Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

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

на тему: «Обход графа в ширину»

Выполнили:

студенты группы 20ВВ3

Горожанин Я. А.

Тихонов А. А.

Скирдова В. М.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2021

**Название**

Обход графа в ширину

**Цель работы**

Разработка алгоритма обхода в ширину

**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 обхода в ширину, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.

**Задание 2\***

1. Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.
2. Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Листинг**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

#include <iostream>

#include <queue>

using namespace std;

struct node {

int numb;

node\* next;

};

struct graf {

node\*\* nodes;

int size;

};

struct que\_node {

int numb;

que\_node\* next;

};

struct quqech {

que\_node\* front;

que\_node\* back;

int size;

};

graf\* sozdat(int versh) {

graf\* grafon = new graf;

grafon->size = versh;

grafon->nodes = new node \* [versh];

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

grafon->nodes[i] = NULL;

}

return grafon;

}

node\* sozdatnode(int index) {

node\* newnode = new node;

newnode->numb = index;

newnode->next = NULL;

return newnode;

}

quqech\* create\_que() {

quqech\* que = new quqech;

que->size = NULL;

que->front = NULL;

que->back = NULL;

return que;

}

void push(quqech\* que, int index) {

que\_node\* newnode = new que\_node;

newnode->numb = index;

newnode->next = NULL;

if (que->back != NULL) {

que->back->next = newnode;

que->back = newnode;

}

else

if (que->front == NULL)

que->front = newnode;

else {

que->back = newnode;

que->front->next = newnode;

}

que->size++;

}

void pop(quqech\* que) {

if (que->size){

que\_node\* temp = que->front;

que->front = que->front->next;

que->size--;

delete(temp);

}

}

void Bfg9000\_matrix(int\*\* a, int num, int\* versh, int size) {

queue <int> que;

versh[num] = 1;

que.push(num);

while (!que.empty()) {

num = que.front();

cout << num << " -> ";

que.pop();

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

if (a[num][i] == 1 && versh[i] != 1) {

que.push(i);

versh[i] = 1;

}

}

}

}

void Bfg9000\_spisok(graf\* grafon, int size, int\* versh, int num ){

queue <int> q;

versh[num] = 1;

q.push(num);

node\* buf;

while(!q.empty()) {

num = q.front();

buf = grafon->nodes[num];

q.pop();

cout << num << " -> ";

while (buf != NULL) {

if (versh[buf->numb] == 0) {

q.push(buf->numb);

versh[buf->numb] = 1;

}

buf = buf->next;

}

}

}

void fuck\_you\_leatherman(int\*\* a, int num, int\* versh, int size) {

quqech\* que = create\_que();

versh[num] = 1;

push(que, num);

while (que->size) {

num = que->front->numb;

cout << num << " -> ";

pop(que);

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

if (a[num][i] == 1 && versh[i] != 1) {

push(que, i);

versh[i] = 1;

}

}

}

}

void leather\_club\_is\_two\_blocks\_down(int\*\* a, int num, int\* versh, int size) {

int\* gym\_lockers;

int last\_boy\_next\_door = 0;

gym\_lockers = (int\*)malloc(sizeof(int) \* size);

versh[num] = 1;

gym\_lockers[0] = num;

while (last\_boy\_next\_door+1) {

num = gym\_lockers[0];

cout << num << " -> ";

gym\_lockers = &gym\_lockers[1];

last\_boy\_next\_door--;

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

if (a[num][i] == 1 && versh[i] != 1) {

last\_boy\_next\_door++;

gym\_lockers[last\_boy\_next\_door] = i;

versh[i] = 1;

}

}

}

}

void addgran(graf\* grafon, int from, int to) {

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;

}

void main(){

srand(time(0));

setlocale (LC\_ALL, "Russian");

int size;

cout << " Введите размерность матрицы: "<< " ";

cin >> size;

cout << endl;

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

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

arr[count] = new int[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;

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;

auto versh = new int[size];

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

versh[i] = 0;

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

int num;

cin >> num;

cout << endl << " ";

clock\_t start, end;

start = clock();

Bfg9000\_matrix(arr, num, versh, size);

end = clock();

cout << endl << endl << " " << "Время выполнения с очередью queue " << (double)difftime(end, start) / CLOCKS\_PER\_SEC << endl << endl << " ";

system("pause");

cout << endl << endl << "-----------------------------------------------------------" << endl;

cout << " Обход в глубину списков смежности" << endl;

cout << " " << endl;

graf\* grafon = sozdat(size);

int j = 1;

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

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

cout << i;

while (temp) {

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

temp = temp->next;

}

cout << endl << " ";

}

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

versh[i] = 0;

cout << endl << " ";

Bfg9000\_spisok(grafon, size, versh, num);

cout << endl << endl << " ";

system("pause");

cout << endl << endl << "-----------------------------------------------------------" << endl;

cout << " Обход в глубину c очередью составленной списками" << endl;

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

versh[i] = 0;

cout << endl << " ";

start = clock();

fuck\_you\_leatherman(arr, num, versh, size);

end = clock();

cout << endl << endl << " Время выполнения с очередью списком " << (double)difftime(end, start) / CLOCKS\_PER\_SEC << endl << endl << " ";

system("pause");

cout << endl << endl << "-----------------------------------------------------------" << endl;

cout << " Обход в глубину c очередью составленной массивом" << endl;

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

versh[i] = 0;

cout << endl << " ";

start = clock();

leather\_club\_is\_two\_blocks\_down(arr, num, versh, size);

end = clock();

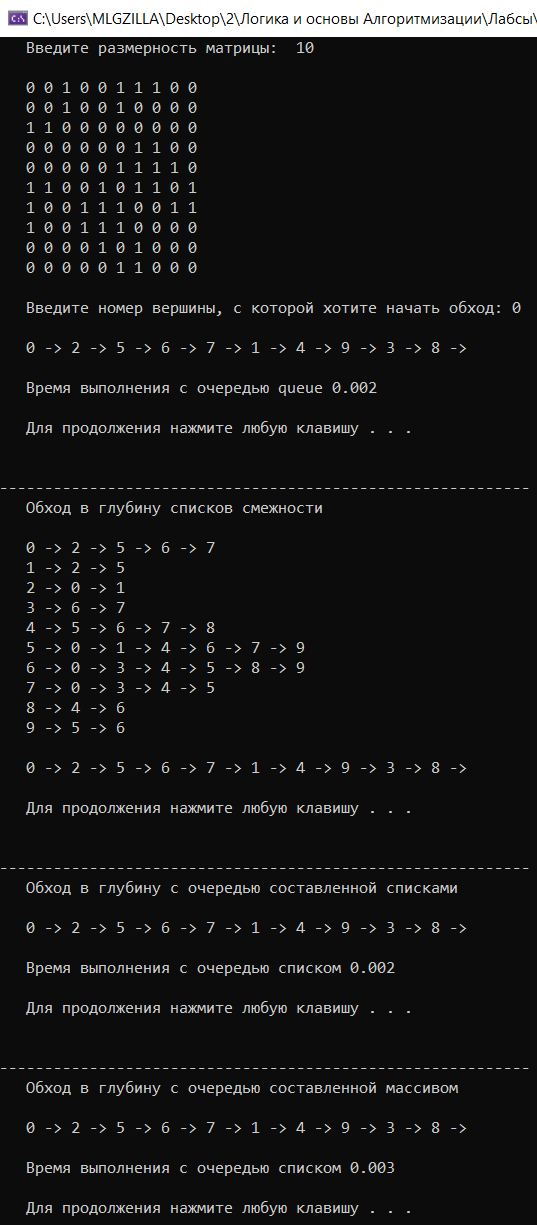
cout << endl << endl << " Время выполнения с очередью списком " << (double)difftime(end, start) / CLOCKS\_PER\_SEC << endl << endl << " ";

system("pause");

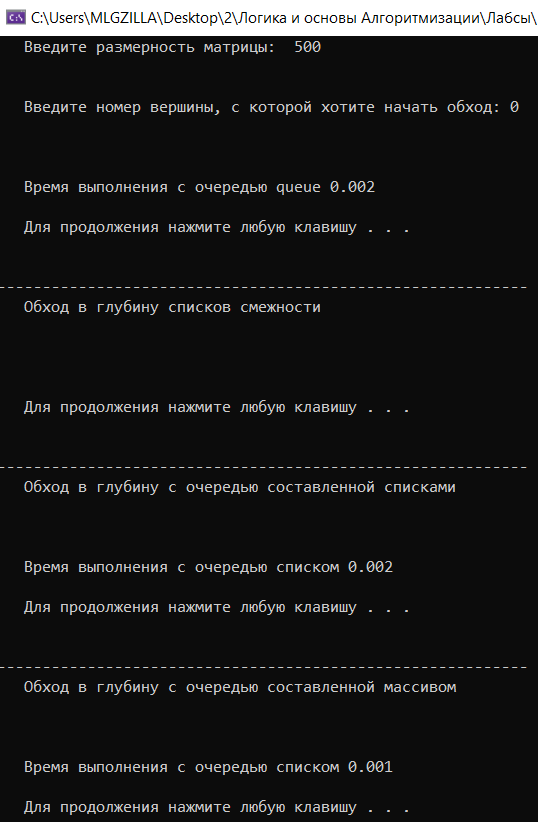
}

**Результат работы программы**

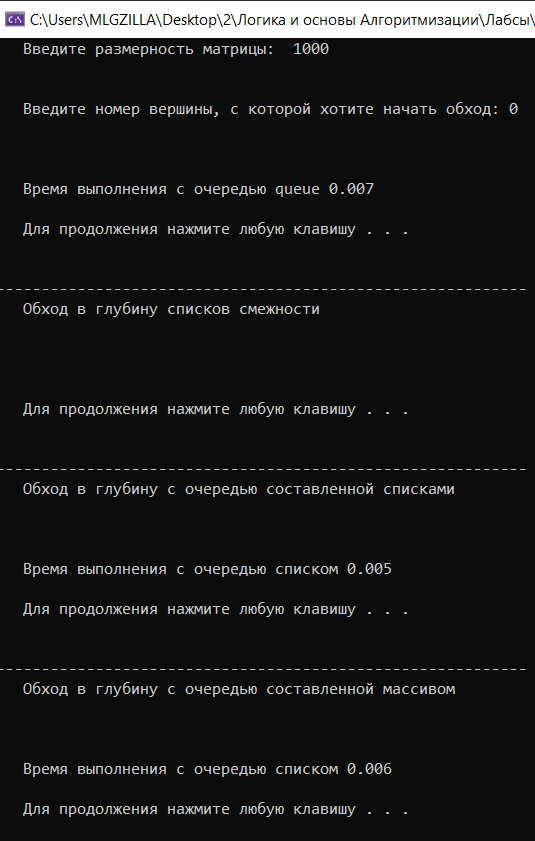
1. Матрица 10х10



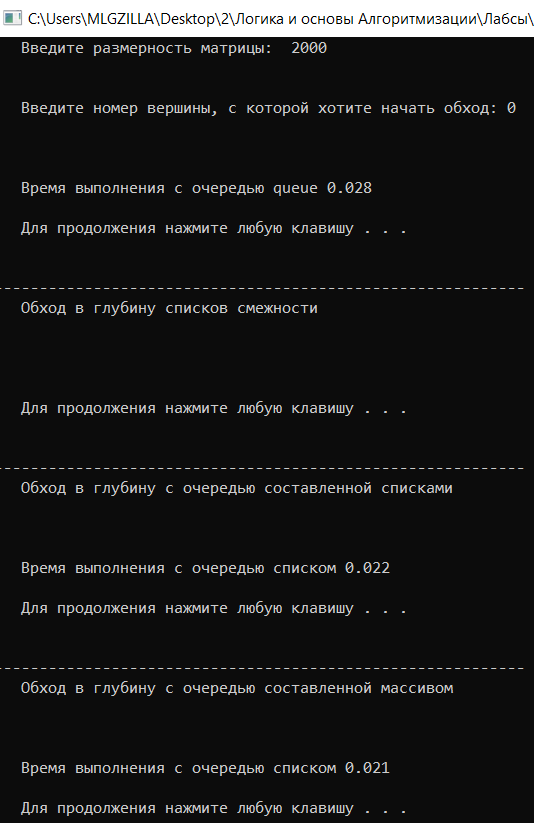
1. Матрица 500х500



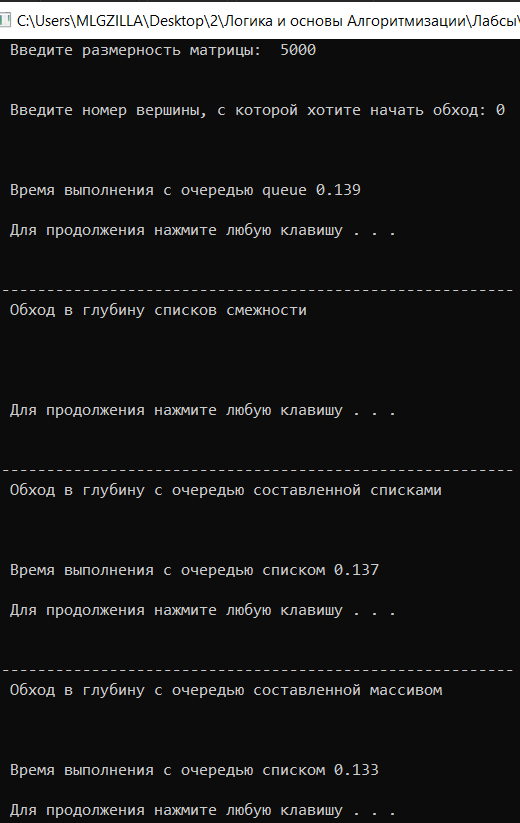
1. Матрица 1000х1000



1. Матрица 2000х2000



1. Матрица 5000х5000



**Вывод**

В ходе выполнения данной лабораторной работы были получены навыки разработки алгоритма в ширину матрицами и списками смежности, а также получили опыт использования функций стандартной библиотеки queue. Научились реализовывать очередь через структуры данных — список и массив. Сравнили алгоритмы их работы с алгоритмом работы очереди queue. В результате тестирования установили, что очередь queue при любой размерности матрицы самая медленная, кроме тех случаев когда размерность исходной матрицы не велика, тогда она идет на ровне со списком и немного выигрывает у массива. Но чем больше вершин в графе, тем очевиднее становится ее отставание. Очередь списком показывает себя достойно, как при малых, так и при великих размерностях матрицы, и тем не менее проигрывает очереди массивом, чем больше вершин в графе.