Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

по курсу «Л и ОА в ИЗ»

на тему «Поиск расстояний во взвешенном графе»

**Выполнили:**

студентом группы 19ВВ3

Земляков В.Д.

**Принял:**

Митрохин М.А.

Пенза2020

**Цель работы:**

Изучить поиск расстояний во взвешенном графе.

**Ход работы:**

Во взвешенном графе в отличие от не взвешенного каждое ребро имеет

вес, отличный от нуля. Поэтому в матрице смежности взвешенного графа

содержится информация не только о наличии ребра, но и о его весе.Поиск расстояний между вершинами в таком графе также возможно

построить используя процедуры обхода графа. Отличие от поиска расстояний

в не взвешенном графе будет состоять в том, что при обновлении расстояния

до вершины при ее посещении оно будет увеличиваться не на 1, а на

величину веса ребра.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include<random>

#include <time.h>

#include <queue>

using namespace std;

void output(int\*\* Array, int N) {

printf("\n");

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

{

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

{

printf("%d\t", Array[i][j]);

}

printf("\n");

}

printf("\n");

}

void GenWeighed(int\*\* Array, int N) {

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

{

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

{

int z = rand() % 100;

if (z > 80) {

Array[i][j] = rand() % N;

}

else {

Array[i][j] = 0;

}

Array[j][i] = Array[i][j];

if (i == j) {

Array[i][j] = 0;

}

}

}

output(Array, N);

}

void GenWeighedOriented(int\*\* Array, int N) {

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

{

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

{

int z = rand() % 100;

if (z > 80) {

Array[i][j] = rand() % N;

}

else {

Array[i][j] = 0;

}

if (i == j) {

Array[i][j] = 0;

}

}

}

output(Array, 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 GenOriented(int\*\* Array, int N) {

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

{

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

{

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

if (i == j) {

Array[i][j] = 0;

}

}

}

output(Array, N);

}

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

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

Dist[i] = 1000;

}

printf("\n");

}

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

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

Dist[i] = -1;

}

printf("\n");

}

void DBFSWeighed(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] > 0 && DIST[i] > DIST[v] + Array[v][i]) {

Q.push(i);

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

}

}

}

}

void DBFS(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;

}

}

}

}

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

{

srand(time(NULL));

int N = 0;

int\*\* ArrayM1 = NULL;

int a = 0;

int index = 0;

int\* DIST = NULL;

if (argc > 1) {

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

if (strcmp(argv[i], "-Z") == 0) {

N = atoi(argv[i + 1]);

index = 1;

}

}

if (index == 0) {

printf("Enter size");

scanf("%d", &N);

}

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

//ориентированный не взвешенный направленный//

if (strcmp(argv[i], "-O") == 0) {

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

ClearDistance(DIST, N);

ArrayM1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

ArrayM1[i] = (int\*)malloc(N \* sizeof(int));

}

GenOriented(ArrayM1, N);

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

DBFS(ArrayM1, i, DIST, N);

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

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

}

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

DIST[j] = -1;

}

printf("\n");

}

}

//ориентированный не взвешенный не направленный//

if (strcmp(argv[i], "-N") == 0) {

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

ClearDistance(DIST, N);

ArrayM1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

ArrayM1[i] = (int\*)malloc(N \* sizeof(int));

}

Gen(ArrayM1, N);

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

DBFS(ArrayM1, i, DIST, N);

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

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

}

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

DIST[j] = -1;

}

printf("\n");

}

}

//ориентированный взвешенный направленный//

if (strcmp(argv[i], "-OW") == 0) {

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

ArrayM1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

ArrayM1[i] = (int\*)malloc(N \* sizeof(int));

}

printf("Weighted orient:\n");

GenWeighedOriented(ArrayM1, N);

ClearDistanceWeighed(DIST, N);

printf("Distance\n");

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

DBFSWeighed(ArrayM1, i, DIST, N);

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

if (DIST[j] == 1000) {

printf("-\t");

}

else

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

}

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

DIST[j] = 1000;

}

printf("\n");

}

}

//ориентированный взвешенный не направленный//

if (strcmp(argv[i], "-NW") == 0) {

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

ArrayM1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

ArrayM1[i] = (int\*)malloc(N \* sizeof(int));

}

printf("Weightd\n");

GenWeighed(ArrayM1, N);

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

ClearDistanceWeighed(DIST, N);

printf("Distance\n");

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

DBFSWeighed(ArrayM1, i, DIST, N);

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

if (DIST[j] == 1000) {

printf("-\t");

}

else

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

}

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

DIST[j] = 1000;

}

printf("\n");

}

}

}

}

else

{

printf("Enter arguments.");

}

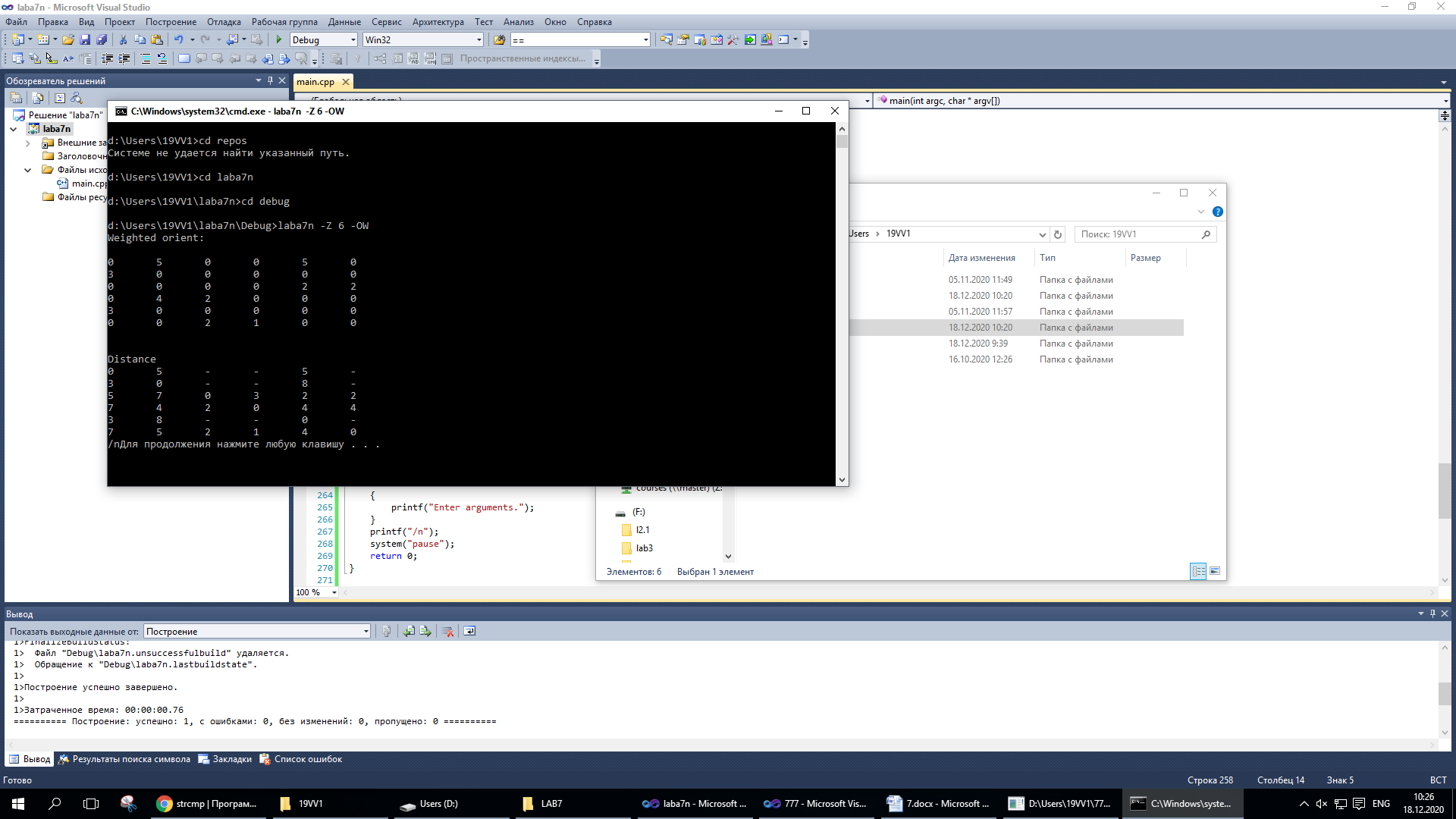
printf("/n");

system("pause");

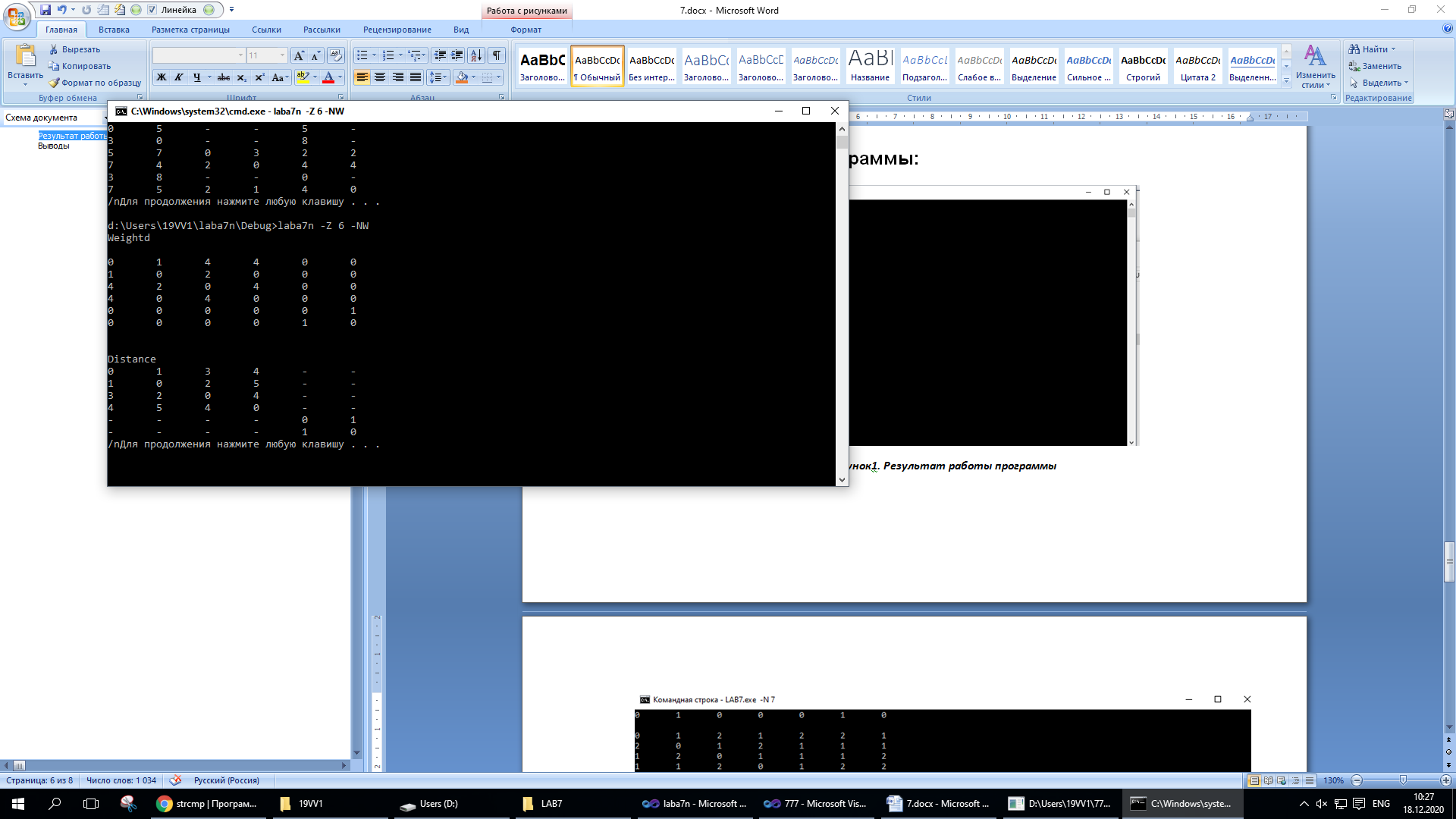
return 0;

}

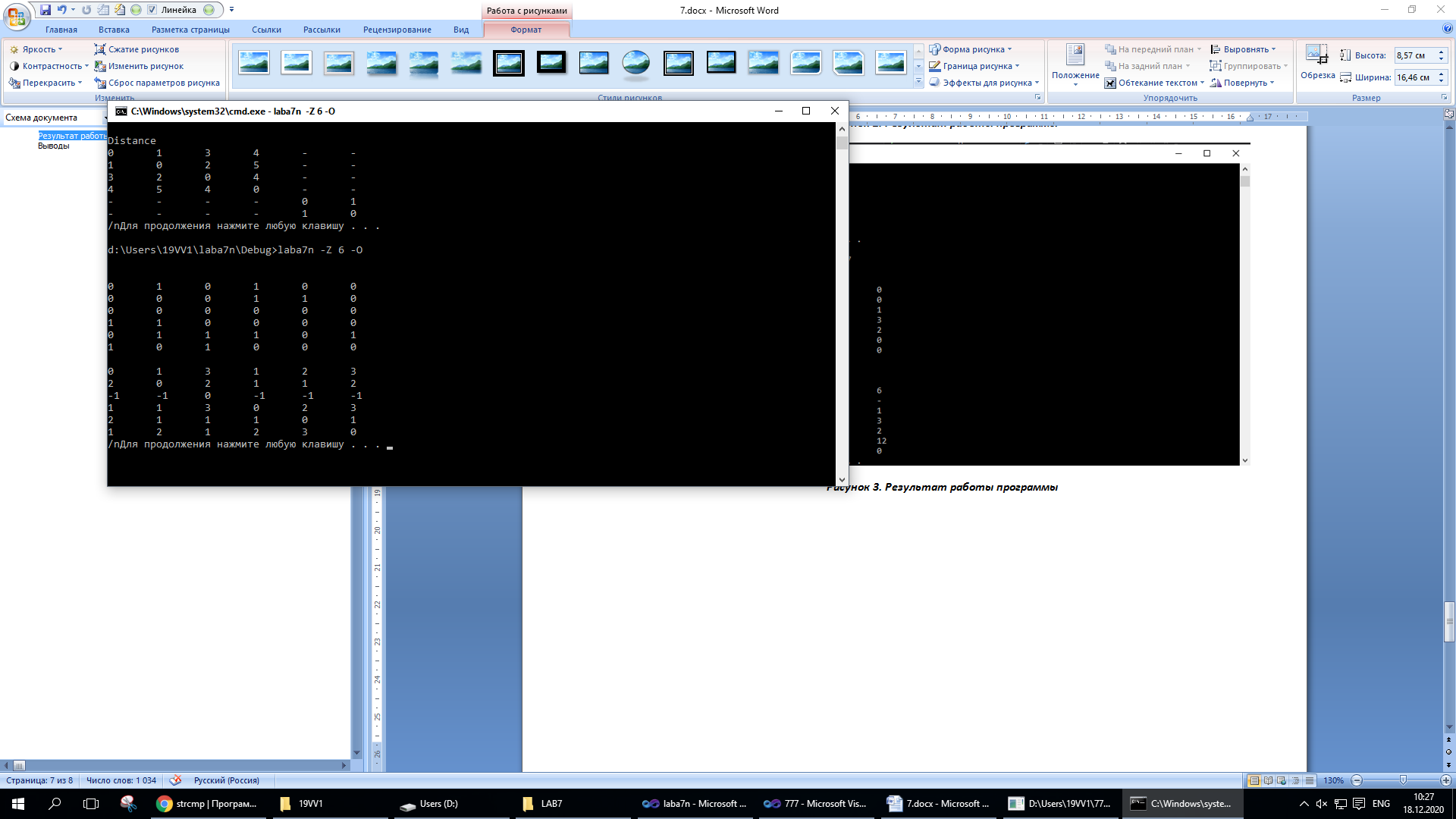
### Результат работы программы:



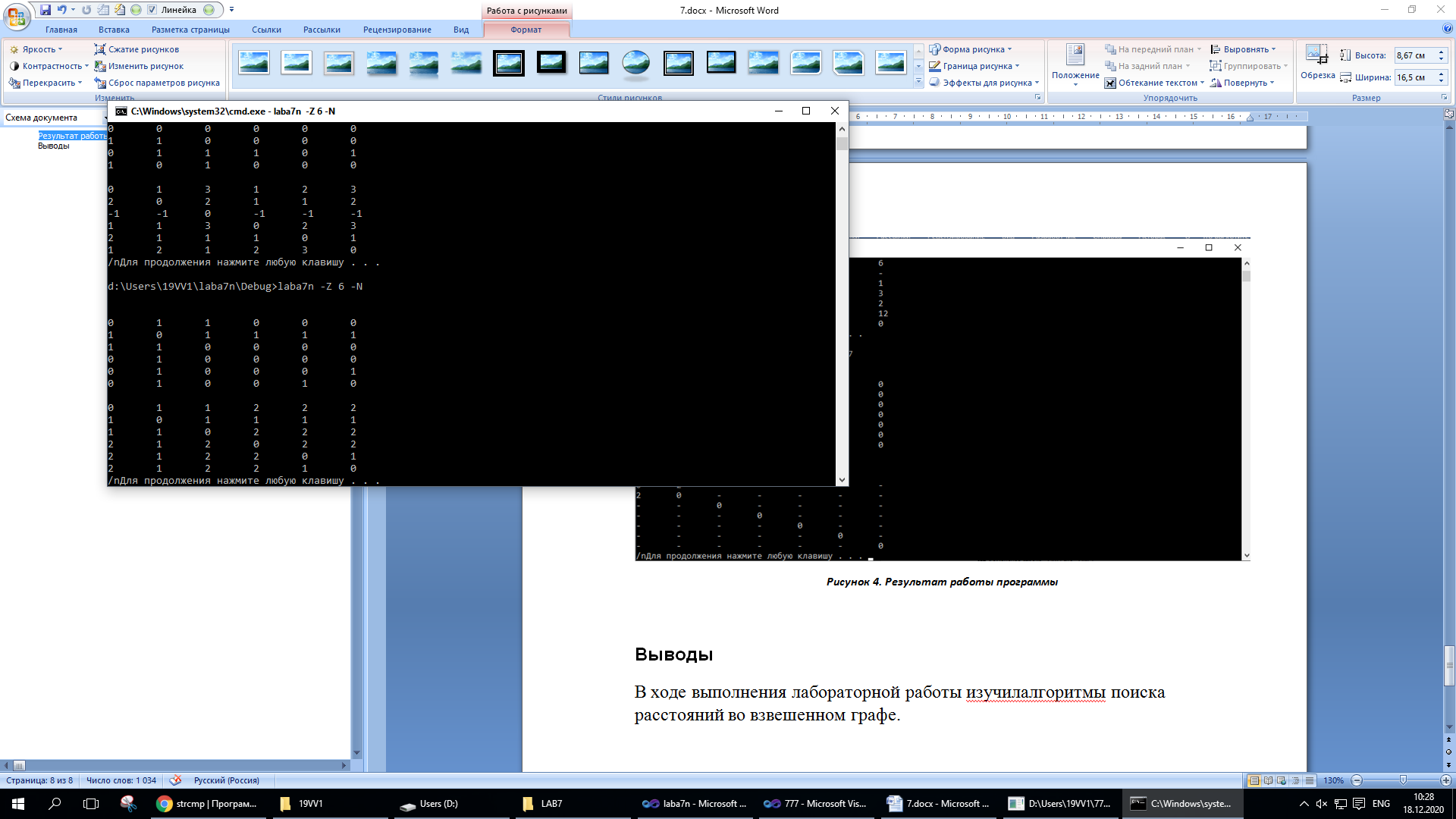
**Рисунок1. Результат работы программы**



**Рисунок 2. Результат работы программы**



**Рисунок 3. Результат работы программы**



**Рисунок 4. Результат работы программы**

### Выводы

В ходе выполнения лабораторной работы изучилалгоритмы поиска расстояний во взвешенном графе.