Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Л и ОА в ИЗ»

на тему «Обход графа в ширину»

**Выполнили:**

студентом группы 19ВВ3

Земляков В.Д.

**Принял:**

Митрохин М.А.

Пенза 2020

**Цель работы:**

Изучить обход графа в ширину.

**Ход работы:**

Обход графа в ширину – еще один распространенный способ обхода

графов.

Основная идея такого обхода состоит в том, чтобы посещать вершины

по уровням удаленности от исходной вершины. Удалённость в данном

случае понимается как количество ребер, по которым необходимо прейти до

достижения вершины.

**Листинг:**

**Header.h:**

#pragma once

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <conio.h>

#include<locale.h>

#include <String.h>

#include <time.h>

#include <stack>

#include <queue>

#include <iostream>

#include <limits.h>

**lab5.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");

}

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");

}

printf("\n");

}

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 BFS(int \*\*Array, int v, int\* used,int N) {

std::queue <int> Q;

Q.push(v);

used[v] = 1;

while (!Q.empty()) {

v = Q.front();

Q.pop();

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

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

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

Q.push(i);

used[i] = 1;

}

}

}

}

void BFSLIST(LIST\* list, int v, int\* used) {

std::queue <int> Q;

Q.push(v);

used[v] = 1;

while (!Q.empty()) {

v= Q.front();

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

Q.pop();

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

while (active) {

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

Q.push(active->v);

used[active->v] = 1;

}

active = active->Next;

}

}

}

void myBFS(int N, int\*\* Array,int\* used, int v) {

int\* List = new int[N];

int Count, Head;

int i;

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

List[i] = 0;

Count = Head = 0;

List[Count++] = v;

used[v] = 1;

while (Head < Count) {

v = List[Head++];

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

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

if (Array[v][i] && !used[i]) {

List[Count++] = i;

used[i] = 1;

}

}

}

void zeroing(int\* used,int N) {

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

used[i] = 0;

}

printf("\n");

}

void main() {

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

int M = 0;

int N = 0;

int v = 0;

int\*\* ArrayM1 = NULL;

int\* used = 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);

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

LIST\* list = CreateLIST(N);

transfer(ArrayM1, N, list);

outputLIST(list);

zeroing(used,N);

double start = clock();

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

if (used[i] == 0) {

BFS(ArrayM1,i , used, N);

}

}

double end = clock();

printf("%f", (end - start) / CLOCKS\_PER\_SEC);

zeroing(used, N);

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

if (used[i] == 0) {

BFSLIST(list, i, used);

}

}

zeroing(used, N);

double start2 = clock();

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

if (used[i] == 0) {

myBFS(N, ArrayM1, used, v);

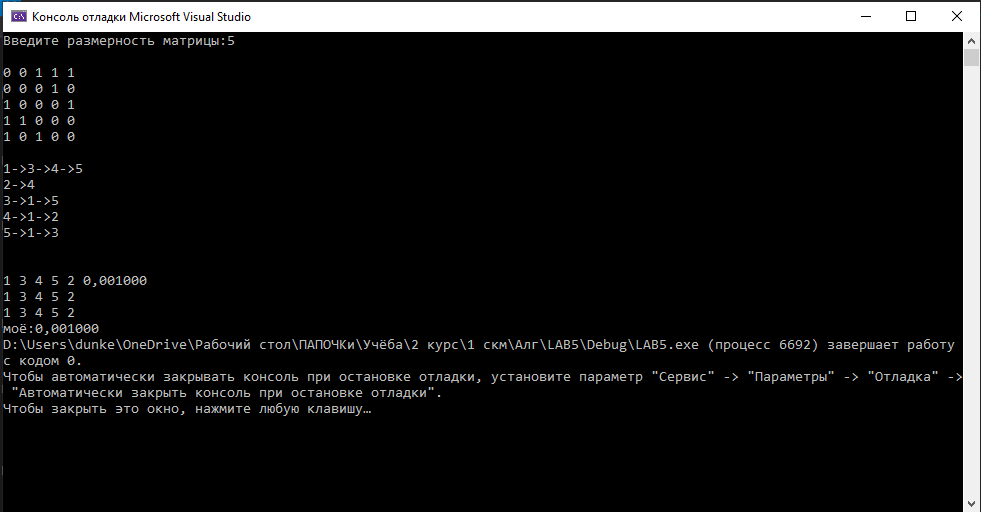
}

}

double end2 = clock();

printf("\nмоё:%f", (end2 - start2) / CLOCKS\_PER\_SEC);}

### Результат работы программы:



**Рисунок1. Результат работы программы**

### Выводы

В ходе выполнения лабораторной работы изучил обход графа в ширину.