Shortest Path Algorithms

1. **BFS**
   Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

   Algorithm

```
1  procedure BFS(G, root) is
2      let Q be a queue
3      label root as explored
4      Q.enqueue(root)
5      while Q is not empty do
6          v := Q.dequeue()
7          if v is the goal then
8              return v
9          for all edges from v to w in G.adjacentEdges(v) do
10             if w is not labeled as explored then
11                 label w as explored
12                 Q.enqueue(w)
```

   Time Complexity
   The time complexity can be expressed as **O(|V|+|E|)**, since every vertex and every edge will be explored in the worst case. **|V|** is the number of vertices and **|E|** is the number of edges in the graph. Note that **O(|E|)** may vary between **O(1)** and **O(|V|^{2})**, depending on how sparse the input graph is.

2. **Bellman-Ford**
   The Bellman-Ford algorithm is an algorithm that computes the shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.

   Algorithm

   - The outer loop traverses from 0 to n - 1.
   - Loop over all edges, check if the next node distance > current node distance + edge weight, in this case, update the next node distance to "current node distance + edge weight".

   Input: Graph and a source vertex src
   Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, a negative weight cycle is reported.

1) This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is the source vertex.
2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
…..a) Do following for each edge u-v
……If dist[v] > dist[u] + weight of edge uv, then update dist[v]
…….dist[v] = dist[u] + weight of edge uv
3) This step reports if there is a negative weight cycle in the graph. Do the following for each edge u-v
……If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”
The idea of step 3 is, that step 2 guarantees the shortest distances if the graph doesn’t contain a negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

Time Complexity
Bellman-Ford runs in **O(|V|*|E|)** time, where **|V|** and **|E|** are the number of vertices and edges respectively.

3. **Dijkstra**

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds the shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.

Algorithm

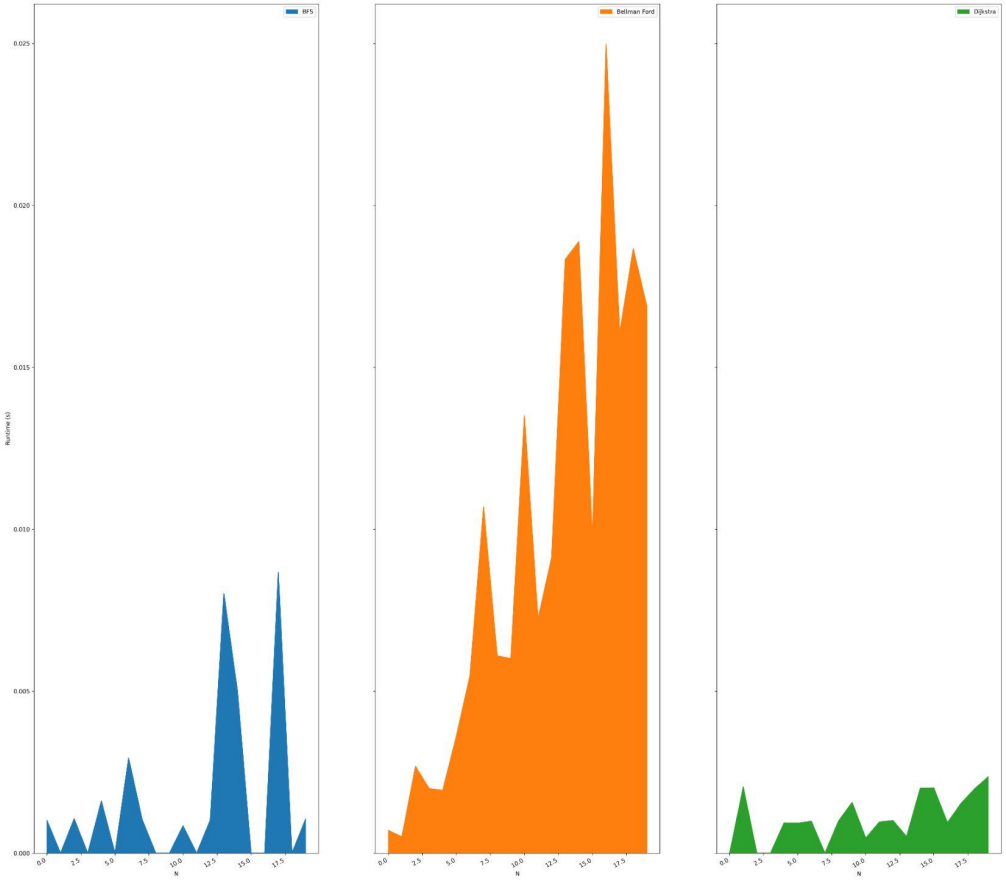
- Set all vertices distances = infinity except for the source vertex, set the source distance = 0.
- Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to vertices distances.
- Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).
- Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push the vertex with the new distance to the priority queue.
- If the popped vertex is visited before, just continue without using it.

- Apply the same algorithm again until the priority queue is empty.

Time Complexity

Time Complexity of Dijkstra's Algorithm is $O(V^2)$ but with min-priority queue it drops down to $O(V + ElogV)$

Analysis while increasing n



We can observe that Dijkstra performs better compared to Bellman-Ford whose runtime increases significantly due to its high time complexity.