Пензенский государственный университет  
Кафедра «Вычислительной техники»

**Отчет**по лабораторной работе №5  
по курсу "ЛиОАвИЗ"

**Выполнили студент группы 20ВВ1:**

Вяльмисов М.О.

**Приняли:**

Юрова О.В.

Акифьев И.В.

Пенза 2021

**Лабораторные задания:**

### **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 алгоритмов:**

**MatrixGraph:**

public List<int> BreadthFirstSearch()

{

\_walkedList = new bool[\_size];

for (int i = 0; i < \_size; i++)

{

if (!\_walkedList[i])

{

UnderBreadthFirstSearch(i);

}

}

return \_walkList;

}

private void UnderBreadthFirstSearch(int v)

{

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

queue.Enqueue(v);

\_walkedList[v] = true;

while(queue.Count > 0)

{

v = queue.Dequeue();

\_walkList.Add(v);

for(int i = 0; i < \_size; i++)

{

if(\_matrix[v,i] == 1 && !\_walkedList[i])

{

queue.Enqueue(i);

\_walkedList[v] = true;

}

}

}

}

public List<int> MyBreadthFirstSearch()

{

\_walkedList = new bool[\_size];

for (int i = 0; i < \_size; i++)

{

if (!\_walkedList[i])

{

UnderBreadthFirstSearch(i);

}

}

return \_walkList;

}

private void MyUnderBreadthFirstSearch(int v)

{

queueuueue queue = new queueuueue();

queue.Push(v);

\_walkedList[v] = true;

while (queue.Count > 0)

{

v = queue.Pop();

\_walkList.Add(v);

for (int i = 0; i < \_size; i++)

{

if (\_matrix[v, i] == 1 && !\_walkedList[i])

{

queue.Push(i);

\_walkedList[v] = true;

}

}

}

}

**ListGraph:**

public List<int> BreadthFirstSearch()

{

\_walkedList = new bool[\_list.Count];

for (int i = 0; i < \_list.Count; i++)

{

if (!\_walkedList[i])

{

UnderBreadthFirstSearch(i);

}

}

return \_walkList;

}

private void UnderBreadthFirstSearch(int v)

{

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

queue.Enqueue(v);

\_walkedList[v] = true;

while (queue.Count > 0)

{

v = queue.Dequeue();

\_walkList.Add(v);

foreach(var el in \_list[v])

{

if (!\_walkedList[el-1])

{

queue.Enqueue(el-1);

\_walkedList[v] = true;

}

}

}

}