Assignment: Comparison of Prim and Dijkstra's Algorithm

Low level Pseudo code:

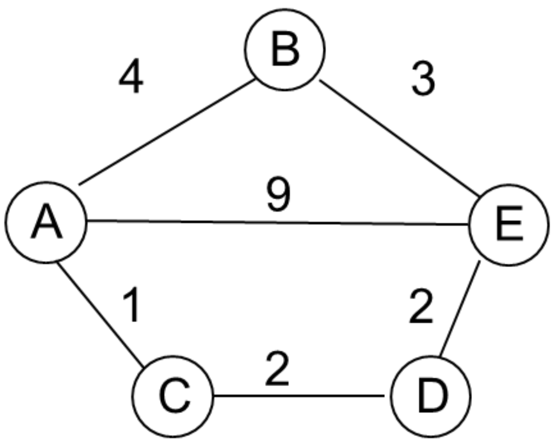
```
Algorithm: Prim-MST (G)
Input: Graph G=(V,E) with edge-weights.


   // Initialize priorities and place in priority
queue.
1.   priority[i] = infinity for each vertex i
2.   Insert vertices and priorities into
priorityQueue;
  // Set the priority of vertex 0 to 0.
3.   priorityQueue.decreaseKey (0, 0)  //

// Process vertices one by one in order of priority

4.   while priorityQueue.notEmpty()
         // Get "best" vertex out of queue.
5.       v = priorityQueue.extractMin()
6.       Add v to MST;
               // Explore edges from v.
7.       for each edge e=(v, u) in adjList[v]
8.           w = weight of edge e=(v, u);

// If it's shorter to get to MST via v, then update.
9.           if priority[u] > w
10.               priorityQueue.decreaseKey (u, w)
11.              predecessor[u] = v
12.          endif
13.      endfor
14.  endwhile

15.  Build MST;
16.  return MST

Output: A minimum spanning tree of the graph G.
```

Source: pseudo code modified by tjk
http://www.seas.gwu.edu/~simhaweb/alg/lectures/module8/module8.html

Low level Pseudo code:

```
Algorithm: Dijkstra-SPT (G, s)
Input: Graph G=(V,E) with non-negative edge weights
and designated source vertex s.

   // Initialize priorities and place in priority
queue.
1. priority[i] = infinity for each vertex i;
2. Insert vertices and priorities into priorityQueue;
   // Source s has priority 0
3. priorityQueue.decreaseKey (s, 0)

// Process vertices one by one in order of priority

4. while priorityQueue.notEmpty()
       // Get "best" vertex out of queue.
5.     v = priorityQueue.extractMin()
6.     Add v to SPT;
           // Explore edges from v.
7.     for each edge e=(v, u) in adjList[v]
8.       w = weight of edge e=(v, u);

// If it's shorter to get to u from s via v, update.
9.      if priority[u] > priority[v] + w
10.          priorityQueue.decreaseKey (u,
priority[v]+w)
11.          predecessor[u] = v
12.        endif
13.    endfor
14. endwhile

15. Build SPT;
16. return SPT

Output: Shortest Path Tree (SPT) rooted at s.
```

Assignment: Comparison of Prim and Dijkstra's Algorithm



PRIM'S ALGORITHM (prev, dist)

MST	A	B	C	D	E
INIT/SetStart	(–, 0)	(–, ∞)	(–, ∞)	(–, ∞)	(–, ∞)
A	(–, 0)	(A, 4)	(A, 1)	(–, ∞)	(A, 9)
C	(–, 0)	(A, 4)	(A, 1)	(C, 2)	(A, 9)
D	(–, 0)	(A, 4)	(A, 1)	(C, 2)	(D, 2)
E	(–, 0)	(E, 3)	(A, 1)	(C, 2)	(D, 2)
B	(–, 0)	(E, 3)	(A, 1)	(C, 2)	(D, 2)

DIJKSTRA'S ALGORITHM (prev, dist)

MST	A	B	C	D	E
INIT/ SetStart	(–, 0)	(–, ∞)	(–, ∞)	(–, ∞)	(–, ∞)
A	(–, 0)	(A, 4)	(A, 1)	(–, ∞)	(A, 9)
C	(–, 0)	(A, 4)	(A, 1)	(C, 3)	(A, 9)
D	(–, 0)	(A, 4)	(A, 1)	(C, 3)	(D, 5)
B	(–, 0)	(A, 4)	(A, 1)	(C, 3)	(D, 5)
E	(–, 0)	(A, 4)	(A, 1)	(C, 3)	(D, 5)