Министерство образование Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

Выполнила:

студентка группы 20ВВ1:

Зацепилина Е.С.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза 2021 г.

**Цель работы**: освоить алгоритм обхода графа в ширину в его списочном и матричном представлении.

**Задание на лабораторную работу:**

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. При реализации алгоритма в качестве очереди используйте класс queue из стандартной библиотеки С++.

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

Задание 2\*

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

3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

**Ход выполнения задания:**

Для выполнения задания на лабораторную работу были написана программа 6 на языке С++ в среде Visual Studio 2019. Полный текст программы смотрите в п. «Листинг».

**1.1** Для осуществления генерации матрицы смежности для неориентированного графа применялась динамическая структура данных стандартной библиотеки «vector» - vector. Vector использовался для удобного и безопасного добавления и удаления строк\столбцов в матрицу смежности графа. С использованием этой структуры в памяти резервируется место для «вектора векторов», который имеет такую же структуру и доступ к элементам, как двумерный массив:

m.resize(n);

for (int i = 0; i < m.size(); i++) {

m.at(i).resize(n);

}

Заполнение матрицы происходит случайным образом. В матрицу записываются 0 или 1 по следующим законам:

1) m[i][j] = m[j][i] – матрица симметрична относительно главной диагонали;

2) m[i][j] = 0, если i = j – главная диагональ всегда заполнена нулями;

3) Шанс выпадения 0 или 1 определяется соотношением 30/70 соответственно.

Заполнение матриц смежности для графа:

srand(time(NULL));

for (int i = 0; i < m.size(); i++) {

for (int j = 0; j < m[i].size(); j++) {

if (i == j) {

m[i][j] = 0;

break;

}

else {

int z = rand() % 101;

if (z < 30) { m[i][j] = 0; }

else { m[i][j] = 1; }

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

}

}

}

**1.2**

Функция BFS\_m(arr, v) реализует обход графа в ширину для поиска расстояний в виде матрицы смежности:

void BFS\_m(vector <vector<int>> arr, int v) {

queue <int> q;

q.push(v);

DIST[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

for (int i = 0; i < arr.size(); i++) {

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

q.push(i);

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

}

}

}

}

Функция реализована также, как и в Л.Р. №5, но вместо массива с булевскими значениями используется массив целых чисел, где не посещенная вершина графа помечается как -1, посещенная 0, а расстояние до каждой вершины сохраняется именно в этом массиве путем прибавления к значению 1 по мере надобности.

**1.3\***

Функция BFS\_g(g, v) реализует обход графа в ширину для поиска расстояний в виде списка смежности:

void BFS\_g(vector <\_queue>& g, int v) {

queue <int> q;

q.push(v);

DIST[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

while (!g[v].empty()) {

if (DIST[poisk\_name(g, g[v].first())] == -1) {

q.push(g[v].first() - 1);

DIST[poisk\_name(g, g[v].first())] = DIST[v] + 1;

}

g[v].pop();

}

}

}

Функция реализована также, как и в Л.Р. №5, но вместо массива с булевскими значениями используется массив целых чисел, где не посещенная вершина графа помечается как -1, посещенная 0, а расстояние до каждой вершины сохраняется именно в этом массиве путем прибавления к значению 1 по мере надобности.

**2.1\***

Функция DFS\_m\_p(arr, v) – реализация обхода графа в для поиска расстояний в виде матрицы смежности:

void DFS\_m(vector <vector<int>> arr, int v, int k) {

DIST[v] = k;

k++;

for (int i = 0; i < arr.size(); i++) {

if ((arr[v][i] == 1) && (DIST[i]) > k) {

DFS\_m(arr, i, k);

}

}

}

В отличие от обхода в глубину в Л.Р. №4 в функцию также передается значение k – уровень вершины, которым и определяется дистанция. Также, если в процессе реализации рекурсии k меньше текущего расстояния, значит до вершины найден более короткий путь.

**2.3\***

Скорость работы алгоритмов поиска расстояний в графе можно увидеть в таблице 1.

Таблица 1 – скорость работы алгоритма обхода в ширину.

|  |  |  |  |
| --- | --- | --- | --- |
| **Кол-во вершин** | **BFS\_m** | **BFS\_g** | **DFS\_m** |
| 10 | 0,006 | 0,009 | 0,009 |
| 100 | 0,059 | 0,154 | 1,29 |
| 250 | 0,166 | 1,629 | 15,046 |

Отсюда можно сделать вывод, что при поиске расстояний гораздо лучше себя показывает алгоритм обхода графа в ширину.

**Вывод:** в результате выполнения лабораторной работы были получены навыки реализации поиска расстояний в графе при обходе в глубину и ширину. Также было проведено сравнение эффективности обходов при поиске расстояний. Испытания показали, что обход графа в ширину гораздо эффективнее при поиске расстояний.

**Листинг:**

**6.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <locale.h>

#include <windows.h>

#include <vector>

#include <queue>

using namespace std;

class \_queue {

private:

//Класс элемента очереди

class item {

public:

int val;

item\* next;

item(int a) {

val = a;

next = NULL;

}

};

//Указатель на первый элемент очереди

item\* it\_first;

public:

int name; //Служит только для обозначения имени вершины графа

//Пуста ли очередь

boolean empty() {

if (it\_first == NULL) return true;

else return false;

}

//Конструктор

\_queue() {

it\_first = NULL;

name = NULL;

}

\_queue(int a) {

it\_first = NULL;

name = a;

}

//Удаление первого эл-та очереди

void pop() {

if (!empty()) {

item\* it = it\_first->next;

delete it\_first;

it\_first = it;

}

}

//Добавление в конец очереди

void push(int a) {

if (it\_first == NULL) {

it\_first = new item(a);

}

else {

item\* it1 = it\_first;

while (it1->next != NULL) {

it1 = it1->next;

}

it1->next = new item(a);

}

}

//Отчистка очереди

void clear() {

while (!empty()) {

pop();

}

}

//Возвращает val первого элемента очереди

int first() {

if (empty()) return NULL;

else return it\_first->val;

}

//Вывод очереди

void vivod() {

item\* it = it\_first;

while (it != NULL) {

printf(" %d", it->val);

it = it->next;

}

}

};

vector <int> DIST;

void vivod\_g(vector <\_queue> g) {//Вывод списка смежности графа

for (int i = 0; i < g.size(); i++) {

printf("%d:", g[i].name);

g[i].vivod();

printf("\n");

}

}

void vivod\_m(vector <vector<int>> arr) {//Функция вывода одной матрицы

cout << (" |");

for (int i = 0; i < arr.size(); i++) {

printf("%-2d|", i + 1);

}

cout << ("\n");

for (int i = 0; i < arr.size(); i++) {

printf("|");

printf("%-2d|", i + 1);

for (int j = 0; j < arr[i].size(); j++) {

printf("%-2d|", arr[i][j]);

}

printf("\n");

}

}

void udalenie(vector <vector<int>>& arr, int v) {//Удаление вершины из матрицы

for (int i = 0; i < arr.size(); i++) {

auto iter = arr[i].cbegin();

arr[i].erase(iter + v - 1);

}

auto iter = arr.cbegin();

arr.erase(iter + v - 1);

}

void clear(vector <vector<int>>& arr) {//Удаление всех вершин, у которых отсутвуют рёбра (отчистка матрицы)

int val;

for (int i = 0; i < arr.size(); i++) {

val = 0;

for (int j = 0; j < arr[i].size(); j++) {

if (arr[i][j] != 0) { val++; }

}

if (val == 0) { udalenie(arr, i + 1); }

}

}

int poisk\_name(vector <\_queue> g, int v) {

for (int i = 0; i < g.size(); i++) {

if (g[i].name == v) return i;

}

return -1;

}

void BFS\_m(vector <vector<int>> arr, int v) {

queue <int> q;

q.push(v);

DIST[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

for (int i = 0; i < arr.size(); i++) {

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

q.push(i);

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

}

}

}

}

void BFS\_g(vector <\_queue>& g, int v) {

queue <int> q;

q.push(v);

DIST[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

while (!g[v].empty()) {

if (DIST[poisk\_name(g, g[v].first())] == -1) {

q.push(g[v].first() - 1);

DIST[poisk\_name(g, g[v].first())] = DIST[v] + 1;

}

g[v].pop();

}

}

}

void DFS\_m(vector <vector<int>> arr, int v, int k) {

DIST[v] = k;

k++;

for (int i = 0; i < arr.size(); i++) {

if ((arr[v][i] == 1) && (DIST[i]) > k) {

DFS\_m(arr, i, k);

}

}

}

int main(void)

{

perez:

char\* locale = setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

system("cls");

vector < vector <int> > m;

int n;

printf("Введите кол-во вершин графа: ");

cin >> n;

m.resize(n);

for (int i = 0; i < m.size(); i++) {

m.at(i).resize(n);

}

srand(time(NULL));

for (int i = 0; i < m.size(); i++) {

for (int j = 0; j < m[i].size(); j++) {

if (i == j) {

m[i][j] = 0;

break;

}

else {

int z = rand() % 101;

if (z < 30) { m[i][j] = 0; }

else { m[i][j] = 1; }

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

}

}

}

clear(m);

if (n > m.size()) printf("\nВершины без рёбер удалены.\n");

printf("\nМатрица смежности графа\n\n");

vivod\_m(m);

printf("\nСписок смежности графа\n\n");

vector <\_queue> g;

g.resize(m.size());

for (int i = 0; i < m.size(); i++) {

g[i].name = i + 1;

for (int j = 0; j < m[i].size(); j++) {

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

g[i].push(j + 1);

}

}

}

vivod\_g(g);

int v;

printf("\nВведите кол-во вершину, для которой нужно найти расстояния: ");

cin >> v;

DIST.resize(m.size());

for (int i = 0; i < DIST.size(); i++) {

DIST[i] = -1;

}

////////

printf("\nОбход матрицы в ширину: \n");

clock\_t start = clock();

BFS\_m(m, v - 1);

printf("\n");

for (int i = 0; i < DIST.size(); i++) {

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

}

printf("\n");

for (int i = 0; i < DIST.size(); i++) {

printf("%d| ", DIST[i]);

}

clock\_t end = clock();

printf("\n");

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

printf("Обошли граф за %f секунд.\n", time\_spent);

for (int i = 0; i < DIST.size(); i++) {

DIST[i] = -1;

}

///////

printf("\nОбход списка в ширину: \n");

start = clock();

BFS\_g(g, v - 1);

printf("\n");

for (int i = 0; i < DIST.size(); i++) {

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

}

printf("\n");

for (int i = 0; i < DIST.size(); i++) {

printf("%d| ", DIST[i]);

}

end = clock();

printf("\n");

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

printf("Обошли граф за %f секунд.\n", time\_spent);

for (int i = 0; i < DIST.size(); i++) {

DIST[i] = n;

}

///////

printf("\nОбход матрицы в глубину: \n");

start = clock();

DFS\_m(m, v - 1, 0);

printf("\n");

for (int i = 0; i < DIST.size(); i++) {

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

}

printf("\n");

for (int i = 0; i < DIST.size(); i++) {

printf("%d| ", DIST[i]);

}

end = clock();

printf("\n");

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

printf("Обошли граф за %f секунд.\n", time\_spent);

for (int i = 0; i < DIST.size(); i++) {

DIST[i] = 10;

}

int num;

printf("\nДля перезапуска программы введите 0, для выхода - 1: ");

cin >> num;

switch (num) {

case 0: goto perez;

case 1: return 0;

}

}

**Результаты:**

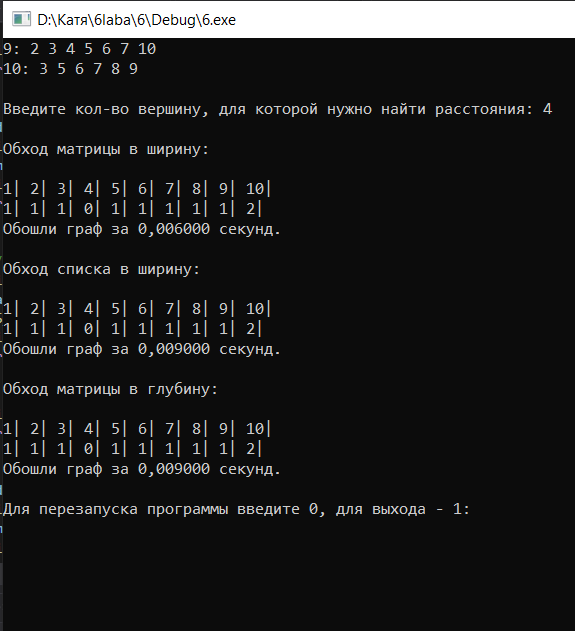


Рисунок 1 – Результаты работы программы