Министерство образования и науки Российской Федерации

Пензенский государственный университет

Факультет вычислительной техники

**Отчёт**

По лабораторной работе №6

По курсу "ЛиОАвИЗ"

Выполнили:

Студент группы 20ВВ1

Репин И.В.

Приняла:

Юрова О.В.

Пенза 2021

**Название:** Поиск расстояний в графе

**Цель работы:** выполнить ряд заданий

**Лабораторное задание:**

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного графа G. Выведите матрицу на экран.

2. Для сгенерированного графа осуществите процедуру поиска

расстояний, реализованную в соответствии с приведенным выше описанием.

3.\* Реализуйте процедуру поиска расстояний для графа, представленного

списками смежности.

Задание 2\*

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.

2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.

3. Оцените время работы реализаций алгоритмов поиска расстояний на

основе обхода в глубину и обхода в ширину для графов разных порядков.

**Описание метода решения задачи:**

Задание 1.

1. Заполняем двумерный массив случайными 0 и 1, следим за тем что бы значения были симметричны относительно главное оси
2. Алгоритм BFSD(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. Обновить вектор расстояний DIST [ x ] = 0;

2.4. ПОКА Q != Æ очередь не пуста ВЫПОЛНЯТЬ

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.9. ЕСЛИ G(v,i) = = 1И DIST = = -1

2.10. ТО

2.11. Поместить i в очередь Q.push(i);

2.12. Обновить вектор расстояний DIST [ i ] = DIST [ v ] + 1;

1. Аналогичную функцию реализуем для графа представленного списками смежности

**Листинг:**

// lb5.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <queue>

#include <time.h>;

using namespace std;

int\*\* m;

int\*\* dfsvislst;

int num;

int\* vis;

int\* vislst;

int\* dfsvis;

int\* dfscntrlst;

queue <int> q;

void BFS(int s);

struct list\* init(int a);

struct list\* addelem(list\* lst, int number, int id, int line);

void listprint(list\* lst);

void BFSlst(int s, list\* lst);

void DFS(int num);

void DFSlst(int n, list\* lst, int j);

struct list

{

int field, id, line; // поле данных

struct list\* ptr; // указатель на следующий элемент

};

struct list\* init(int a) // а- значение первого узла

{

struct list\* lst;

// выделение памяти под корень списка

lst = (struct list\*)malloc(sizeof(struct list));

lst->field = a;

lst->id = a;

lst->line = -1;

lst->ptr = NULL; // это последний узел списка

return(lst);

}

struct list\* addelem(list\* lst, int number, int id, int line)

{

struct list\* temp, \* p;

temp = (struct list\*)malloc(sizeof(list));

p = lst->ptr; // сохранение указателя на следующий узел

lst->ptr = temp; // предыдущий узел указывает на создаваемый

temp->field = number; // сохранение поля данных добавляемого узла

temp->id = id;

temp->line = line;

temp->ptr = p; // созданный узел указывает на следующий элемент

return(temp);

}

void listprint(list\* lst)

{

struct list\* p;

p = lst->ptr;

int line, id;

line = 0;

id = 0;

do {

if (p->line > line) {

printf("\n");

line++;

id = 0;

}

printf("%4i", p->id);

p = p->ptr; // переход к следующему узлу

} while (p != NULL);

}

void BFS(int s) {

vis[s] = 0;

q.push(s);

while (!q.empty()) {

int v = q.front();

q.pop();

printf("%4i", v);

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

if ((m[v][i] == 1) and (vis[i] == -1)) {

q.push(i);

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

}

}

}

printf("\nDists: ");

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

printf("%4i", vis[i]);

}

}

void BFSlst(int s, list\* lst) {

q.push(s);

struct list\* p;

p = lst->ptr;

vislst[s] = 0;

while (p->line != s) {

p = p->ptr;

}

int i = 0;

while (!q.empty()) {

int v = q.front();

q.pop();

printf("%4i", v);

p = lst->ptr;

while (p->line != v) {

p = p->ptr;

}

int oline = p->line;

int nline = p->line;

while (oline == nline) {

i = p->id;

if (vislst[i] == -1) {

q.push(i);

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

}

p = p->ptr;

if (!p == NULL) {

nline = p->line;

}

else {

nline = -1;

}

}

}

printf("\nDists: ");

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

printf("%4i", vislst[i]);

}

}

void DFS(int n) {

dfsvis[n] = vislst[n] + 1;

printf("%4i", n);

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

if ((m[n][i] == 1) && (dfsvis[i] == -1)) {

DFS(i);

}

}

}

void DFSlst(int n, list\* lst, int o) {

struct list\* p;

int i;

p = lst->ptr;

dfscntrlst[n] = dfscntrlst[o] + 1;

printf("%4i", n);

while (p->line != n) {

p = p->ptr;

}

while (!p == NULL) {

i = p->id;

if (n != o) {

struct list\* check;

check = lst->ptr;

while (check->line != o) {

check = check->ptr;

}

int newline = o;

bool connected = false;

while (newline == o) {

int checki = check->id;

if (checki == i) {

connected = true;

}

check = check->ptr;

if (!check == NULL) {

newline = check->line;

}

else {

newline = -1;

}

}

if ((dfscntrlst[i] == -1) and (connected == false)) {

DFSlst(i, lst, n);

}

}

else {

if (dfscntrlst[i] == -1) {

DFSlst(i, lst, n);

}

}

p = p->ptr;

}

}

int main()

{

//generate array

clock\_t start, end;

int generate, choice;

printf("Input array size: ");

scanf("%i", &num);

m = new int\* [num];

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

m[i] = new int[num];

}

srand(time(NULL));

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

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

if (j == i) {

m[i][j] = 0;

}

else if (j > i) {

generate = rand() % 100;

if (generate >= 30) {

m[i][j] = 1;

}

else {

m[i][j] = 0;

}

}

else {

m[i][j] = m[j][i];

}

}

}

//print array

printf("array:");

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

printf("\n");

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

printf("%4i", m[i][j]);

}

}

printf("\nInput start point: ");

scanf("%i", &choice);

//generate list

struct list\* lst = init(0);

for (int i = (num - 1); i >= 0; i--) {

for (int j = (num - 1); j >= 0; j--) {

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

addelem(lst, 1, j, i);

}

}

}

//print list

printf("\nList: \n");

listprint(lst);

//rounds

vis = new int[num];

vislst = new int[num];

dfsvis = new int[num];

dfscntrlst = new int[num];

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

vis[i] = -1;

vislst[i] = -1;

dfscntrlst[i] = -1;

dfsvis[i] = -1;

}

//result

start = clock();

printf("\nBFS: ");

BFS(choice);

end = clock();

double time\_spent = (double)(end - start) \* 1000000 / CLOCKS\_PER\_SEC;

printf(" BFS complited in %f ns", time\_spent);

start = clock();

printf("\nBFSlst: ");

BFSlst(choice, lst);

end = clock();

time\_spent = (double)(end - start) \* 1000000 / CLOCKS\_PER\_SEC;

printf(" BFSlst complited in %f ns", time\_spent);

start = clock();

printf("\nDFS: ");

DFS(choice);

printf("\nDists: ");

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

dfsvis[i] = dfsvis[i] - 1;

printf("%4i", dfsvis[i]);

}

end = clock();

time\_spent = (double)(end - start) \* 1000000 / CLOCKS\_PER\_SEC;

printf(" DFS complited in %f ns", time\_spent);

start = clock();

printf("\nDFSlst: ");

DFSlst(choice,lst, choice);

printf("\nDists: ");

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

/\* if(!i==choice){

if (dfsvislst[i] == 0) {

dfsvislst[i] = 1;

}

else {

dfsvislst[i] = -dfsvislst[i];

dfsvislst[i] = dfsvislst[i] + 1;

}

}\*/

printf("%4i", dfscntrlst[i]);

}

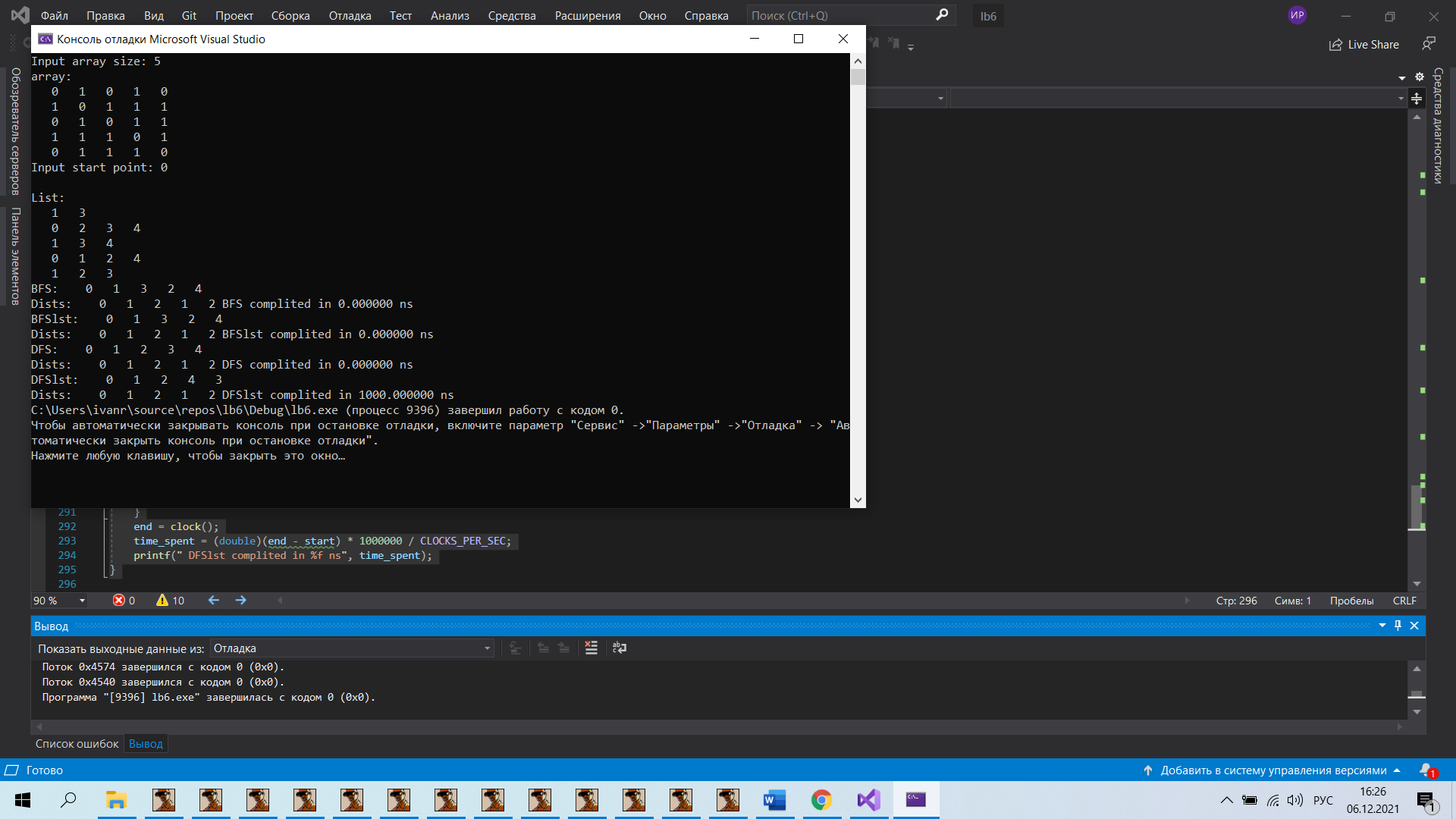
end = clock();

time\_spent = (double)(end - start) \* 1000000 / CLOCKS\_PER\_SEC;

printf(" DFSlst complited in %f ns", time\_spent);

}

**Результаты работы программы:**



**Вывод:** научились искать расстояния в графе методами обхода в ширину и в глубину