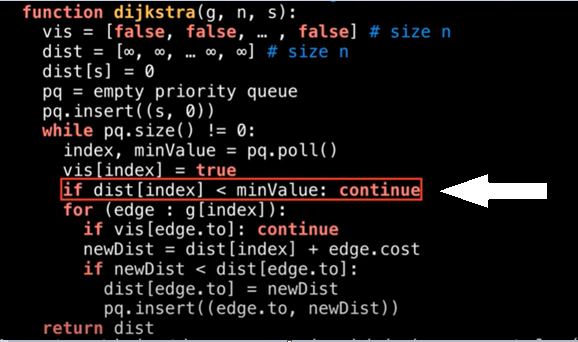
**Dijkstra algorithm** is a Single Source Shortest Path algorithm for graphs with non-negative edge weights.

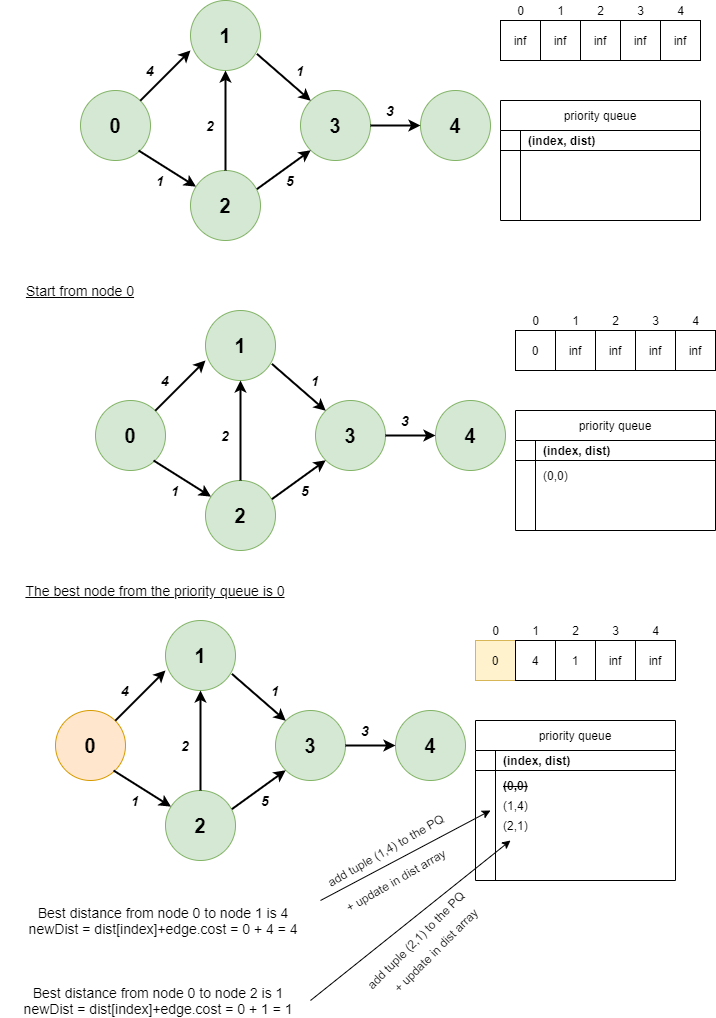
Single Source = find shortest paths from a given ***source*** vertex s to every other vertex in the graph.

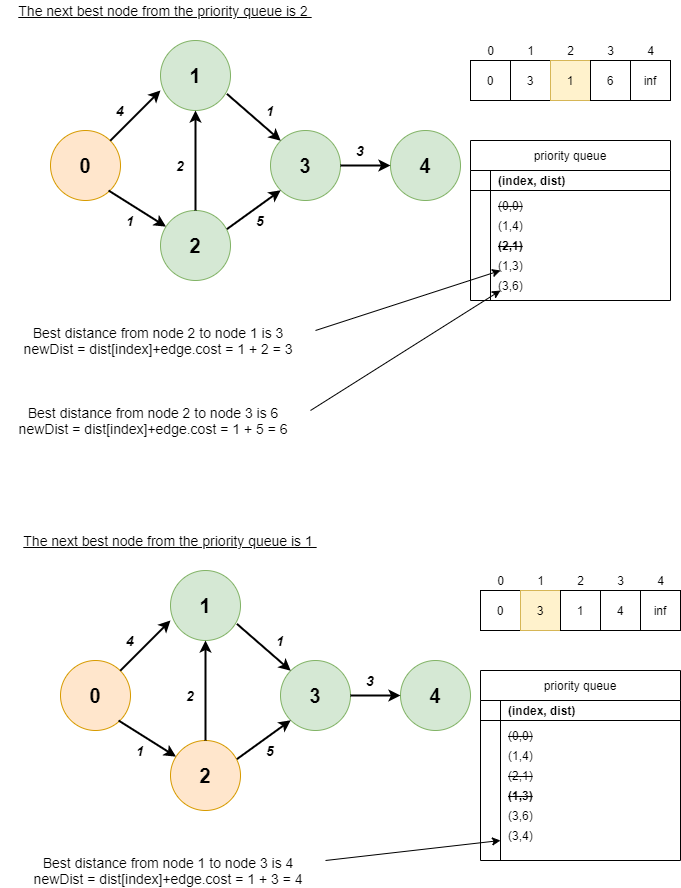
 the priority queue provided by the Java library (and other languages also) does not support the decrease-key operation => skip all nodes that are worse performing *(minValue > dist[index])*

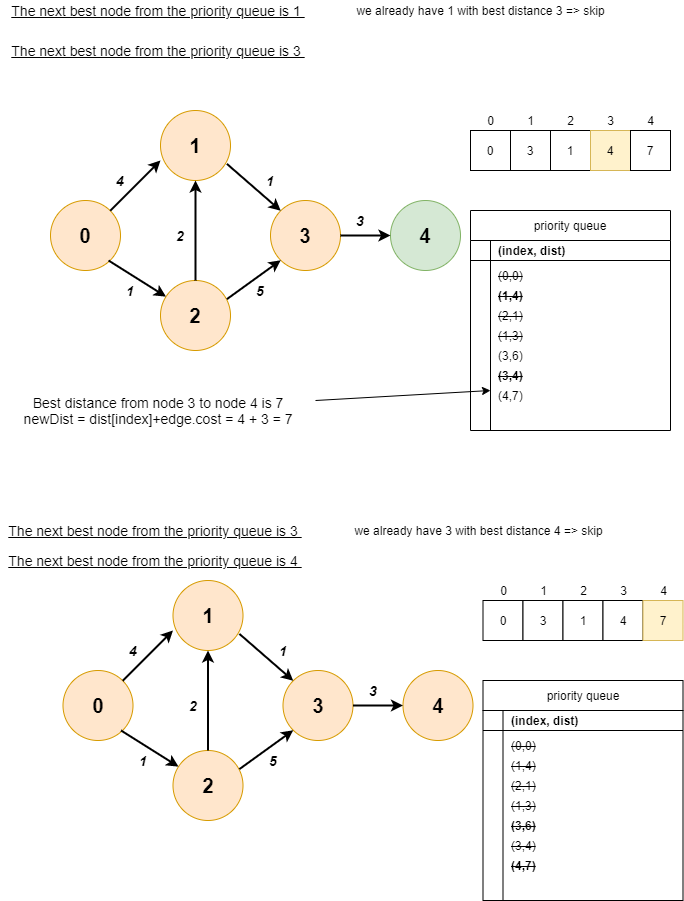
*Decrease-key operation =* ***decrease****the* ***value*** *of a certain****key****inside the data structure*

In the lecture you have an example with the decrease-key operation but in practice is not so common for a PriorityQueue to have this type of operation.

Example:

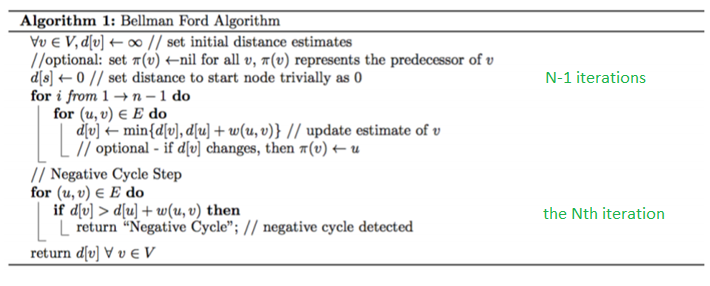






**Bellman-Ford algorithm** is a Single Source Shortest Path algorithm for graphs that **allows** non-negative edge weights (but no negative cycles).

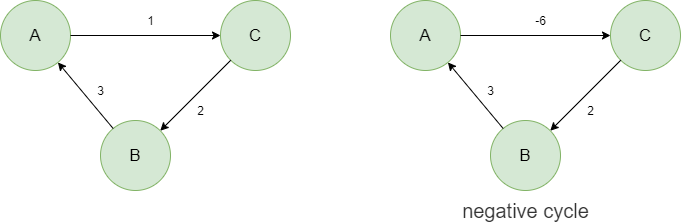
* The first for loop relaxes each of the edges in the graph n − 1 times. We claim that after n − 1 iterations, the distances are guaranteed to be correct.
* The second for loop in this algorithm detects negative cycles.



**Why perform the Nth loop ?**

After completing iteration (N-1) we get the shortest distance. Now in order to check if there is no negative cycle we have to perform the Nth iteration.

If there is **no change** in the value of distance array d[ ] in the Nth loop then the graph *has no negative cycle* which means we can find the shortest path. Otherwise the graph has negative cycle and we cannot find the shortest path.



**Example**

