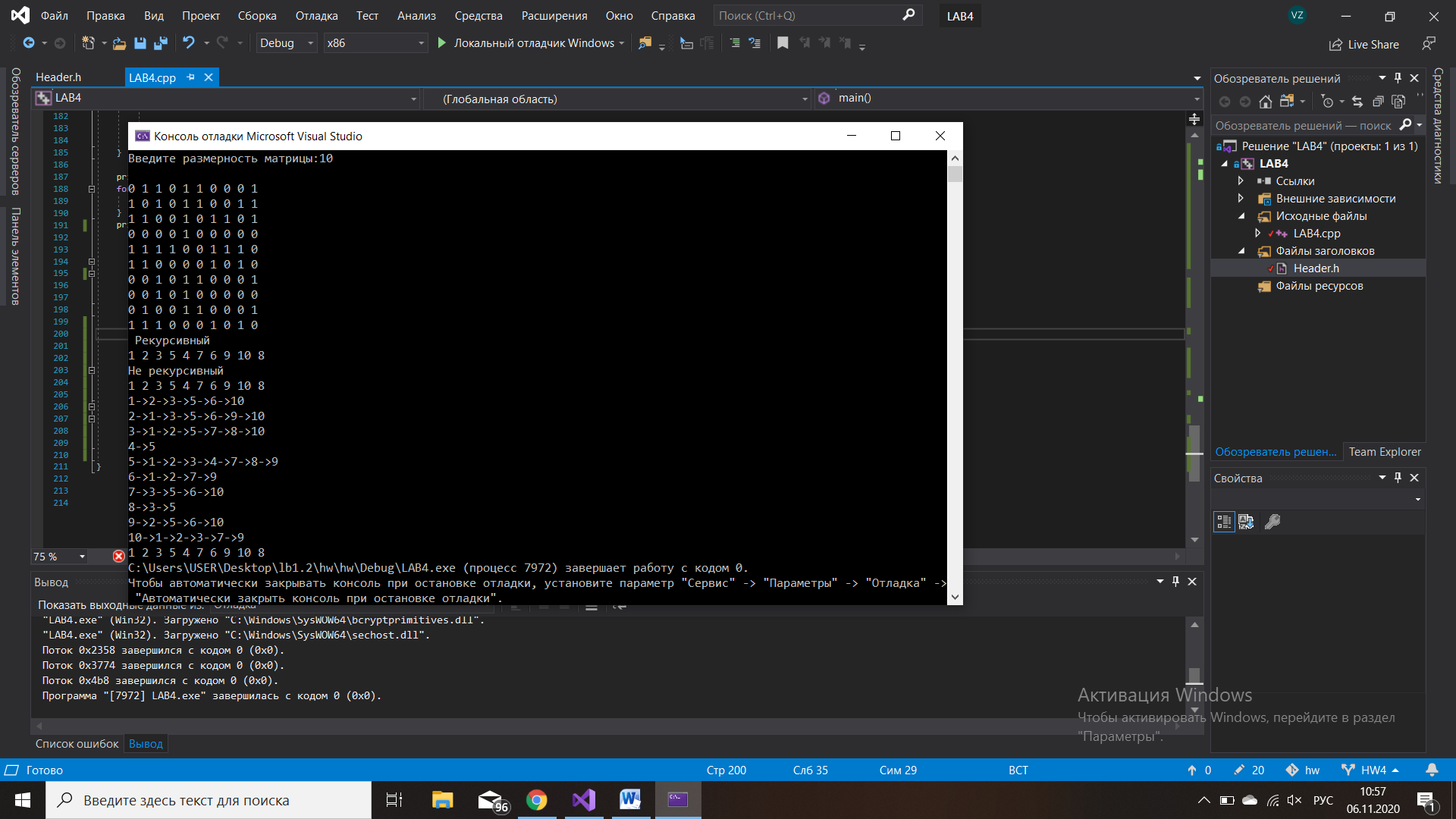
DFSLIST(i, list, Visitedarray, N);

}

}

}

### Результат работы программы:



**Рисунок1. Результат**

### Выводы

В ходе выполнения лабораторной работы изучил алгоритмы обхода графов глубину.