Lab 4
Shortest Paths Algorithms

Name : Ahmed Khaled

ID : 9

-Bellman-Ford :

```
Bellman_Ford( ArrayList<ArrayList<Edge>> g ,int s , int t)
 {
      init(g.size(),s);
      boolean relaxation = true ;

      for (int i = 0;  i < g.size()-1 && relaxation ; i++) {

           relaxation = false ;
           for (int j = 0; j < g.size(); j++) {
                for (Edge e : g.get(j)) {

                     if(dist[e.to] > dist[e.from] + e.w)
                     {
                          dist[e.to] = dist[e.from] + e.w;
                          parent[e.to] = e.from;
                          relaxation = true ;
                     }
                }

           }
      }
      for (int j = 0; j < g.size(); j++) {
           for (Edge e : g.get(j)) {

                if(dist[e.to] > dist[e.from] + e.w)
                {
                     return  null ;
                }
           }

      }
      return dist[t];
 }
```

-Algorithm description :

-Initialize the distance array from the source to all other nodes with infinity.

-Try to relax the nodes v-1 times using all edges , if there isn't any relaxation in any iteration the algorithm will terminate and return the shortest path value .

-Try to relax using all edges one more time , The occurrence of any relaxation indicates that there is a negative cycle .

–Time complexity:

- Time complexity of the algorithm = O(|V|*|E|) .

-Bellman-Ford Moore :

```
    BellmanFord_moore( ArrayList<ArrayList<Edge>> g ,int s , intt)
    {
        init(g.size(),s);
        Queue<Integer> queue = new LinkedList<Integer>();
        queue.add(s);

        while(!queue.isEmpty())
        {
            int node = queue.poll();
            marked[node] = false ;
            for (Edge e : g.get(node))
            {
                if(dist[e.to] > dist[e.from]+e.w)
                {
                    dist[e.to] = dist[e.from]+e.w ;
                    parent[e.to] = e.from;
                    if(!marked[e.to])
                    {
                        queue.add(e.to);
                        marked[e.to] = true ;
                    }
                }

            }
        }

        return dist[t];
}
```

-Algorithm description :

-Initialize the distance array from the source to all other nodes with infinity , and a Boolean array with false.

-Start with the source node and add it to a queue .

-Pop a node from the queue and try to relax all its adjacent nodes ,If any node has been relaxed , add it to a queue if it wasn't added before and mark it .

- Repeat the previous step until the queue become empty .

-The algorithm will try to relax the nodes that are adjacent to the nodes have been relaxed in a previous iteration .

–Time complexity:

- Time complexity of the algorithm = $O(|V|*|E|)$.

-Dijkstra:

```
dijkstra(ArrayList<ArrayList<Edge>> g ,int s , int t)
	{
		init(g.size(),s);
		PriorityQueue<Node> pq = new PriorityQueue<Node>();
		pq.add(new Node(s,0));


		Node temp ;
		while(!pq.isEmpty())
		{
			temp = pq.poll();

			if(!marked[temp.i])
			{
				marked[temp.i] = true ;
				for (Edge e : g.get(temp.i)) {

					if(dist[e.to]>dist[temp.i]+e.w)
					{
						dist[e.to]=dist[temp.i]+e.w ;
						parent[e.to] = temp.i ;
						pq.add(new Node(e.to,dist[e.to]));
					}
				}
			}

		}
		return dist[t] ;
}
```

-Algorithm description :

-Initialize the distance array from the source to all other nodes with infinity , a Boolean array with false and a binary heap .

-Add the source node to the binary heap .

- Extract the minimum node from the heap try to relax all its adjacent nodes , If any node has been relaxed , add it to the heap with its new distance from the source (decreasing its value in the heap) .

- Repeat the previous step until the heap become empty .

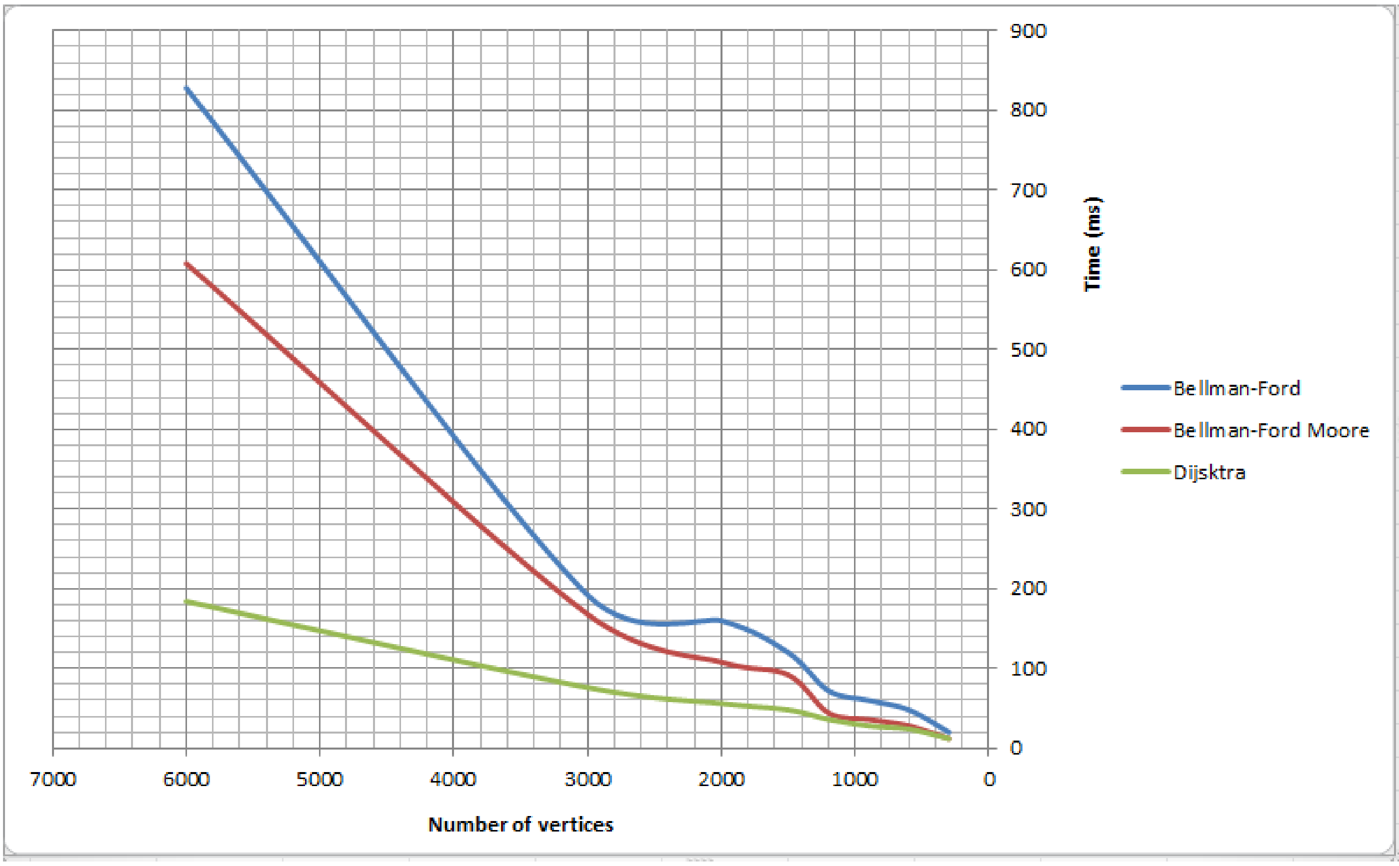
-A single node may be relaxed more than once , so it will add to the heap more than once with different values , so after extract the minimum value of this node the algorithm need to mark that node as scanned with its last and minimum distance to the source , and the algorithm will check each node polled from the heap to be unmarked before trying to relax its adjacent nodes .

–Time complexity:

- Time complexity of the algorithm = $O((|V|+|E|) * \text{Log } |V|)$.

-Running time comparison:

- Number of vertices vs running time:



- Number of edges vs running time :