

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №8
«Обход графа в ширину»
По дисциплине «Л и ОА в ИЗ»

Выполнили: ст. гр. 22ВВ4
Жуков Илья
Чумаев Сабит

Приняли: Юрова О.В.
Акифьев И.В.

Цель работы:

Написать код программы, выполнив следующие задания:

По заданию 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 C++.
- 3.* Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.

По заданию 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.

Ход работы:

Описание кода программы по заданию 1.1 - 1.2:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Laba8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите размер матрицы: ");

            int size = Convert.ToInt32(Console.ReadLine());

            int[,] adjacencyMatrix = GenerateAdjacencyMatrix(size);

            Console.WriteLine("Матрица смежности для графа G1:");
            PrintMatrix(adjacencyMatrix);

            Console.Write("Введите вершину, с которой начать обход:");

            int startVertex = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Результат обхода в ширину:");
            BreadthFirstSearch(adjacencyMatrix, startVertex);
        }

        //Метод GenerateAdjacencyMatrix генерирует случайную матрицу
        смежности для графа.
```

```

private static int[,] GenerateAdjacencyMatrix(int size)
{
    Random r = new Random();

    int[,] matrix = new int[size, size];

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (i != j)
            {
                //для каждой пары вершин (i, j) генерируется
случайное число 0 или 1, которое указывает наличие или отсутствие
ребра между вершинами.

                matrix[i, j] = r.Next(2);
                matrix[j, i] = matrix[i, j];
            }
        }
    }
    return matrix;
}

//Метод PrintMatrix выводит матрицу смежности на экран.
static void PrintMatrix(int[,] matrix)
{
    int size = matrix.GetLength(0);

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

```

```

        Console.Write(matrix[i, j] + " ");
    }
    Console.WriteLine();
}
Console.WriteLine();
}

//метод, выполняющий обход в ширину графа, начиная с указанной
начальной вершины

//В процессе обхода каждая посещенная вершина выводится на
экран.Обход осуществляется с помощью очереди:

//начальная вершина добавляется в очередь, затем извлекается
из очереди и все её смежные не посещенные

//вершины добавляются в очередь.Этот процесс продолжается,
пока очередь не опустеет.

private static void BreadthFirstSearch(int[,] adjacencyMatrix,
int startVertex)
{
    int size = adjacencyMatrix.GetLength(0);
    bool[] visited = new bool[size];
    Queue<int> queue = new Queue<int> ();

    //Помечаем начальную вершину как посещенную
    visited[startVertex] = true;

    //Добавляем начальную вершину в очередь
    queue.Enqueue(startVertex);

    while(queue.Count > 0)
    {
        //Извлекаем следующую вершину из очереди
        int currentVertex = queue.Dequeue();
    }
}

```


//Конструктор класса принимает размер графа и инициализирует пустой список adjacencyList, который будет хранить смежные вершины для каждой вершины графа.

```
public Graph(int size)
{
    adjacencyList = new List<List<int>>();
    for (int i = 0; i < size; i++)
    {
        adjacencyList.Add(new List<int>());
    }
}
```

//Метод AddEdge принимает две вершины - from и to, и добавляет их в список смежности друг друга.

//Это позволяет установить связь между вершинами графа.

```
public void AddEdge(int from, int to)
{
    adjacencyList[from].Add(to);
    adjacencyList[to].Add(from);
}
```

//Метод GetNeighbors принимает вершину графа и возвращает список ее смежных вершин.

```
public List<int> GetNeighbors(int vertex)
{
    return adjacencyList[vertex];
}
```

//Метод PrintGraph выводит на экран информацию о графе. Он перебирает все вершины графа и для каждой вершины выводит ее номер

номера. //, а затем перебирает все смежные вершины и выводит их номера.

```
public void PrintGraph()
{
    for (int i = 0; i < adjacencyList.Count; i++)
```

```

        {
            Console.Write($"Вершина {i + 1}: ");
            foreach (var vertex in adjacencyList[i])
            {
                Console.Write($"{vertex + 1} ");
            }
            Console.WriteLine();
        }
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Console.Write("Введите размер графа: ");
        int size = Convert.ToInt32(Console.ReadLine());

        Graph graph = GenerateAdjacencyList(size);

        Console.WriteLine("Граф смежности для графа G1:");
        graph.PrintGraph();

        //массив visited, который используется для отслеживания
        //посещенных вершин.
        bool[] visited = new bool[size];

        //цикл, который проходит по всем вершинам графа.
        for (int startVertexIndex = 0; startVertexIndex < size;
startVertexIndex++)
        {

```



```

        //Если текущая вершина не была посещена, то
        выполняется обход в глубину с использованием функции DepthFirstSearch

        if (!visited[startVertexIndex])
        {
            Console.WriteLine("Обход в глубину, начиная с
            вершины " + (startVertexIndex + 1) + ":");

            DepthFirstSearch(startVertexIndex, graph,
            visited);
        }
    }

    //Функция GenerateAdjacencyList генерирует случайный граф
    заданного размера size в виде списка смежности.

    private static Graph GenerateAdjacencyList(int size)
    {
        Random r = new Random();

        Graph graph = new Graph(size);

        //двойной цикл, который перебирает все возможные
        комбинации вершин графа.

        for (int i = 0; i < size; i++)
        {
            for (int j = i + 1; j < size; j++)
            {
                //Если сгенерированное число равно 1, то
                вызывается метод AddEdge объекта graph для добавления ребра между
                вершинами i и j.

                if (r.Next(2) == 1)
                {
                    graph.AddEdge(i, j);
                }
            }
        }
    }
}

```

```
        //После завершения циклов возвращается объект graph,  
        содержащий случайно сгенерированный граф в виде списка смежности.  
  
        return graph;  
  
    }
```

```
    static void DepthFirstSearch(int startVertexIndex, Graph  
graph, bool[] visited)
```

```
    {  
  
        //Алгоритм использует очередь для хранения вершин, которые  
        нужно посетить.
```

```
        Queue<int> queue = new Queue<int> ();
```

```
        //Начальная вершина помещается в очередь  
        queue.Enqueue(startVertexIndex);
```

```
        visited[startVertexIndex] = true;
```

```
        //а затем пока очередь не пустая, извлекается текущая  
        вершина из очереди.
```

```
        while (queue.Count > 0)
```

```
        {
```

```
            int currentVertexIndex = queue.Dequeue();
```

```
            //Затем выводится информация о посещенной вершине(в  
            данном случае просто ее номер)
```

```
            Console.WriteLine("Посещена вершина: №" +  
(currentVertexIndex + 1));
```

```
            //и для каждого соседа текущей вершины, который еще не  
            был посещен, он добавляется в очередь и отмечается как посещенный.
```

```
            List<int> neighbors =  
graph.GetNeighbors(currentVertexIndex);
```



```

        Console.WriteLine("Матрица смежности для графа G1:");
        PrintMatrix(adjacencyMatrix);

        Console.Write("Введите вершину, с которой начать обход:");
    };

    int startVertex = Convert.ToInt32(Console.ReadLine());

    DateTime startTime = DateTime.Now;

    Console.WriteLine("Результат обхода в ширину(List): ");
    BFS(adjacencyMatrix, startVertex);

    DateTime endTime = DateTime.Now;
    TimeSpan listTime = endTime - startTime;

    Console.WriteLine("Время работы обхода в ширину(с использованием класса List): " + listTime.TotalMilliseconds + " миллисекунд");

    DateTime startTime2 = DateTime.Now;

    Console.WriteLine("\n" + "Результат обхода в ширину(Queue):");
    BreadthFirstSearch(adjacencyMatrix, startVertex);

    DateTime endTime2 = DateTime.Now;
    TimeSpan listTime2 = endTime2 - startTime2;

    Console.WriteLine("Время работы обхода в ширину(с использованием класса Queue): " + listTime2.TotalMilliseconds + " миллисекунд");

```

```
}
```

```
static void BFS(int[,] graph, int startVertex)
{
    int size = graph.GetLength(0);

    //Создается массив "visited" размером "size", который
    //будет хранить информацию о посещенных вершинах.
    bool[] visited = new bool[size];
    for (int i = 0; i < size; i++)
    {
        //В начале все элементы массива устанавливаются в
        //значение "false".
        visited[i] = false;
    }

    //Создается пустой список "queue", который будет
    //использоваться для хранения вершин, ожидающих обработки.
    List<int> queue = new List<int>();

    //Метод начинает с посещения стартовой вершины
    //startVertex, устанавливая соответствующий элемент в массиве
    //visited в значение "true" и добавляя вершину в "queue".
    visited[startVertex] = true;
    queue.Add(startVertex);

    //пока "queue" не пуст, выполняется следующее:
    while (queue.Count > 0)
    {
        //Извлекается первая вершина из "queue" и сохраняется
        //в переменной "currentVertex".
        int currentVertex = queue[0];
```

```

        //Посещенная вершина "currentVertex" выводится на
консоль.

        Console.WriteLine("Посещена вершина: №" +
currentVertex);

        queue.RemoveAt(0);

        for (int i = 0; i < size; i++)
        {
            //Для каждой смежной вершины "i" с
"currentVertex", если она еще не посещена (элемент "visitedi" равен
"false") и есть ребро между "currentVertex" и "i" (элемент
"graph[currentVertex, i]" равен 1), выполняется:

            if (graph[currentVertex, i] == 1 && !visited[i])
            {
                //Пометить "i" как посещенную

                visited[i] = true;

                //Добавить "i" в "queue".

                queue.Add(i);
            }
        }
    }

    private static void BreadthFirstSearch(int[,] adjacencyMatrix,
int startVertex)
    {
        //Создается массив "visited" размером "size", который
будет хранить информацию о посещенных вершинах.

        int size = adjacencyMatrix.GetLength(0);

        bool[] visited = new bool[size];

        //Создается пустая очередь "queue", которая будет
использоваться для хранения вершин, ожидающих обработки.

        Queue<int> queue = new Queue<int>();

```

```

//Помечаем начальную вершину как посещенную
visited[startVertex] = true;

//Добавляем начальную вершину в очередь
queue.Enqueue(startVertex);

while (queue.Count > 0)
{
    //Извлекаем следующую вершину из очереди
    int currentVertex = queue.Dequeue();

    //Посещенная вершина "currentVertex" выводится на
    консоль.
    Console.WriteLine("Посещена вершина: №" +
currentVertex);

    for (int i = 0; i < size; i++)
    {
        //Для каждой смежной вершины "i" с
        "currentVertex", если она еще не посещена (элемент "visitedi" равен
        "false") и есть ребро между "currentVertex" и "i" (элемент
        "adjacencyMatrix[currentVertex, i]" равен 1),
        if (adjacencyMatrix[currentVertex, i] == 1
        && !visited[i])
        {
            //Помечаем смежную вершину как посещенную
            visited[i] = true;

            //Добавляем смежную вершину в очередь
            queue.Enqueue(i);
        }
    }
}

```

```

    }
}

//Метод GenerateAdjacencyMatrix генерирует случайную матрицу
смежности для графа.

private static int[,] GenerateAdjacencyMatrix(int size)
{
    Random r = new Random();

    int[,] matrix = new int[size, size];

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (i != j)
            {
                //для каждой пары вершин (i, j) генерируется
                случайное число 0 или 1, которое указывает наличие или отсутствие
                ребра между вершинами.

                matrix[i, j] = r.Next(2);
                matrix[j, i] = matrix[i, j];
            }
        }
    }

    return matrix;
}

//Метод PrintMatrix выводит матрицу смежности на экран.
static void PrintMatrix(int[,] matrix)
{
    int size = matrix.GetLength(0);

```



```
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                Console.Write(matrix[i, j] + " ");
            }
            Console.WriteLine();
        }
        Console.WriteLine();
    }
}
```

Результат работы программы 1.1-1.2:

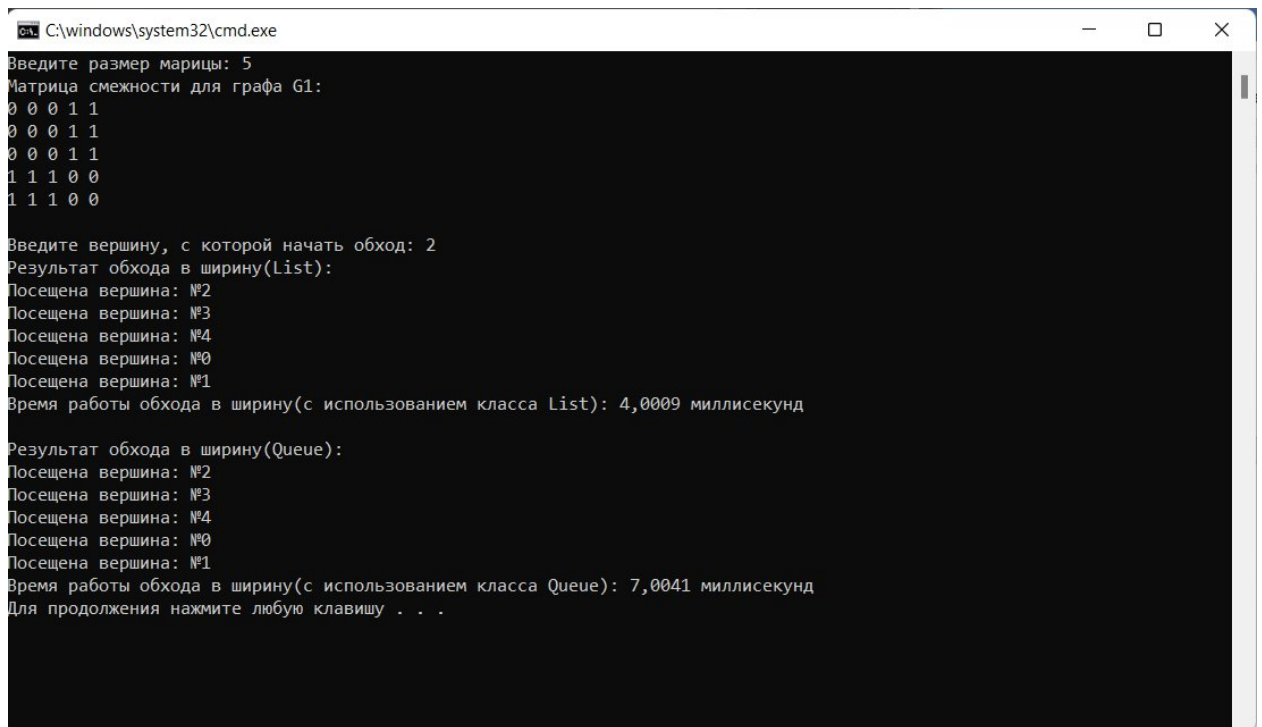
```
C:\windows\system32\cmd.exe
Введите размер матрицы: 4
Матрица смежности для графа G1:
0 0 1 0
0 0 0 1
1 0 0 1
0 1 1 0

Введите вершину, с которой начать обход: 0
Результат обхода в ширину:
Посещена вершина: №0
Посещена вершина: №2
Посещена вершина: №3
Посещена вершина: №1
Для продолжения нажмите любую клавишу . . .
```

Результат работы программы 1.3:

```
C:\windows\system32\cmd.exe
Введите размер графа: 4
Матрица смежности для графа G1:
Вершина 1: 2 3
Вершина 2: 1
Вершина 3: 1
Вершина 4:
Обход в глубину, начиная с вершины 1:
Посещена вершина: №1
Посещена вершина: №2
Посещена вершина: №3
Обход в глубину, начиная с вершины 4:
Посещена вершина: №4
Для продолжения нажмите любую клавишу . . .
```

Результат работы программы 2:



```
C:\windows\system32\cmd.exe
Введите размер матрицы: 5
Матрица смежности для графа G1:
0 0 0 1 1
0 0 0 1 1
0 0 0 1 1
1 1 1 0 0
1 1 1 0 0

Введите вершину, с которой начать обход: 2
Результат обхода в ширину(List):
Посещена вершина: №2
Посещена вершина: №3
Посещена вершина: №4
Посещена вершина: №0
Посещена вершина: №1
Время работы обхода в ширину(с использованием класса List): 4,0009 миллисекунд

Результат обхода в ширину(Queue):
Посещена вершина: №2
Посещена вершина: №3
Посещена вершина: №4
Посещена вершина: №0
Посещена вершина: №1
Время работы обхода в ширину(с использованием класса Queue): 7,0041 миллисекунд
Для продолжения нажмите любую клавишу . . .
```

Вывод: в ходе лабораторной работы были получены навыки реализации обхода графа в ширину для матрицы смежности и списка смежности. По 1 заданию - в качестве очереди был использован класс queue, для 2 задания - на основе структуры данных «список». Также было оценено время работы двух реализаций алгоритмов обхода в ширину. В ходе которого метод, использующий класс Queue выполнил обход графа медленнее, чем метод, использующий очередь, реализованную самостоятельно.