Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Логика и основы алгоритмизации в ИС»

на тему «Обход графа в ширину»

Выполнили

студенты группы 20ВВ2:

Тумасов Вадим

Портнов Никита

Принял:

д.т.н Митрохин М.А.

д.т.н Юрова О.В.

Пенза 2021

**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

глубину, реализованную в соответствии с приведенным выше описанием.

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:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <ctime> // Библиотека времени

#include <cmath>

#include <locale.h>

#include <iostream> // Библиотека Ввода,Вывода

#include <conio.h>

#include < queue >

#include <time.h>

int\*\* matrix;

using namespace std;

struct bfsQ {

int data;

bfsQ \*next, \*prev;

};

bfsQ \*head, \*tail;

typedef bfsQ \*PbfsQ;

bfsQ\* tmp;

bfsQ\* n\_tmp;

void push(int versh)

{

n\_tmp = new bfsQ;

n\_tmp->data=versh;

n\_tmp->next = NULL;

n\_tmp->prev = NULL;

if (tail == NULL)

{

tmp = n\_tmp;

head= tmp;

tail = tmp;

}

else

{

n\_tmp->next = tmp;

tmp->prev = n\_tmp;

tmp = n\_tmp;

tail = tmp;

}

}

void pop()

{

bfsQ\* m\_tmp;

if ((head) != NULL)

{

n\_tmp = head;

if ((n\_tmp->prev) !=NULL)

{

m\_tmp = n\_tmp->prev;

free(n\_tmp);

m\_tmp->next = NULL;

head = m\_tmp;

}

else

{

free(n\_tmp);

head = NULL;

tail = NULL;

}

}

}

int front()

{

if ((head) != NULL)

{

return (head->data);

}

}

bool empty\_1()

{

if ((head) == NULL)

{

return true;

}

else

{

return false;

}

}

struct spisok\_smzh {

int versh;

spisok\_smzh\* next;

};

struct spiski\_smzh {

spisok\_smzh\* spisok;

spiski\_smzh\* next;

};

spiski\_smzh\* p\_spiski = NULL;

spiski\_smzh\* n\_spiski = NULL;

spiski\_smzh\* p\_spiski\_start = NULL;

spisok\_smzh\* p\_spisok = NULL;

spisok\_smzh\* p\_spisok\_start = NULL;

spisok\_smzh\* n\_spisok = NULL;

void bfs2(int v, int n)

{

clock\_t time;

time = clock();

int p = 0, i = 0, u = 0;

char\* NUM;

NUM = new char[n];

for (u = 0; u < n; u++)

{

NUM[u] = 0;

}

cout << v;

//i = s;

queue <int> plan; // план посещения в виде очереди

p\_spiski = p\_spiski\_start;

spisok\_smzh\* p\_spisok\_loc = NULL;

plan.push(v); // мы нумеруем с 0, а не с 1

NUM[v] = 1; // отмечаем, что эта вершина уже заносилась в план

//int counter = 1; // начальную уже сосчитали

while (!plan.empty()) {

v = plan.front(); // посещаем следующую по плану вершину

plan.pop(); // удаляем ее из плана посещения

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

p\_spiski = p\_spiski->next;

p\_spisok\_loc = p\_spiski->spisok;

p\_spiski = p\_spiski\_start;

//cout « v;

while (p\_spisok\_loc != NULL) { // перебираем соседние с ней

if (!NUM[p\_spisok\_loc->versh]) { // если новая, то

plan.push(p\_spisok\_loc->versh); // добавляем ее в план

cout << p\_spisok\_loc->versh;

NUM[p\_spisok\_loc->versh] = 1; // отмечаем, что уже не новая

}

p\_spisok\_loc = p\_spisok\_loc->next;

}

}

time = clock() - time;

cout << endl;

printf("%.30f", (double)time / CLOCKS\_PER\_SEC);

}

void bfs(int v, int n)

{

clock\_t time;

time = clock();

bfsQ\* head = NULL;

bfsQ\* tail = NULL;

int p = 0, i = 0, u = 0;

char\* NUM;

NUM = new char[n];

for (u = 0; u < n; u++)

{

NUM[u] = 0;

}

cout << v;

//i = s;

//queue <int> plan; // план посещения в виде очереди

p\_spiski = p\_spiski\_start;

spisok\_smzh\* p\_spisok\_loc = NULL;

push(v); // мы нумеруем с 0, а не с 1

NUM[v] = 1; // отмечаем, что эта вершина уже заносилась в план

//int counter = 1; // начальную уже сосчитали

while (!empty\_1()) {

v = front(); // посещаем следующую по плану вершину

pop(); // удаляем ее из плана посещения

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

p\_spiski = p\_spiski->next;

p\_spisok\_loc = p\_spiski->spisok;

p\_spiski = p\_spiski\_start;

//cout « v;

while (p\_spisok\_loc != NULL) { // перебираем соседние с ней

if (!NUM[p\_spisok\_loc->versh]) { // если новая, то

push(p\_spisok\_loc->versh); // добавляем ее в план

cout << p\_spisok\_loc->versh;

NUM[p\_spisok\_loc->versh] = 1; // отмечаем, что уже не новая

}

p\_spisok\_loc = p\_spisok\_loc->next;

}

}

time = clock() - time;

cout << endl;

printf("%.30f", (double)time / CLOCKS\_PER\_SEC);

}

int main()

{

int p = 0, i = 0, n,j;

int v = 0;

//int s;

setlocale(LC\_ALL, "Russian"); // подключение русского

//srand(time(NULL)); // новый отсчёт времени

cout << "n="; // выводит данные

cin >> n; // помещает данные

matrix = new int\* [n];

//cout « "n=";

//cin » n;

cout << "Матрица: \n";

int z;

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

{

matrix[i] = new int[n];

}

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

{

for (p = 0; p < n; p++)

{

matrix[i][p] = rand() % 2;

}

}

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

{

cout << endl;

for (p = 0; p < n; p++)

{

if (p == i)

{

matrix[i][p] = 0;

}

else

{

matrix[i][p] = matrix[p][i];

}

cout << matrix[i][p]<<" ";

}

}

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

{

n\_spiski = (spiski\_smzh\*)malloc(sizeof(spiski\_smzh));

p\_spisok = NULL;

p\_spisok\_start = NULL;

for (j = 0; j < n; j++)

{

if (matrix[i][j] == 1)

{

n\_spisok = (spisok\_smzh\*)malloc(sizeof(spisok\_smzh));

n\_spisok->versh = j;

if (p\_spisok == NULL)

{

p\_spisok = n\_spisok;

p\_spisok\_start = p\_spisok;

}

else

{

p\_spisok->next = n\_spisok;

p\_spisok = n\_spisok;

}

n\_spisok->next = NULL;

}

}

n\_spiski->spisok = p\_spisok\_start;

if (p\_spiski == NULL)

{

p\_spiski = n\_spiski;

p\_spiski\_start = p\_spiski;

}

else

{

p\_spiski->next = n\_spiski;

p\_spiski = n\_spiski;

}

}

p\_spiski = p\_spiski\_start;

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

{

p\_spisok = p\_spiski->spisok;

p\_spisok\_start = p\_spisok;

cout << "\n";

cout << "{";

while (p\_spisok != NULL)

{

cout << p\_spisok->versh << " ";

p\_spisok = p\_spisok->next;

}

cout << "}" << " Вершина" << i;

p\_spisok = p\_spisok\_start;

p\_spiski = p\_spiski->next;

}

cout << "\n";

p\_spiski = p\_spiski\_start;

cout << "\n";

cout << "\n";

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

{

// used[i] = false;

for (p = 0; p < n; p++)

cout << matrix[i][p] << " "; // выводит матрицу

cout << endl;

}

cout << "Стартовая вершина » ";

//cin » s;

cin >> v;

//bool\* vis = new bool[n];

cout << "Порядок обхода: ";

//time = clock();

bfs(v, n);

//time = clock() - time;

cout << endl;

//printf("%.30f", (double)time / CLOCKS\_PER\_SEC);

cout << endl;

cout << "Порядок обхода: ";

//time = clock();

bfs2(v, n);

//time = clock() - time;

cout << endl;

//printf("%.30f", (double)time / CLOCKS\_PER\_SEC);

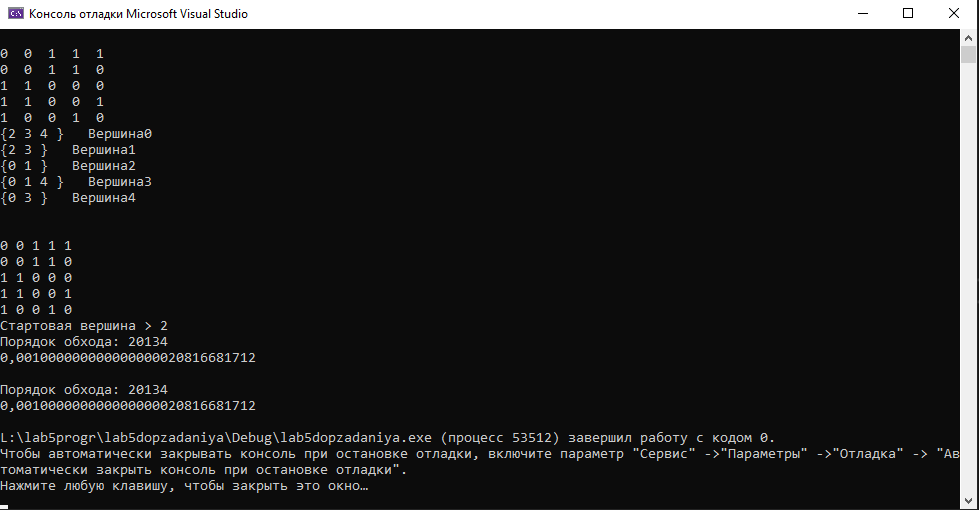
//delete[] used;

\_getch();

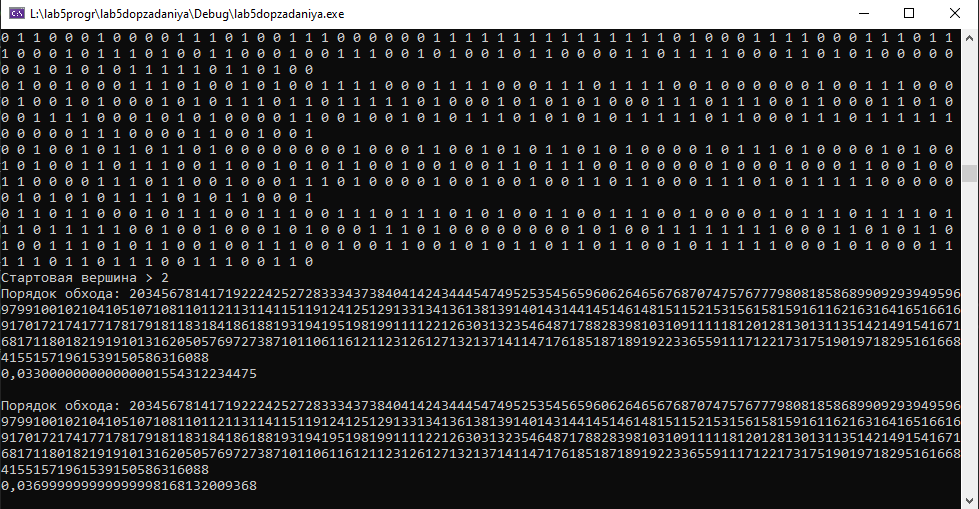
}

**Результат работы программы:**

Количество вершин=5



Количество вершин=200



**Вывод**: Мы сгенерировали матрицу смежности для неориентированного графа, организовали обход в ширину с помощью bfs и класса queue.Также в качестве эксперимента быстродействия алгоритмов мы создали обход в ширину на основе собственной очереди и функций,а не класса queue.В ходе данного эксперимента мы обнаружили то,что данные алгоритмы схожи по быстродействию при работе с маленьким количеством данных,а на больших количествах видны различия:наш алгоритм работает быстрее чем стандартный,но в ходе подобного эксперимента днем ранее мы видели прямо противоположную картину,следовательно можно сделать вывод о том,что данные алгоритмы впринципе очень схожи по эффективности и имеется слишком много факторов,которые могут повлиять на определение самого эффективного.