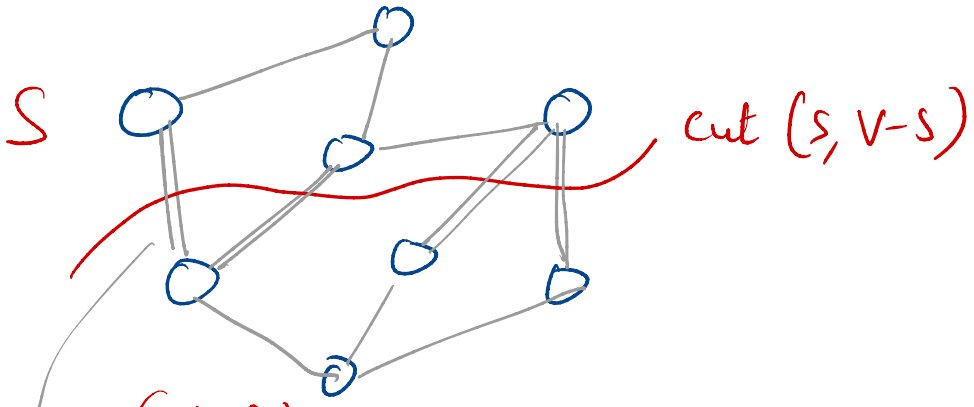
Graphs - IV

Minimum Spanning Tree

Definition: A **cut** $(S, V-S)$ of an undirected graph $G = (V, E)$ is a partition of V. An edge $(u, v) \in E$ **crosses** the cut $(S, V-S)$ if one of its end-point is in S and the other in $(V-S)$



→ A cut **respects** a set A of edges if no edge in A crosses the cut.

An edge is a **light edge** crossing a cut if its weight is minimum of any edge crossing the cut.

Kruskal's algorithm

- 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
- Find-Set(u) returns a representative element from the set that contains u.
  - a function
  - Use it to check if two vertics belong to the same tree.
- The union procedure combines trees.

MST-Kruskal(G, w)

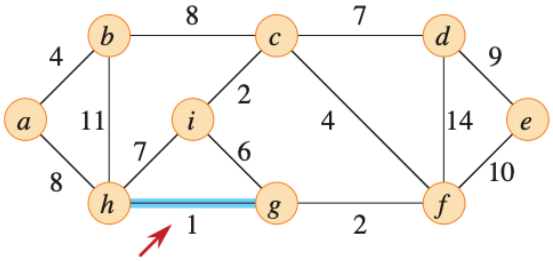
```
A = ∅  // set of edges that represent MST
For each vertex v ∈ G.V
    Make-Set(v)
Create a single list of edges in G.E
Sort the list of edges into monotonically
    increasing order by the weight w
For each edge (u,v) taken from sorted list in order
    if Find-set(u) ≠ Find-set(v)
        A = A ∪ {(u,v)}
        Union(u,v)
return A
```

else it will create a cycle.

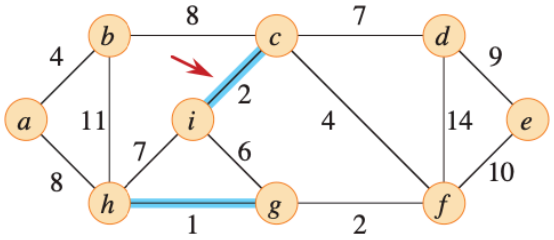
Running time $O(E \lg V)$

Example 5

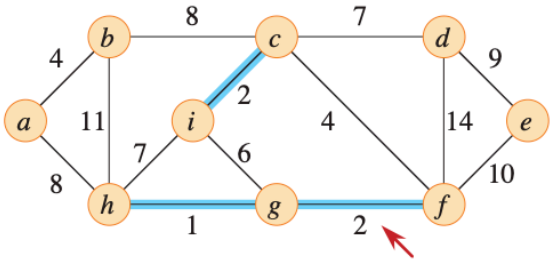
(a)



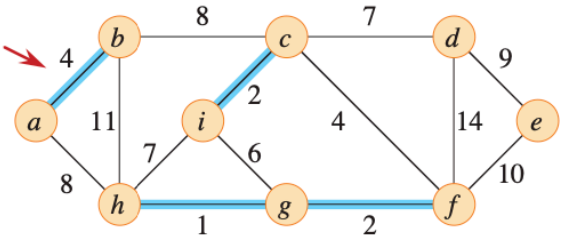
(b)



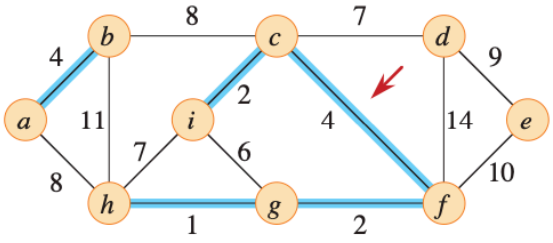
(c)



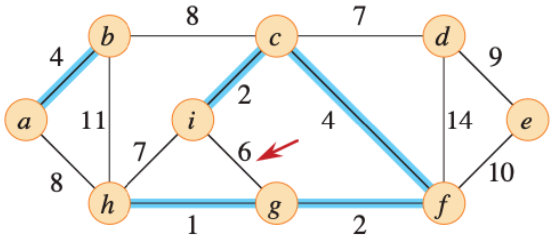
(d)



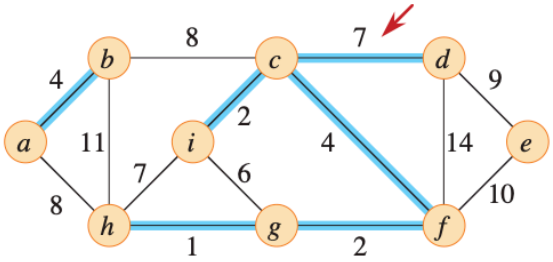
(e)



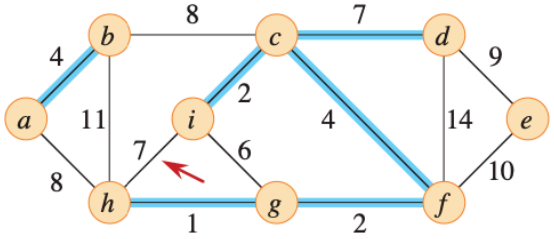
(f)



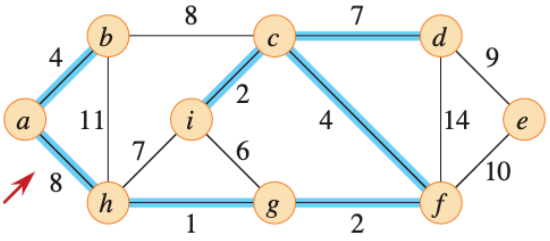
(g)



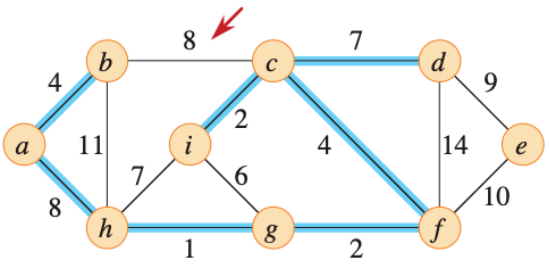
(h)



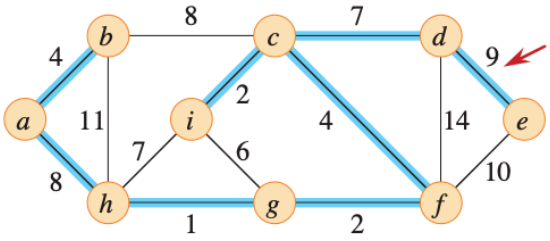
(i)



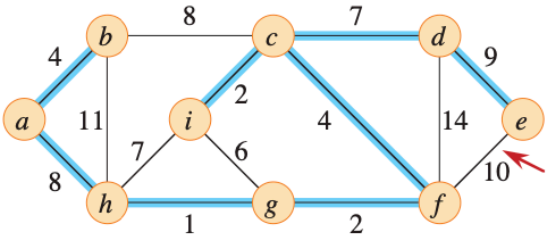
(j)



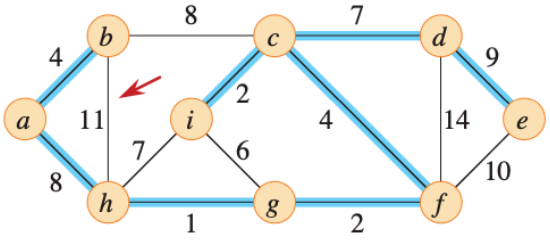
(k)



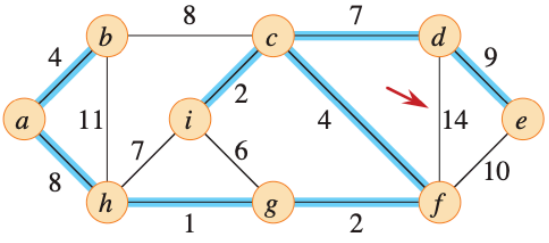
(l)



(m)



(n)

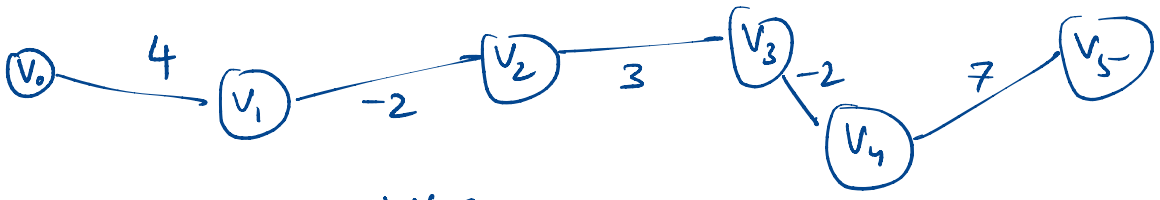


Shortest Path

Consider a digraph $G(V,E)$ with edge wt. functin $W: E \to \mathbb{R}$

The weight of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is defined as:

$$W(p) = \sum_{i=0}^{k} w(v_i, v_{i+1})$$



$W(p) = 10$

Shortest Path: A shortest path from u to v is a path of minimum weight from u to v. The shortest-path weight is defined as:

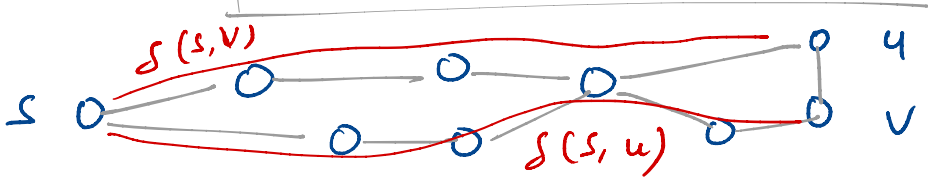
$$\delta(u,v) = \begin{cases} \min\{W(p) : u \xrightarrow{p} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise.} \end{cases}$$

Theorem: A sub path of a shortest path is a shortest path.

Theorem: Triangle inequality

Let $G = (V,E)$ be weighted digraph with weight fn. $W: E \to \mathbb{R}$ and a source vertex s. Then for all edges $(u,v) \in E$

$$\delta(s,v) \leq \delta(s,u) + W(u,v)$$



Proof : Suppose p is the shortest path bet. s & v. then $\delta(p)$ has the least weight for any path bet. s & v. In particular it is also true for the path $s \to u$ and then u to v.

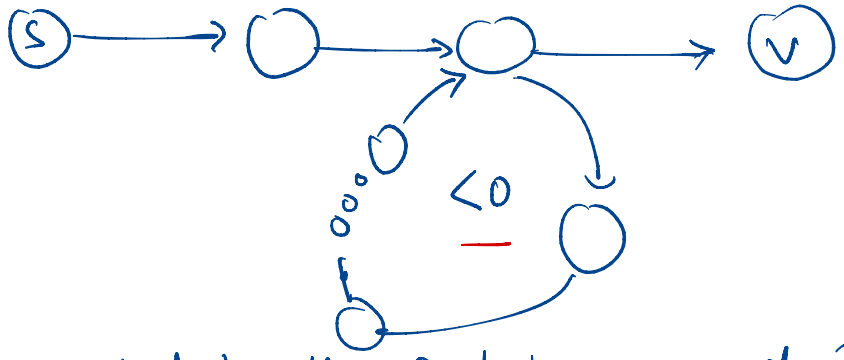
Negative weight edges

(a cycle with neg. weight)

if a graph G contains no negative weight cycle reachable from source s, then for all vertices $v \in V$, the weight $\delta(s, v)$ remains well defined.

↳ On the contrary if G contains a -ve weight cycle reachable from s, then shortest path weight are not well defined.

↳ One can always find a path of lower weight by traversing the -ve wt. cycle arbitrary number of times



Q : Can a shortest path contain a cycle? NO

↳ Negative wt cycles are certainly not possible

↳ Positive weight cycles will imply that one can remove them to obtain a path of smaller weight.

Single - Source Shortest path

From a given vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all vertices $v \in V$

↳ If all edge weight are non-negative then all shortest path weight must exist.

Greedy approach:

- Maintain a set S of vertices whose shortest path distances from s are known
- At each step add to S the vertex $v \in V - S$ whose distance estimate from s is minimal.
- Update distance estimates of vertices adjacent to v.

Dijkstra's Algorithm

Given: Directed graph $G = (V, E)$, non-negative weights $w(u, v) \geq 0$ for each edge $(u, v) \in E$

↳ Generalizes BFS algorithm.

S: Set of vertices whose final shortest path weights from s' have been determined.

Q: A min-priority queue of vertices with key d

DIJKSTRA (G, w, s')

For each vertex $v \in G.V$
  $v.d = \infty$ → shortest path distance from s'
  $v.\pi = NIL$

$s'.d = 0$

$S = \phi$

$Q = \phi$

For each vertex $u \in G.V$
  Insert (Q, u)

while $Q \neq \phi$

  u = Extract-Min (Q)

  $S = S \cup \{u\}$

  for each vertex v in $G.Adj[u]$

    if $v.d > u.d + w(u,v)$

      $v.d = u.d + w(u,v)$

      $v.\pi = u$

      Decrease-key (Q, v, v.d)

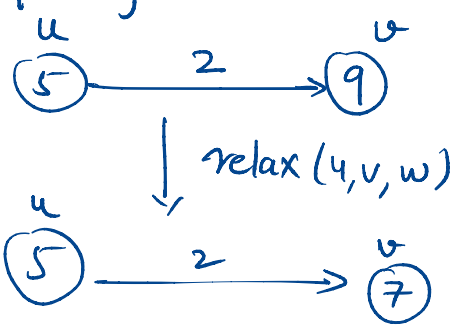
Complexity.

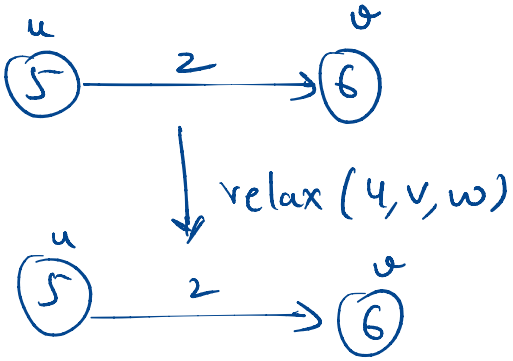
$O(E \lg V)$: if using bin. heap

$O(V^2)$: if using an array.

Relaxation step that tests if going through u will improve shortest path between s' and v.

Example of relaxation:





Example:

Blue vertices are part of S