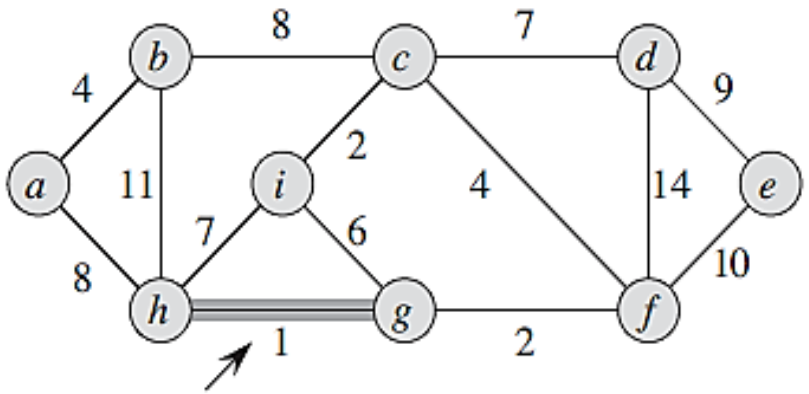
Lecture

Kruskal’s algorithm with Disjoint sets

Single Source Shortest Paths: Dijkstra's Algorithm

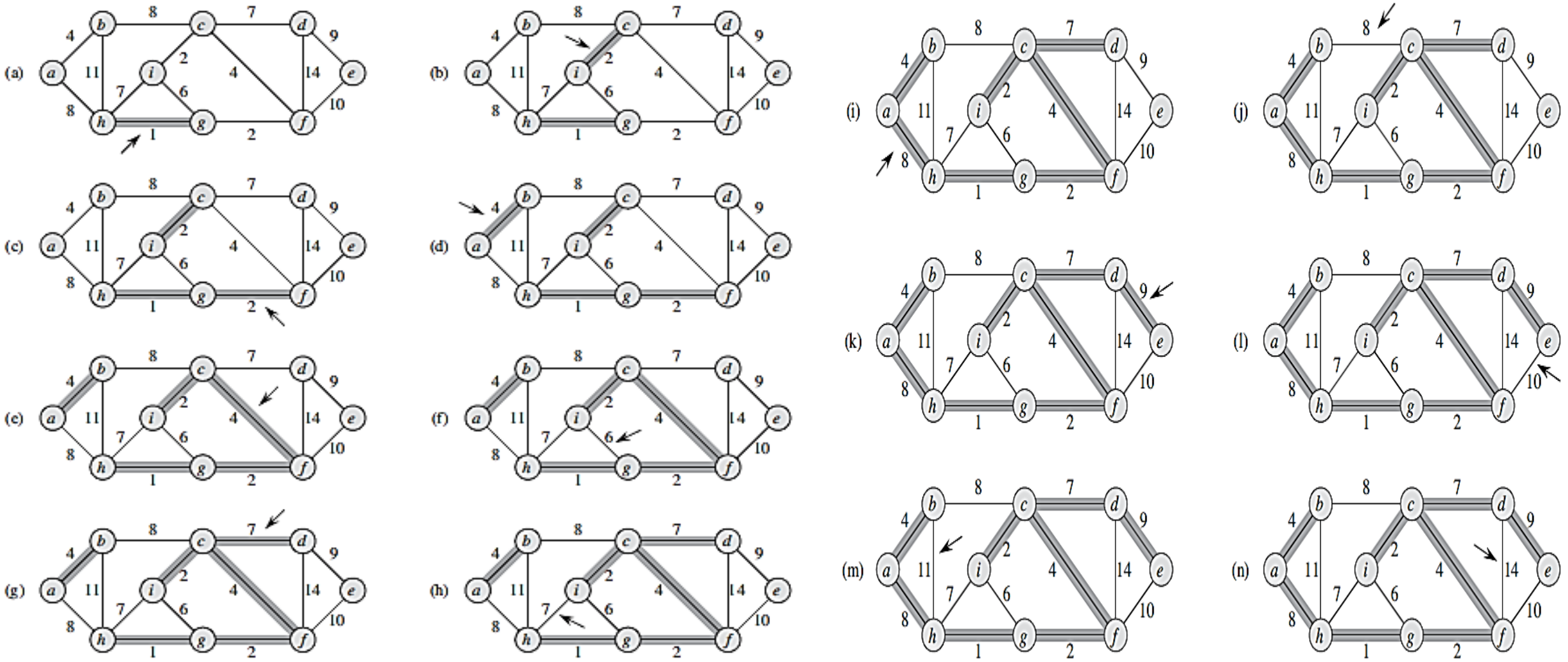
Minimum Cost Spanning Tree: Kruskal's algorithm

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weigh
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```



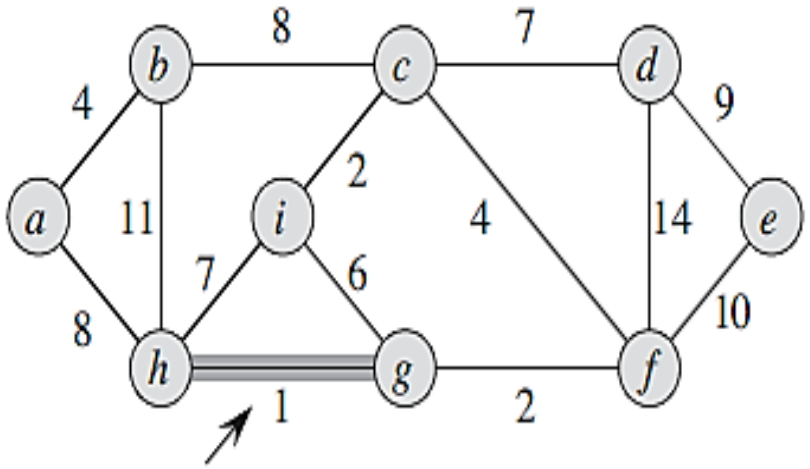
- ✓ Kruskal's algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of least weight.
- ✓ It uses a disjoint-set data structure to maintain several disjoint sets of elements.
- ✓ Each set contains the vertices in one tree of the current forest.
- ✓ The operation FIND-SET (u) returns a representative element from the set that contains u. It helps in determining whether two vertices u and v belong to the same tree by testing whether FIND-SET (u) equals FIND-SET (v).
- ✓ To combine trees, Kruskal's algorithm calls the UNION procedure.

Kruskal's algorithm: Process



Kruskal's algorithm: Example

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```



Kruskal's algorithm: Complexity Analysis

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

Time Complexity = O(ElogV)

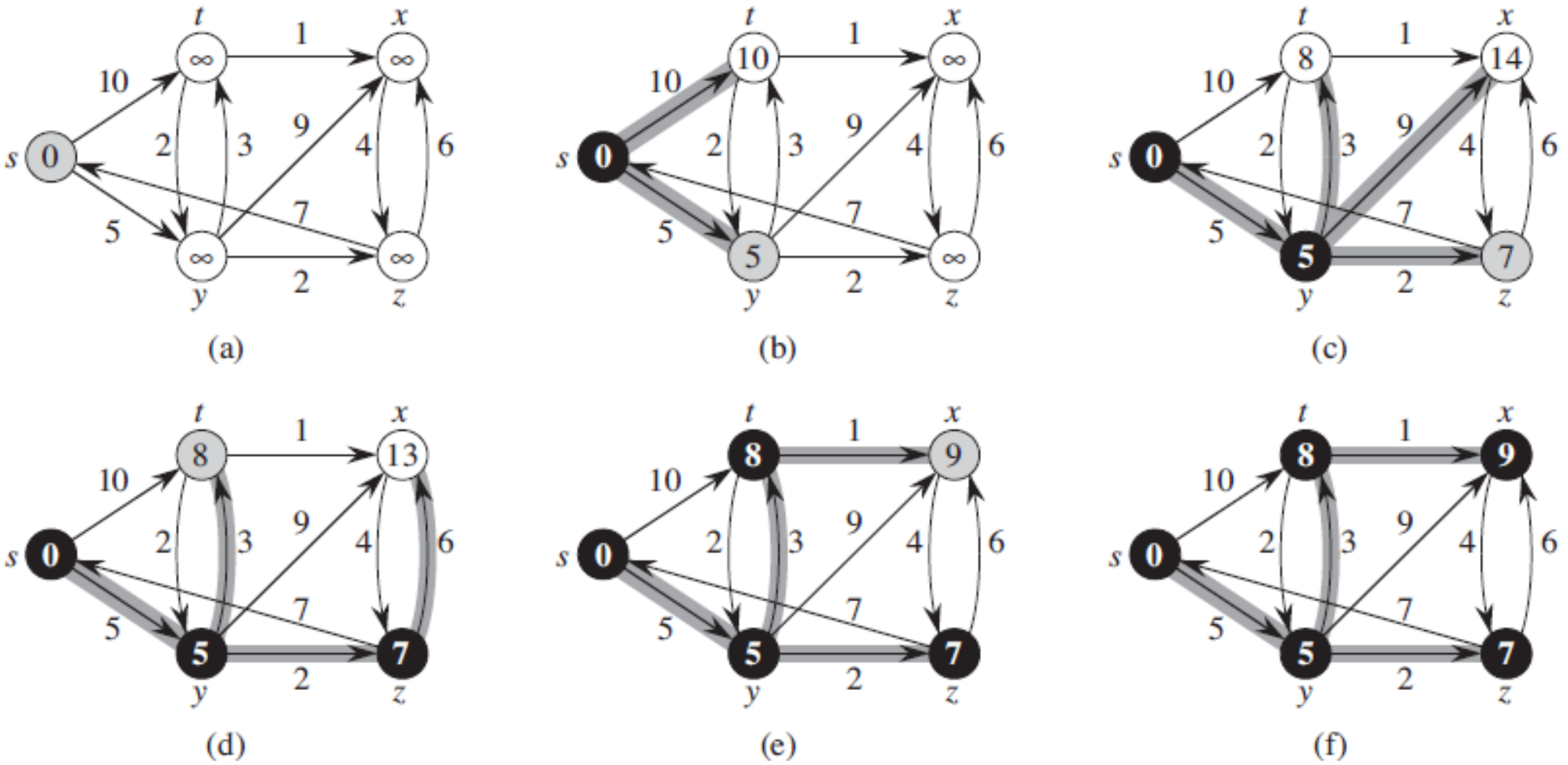
Single Source Shortest Path

Single Source Shortest Path: Dijkstra's Algorithm

- ✓ Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph G = (V, E) for the case in which all edge weights are nonnegative (w (u,v) ≥ 0 for each edge (u, v) ∈ E).
- ✓ Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.
- ✓ The algorithm repeatedly selects the vertex u ∈ V - S with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u.
- ✓ For implementation, we use a min-priority queue Q of vertices, keyed by their d values.

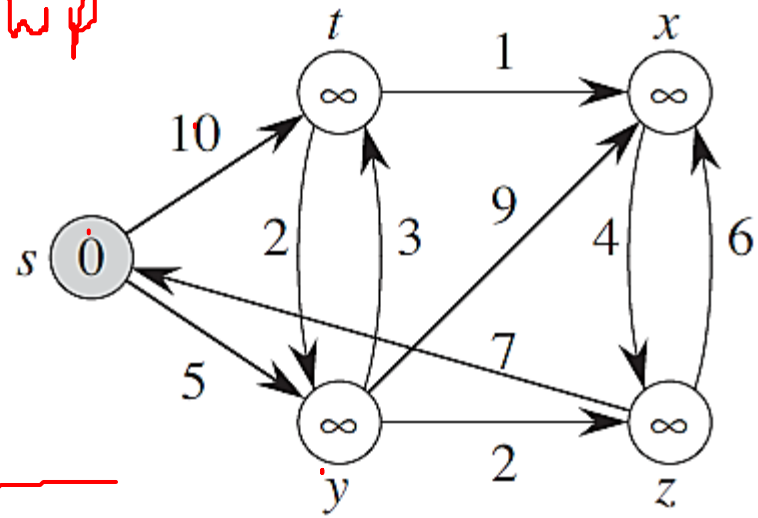
```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

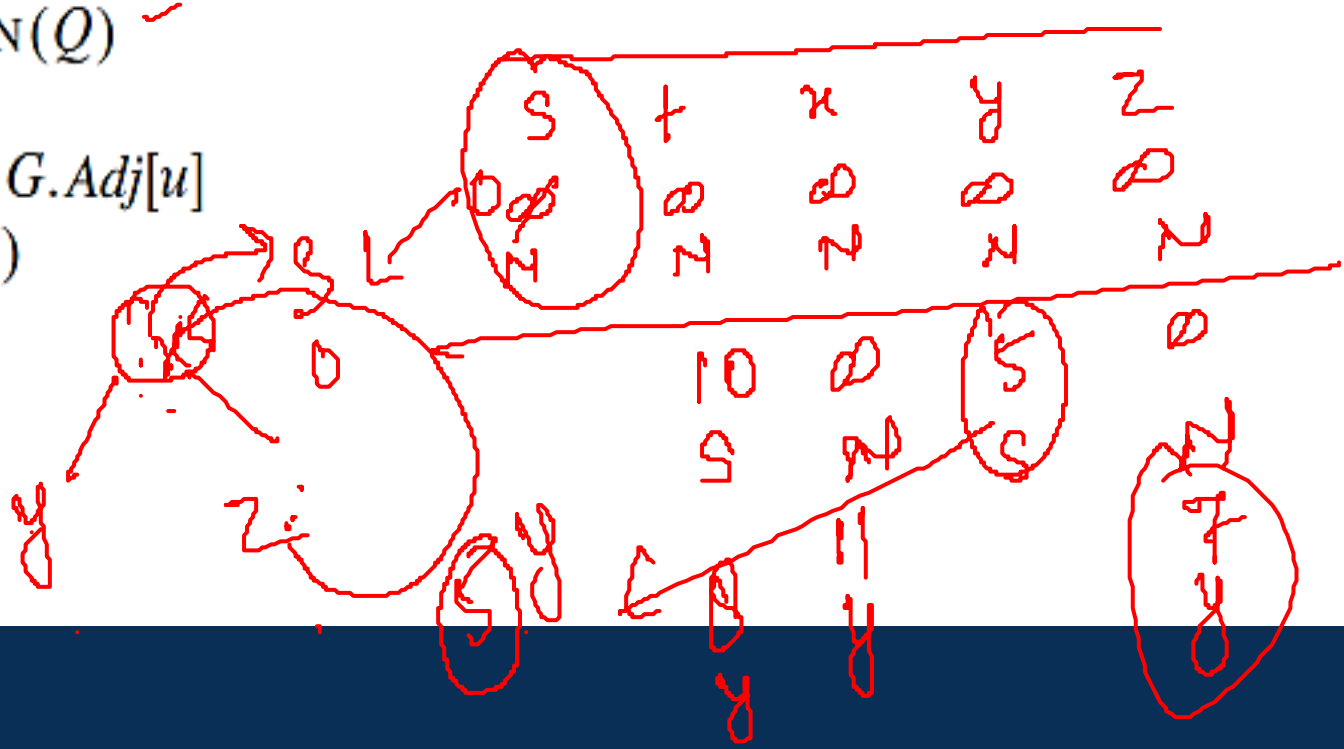
Dijkstra's Algorithm: Process



Dijkstra's Algorithm: Example

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```





Dijkstra's Algorithm: Complexity Analysis

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

Time Complexity:
Using Binary Min Heap: $O[(V + E)logV] = O(ELogV)$
Using Fibonacci Min Heap: $O[E + VlogV]$

Thank You