Bellman Ford

The Dijkstra algorithm fails to produce optimal results for negative weights. To determine a single source shortest path SSSP for negative weights graph the “Bellman ford” is the best algorithm that uses dynamic programming approach.

Each node is assigned an attribute “**cost**” which will provide you the minimum distance from source vertex and “π ***or PI***” representing the predecessor.
Input parameters: A graph **G(V,E)** in the form of adjacency list, A matrix **“w”** containing the weight of all the edges of graph, and “**src**” source vertex.

```
Bellman Ford (G, w, src)
 {
   Initialize (G, src)  //This function initialize the attributes “cost” and “π" of each vertex of graph
   For (i=1 to |G.V|)   //|G.V| will provide the total number of vertices in the graph
   {
// Iterate for all the edges, the order in which they are selected is similar to the order in which
adjacency list is maintained.
      for each edge (u, v) ε G.E    // iterate for all the edges of the graph.
         Relax (u, v, w)    // function call to check whether the edge between vertices “u” and “v” will
                            // minimize the distance to vertex “v” from “src” or not.
   }
}
Relax (u, v, w){
   if (v. cost > u. cost + w (u, v))
   {
      v. cost = u. cost + w (u, v)
      v. π = u
   }
}
Initialize (G, src)
{
   for each "v" belongs to G.V
      v. cost = infinity
      v. π = NULL

   src. cost = 0
}
```

Time complexity: *O*(V*E)
E = V*(V-1) = V^2
Overall time complexity: ***O(V^3)***

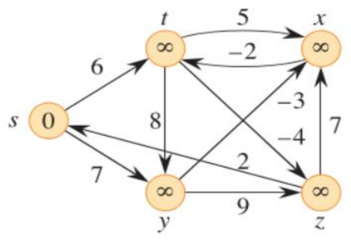
A minor improvement and negative cycle detection.
If any edge in the graph is relaxed, it requires repeating the edge relaxation process because there is a high likelihood that this relaxation will trigger the relaxation of additional edges in subsequential iterations. If, during any iteration of the outer loop, no edge relaxation occurs, there’s no need to repeat the process. If any of the edges relaxed even after the termination of outer loop, then it means that there is a negative cycle in the graph.

```
// Here is the improved version. The highlighted statements represent the changes.
Bellman Ford (G, w, src)
{
   Initialize (G, src)

   flag = true   // use a Boolean variable initialized with true to enter the loop in the 1st iteration
// we will iterate either for total number of vertices or until the flag is true.
   for (i=1 to |G.V| AND flag == true)
   {
Initialize the flag with false before entering the inner loop. If it remains false after all the inner loop
iterations, it indicates no edge in the graph was relaxed, thus no further iterations are required.
      flag = false
      for each edge (u,v) G.E
         flag = flag || Relax(u,v,w)   // we are using OR operator so that if any of the edge gets relaxed
                                        //  the flag will remain true.
   }

// To determine whether there exists a negative cycle in the graph, we need to iterate all the edges
and if any of the edge gets relaxed then it means that there is a negative cycle.
   for each edge (u,v) belongs to G.E
      if (v.cost > u.cost + w(u,v))
         // Negative cycle detected in graph
}
Relax (u, v, w)
{
   If (v.cost > u.cost + w(u,v))
   {
      v.cost = u.cost + w(u,v)
      v.PI = u
      return true
   }
   return false
}
```

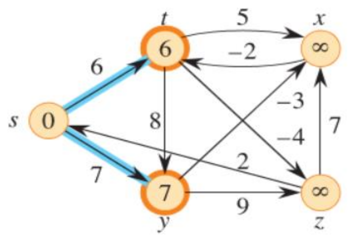
Sample Example:

Blue edges in the following image represent the predecessors. In this particular example, each pass relaxes the edges in the order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

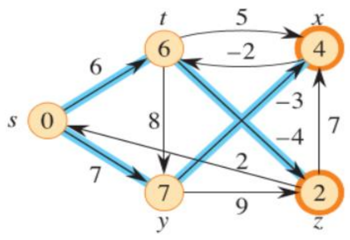
Fig-(a) The situation just before the first pass over the edges. **Figures (b)–(e)** The situation after each successive pass over the edges. Vertices whose shortest path estimates and predecessors have changed due to a pass are highlighted in orange. The edges (s, t) and (s, y) will be relaxed in the first pass. The edges (y, x) and (t, z) will be relaxed in the second pass. The edge (x, t) will be relaxed in third pass and the edge (t, z) will be relaxed again in the fourth pass. Then there will be no relaxation in any of the edges in the fifth pass. The process will be terminated after the fifth pass and there are a total of five vertices in the graph. So, there are total of "v" passes and in each pass there are "E" iterations.



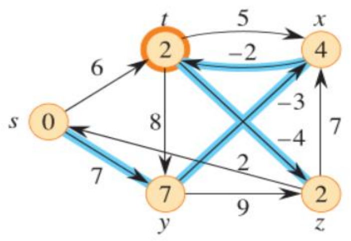
(a)



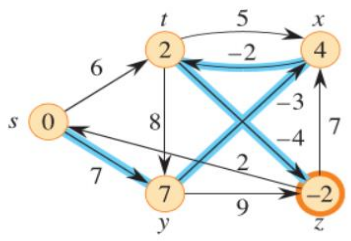
(b)



(c)



(d)



(e)