Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в ИЗ»

## на тему «Обход графа в ширину»

Выполнил:

студент групп 22ВВВ2

Изосин М.А.

Приняли:

д.т.н. профессор Митрохин М.А.

к.э.н. доцент Акифьев И.В.

Пенза 2023

**Название**

Обход графа в ширину

**Цель работы**

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++.

**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++.

### 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 <stdlib.h>

#include <locale.h>

#include <string.h>

#include <time.h>

#define Max\_Size\_Q 100

// Функции добавления элемента, удаления

void spstore(int x);

char find\_el[256];

struct node \*get\_struct(int x); // функция создания элемента

struct node \*head = NULL, \*last = NULL, \*f = NULL; // указатели на первый и последний элементы списка

struct node

{

int vertex; // номер вершины

struct node \*next; // ссылка на следующий элемент

};

struct node \*get\_struct(int x)

{

struct node \*p = NULL;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->vertex = x;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

void spstore(int x)

{

struct node \*p = NULL;

p = get\_struct(x);

if (head == NULL && p != NULL) // если списка нет, то устанавливаем голову списка

{

head = p;

last = p;

}

else if (head != NULL && p != NULL) // список уже есть, то вставляем в конец

{

last->next = p;

last = p;

}

return;

}

void find\_och(int x, int\*\* k, int m, int\* D)

{

spstore(x);

D[x - 1] = 1;

while (x != 0)

{

struct node \*struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

printf("%d\n", struc->vertex);

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

{

if (k[x - 1][i] == 1 && D[i] == 0)

{

spstore(i + 1);

D[i] = 1;

}

}

if (head == NULL) // если голова списка равна NULL, то список пуст

{

printf("Список пуст\n");

return;

}

head = struc->next; // установливаем голову на следующий элемент

free(struc); // удаляем первый элемент

struc = head; // устанавливаем указатель для продолжения поиска

if(struc != 0)

x = struc->vertex;

else

x = 0;

}

}

void find\_Q(int x, int\*\* k, int m, int\* D)

{

int Q[Max\_Size\_Q] = { 0 };

int n = 0;

int j = 0;

Q[j] = x;

D[x - 1] = 1;

while (x != 0)

{

printf("%d\n", Q[j]);

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

{

if (k[x - 1][i] == 1 && D[i] == 0)

{

n++;

Q[n] = i + 1;

D[i] = 1;

}

}

j++;

x = Q[j];

}

}

int main ()

{

setlocale(LC\_ALL, "rus");

clock\_t start, end; // объявляем переменные для определения времени выполнения

int m;

int l;

int x; // Вершина

srand(time(NULL));

printf("Введите размер матрицы\n");

scanf("%d", &m);

int\*\* k = (int\*\*)malloc(m \* sizeof(int\*));

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

{

k[i] = (int\*)malloc(m \* sizeof(int));

}

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

{

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

{

k[i][j] = 0;

}

}

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

{

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

{

int t = rand() % 2;

k[i][j] = t;

k[j][i] = t;

k[i][i] = 0;

}

}

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

{

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

{

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

}

printf("\n");

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int\* D = (int\*)malloc(m \* sizeof(int));

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

{

D[i] = 0; //Массив вершин

}

printf("Введите номер вершины - ");

scanf("%d", &l);

x = l;

start = clock(); // в переменную start записываем время от начала запуска программы

find\_och(x, k, m, D);

end = clock(); // в переменную end записываем время от начала запуска программы

double time = (end - start) \* 1000.0 / CLOCKS\_PER\_SEC; // вычисляем разность(т.е. время, затраченное на умножение матриц)

printf("%lf ms\n", time); // выводим результат работы программы в секундах

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

{

D[i] = 0; //Массив вершин

}

x = l;

start = clock(); // в переменную start записываем время от начала запуска программы

find\_och(x, k, m, D);

end = clock(); // в переменную end записываем время от начала запуска программы

time = (end - start) \* 1000.0 / CLOCKS\_PER\_SEC; // вычисляем разность(т.е. время, затраченное на умножение матриц)

printf("%lf ms\n", time); // выводим результат работы программы в секундах

free(D);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

getchar();getchar();

return 0;

}

**Результат работы программы**

Результат работы программы показан на рисунках.

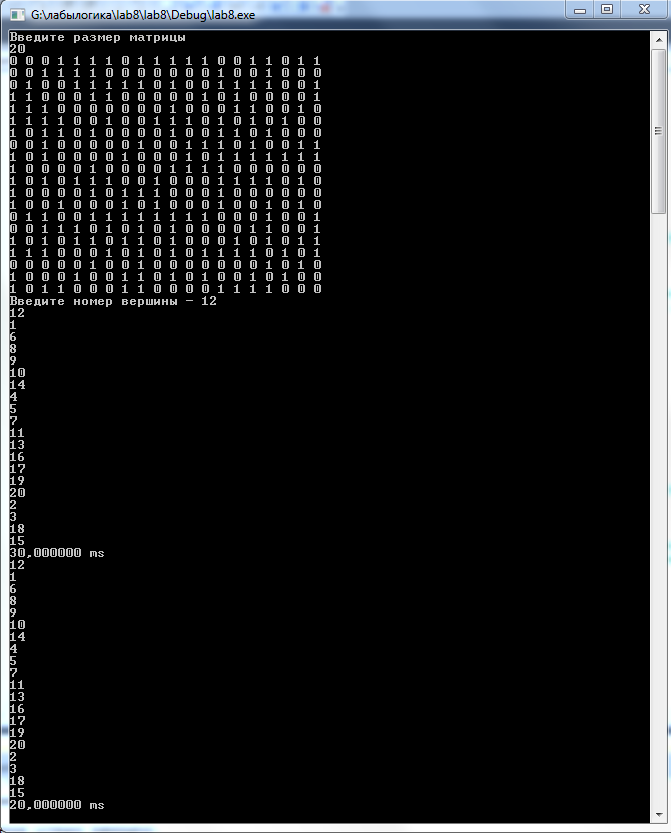


Рис 1.

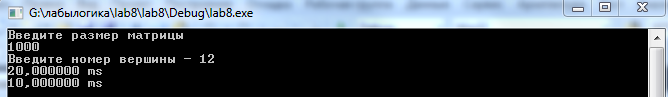


Рис 2.



Рис 3.

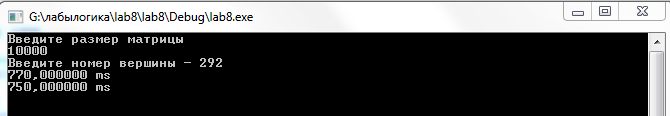


Рис 4.

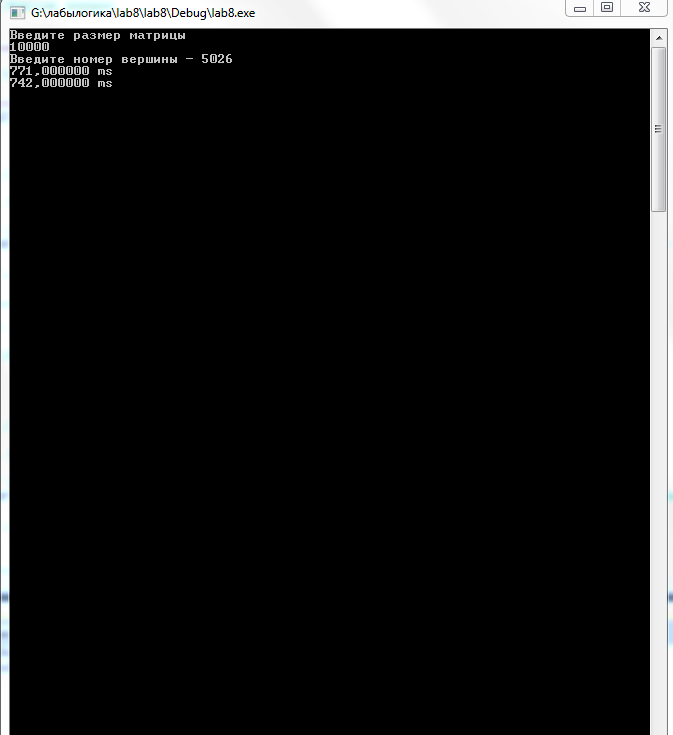


Рис 5.

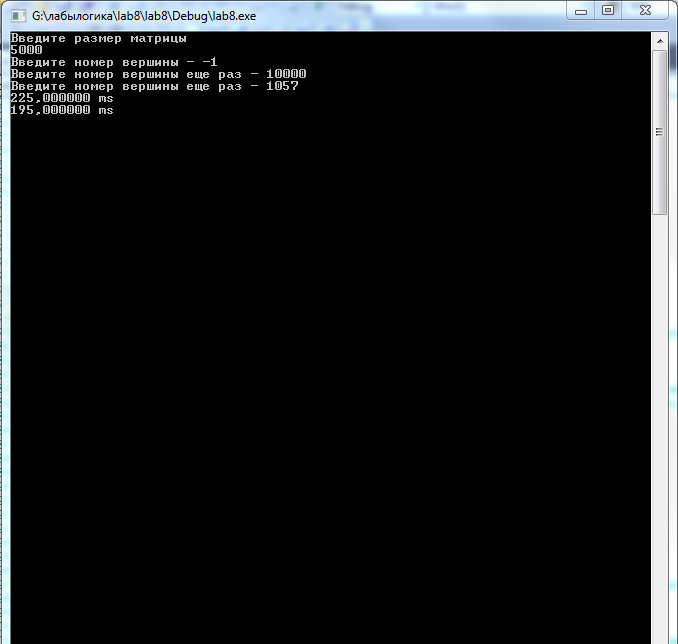


Рис 6.

**Вывод:** в ходе выполнения лабораторной работы мы научились совершать обход графа в ширину, используя стандартный класс **queue** и очередь, реализованную самостоятельно. В результате подсчета времени, затраченного на выполнение программ выяснилось, что программа, реализованная очередью выполняет обход графа быстрее.

Ссылка на репозиторий: