|  |
| --- |
| 0Министерство образования Российской Федерации  Пензенский государственный университет  Кафедра «Вычислительная техника» |
| Отчет  по лабораторной работе №7  по курсу «Логика и основы алгоритмизации в инженерных задачах»  на тему «Поиск расстояний во взвешанном графе» |
|  |
|  |
| Выполнил студент группы 19ВВ3:  Табрисов С.А.  Принял:  Митрохин М. А. |
| Пенза  2020 |

**Цель работы**

Реализовать алгоритм поиска расстояний во взвешенном/невзвешенном и ориентированном/неориентированном графе, используя алгоритм обхода в ширину.

**Листинг программы**

#include <stdio.h>

#include <time.h>

#include <windows.h>

#include <conio.h>

#include <queue>

#include <string.h>

int\* DIST;

int\* DIST1;

int\*\* arr1;

int\*\* arr2;

int\*\* arr3;

int\*\* arr4;

int\*\* dynamic(int count) {

int\*\* mass;

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

for (int i = 0; i < count; i++) mass[i] = (int\*)malloc(count \* sizeof(int));

return mass;

}

void generate\_typeNV\_Nroute(int\*\* mass, int count) { //невзвешенный неориентированный

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

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

if (j > i) {

mass[i][j] = rand() % 2;

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

}

else if (i == j) mass[i][j] = 0;

}

}

}

void generate\_typeV\_Nroute(int\*\* mass, int count) { // взвешенный неориентированный

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

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

if (j > i) {

mass[i][j] = rand() % 7 + 0;

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

}

else if (i == j) mass[i][j] = 0;

}

}

}

void generate\_typeNV\_route(int\*\* mass, int count) { //невзвешенный ориентированный

int a;

int b;

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

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

a = rand() % 101;

if (a < 70) {

mass[j][i] = 1;

mass[i][j] = 0;

}

else mass[j][i] = 0;

if (i == j) mass[j][i] = 0;

}

}

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

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

b = rand() % 101;

if ((b < 50) && (mass[j][i]==1)) {

mass[i][j] = 1;

mass[j][i] = 0;

}

if (i == j) mass[i][j] = 0;

}

}

}

void generate\_typeV\_route(int\*\* mass, int count) { //взвешенный ориентированный

int a;

int b;

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

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

a = rand() % 101;

if (a < 70) {

mass[j][i] = rand() % 6 + 1;

mass[i][j] = 0;

}

else mass[j][i] = 0;

if (i == j) mass[j][i] = 0;

}

}

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

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

b = rand() % 101;

if ((b < 50) && (mass[j][i] >= 1)) {

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

mass[j][i] = 0;

}

if (i == j) mass[i][j] = 0;

}

}

}

void print(int\*\* mass, int count) {

printf(" ");

for (int k = 1; k <= count; k++) printf("%d ", k);

printf("\n -----------\n");

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

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

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

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

}

printf("\n");

}

}

int\* dynamic\_dist(int count) {

int\* dist;

dist = (int\*)malloc(count \* sizeof(int));

return dist;

}

void generate\_dist\_v(int\* DISTV, int count) {

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

DISTV[i] = -1;

}

}

void generate\_dist\_nv(int\* DISTNV, int count) {

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

DISTNV[i] = -1;

}

}

void BFSD\_V(int v, int a, int\*\* arr) {

std::queue <int> qu;

qu.push(v);

DIST[v] = 0;

while (!qu.empty()) {

v = qu.front();

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

qu.pop();

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

if ((arr[v][i] > 0) && (DIST[i] == -1)) {

qu.push(i);

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

}

}

}

}

void BFSD\_NV(int v, int a, int\*\* arr) {

std::queue <int> qu1;

qu1.push(v);

DIST1[v] = 0;

while (!qu1.empty()) {

v = qu1.front();

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

qu1.pop();

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

if ((arr[v][i] == 1) && (DIST1[i] == -1)) {

qu1.push(i);

DIST1[i] = DIST1[v] + 1;

}

}

}

}

int main(int argc, char\* argv[])

{

int begin;

int n=0;

int min = 1000;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

if (argc < 2) {

printf("Введите параметры в командной строке и попробуйте снова.\n");

printf("Варианты параметров:\n");

printf("Взвешенный граф - typev.\n");

printf("Невзвешенный граф - typenv.\n");

printf("Ориентированный граф - route.\n");

printf("Неориентированный граф - nroute.\n");

return 1;

}

if ((argc >= 3) && (strcmp(argv[1], "typev")==0) && (strcmp(argv[2], "route")==0)) {

if (argc > 4) {

printf("Слишком много параметров(Больше 4). Попробуйте снова!\n");

return 1;

}

if (argv[3] != NULL) {

n = atoi(argv[3]);

}

else if (n == 0) {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6;

}

else {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6;

}

arr1 = dynamic(n);

DIST = dynamic\_dist(n);

generate\_dist\_v(DIST, n);

generate\_typeV\_route(arr1, n);

printf("Матрица смежности взвешенного ориентированного графа:\n\n");

print(arr1, n);

printf("Введите вершину с которой хотите начать обход - ");

scanf\_s("%d", &begin);

BFSD\_V(begin - 1, n, arr1);

printf("\n--------------------------------------------\n");

printf("| Вершины |");

for (int i = 1; i <= n; i++) {

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

}

printf("\n--------------------------------------------\n");

printf("| Расстояние |");

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

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

}

printf("\n--------------------------------------------\n");

free(arr1);

free(DIST);

system("pause");

return 0;

}

if ((argc >= 3) && (strcmp(argv[1], "typev") == 0) && (strcmp(argv[2], "nroute") == 0)) {

if (argc > 4) {

printf("Слишком много параметров(Больше 4). Попробуйте снова!\n");

return 1;

}

if (argv[3] != NULL) {

n = atoi(argv[3]);

}

else if (n == 0) {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6;

}

else {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6; }

arr2 = dynamic(n);

generate\_typeV\_Nroute(arr2, n);

DIST = dynamic\_dist(n);

generate\_dist\_v(DIST, n);

printf("Матрица смежности взвешенного неориентированного графа:\n\n");

print(arr2, n);

printf("Введите вершину с которой хотите начать обход - ");

scanf\_s("%d", &begin);

BFSD\_V(begin - 1, n, arr2);

printf("\n--------------------------------------------\n");

printf("| Вершины |");

for (int i = 1; i <= n; i++) {

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

}

printf("\n--------------------------------------------\n");

printf("| Расстояние |");

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

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

}

printf("\n--------------------------------------------\n");

free(arr2);

free(DIST);

system("pause");

return 0;

}

if ((argc >= 3) && (strcmp(argv[1], "typenv") == 0) && (strcmp(argv[2], "route") == 0)) {

if (argc > 4) {

printf("Слишком много параметров(Больше 4). Попробуйте снова!\n");

return 1;

}

if (argv[3] != NULL) {

n = atoi(argv[3]);

}

else if (n == 0) {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6;

}

else {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6;

}

arr3 = dynamic(n);

generate\_typeNV\_route(arr3, n);

DIST1 = dynamic\_dist(n);

generate\_dist\_nv(DIST1, n);

printf("Матрица смежности невзвешенного ориентированного графа:\n\n");

print(arr3, n);

printf("Введите вершину с которой хотите начать обход - ");

scanf\_s("%d", &begin);

BFSD\_NV(begin - 1, n, arr3);

printf("\n--------------------------------------------\n");

printf("| Вершины |");

for (int i = 1; i <= n; i++) {

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

}

printf("\n--------------------------------------------\n");

printf("| Расстояние |");

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

printf("| %d |", DIST1[i]);

}

printf("\n--------------------------------------------\n");

free(arr3);

free(DIST1);

system("pause");

return 0;

}

if ((argc >= 3) && (strcmp(argv[1], "typenv")==0) && (strcmp(argv[2], "nroute")==0)) {

if (argc > 4) {

printf("Слишком много параметров(Больше 4). Попробуйте снова!\n");

return 1;

}

if (argv[3] != NULL) {

n = atoi(argv[3]);

}

else if (n == 0) {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6;

}

else {

printf("Число не обнаружено, количество вершин равно 6!\n");

n = 6;

}

arr4 = dynamic(n);

generate\_typeNV\_Nroute(arr4, n);

DIST1 = dynamic\_dist(n);

generate\_dist\_nv(DIST1, n);

printf("Матрица смежности невзвешенного неориентированного графа:\n\n");

print(arr4, n);

printf("Введите вершину с которой хотите начать обход - ");

scanf\_s("%d", &begin);

BFSD\_NV(begin - 1, n, arr4);

printf("\n--------------------------------------------\n");

printf("| Вершины |");

for (int i = 1; i <= n; i++) {

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

}

printf("\n--------------------------------------------\n");

printf("| Расстояние |");

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

printf("| %d |", DIST1[i]);

}

printf("\n--------------------------------------------\n");

free(arr4);

free(DIST1);

system("pause");

return 0;

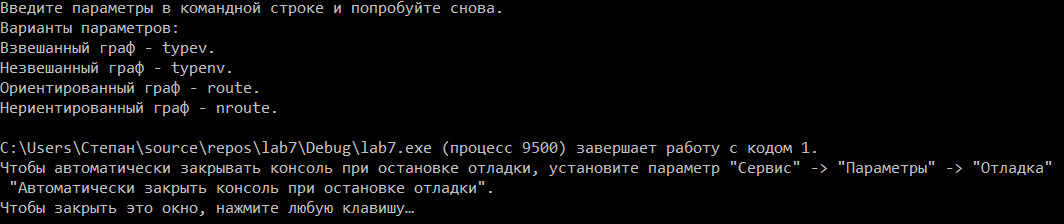
}

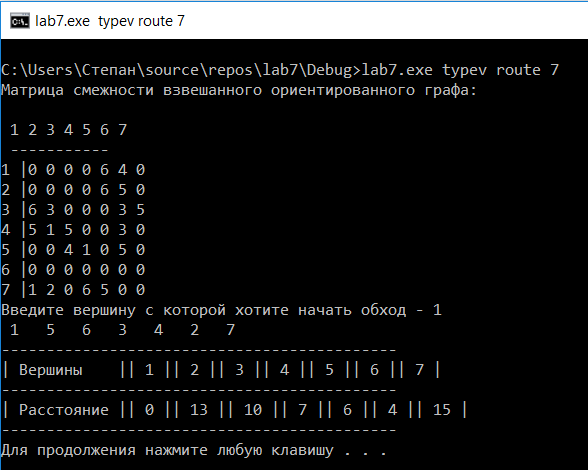
system("pause");

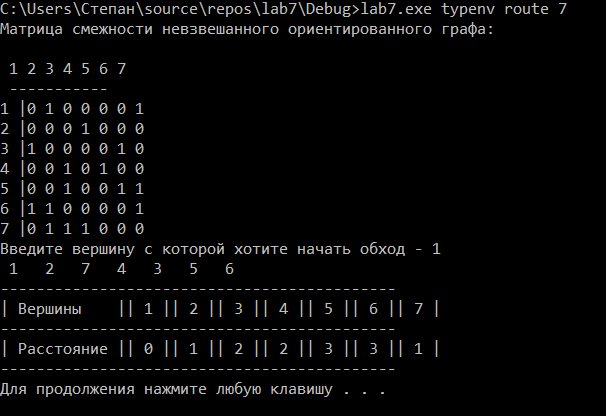
return 0;

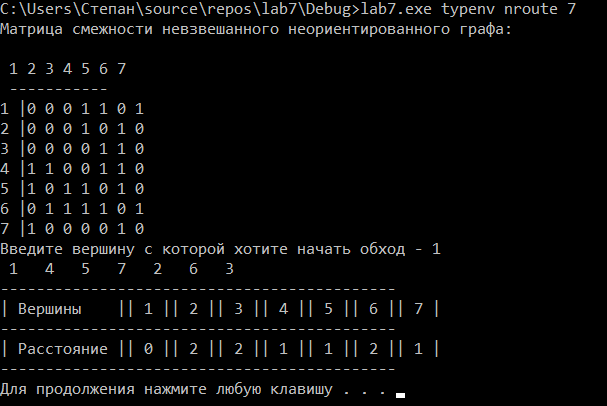
}

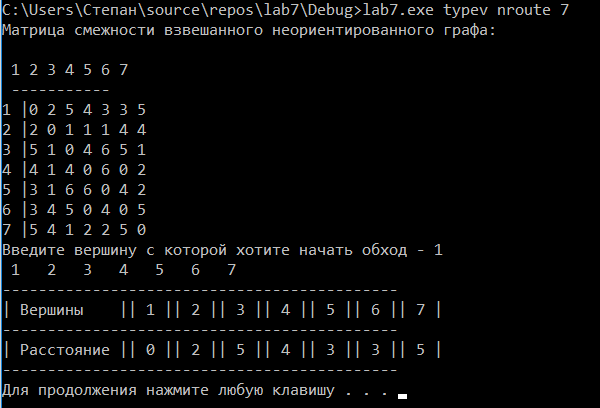
### **Результат работы программы:**



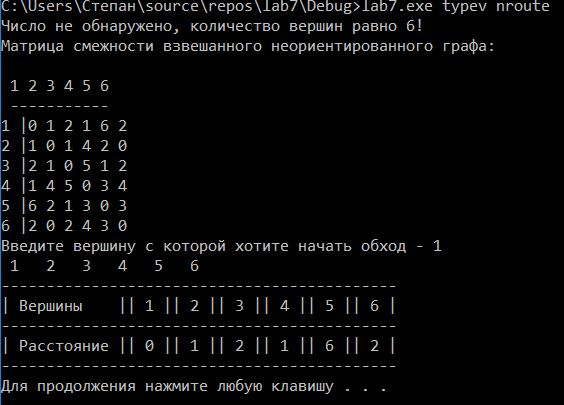












**Вывод:** Реализовал алгоритм поиска расстояний во взвешенном/невзвешенном и ориентированном/неориентированном графе, используя алгоритм обхода в ширину.