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»

**Выполнил:**

Студент группы 19ВВ3

Субботкин М.В.

**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:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <time.h>

#include <stdio.h>

#include <random>

#include <queue>

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 matrix\_into\_list(int\*\* mass, int f, LIST\* list) {

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

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

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

NODE\* NewNODE = CreateNODE(j);

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

}

}

}

}

void BFS(int\*\* mass, int v, int\* visited, int f) {

std::queue <int> vertex\_queue;

vertex\_queue.push(v);

visited[v] = 1;

while (!vertex\_queue.empty()) {

v = vertex\_queue.front();

vertex\_queue.pop();

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

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

if (mass[v][i] == 1 && visited[i] == 0) {

vertex\_queue.push(i);

visited[i] = 1;

}

}

}

}

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

std::queue <int> vertex\_queue;

vertex\_queue.push(v);

visited[v] = 1;

while (!vertex\_queue.empty()) {

v = vertex\_queue.front();

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

vertex\_queue.pop();

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

while (active) {

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

vertex\_queue.push(active->v);

visited[active->v] = 1;

}

active = active->Next;

}

}

}

void new\_BFS(int f, int\*\* mass, int\* visited, int v) {

int\* list = new int[f];

int counter, head;

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

list[i] = 0;

counter = head = 0;

list[counter++] = v;

visited[v] = 1;

while (head < counter) {

v = list[head++];

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

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

if (mass[v][i] && !visited[i]) {

list[counter++] = i;

visited[i] = 1;

}

}

}

int main() {

srand(time(NULL));

int f;

int v = 0;

printf("Enter lenght: ");

scanf("%d", &f);

int\* vis = (int\*)malloc(sizeof(int) \* f);

int\* vert = (int\*)malloc(sizeof(int) \* f);

int\*\* mass = (int\*\*)malloc(sizeof(int) \* f);

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

vis[i] = 0;

}

printf("\n");

printf("\n");

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

vert[i] = i + 1;

printf("%d ", vert[i]);

}

printf("\n");

printf("\n");

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

mass[i] = (int\*)malloc(sizeof(int) \* f);

}

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

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

int c = rand() % 101;

if (c <= 50) {

mass[i][j] = 1;

}

else mass[i][j] = 0;

if (i == j) {

mass[i][j] = 0;

}

}

}

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

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

if (mass[i][j] != mass[j][i])

mass[i][j] = mass[j][i];

}

}

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

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

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

}

printf("\n");

}

LIST\* list = CreateLIST(f);

matrix\_into\_list(mass, f, list);

outputLIST(list);

printf("\nDefoult BFS\n");

double start = clock();

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

if (vis[i] == 0) {

BFS(mass, i, vis, f);

}

}

double end = clock();

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

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

vis[i] = 0;

}

printf("\nBFS List\n");

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

if (vis[i] == 0) {

BFSLIST(list, i, vis);

}

}

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

vis[i] = 0;

}

printf("\nMy BFS\n");

double start2 = clock();

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

if (vis[i] == 0) {

new\_BFS(f, mass, vis, v);

}

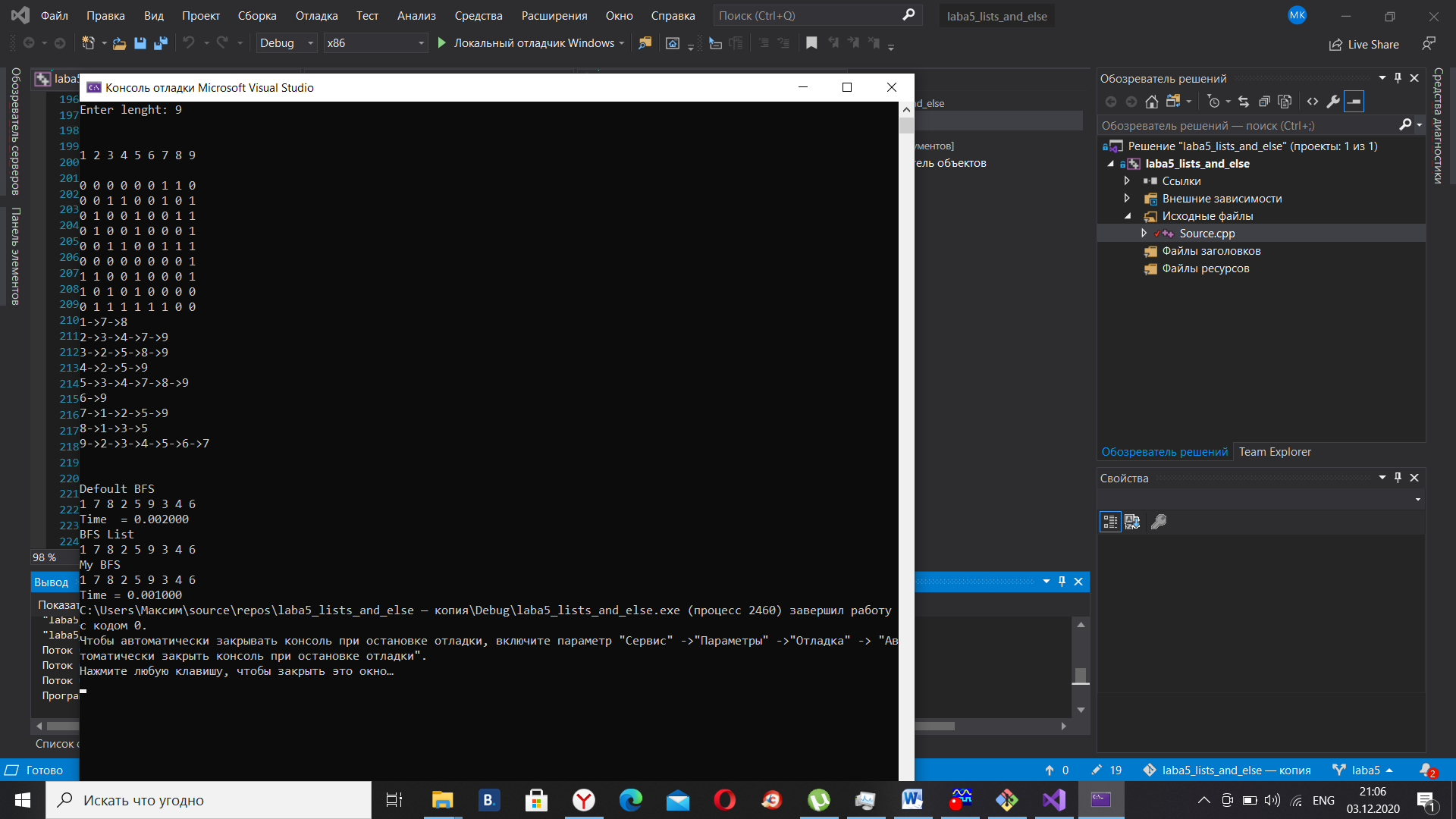
}

double end2 = clock();

printf("\nTime = %f", (end2 - start2) / CLOCKS\_PER\_SEC);

}

### Результат работы программы:



**Рисунок1. Результат работы программы**

### Выводы

В ходе выполнения лабораторной работы изучил обход графа в ширину.