Breadth First Search Algorithm

I have created Vertex, Graph and Searches Class in C#. In Windows Forms App, It shows searched nodes in RichTextBox in a layout.

Vertex has adding neighbor and removing neighbor methods.

Graph has Add and AddPair methods, They are usable for users out of class and some private methods. Add method arranges the path from first node to another. AddPair method arranges the path between each other in nodes.

Searches has Breadth First Search algorithm.

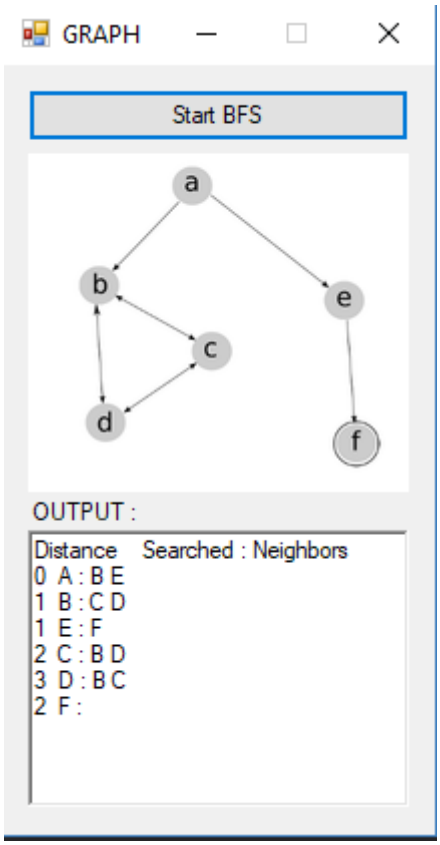
All of the public methods and variables are explaned in the comment lines.

Note: I didn’t expand methods in screenshots because They doesn’t fit in screenshots.

```
//          Represents a node for a Graph Data Structure
// Variables:
//          Value holds the vertex value that is type of T which is readonly.
//          Distance represents the length from parent Vertex that can be arranged manually or in a search method.
//          Neighbors represents neighbors of the vertex that is readonly.
//          IsVisited is a control variable that controls wheter it is visited or not.
//          NeighborCount represents the amount of neighbors.
class Vertex<T>
{
    public T Value { get; }
    public int Distance { get; set; }
    public List<Vertex<T>> Neighbors { get; }
    public bool IsVisited { get; set; }
    public int NeighborCount => Neighbors.Count;
    // Summary:
    //          Initializes the Vertex Class.
    // Parameters:
    //          value is for filling the Value variable of Vertex Class.
    //          neighbors is default null that arrange Neighbors list.
    public Vertex(T value, IEnumerable<Vertex<T>> neighbors = null)...
    // Summary:
    //          Adds vertex parameter into Neighbors list.
    // Parameters:
    //          vertex is a Vertex class.
    public void AddEdge(Vertex<T> vertex)...
    // Summary:
    //          Adds newNeighbors comes as an array into Neighbors list by using AddRange.
    // Parameters:
    //          vertex is a Vertex class.
    public void AddEdges(params Vertex<T>[] newNeighbors)...
    // Summary:
    //          Adds newNeighbors comes as an IEnumerable list into Neighbors list by using AddRange.
    // Parameters:
    //          vertex is a Vertex class.
    public void AddEdges(IEnumerable<Vertex<T>> newNeighbors)...
    // Summary:
    //          Removes parameter vertex from Neighbors.
    // Parameters:
    //          vertex is a Vertex class.
    public void RemoveEdge(Vertex<T> vertex)...

    // Summary:
    //          Overrided ToString method returns neighbors of vertex in a layout.
    public override string ToString()...
```

```
// Summary:
//          Represents a Graph Data Structure
// Variables:
//          Vertices stores Vertex by using list structure.
//          Size represents Vertex list lenght.
class Graph<T>
{
    public List<Vertex<T>> Vertices { get; }
    public int Size => Vertices.Count;

    // Summary:
    //          Initializes the Graph Class.
    // Parameters:
    //          initialNodes is a Vertex IEnumerable list for initializing Vertices that is default null.
    public Graph(IEnumerable<Vertex<T>> initialNodes = null)...

    // Summary:
    //          Adds vertex parameters into Vertex list and make a path from first to second.
    // Parameters:
    //          first is a Vertex that is source in the path hierarchy.
    //          second is a Vertex that is destination in the path hierarchy.
    public void Add(Vertex<T> first,Vertex<T> second)...
    // Summary:
    //          Adds vertex parameters into Vertex list and make a path between each other.
    // Parameters:
    //          first is a Vertex.
    //          second is a Vertex.
    public void AddPair(Vertex<T> first,Vertex<T> second)...

    private void AddToList(Vertex<T> vertex)...

    private void AddNeighbor(Vertex<T> first,Vertex<T> second)...

    private void AddNeighbors(Vertex<T> first, Vertex<T> second)...

    // Summary:
    //          Makes all of the Vertice's IsVisited variable false.
    public void UnvisitAll()...
```

```
// Summary:
//          Includes search algorithms.
// Variables:
//          queue is used for search algorithms.
class Searches<T>
{
    Queue queue;
    // Summary:
    //          Initializes the Searches Class.
    public Searches()
    {
        queue = new Queue();
    }
    // Summary:
    //          Applies Breadth First Search algorithms using Queue Class and arranges distance variable of Vertex.
    // Parameters:
    //          root is used as the beginning node.
    //          richTextBox is RichTextBox reference that shows searched node's datas.
    public void BreadthFirstSearch(Vertex<T> root, RichTextBox richTextBox)
    {
        queue.Enqueue(root);
        while (!queue.isEmpty())
        {
            Vertex<T> temp = (Vertex<T>)queue.Dequeue();
            if (!temp.IsVisited) {
                temp.IsVisited = true;
                TextBoxWriter(temp, richTextBox);
                foreach (var vertex in temp.Neighbors)
                {
                    vertex.Distance = temp.Distance + 1;
                    queue.Enqueue(vertex);
                }
            }
        }
    }
    private void TextBoxWriter(Vertex<T> temp,RichTextBox richTextBox)...
```


GRAPH
Start BFS
a
b
c
d
e
f
OUTPUT :
Distance Searched : Neighbors
0 A : B E
1 B : C D
1 E : F
2 C : B D
3 D : B C
2 F :