GTU Department of Computer Engineering
CSE 222/505 – Spring 2021
Homework 8

OZAN GEÇKİN
1801042103

1) Detailed System Requirements

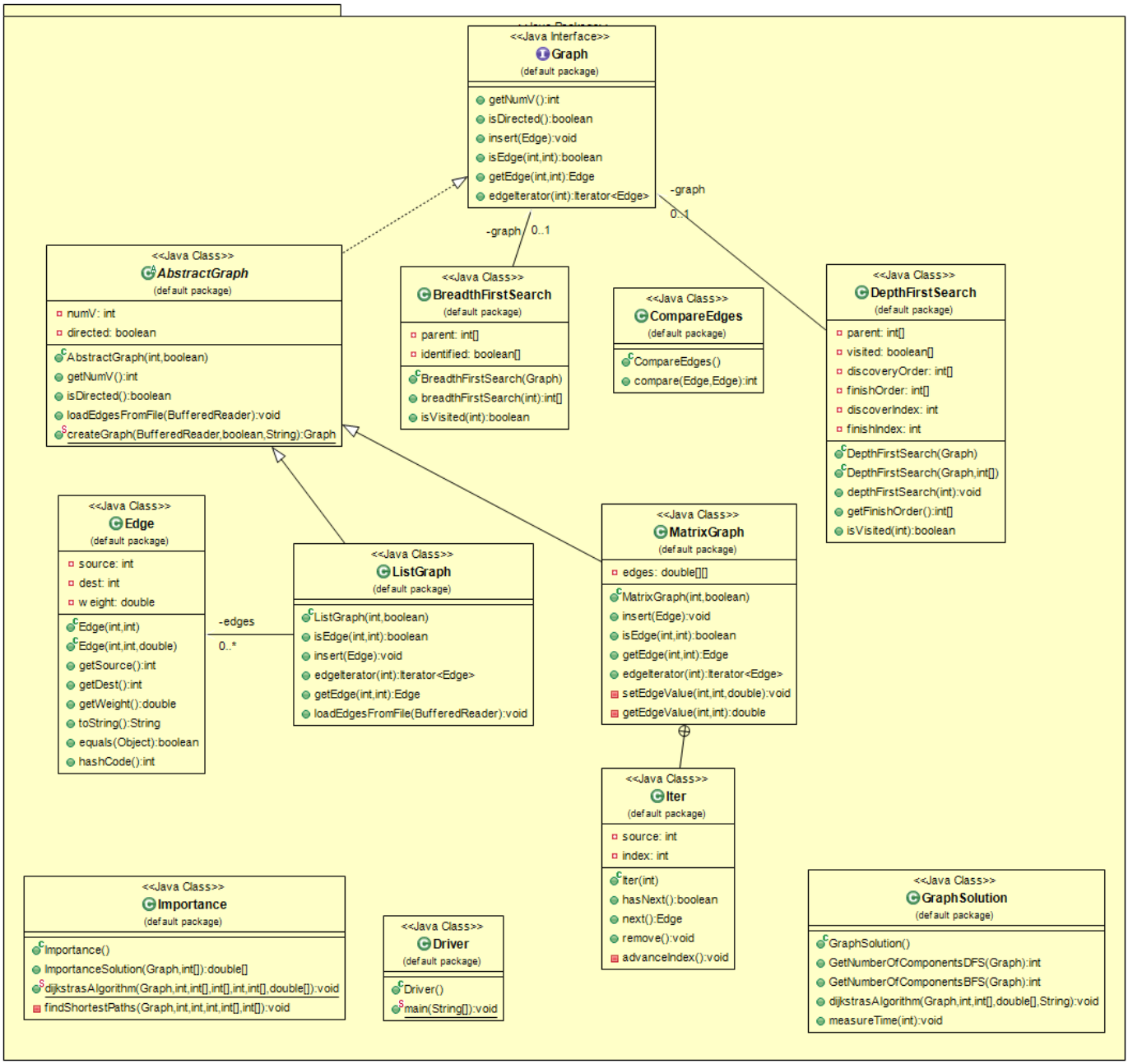
In this system we have created the methods one for each part to do specific goal. We have generalized the implementation of the Dijkstra algorithm to support time or any value on the edge's weights. Then we have created an algorithm to find the number of components in a graph using Depth-First search and Breadth-First search algorithms and generated a large number random graphs to compare the efficiency for both algorithms according to the time. Finally, we calculated the Normal Importance for each vertex in a graph using a new class that we created to do this goal, we used here the Dijkstra algorithm more than one time to get the distances for each separate vertex. All the methods have been implemented using Java programming language and all the method have been tested using many test cases to check the correctness of each method.

According to the first part we didn't change the source idea of the Dijkstra algorithm but just we added new parameter to get the operator type whether it's addition, multiplication or * operator. Also, we have generalized the edges weight to act like a weight or time or value. At last, we have generalized the graph type to be MatrixGraph or ListGraph using a class for each one.

According to the second part, we just tried to do a Breadth-First search or Depth-First search for each unvisited vertex and count how many times we did it. Then we generated random number of edges for each 1000,2000,5000 and 10000 vertices graphs to test the efficiency.

According to the last part, we used the Dijkstra algorithm to calculate the number of shortest paths and store the result.

2) Class Diagram



3) Problem Solution Approach

- **Part 1**
  Here is the method that we have implemented to generalize the Dijkstra algorithm:

```
public static void dijkstrasAlgorithm(Graph graph,int start,int[] pred, double[] dist, String type) {
      int numV = graph.getNumV();
      HashSet < Integer > vMinusS = new HashSet < Integer > (numV);
      // Initialize V﷼S.
      for (int i = 0; i < numV; i++) {
       if (i != start) {
       vMinusS.add(i);
       }
      }
      // Initialize pred and dist.
      for (int v : vMinusS) {
      pred[v] = start;
       dist[v] = graph.getEdge(start, v).getWeight();
      }
      // Main loop
      while (vMinusS.size() != 0) {
       // Find the value u in V﷼S with the smallest dist[u].
       double minDist = Double.POSITIVE_INFINITY;
       int u = -1;
       for (int v : vMinusS) {
        if (dist[v] < minDist) {
        minDist = dist[v];
          u = v;
        }
       }
       // Remove u from vMinusS.
       vMinusS.remove(u);
       // Update the distances.
       for (int v : vMinusS) {
        if (graph.isEdge(u, v)) {
          double weight = graph.getEdge(u, v).getWeight();
          double operatorResult = 0.0;
          if(type == "Multiplication")
             operatorResult = dist[u] * weight;
          else if(type == "*")
             operatorResult = dist[u] + weight - (dist[u] * weight);
          else
             operatorResult = dist[u] + weight;
          if (operatorResult < dist[v]) {
           dist[v] = operatorResult;
           pred[v] = u;
          }
         }
       }
      }
   }
```

Where the blue marked syntaxes are the new syntaxes that we added to the algorithm where the If statement is just to choose the type of the operator, and we are getting it from the parameters. According to the edges we are representing it as integer value whatever it acts. According the type of the graph we can pass it in the parameters and we created inheriting class from graph class that acts the MatrixGraph and ListGraph. So, we can call the method in one of the following ways:

```
Graph g = new ListGraph(4,false);
Edge e0 = new Edge(0,1,1);
Edge e1 = new Edge(0,2,5);
Edge e2 = new Edge(1,2,3);
Edge e3 = new Edge(1,3,5);
Edge e4 = new Edge(2,3,2);
g.insert(e0);
g.insert(e1);
g.insert(e2);
g.insert(e3);
g.insert(e4);
int pr[] = new int[4];
double di[] = new double[4];
dijkstrasAlgorithm(g,0,pr,di,"Addition");
```

Or:

```
Graph g = new MatrixGraph(4,false);
Edge e0 = new Edge(0,1,1);
Edge e1 = new Edge(0,2,5);
Edge e2 = new Edge(1,2,3);
Edge e3 = new Edge(1,3,5);
Edge e4 = new Edge(2,3,2);
g.insert(e0);
g.insert(e1);
g.insert(e2);
g.insert(e3);
g.insert(e4);
int pr[] = new int[4];
double di[] = new double[4];
dijkstrasAlgorithm(g,0,pr,di,"Addition");
```

- **Part 2**
  i. We have created the methods depending on the original Depth-First search and Breadth-First search algorithms. We just called the algorithms on every unvisited vertex. Here is the code:

```
public static int GetNumberOfComponentsDFS(Graph g){
   int ret = 0;
   DepthFirstSearch DFS = new DepthFirstSearch(g);
   for(int i=0;i<g.getNumV();i++){
      if(!DFS.isVisited(i)){
         DFS.depthFirstSearch(i);
         ret++;
      }
   }
   return ret;
}


public static int GetNumberOfComponentsBFS(Graph g){
   int ret = 0;
   BreadthFirstSearch BFS = new BreadthFirstSearch(g);
   for(int i=0;i<g.getNumV();i++){
      if(!BFS.isVisited(i)){
         BFS.breadthFirstSearch(i);
         ret++;
      }
   }
   return ret;
}
```

Which we can call them in the following way:

```
Graph g = new ListGraph(7,false);
Edge e0 = new Edge(0,1,1);
Edge e1 = new Edge(0,2,5);
Edge e2 = new Edge(1,2,3);
Edge e3 = new Edge(1,3,5);
Edge e4 = new Edge(2,3,2);
Edge e5 = new Edge(4,5,2);
Edge e6 = new Edge(5,6,2);
Edge e7 = new Edge(6,4,2);
g.insert(e0);
g.insert(e1);
g.insert(e2);
g.insert(e3);
g.insert(e4);
g.insert(e5);
g.insert(e6);
g.insert(e7);
GetNumberOfComponentsBFS(g)
GetNumberOfComponentsDFS(g)
```

ii. In this part we used the methods that we completed in the part i to measure the time of performance for each type of working, DFS or BFS. We measured the time for graphs of size 1000,2000,5000,10000 and random number of edges for 10 times for each one. Here is the method that we implemented:

```
public static void measureTime(int n) {
    int MaxEdges = n * ((n - 1) / 2);
    Random random = new Random();
    for (int i = 0; i < 10; i++) {
        int edges = i*n - 7;
        Graph g = new ListGraph(n,false);
        for(int j = 0 ; j < edges ; j++){
            int u = random.nextInt(n);
            int v = random.nextInt(n);
            Edge e = new Edge(u,v);
            g.insert(e);
        }
        long startTime = System.nanoTime();
        GetNumberOfComponentsDFS(g);
        long estimatedTime = System.nanoTime() - startTime;
        double elapsedTimeInSecond = (double) estimatedTime / 1_000_000;

        System.out.println("Time to count the number of components for a graph of size "
+ n + " vertices and " + edges + " edge using DFS is " + elapsedTimeInSecond);

        startTime = System.nanoTime();
        GetNumberOfComponentsBFS(g);
        estimatedTime = System.nanoTime() - startTime;
        elapsedTimeInSecond = (double) estimatedTime / 1_000_000;

        System.out.println("Time to count the number of components for a graph of size "
+ n + " vertices and " + edges + " edge using BFS is " + elapsedTimeInSecond);

        System.out.println("..........");
    }
    System.out.println("..........");
}
```

- **Part 3**

  In this part we had to use Dijkstra to get the number of shortest paths for each node in the graph starting from each node in the graph. We modified the parent's array to mean if we passed through vertex v or not to detect whether this path if it's shortest path contains v or not. Here is the class that we have implemented:

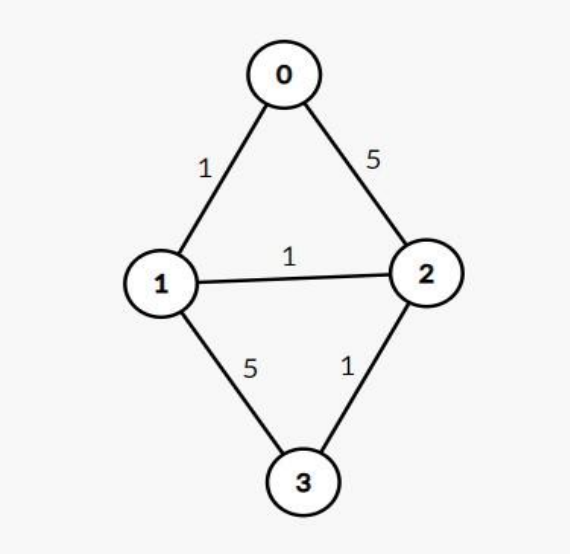
```
public class Importance {

   public double[] Importance(Graph g,int[] importance){
      double ret[] = new double[g.getNumV()];

      for(int i=0;i<g.getNumV();i++)
         ret[i] = 0;

      for(int v = 0 ; v < g.getNumV() ; v++){
         double cur = 0;
         for(int u = 0 ; u < g.getNumV() ; u++){
            if(u==v) continue;
            int[] paths = new int[g.getNumV()];
            int[] pathsUsingR = new int[g.getNumV()];
            findShortestPaths(g,u,v,g.getNumV(),paths,pathsUsingR);
            for(int w=0;w<g.getNumV();w++){
               if(w==u || w==v || paths[w] == 0) continue;
               cur+=pathsUsingR[w]/paths[w];
            }
         }
         ret[v] = cur;
      }

      return ret;
   }

public static void dijkstrasAlgorithm(Graph graph,
                                      int start,
                                      int[] paths,
                                      int[] pathsUsingR,
                                      int r,
                                      int[] par,
                                      double[] dist) {
      int numV = graph.getNumV();
      HashSet < Integer > vMinusS = new HashSet < Integer > (numV);
```

```
  for (int i = 0; i < numV; i++) {
     if (i != start) {
     vMinusS.add(i);
     }
    }
    // Initialize pred and dist.
    for (int v : vMinusS) {
     dist[v] = graph.getEdge(start, v).getWeight();
    }
    // Main loop
    while (vMinusS.size() != 0) {
     // Find the value u in V−S with the smallest dist[u].
     double minDist = Double.POSITIVE_INFINITY;
     int u = -1;
     for (int v : vMinusS) {
      if (dist[v] < minDist) {
      minDist = dist[v];
       u = v;
      }
     }
     // Remove u from vMinusS.
     vMinusS.remove(u);
     // Update the distances.
     for (int v : vMinusS) {
      if (graph.isEdge(u, v)) {
        double weight = graph.getEdge(u, v).getWeight();
        double operatorResult = 0.0;
        operatorResult = dist[u] + weight;
        if (operatorResult < dist[v]) {
        dist[v] = operatorResult;
         if(u == r || par[u] == r)
            par[v] = r;
        }
        else if(dist[v] == dist[u] + weight && par[u] == r){
           pathsUsingR[v]++;
        }
        else if(dist[v] == dist[u] + weight){
           paths[v]++;
        }
       }
     }
    }
}
```

```
   private void findShortestPaths(Graph g,int s,int r, int n,int[] paths,int[] pathsUsingR )
   {
      double[] dist = new double[n];
      int[] par = new int[n];

      for (int i = 0; i < n; i++)
         dist[i] = Integer.MAX_VALUE;

      for (int i = 0; i < n; i++){
         paths[i] = 0;
         par[i] = -1;
         pathsUsingR[i] = 0;
      }

      dijkstrasAlgorithm(g, s, paths,pathsUsingR , r , par ,dist);
   }

}
```

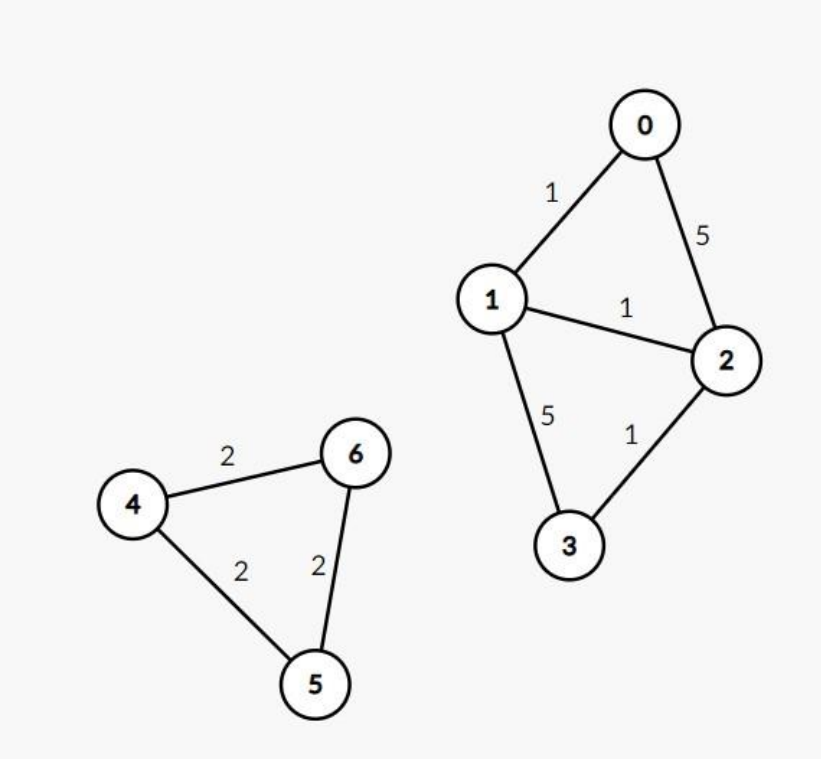
4) Test Cases

- For the part 1 we just used a simple graph of 4 vertices and 5 edges which is represented in the following way:



  Where the shortest path from 0 to 4 is 3 using 0,1,2,3.

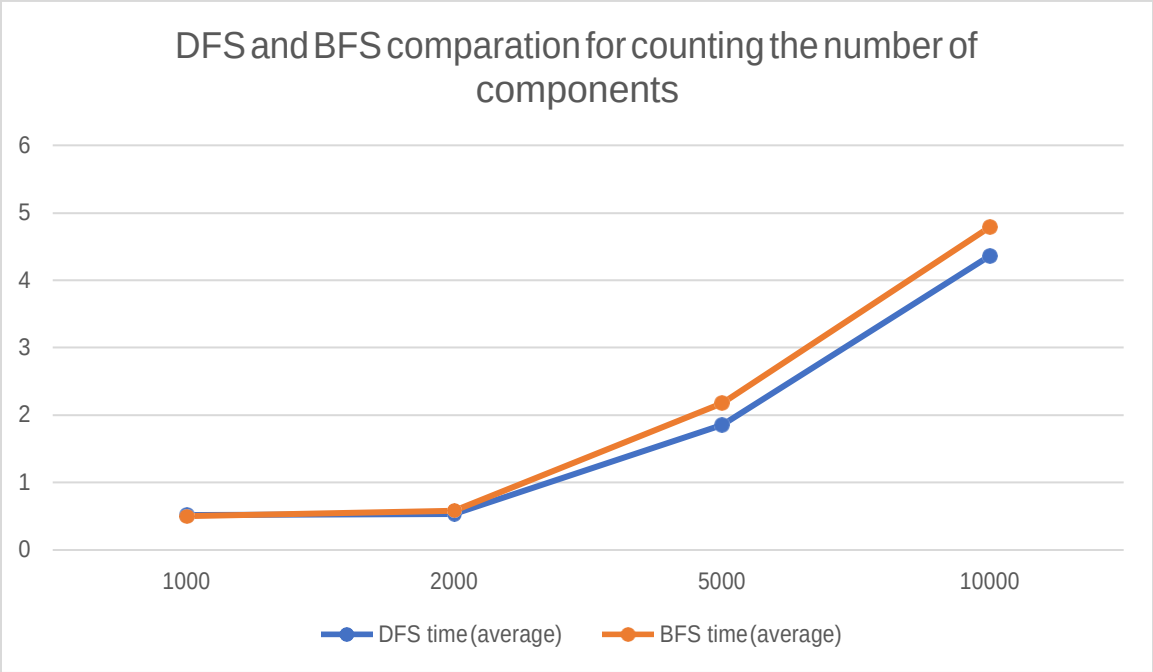
- For part 2 we used the graph below to test it:



Which the number of components is 2. We used DFS and BFS to find the result and for both we get the same result. Then we used the measure Time method that we mentioned before to get the performance and we get these values:

The number of vertices	DFS time (average)	BFS time (average)
1000	0.5185	0.5033900000000001
2000	0.53936	0.5854000000000001
5000	1.8593	2.17892
10000	4.36173	4.7894499999999995

Which will result the following graphs:



We can notice that BFS gives faster results than DFS.

- For part 3 we can use the same sample that we used for the first part which will give us the result after calculating the importance depending on the given formula.

5) Running and Results

```
Here Is the results for part 1:
The distances array using Addition operator is:
0.0
1.0
4.0
6.0
The distances array using Multiplication operator is:
0.0
1.0
3.0
5.0
The distances array using * operator is:
0.0
1.0
1.0
1.0
Here Is the results for part 2.i:
The number of components using DFS1
The number of components using BFS1
Here Is the results for part 2.ii:
Time to count the number of components for a graph of size 1000 vertices and -7 edge using DFS is 0.4257
Time to count the number of components for a graph of size 1000 vertices and -7 edge using BFS is 0.9389
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 993 edge using DFS is 0.4521
Time to count the number of components for a graph of size 1000 vertices and 993 edge using BFS is 0.8147
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 1993 edge using DFS is 0.6021
Time to count the number of components for a graph of size 1000 vertices and 1993 edge using BFS is 0.9073
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 2993 edge using DFS is 1.0063
Time to count the number of components for a graph of size 1000 vertices and 2993 edge using BFS is 0.4579
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 3993 edge using DFS is 0.4813
Time to count the number of components for a graph of size 1000 vertices and 3993 edge using BFS is 0.458
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 4993 edge using DFS is 0.3847
Time to count the number of components for a graph of size 1000 vertices and 4993 edge using BFS is 1.0803
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 5993 edge using DFS is 0.5137
Time to count the number of components for a graph of size 1000 vertices and 5993 edge using BFS is 0.4522
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 6993 edge using DFS is 0.5262
Time to count the number of components for a graph of size 1000 vertices and 6993 edge using BFS is 0.4488
----------------------------------------
 Time to count the number of components for a graph of size 1000 vertices and 7993 edge using DFS is 0.6158
Time to count the number of components for a graph of size 1000 vertices and 7993 edge using BFS is 0.4425
----------------------------------------
Time to count the number of components for a graph of size 1000 vertices and 8993 edge using DFS is 1.4393
Time to count the number of components for a graph of size 1000 vertices and 8993 edge using BFS is 0.8988
----------------------------------------
```

Time to count the number of components for a graph of size 2000 vertices and -7 edge using DFS is 0.1768
Time to count the number of components for a graph of size 2000 vertices and -7 edge using BFS is 0.3122

Time to count the number of components for a graph of size 2000 vertices and 1993 edge using DFS is 0.4848
Time to count the number of components for a graph of size 2000 vertices and 1993 edge using BFS is 0.6704

Time to count the number of components for a graph of size 2000 vertices and 3993 edge using DFS is 0.5271
Time to count the number of components for a graph of size 2000 vertices and 3993 edge using BFS is 0.3838

Time to count the number of components for a graph of size 2000 vertices and 5993 edge using DFS is 0.3342
Time to count the number of components for a graph of size 2000 vertices and 5993 edge using BFS is 0.3534

Time to count the number of components for a graph of size 2000 vertices and 7993 edge using DFS is 0.4121
Time to count the number of components for a graph of size 2000 vertices and 7993 edge using BFS is 0.48

Time to count the number of components for a graph of size 2000 vertices and 9993 edge using DFS is 0.4891
Time to count the number of components for a graph of size 2000 vertices and 9993 edge using BFS is 0.619

Time to count the number of components for a graph of size 2000 vertices and 11993 edge using DFS is 0.7487
Time to count the number of components for a graph of size 2000 vertices and 11993 edge using BFS is 0.6787

Time to count the number of components for a graph of size 2000 vertices and 13993 edge using DFS is 0.6946
Time to count the number of components for a graph of size 2000 vertices and 13993 edge using BFSis 0.6022

Time to count the number of components for a graph of size 2000 vertices and 15993 edge using DFS is 0.7188
Time to count the number of components for a graph of size 2000 vertices and 15993 edge using BFS is 0.7401

Time to count the number of components for a graph of size 2000 vertices and 17993 edge using DFS is 0.9239
Time to count the number of components for a graph of size 2000 vertices and 17993 edge using BFS is 0.8663

Time to count the number of components for a graph of size 5000 vertices and -7 edge using DFS is 0.3252
Time to count the number of components for a graph of size 5000 vertices and -7 edge using BFS is 0.3999

Time to count the number of components for a graph of size 5000 vertices and 4993 edge using DFS is 0.6337
Time to count the number of components for a graph of size 5000 vertices and 4993 edge using BFS is 0.7273

Time to count the number of components for a graph of size 5000 vertices and 9993 edge using DFS is 0.9182
Time to count the number of components for a graph of size 5000 vertices and 9993 edge using BFS is 0.8994

Time to count the number of components for a graph of size 5000 vertices and 14993 edge using DFS is 1.6951
Time to count the number of components for a graph of size 5000 vertices and 14993 edge using BFS is 3.4051

Time to count the number of components for a graph of size 5000 vertices and 19993 edge using DFS is 1.0149
Time to count the number of components for a graph of size 5000 vertices and 19993 edge using BFS is 1.1635

Time to count the number of components for a graph of size 5000 vertices and 24993 edge using DFS is 1.7116
Time to count the number of components for a graph of size 5000 vertices and 24993 edge using BFS is 2.6177

Time to count the number of components for a graph of size 5000 vertices and 29993 edge using DFS is 3.9584
Time to count the number of components for a graph of size 5000 vertices and 29993 edge using BFS is 5.2031

Time to count the number of components for a graph of size 5000 vertices and 34993 edge using DFS is 3.9958
Time to count the number of components for a graph of size 5000 vertices and 34993 edge using BFS is 2.8858

```
Time to count the number of components for a graph of size 5000 vertices and 39993 edge using DFS is 3.3317
Time to count the number of components for a graph of size 5000 vertices and 39993 edge using BFS is 5.6068
---------------------------------------
Time to count the number of components for a graph of size 5000 vertices and 44993 edge using DFS is 2.2634
Time to count the number of components for a graph of size 5000 vertices and 44993 edge using BFS is 2.9709
---------------------------------------
-------------------------------------------------------------------------------
Time to count the number of components for a graph of size 10000 vertices and -7 edge using DFS is 0.0779
Time to count the number of components for a graph of size 10000 vertices and -7 edge using BFS is 0.1233
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 9993 edge using DFS is 0.6065
Time to count the number of components for a graph of size 10000 vertices and 9993 edge using BFS is 0.652
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 19993 edge using DFS is 1.7967
Time to count the number of components for a graph of size 10000 vertices and 19993 edge using BFS is 1.2715
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 29993 edge using DFS is 1.4571
Time to count the number of components for a graph of size 10000 vertices and 29993 edge using BFS is 1.5904
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 39993 edge using DFS is 2.025
Time to count the number of components for a graph of size 10000 vertices and 39993 edge using BFS is 3.4174
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 49993 edge using DFS is 3.3654
Time to count the number of components for a graph of size 10000 vertices and 49993 edge using BFS is 3.9914
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 59993 edge using DFS is 6.3032
Time to count the number of components for a graph of size 10000 vertices and 59993 edge using BFS is 5.5838
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 69993 edge using DFS is 6.4109
Time to count the number of components for a graph of size 10000 vertices and 69993 edge using BFS is 8.922
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 79993 edge using DFS is 8.0046
Time to count the number of components for a graph of size 10000 vertices and 79993 edge using BFS is 10.5387
---------------------------------------
Time to count the number of components for a graph of size 10000 vertices and 89993 edge using DFS is 9.6633
Time to count the number of components for a graph of size 10000 vertices and 89993 edge using BFS is 11.7587
---------------------------------------
-------------------------------------------------------------------------------
BUILD SUCCESSFUL (total time: 0 seconds)
```