Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Л и ОА в ИЗ»

на тему «Поиск расстояний в графе»

**Выполнили:**

студентом группы 19ВВ3

Земляков В.Д.

**Принял:**

Митрохин М.А.

Пенза 2020

**Цель работы:**

Изучить поиск расстояний в графе.

**Ход работы:**

Поиск расстояний – довольно распространенная задача анализа графов.

Для поиска расстояний можно использовать процедуры обхода графа.

Для этого при каждом переходе в новую вершину необходимо запоминать,

сколько шагов до нее мы сделали. При этом вектор, который хранил

информацию о посещении вершин становится вектором расстояний.

Довольно просто модернизировать для поиска расстояний в графе алгоритм

обхода в ширину, т.к. этот алгоритм проходит вершины по уровням

удаленности, то для не ориентированного графа для вершин каждого

следующего уровня глубины расстояние от исходной вершины

увеличивается на 1. 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>

**Lab6.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 zeroing(int\* used, int N) {

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

used[i] = 0;

}

printf("\n");

}

void ClearDistance(int\* Dist, int N) {

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

Dist[i] = -1;

}

printf("\n");

}

void SBFS(int\*\* Array, int v, int\* DIST, int N) {

std::queue <int> Q;

Q.push(v);

DIST[v] = 0;

while (!Q.empty()) {

v = Q.front();

Q.pop();

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

if (Array[v][i] == 1 && DIST[i] == -1) {

Q.push(i);

DIST[i] = DIST[v]+1;

}

}

}

}

void SBFSList(LIST\* list, int v, int\* DIST, int N) {

std::queue <int> Q;

Q.push(v);

DIST[v] = 0;

while (!Q.empty()) {

v = Q.front();

Q.pop();

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

while (active) {

if (DIST[active->v] == -1) {

Q.push(active->v);

DIST[active->v] = DIST[v] + 1;

}

active = active->Next;

}

}

}

void SDFS(int v, int\*\* Array, int\* DIST, int N)

{

std::stack<int>S;

S.push(v);

DIST[v] = 0;

while (!S.empty()) {

v = S.top();

S.pop();

int i = N - 1;

while (i >= 0) {

if (Array[v][i] && DIST[i] == -1) {

S.push(i);

DIST[i] = DIST[v] + 1;

}

i--;

}

}

}

void SDFSList(int v, LIST\*list, int\* DIST, int N)

{

std::stack<int>S;

S.push(v);

DIST[v] = 0;

while (!S.empty()) {

v = S.top();

S.pop();

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

int i = N - 1;

while (active) {

if (DIST[active->v] == -1) {

S.push(active->v);

DIST[active->v] = DIST[v] + 1;

}

active = active->Next;

}

}

}

void main() {

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

int M = 0;

int N = 0;

int v = 0;

int\*\* ArrayM1 = NULL;

int\* used = NULL;

int\* DIST = 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));

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

LIST\* list = CreateLIST(N);

transfer(ArrayM1, N, list);

outputLIST(list);

ClearDistance(DIST,N);

double start = clock();

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

SBFS(ArrayM1, i, DIST, N);

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

printf("%d ", DIST[j]);

}

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

DIST[j] = -1;

}

printf("\n");

}

double end = clock();

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

ClearDistance(DIST, N);

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

SBFSList(list, i, DIST, N);

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

printf("%d ", DIST[j]);

}

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

DIST[j] = -1;

}

printf("\n");

}

ClearDistance(DIST, N);

double start2 = clock();

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

SDFS(i,ArrayM1,DIST,N);

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

printf("%d ", DIST[j]);

}

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

DIST[j] = -1;

}

printf("\n");

}

double end2 = clock();

printf("в глубину:%f\n", (end2 - start2) / CLOCKS\_PER\_SEC);

ClearDistance(DIST, N);

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

SDFSList(i, list, DIST, N);

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

printf("%d ", DIST[j]);

}

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

DIST[j] = -1;

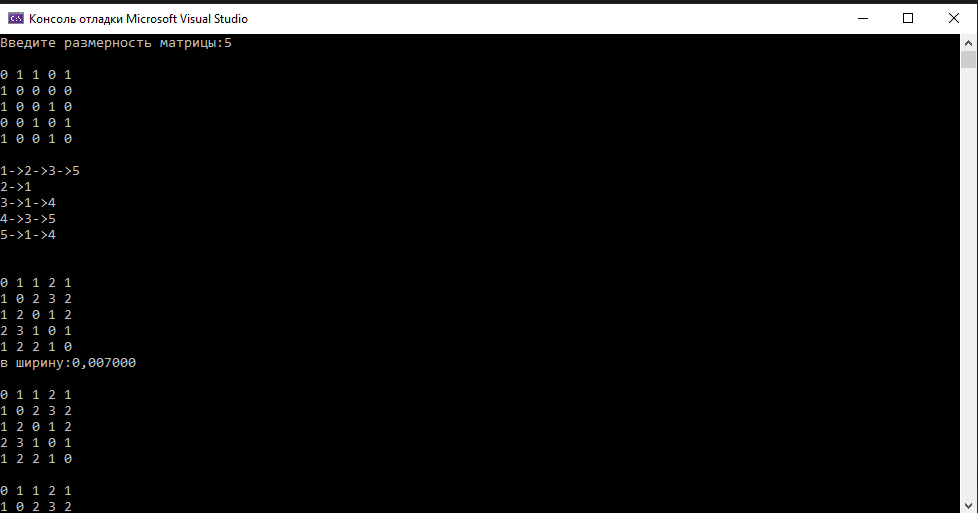
}

printf("\n");

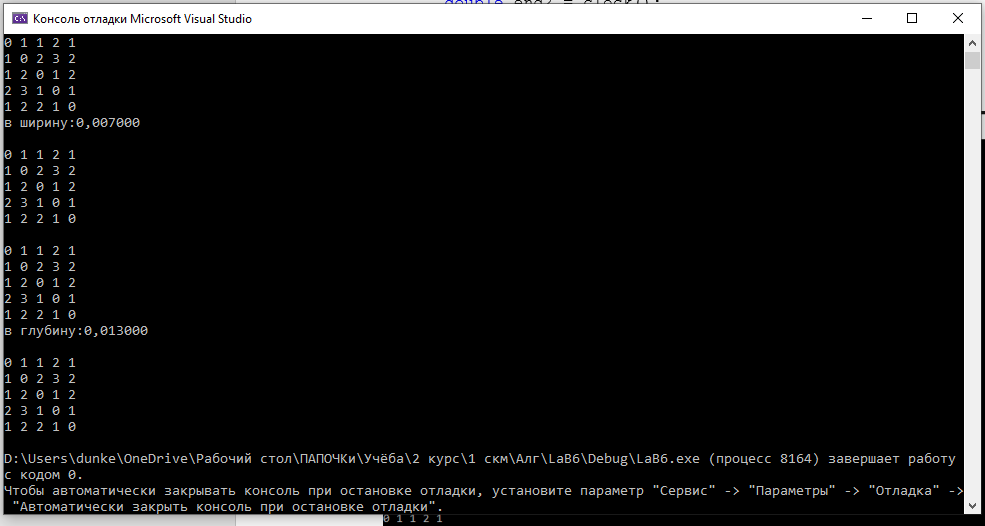
}

}

### Результат работы программы:



**Рисунок1. Результат работы программы**



**Рисунок 2. Результат работы программы**

### Выводы

В ходе выполнения лабораторной работы изучил алгоритмы поиска расстояний в графах.