Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

# по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»

на тему: «Поиск расстояний в графе»

Выполнили:

студенты группы 20ВВ3

Горожанин Я. А.

Тихонов А. А.

Скирдова В. М.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2021

**Название**

Поиск расстояний в графе

**Цель работы**

Разработка алгоритма поиска расстояний в графе

**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. 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.
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.

**Листинг**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

#include<locale.h>

#include<iostream>

#include<stack>

#include<queue>

usingnamespace std;

structnode {

int numb;

node\* next;

};

structgraf {

node\*\* nodes;

int size;

};

voidJohny\_Depp(int\*\* a, int\* dist, intcurrent\_vertex, intsize, intpath\_length) {

dist[current\_vertex] = path\_length;

for (inti = 0; i<size; i++) {

if (a[current\_vertex][i] != 0 &&dist[i] >path\_length + 1) {

Johny\_Depp(a, dist, i, size, path\_length + 1);

}

}

}

voidCraigslist(graf\* grafon, int\* dist, intcurrent\_vertex, intsize, intpath\_length) {

dist[current\_vertex] = path\_length;

node\* temp = grafon->nodes[current\_vertex];

while (temp != NULL) {

if (dist[temp->numb] >path\_length + 1) {

Craigslist(grafon, dist, temp->numb, size, path\_length + 1);

}

temp = temp->next;

}

}

voidGoogle\_search(int\*\* a, intnum, int\* dist, intsize) {

queue<int> q;

dist[num] = 0;

q.push(num);

while(!q.empty()) {

num = q.front();

q.pop();

for (inti = 0; i<size; i++) {

if (a[num][i] == 1 &&dist[i]==-1) {

q.push(i);

dist[i] = dist[num] + 1;

}

}

}

}

voidGoogle\_search\_spis(graf\* grafon, intnum, int\* dist, intsize) {

queue<int> q;

node\* buff;

dist[num] = 0;

q.push(num);

while(!q.empty()) {

num = q.front();

buff = grafon->nodes[num];

q.pop();

while(buff != NULL ) {

if (dist[buff->numb] == -1) {

q.push(buff->numb);

dist[buff->numb] = dist[num] + 1;

}

buff = buff->next;

}

}

}

graf\* sozdat(intversh) {

graf\* grafon = newgraf;

grafon->size = versh;

grafon->nodes = newnode \* [versh];

for (inti = 0; i<versh; i++) {

grafon->nodes[i] = NULL;

}

returngrafon;

}

node\* sozdatnode(intindex) {

node\* newnode = newnode;

newnode->numb = index;

newnode->next = NULL;

returnnewnode;

}

voidaddgran(graf\* grafon, intfrom, intto) {

node\* newnode = sozdatnode(from);

if (grafon->nodes[to] == 0) {

grafon->nodes[to] = newnode;

newnode = NULL;

}

node\* buf = grafon->nodes[to];

while (buf->next != NULL) {

buf = buf->next;

}

buf->next = newnode;

newnode = sozdatnode(to);

if (grafon->nodes[from] == 0) {

grafon->nodes[from] = newnode;

return;

}

buf = grafon->nodes[from];

while (buf->next != NULL) {

buf = buf->next;

}

buf->next = newnode;

}

voidmain() {

srand(time(0));

setlocale(LC\_ALL, "Russian");

intsize;

cout<<" Введите размерность матрицы: ";

cin>> size;

cout<<endl;

int\*\* arr = newint\* [size];

for (int count = 0; count < size; ++count)

arr[count] = newint[size];

for (int row = 0; row < size; row++) {

arr[row][row] = 0;

for (int col = row + 1; col < size; col++) {

arr[row][col] = rand() % 2;

if (arr[row][col] != 1)

arr[row][col] = 0;

arr[col][row] = arr[row][col];

}

}

cout<<" ";

for (int row = 0; row < size; row++)

{

for (int col = 0; col < size; col++) {

cout<<arr[row][col] <<" ";

}

cout<<endl<<" ";

}

cout<<endl;

graf\* grafon = sozdat(size);

int j = 1;

for (inti = 0; i< size; i++) {

for (j; j < size; j++) {

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

addgran(grafon, i, j);

}

}

j = j - size + i + 1;

}

cout<<" ";

for (inti = 0; i< size; i++) {

node\* temp = grafon->nodes[i];

cout<<i<<" ";

while (temp) {

cout<<" -> "<< temp->numb;

temp = temp->next;

}

cout<<endl<<" ";

}

autodistG = newint[size];

for (inti = 0; i< size; i++)

distG[i] = -1;

cout<<endl<<" Поиск кратчайшего пути в графе, представленным через матрицу смежности, реализованный на основе алгоритма поиска в ширину с использованием очереди"<<endl<<endl;

cout<<" Введите номер вершины, с которой хотите начать обход: ";

int start;

cin>> start;

clock\_tstart\_t, end;

start\_t = clock();

Google\_search(arr, start, distG, size);

end = clock();

cout<<endl;

for (inti = 0; i< size; i++)

if (distG[i] == -1)

cout<<" Кратчайшийпутьдовершины "<<i<<" = "<< 0 <<endl;

else

cout<<" Кратчайшийпутьдовершины "<<i<<" = "<<distG[i] <<endl;

cout<<endl<<" Времявыполненияалгоритмасоставляет "<< (double)difftime(end, start\_t) / CLOCKS\_PER\_SEC<<endl<<endl;

//////////////////////////////////////////////////////////////////

cout<<" Поиск кратчайшего пути в графе, представленным через список смежности, реализованный на основе алгоритма поиска в ширину с использованием очереди"<<endl<<endl;

for (inti = 0; i< size; i++)

distG[i] = -1;

Google\_search\_spis(grafon, start, distG, size);

for (inti = 0; i< size; i++)

if(distG[i]== -1)

cout<<" Кратчайшийпутьдовершины "<<i<<" = "<< 0 <<endl;

else

cout<<" Кратчайшийпутьдовершины "<<i<<" = "<<distG[i] <<endl;

/////////////////////////////////////////////////////////////

cout<<endl<<" Поиск кратчайшего пути, реализованный на основе алгоритма поиска в глубину с использованием рекурсии"<<endl;

int\* dist = newint[size];

for (int count = 0; count < size; ++count)

dist[count] = INT\_MAX;

start\_t = clock();

Johny\_Depp(arr, dist, start, size, 0);

end = clock();

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

if (dist[count] == INT\_MAX)

dist[count] = 0;

cout<<endl<<" Кратчайшийпутьдовершины "<< count <<" = "<<dist[count];

}

cout<<endl<<endl<<" Времявыполненияалгоритмасоставляет "<< (double)difftime(end, start\_t) / CLOCKS\_PER\_SEC<<endl;

/////////////////////////////////////////////////////////////

int\* versh = newint[size];

for (int count = 0; count < size; ++count)

versh[count] = 0;

cout<<endl<<" Поиск кратчайшего пути в графе, представленным через список, реализованный на основе алгоритма поиска в глубину с использованием рекурсии"<<endl;

for (int count = 0; count < size; ++count)

dist[count] = INT\_MAX;

Craigslist(grafon, dist, start, size, 0);

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

if (dist[count] == INT\_MAX)

dist[count] = 0;

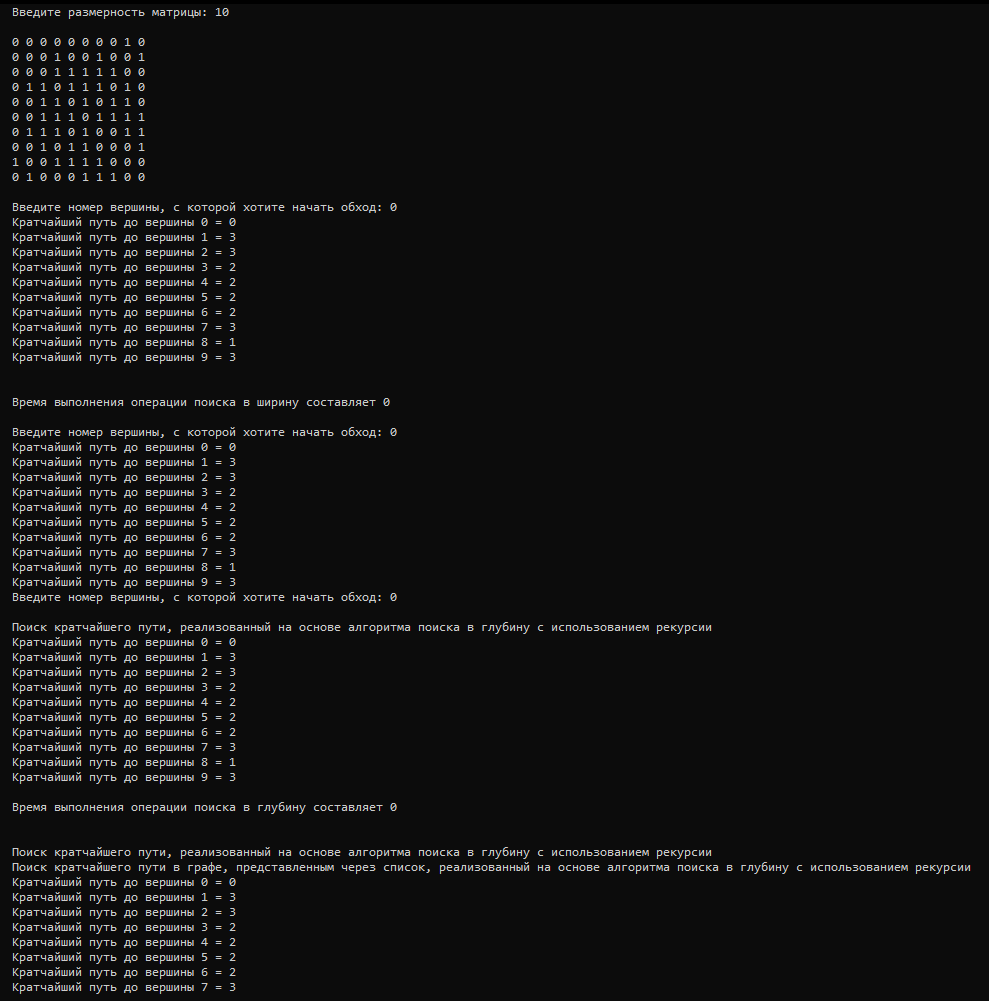
cout<<endl<<" Кратчайшийпутьдовершины "<< count <<" = "<<dist[count];

}

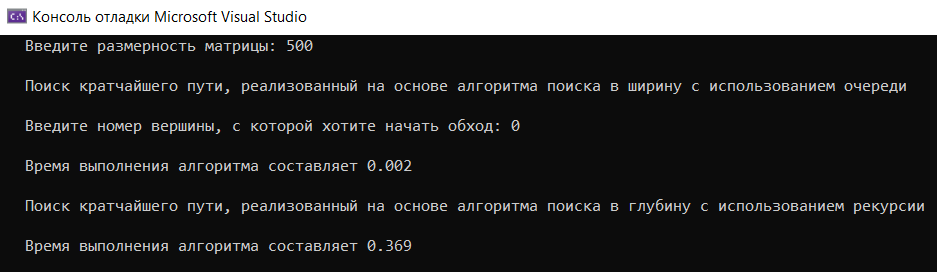
}

**Результат работы программы**

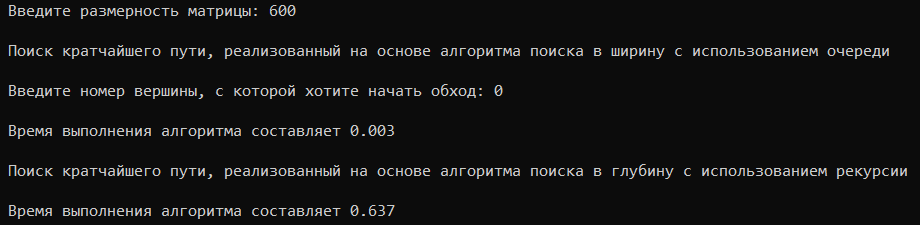
1. Матрица 10х10



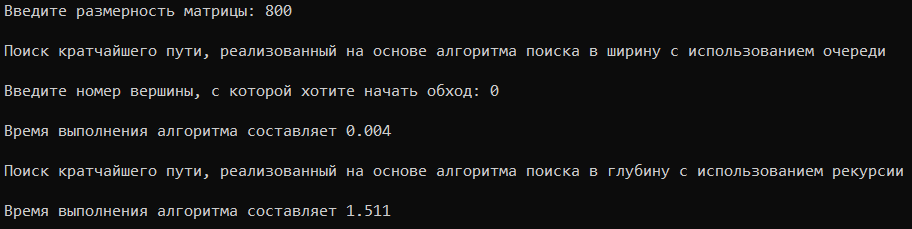
1. Матрица 500х500



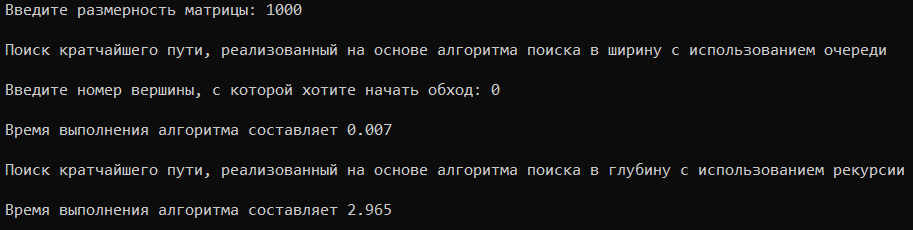
1. Матрица 600х600



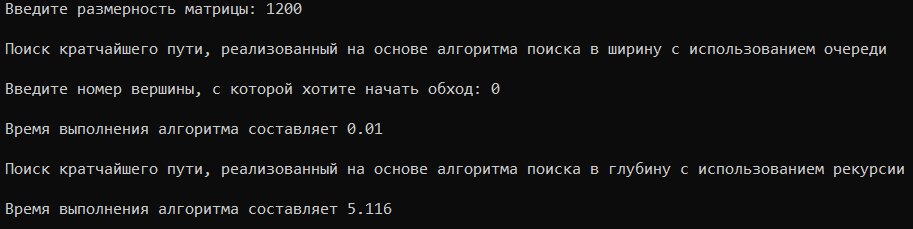
1. Матрица 800х800



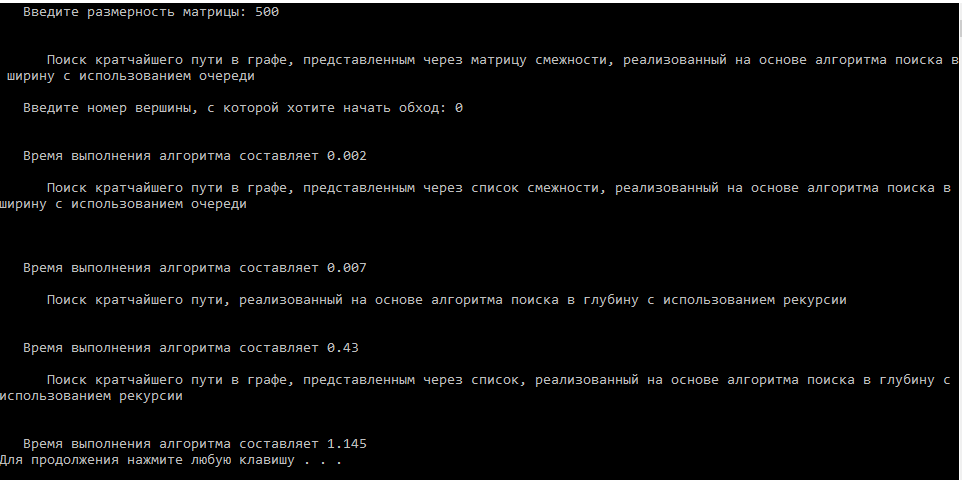
1. Матрица 1000х1000



1. Матрица 1200х1200



1. Сравнение всех алгоритмов

****

**Вывод**

В ходе выполнения данной лабораторной работы были получены навыки разработки алгоритма поиска кратчайшего пути, реализованного на основе алгоритма поиска в ширину с использованием очереди и в глубину с использованием рекурсии, в том числе через списки. В результате тестирования установили, что поиск в ширину через матрицу смежности с использованием очереди работает в быстрее, чем поиск в глубину с использованием рекурсии. А так же было установлено, что эти же алгоритмы поиска пути, но реализованные с помощью списков смежности в два с половиной раза быстрее, чем идентичные алгоритмы, основанные на матрицах смежности.