Topics

- Minimum Spanning Trees
  - Kruskal
  - Prim

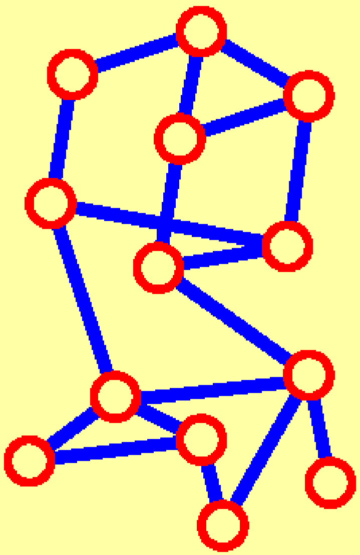
الدكتور
اثير العاني

Minimum Spanning Trees

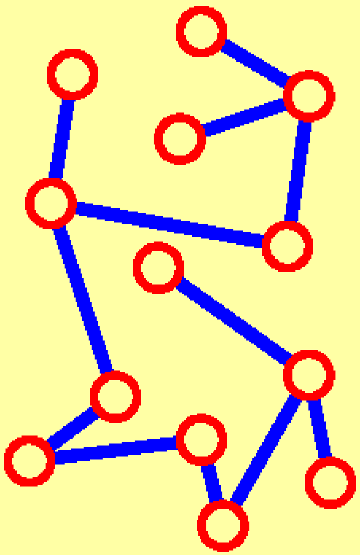
(MST)

Spanning Tree

- A **spanning tree** of **G** is a subgraph which
  - is a tree
  - contains all vertices of **G**



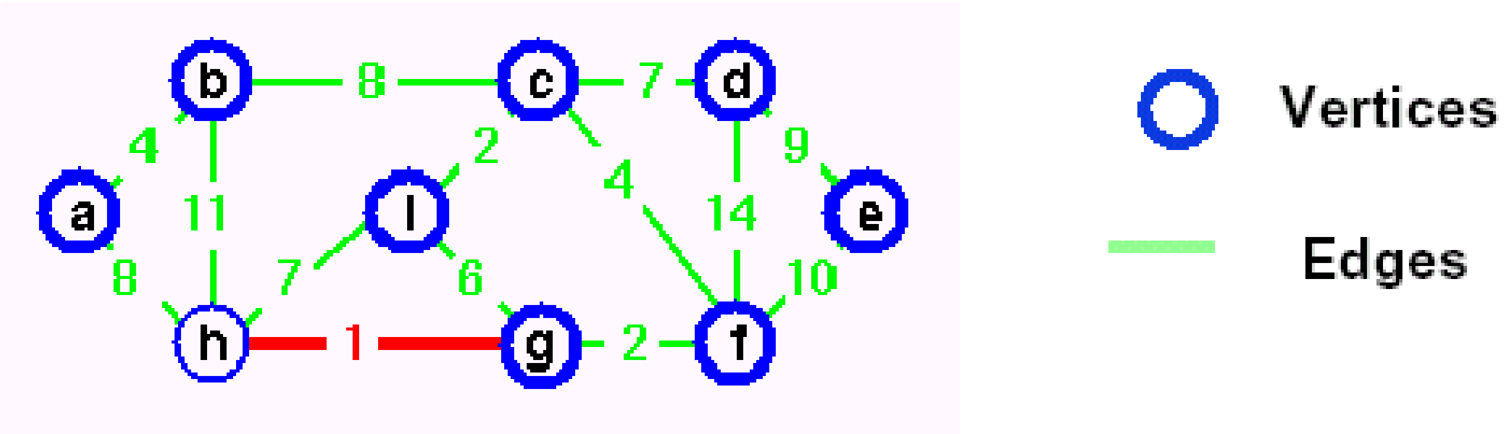
G



spanning tree of **G**

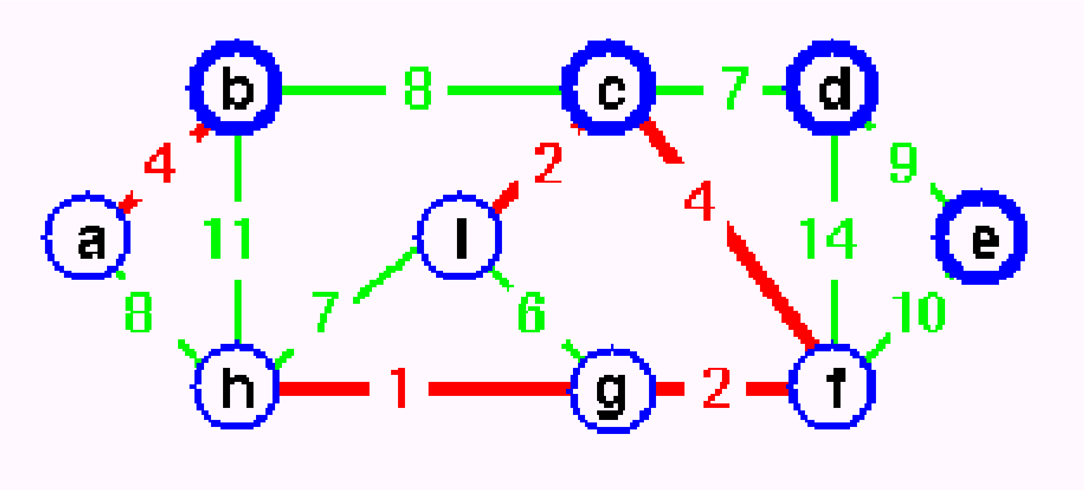
Weighted Graphs - Definitions

- Graph G = (V, E)
  - Set of vertices (nodes) and edges connecting them
  - V is a set of vertices: $V = \{ v_i \}$
  - E is a set of edges: $E = \{ (v_i, v_j) \}$
  - w: E → R, w is the weight function from E to Reals.



Weight Graphs - Definitions

- **Path**
  - A path, p, of length, k, is a sequence of connected vertices
  - $p = \langle v_0, v_1, \ldots, v_k \rangle$ *where* $(v_i, v_{i+1}) \in E$



< i, c, f, g, h >

Path of length 4 and distance 9

< a, b >

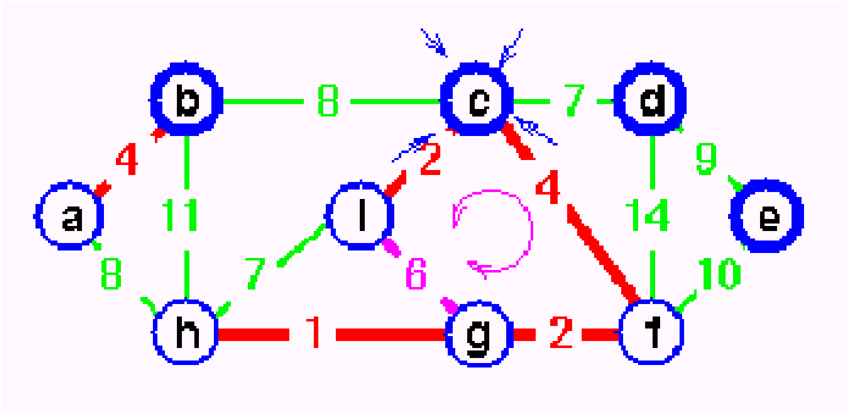
Path of length 1 and distance 4

Weighted Graphs - Definitions

- **Cycle**
  - A graph contains no cycles if there is *no* path

$$p = \langle v_0, v_1, \ldots, v_k \rangle$$

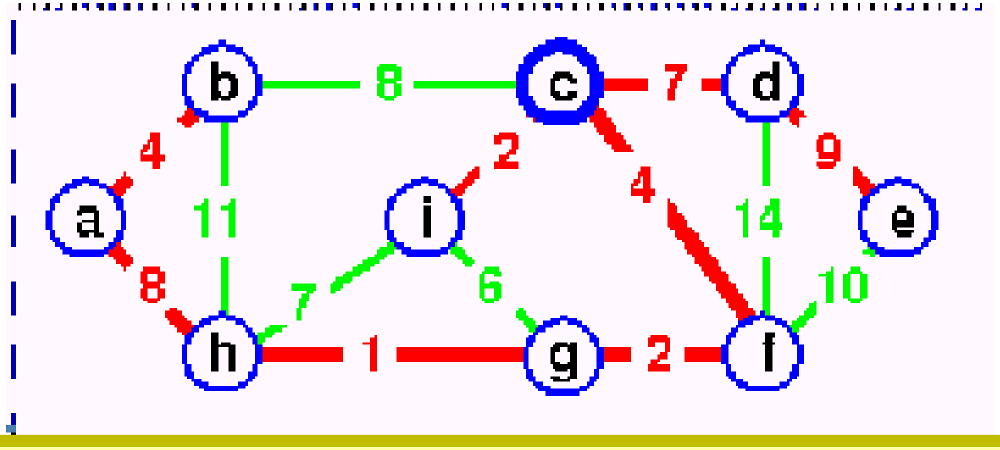
such that $v_0 = v_k$



< i, c, f, g, i >
is a cycle

Weighted Graphs - Definitions

- Spanning Tree
  - A spanning tree is a set of $|V|-1$ edges that connect all the vertices of a graph



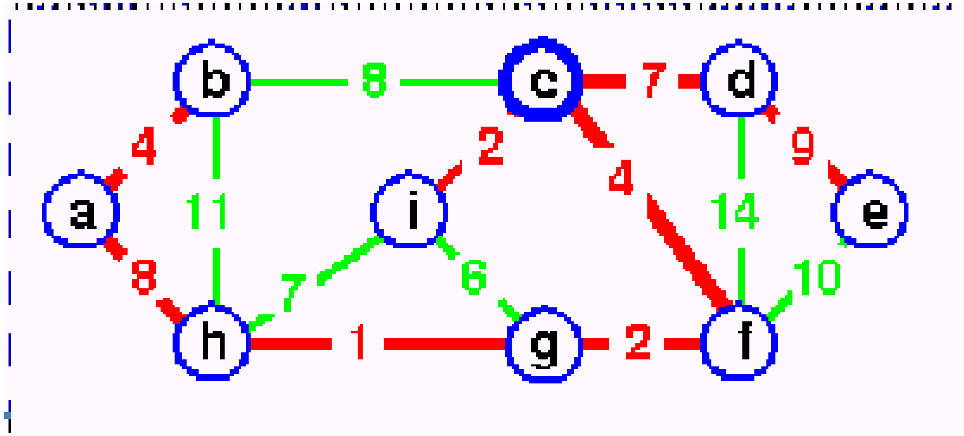
The red path connects all vertices, so it's a spanning tree

Minimum Spanning Tree

- Generally there is more than one spanning tree
- If a weight or cost c_{ij} is associated with edge $e_{ij} = (v_i, v_j)$ then the minimum spanning tree is the set of edges E_{span} such that

$$C = \Sigma \, (\, c_{ij} \mid \forall \, e_{ij} \in \, Espan \,)$$

is a minimum.

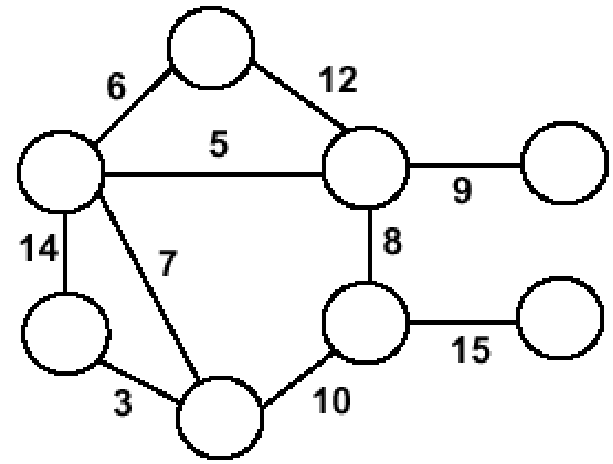


Other ST's can be formed ..
- Replace 2 with 7
- Replace 4 with 11

The red tree is the Min ST

Minimum Spanning Trees

- Undirected, connected graph $G = (V,E)$
- Weight function $W: E \rightarrow R$ (assigning cost or length or other values to edges)



- Spanning tree: tree that connects all the vertices (above?)
- Minimum spanning tree: tree that connects all the vertices and minimizes

$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

Kruskal's Algorithm

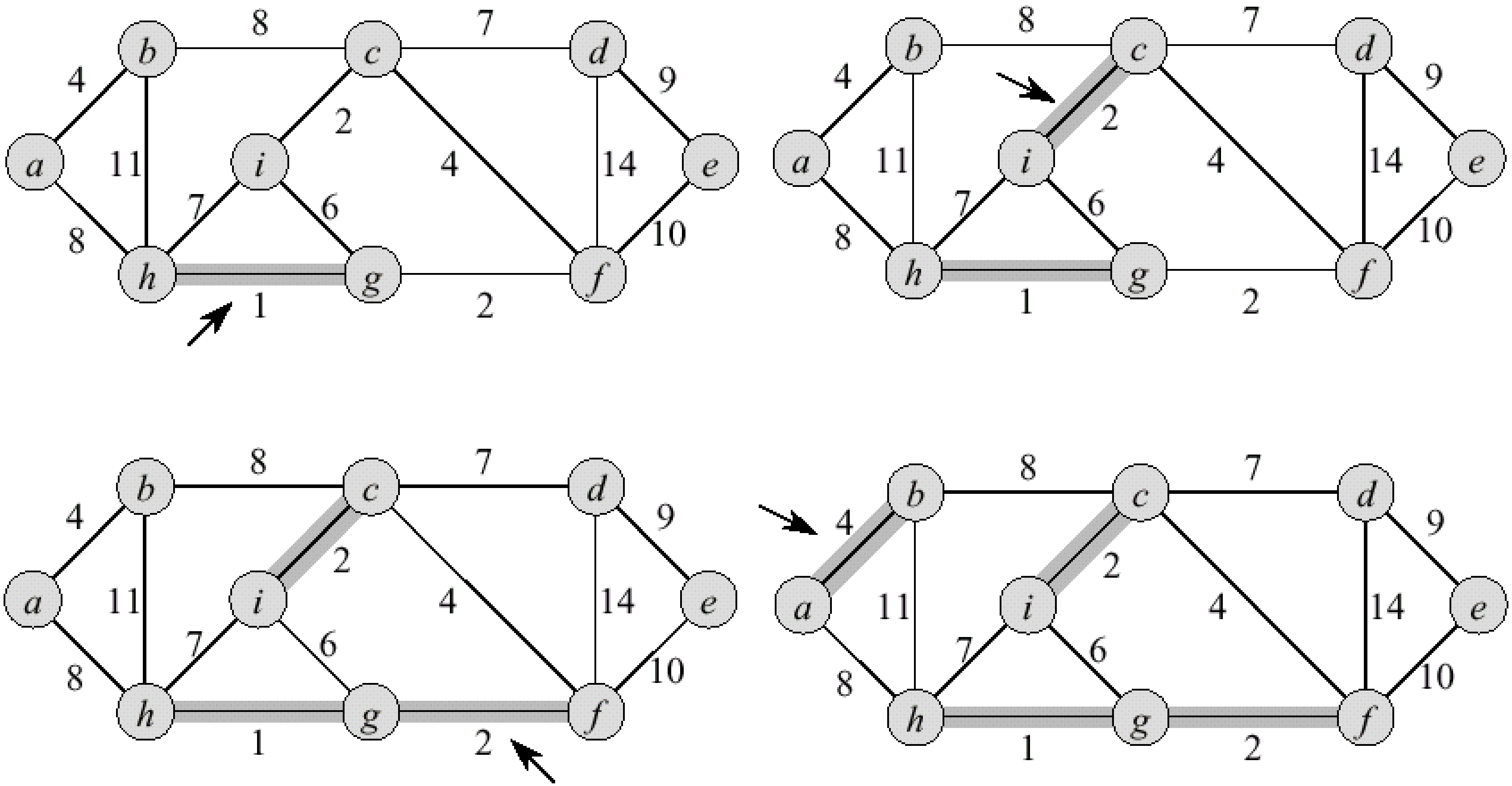
- Edge based algorithm
- Add the edges one at a time, in increasing weight order
- The algorithm maintains *A* – a **forest of trees**. An edge is accepted it if connects vertices of distinct trees
- We need a data structure that maintains a partition, i.e.,a collection of disjoint sets
  - MakeSet(S,x): $S \leftarrow S \cup \{\{x\}\}$
  - Union(S_i,S_j): $S \leftarrow S - \{S_i,S_j\} \cup \{S_i \cup S_j\}$
  - FindSet(S, x): returns unique $S_i \in S$, where $x \in S_i$

Kruskal's Algorithm

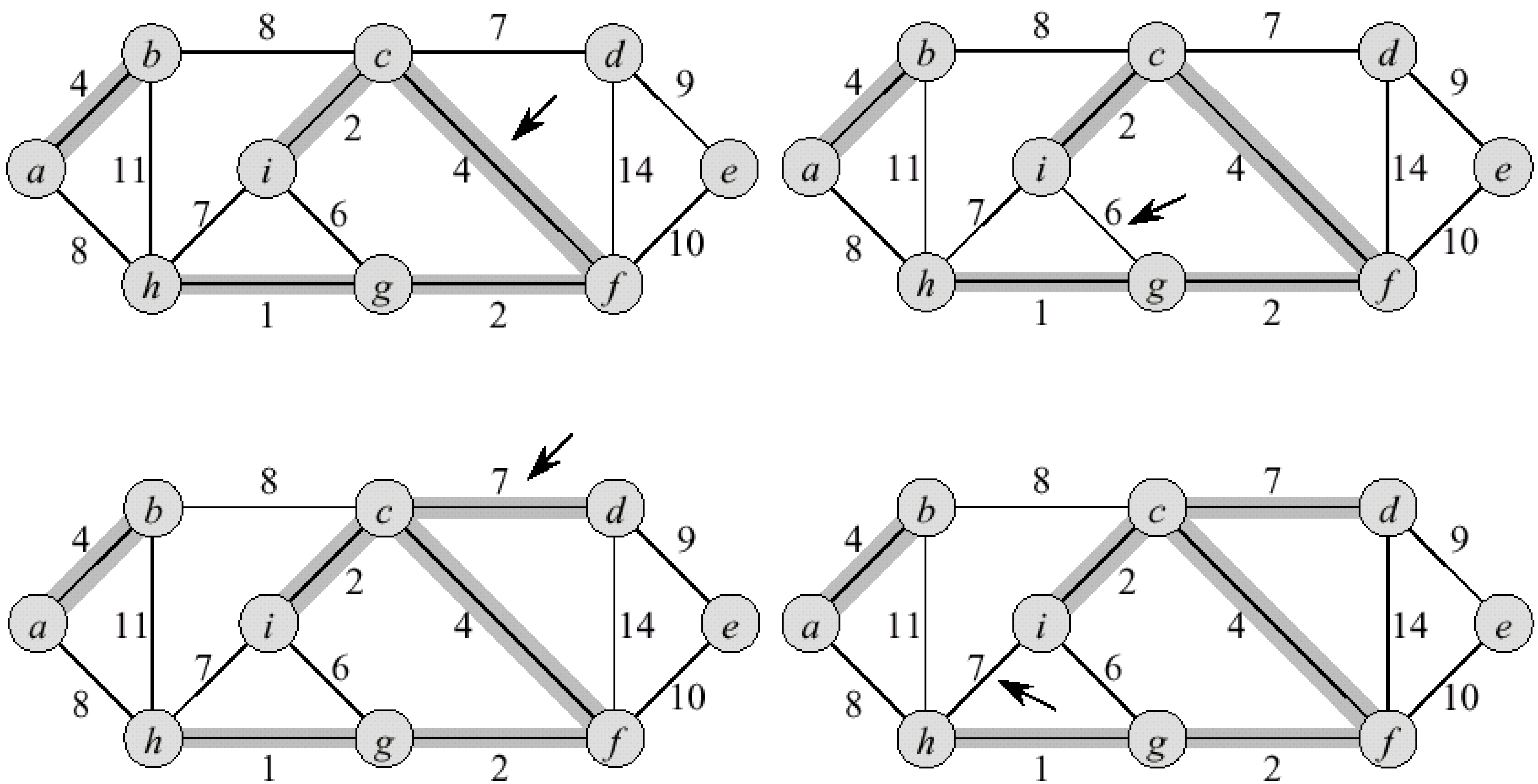
- The algorithm adds the cheapest edge that connects two trees of the forest

```
MST-Kruskal(G,w)
A ← ∅
for each vertex v ∈ V[G] do
   Make-Set(v)
sort the edges of E by non-decreasing weight w
for each edge (u,v) ∈ E, in order by non-decreasing weight do
  if Find-Set(u) ≠ Find-Set(v) then
     A ← A ∪ {(u,v)}
     Union(u,v)
return A
```

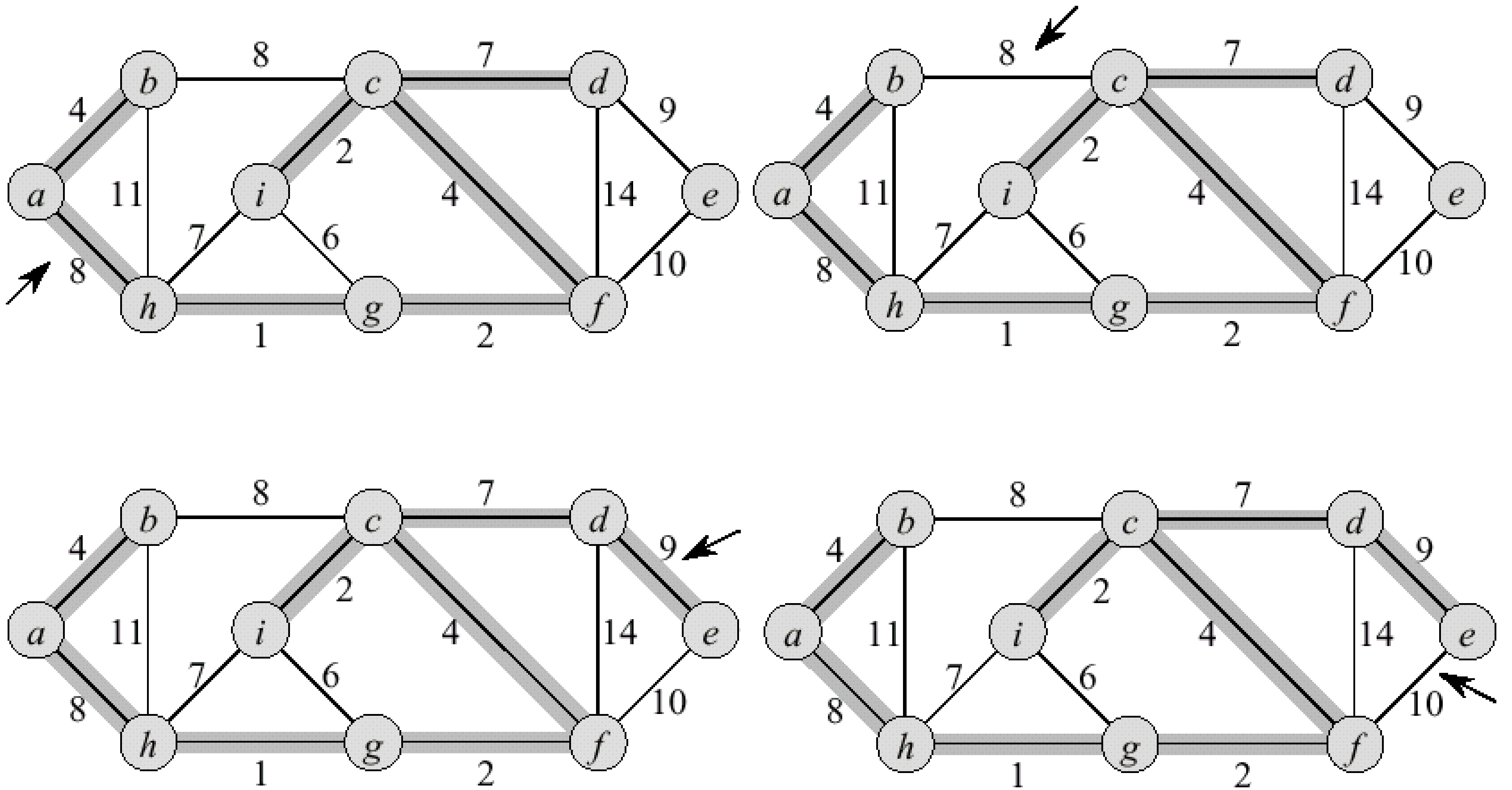
Kruskal Example



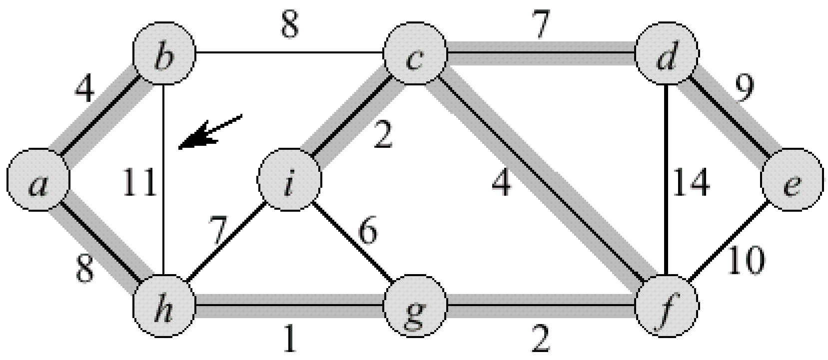
Kruskal Example (2)

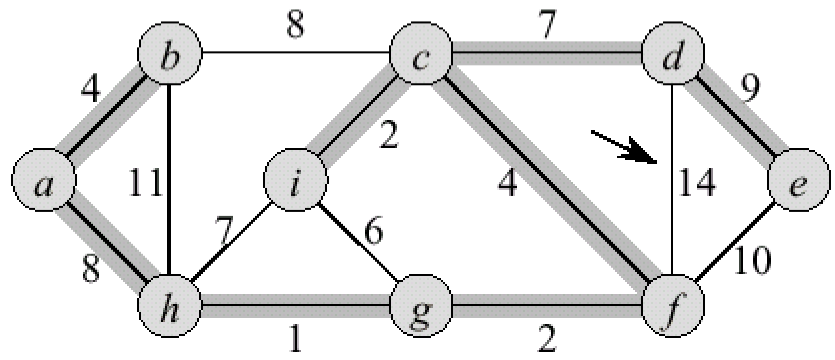


Kruskal Example (3)



Kruskal Example (4)





Kruskal Running Time

- Initialization $O(V)$ time
- Sorting the edges $\Theta(E \lg E) = \Theta(E \lg V)$ (why?)
- $O(E)$ calls to FindSet
- Union costs
  - Let $t(v)$ – the number of times v is moved to a new cluster
  - Each time a vertex is moved to a new cluster the size of the cluster containing the vertex at least doubles: $t(v) \leq \log V$
  - Total time spent doing Union $\sum_{v \in V} t(v) \leq |V| \log |V|$
- Total time: $O(E \lg V)$

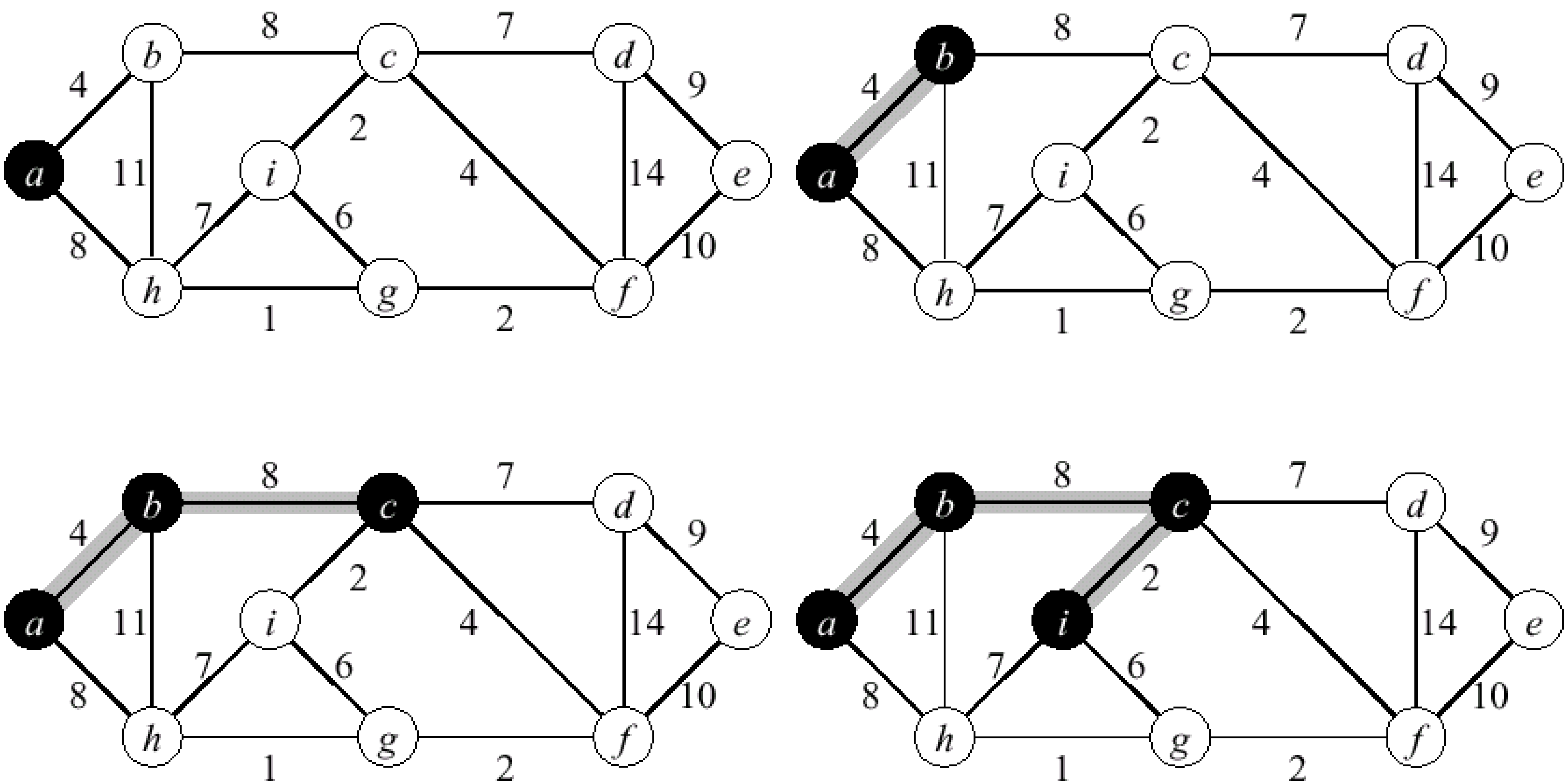
Prim-Jarnik Algorithm

- Vertex based algorithm
- Grows one tree T, **one vertex at a time**
- A cloud covering the portion of T already computed
- Label the vertices *v* outside the cloud with *key*[*v*] – the minimum weigth of an edge connecting *v* to a vertex in the cloud, *key*[*v*] = ∞, if no such edge exists

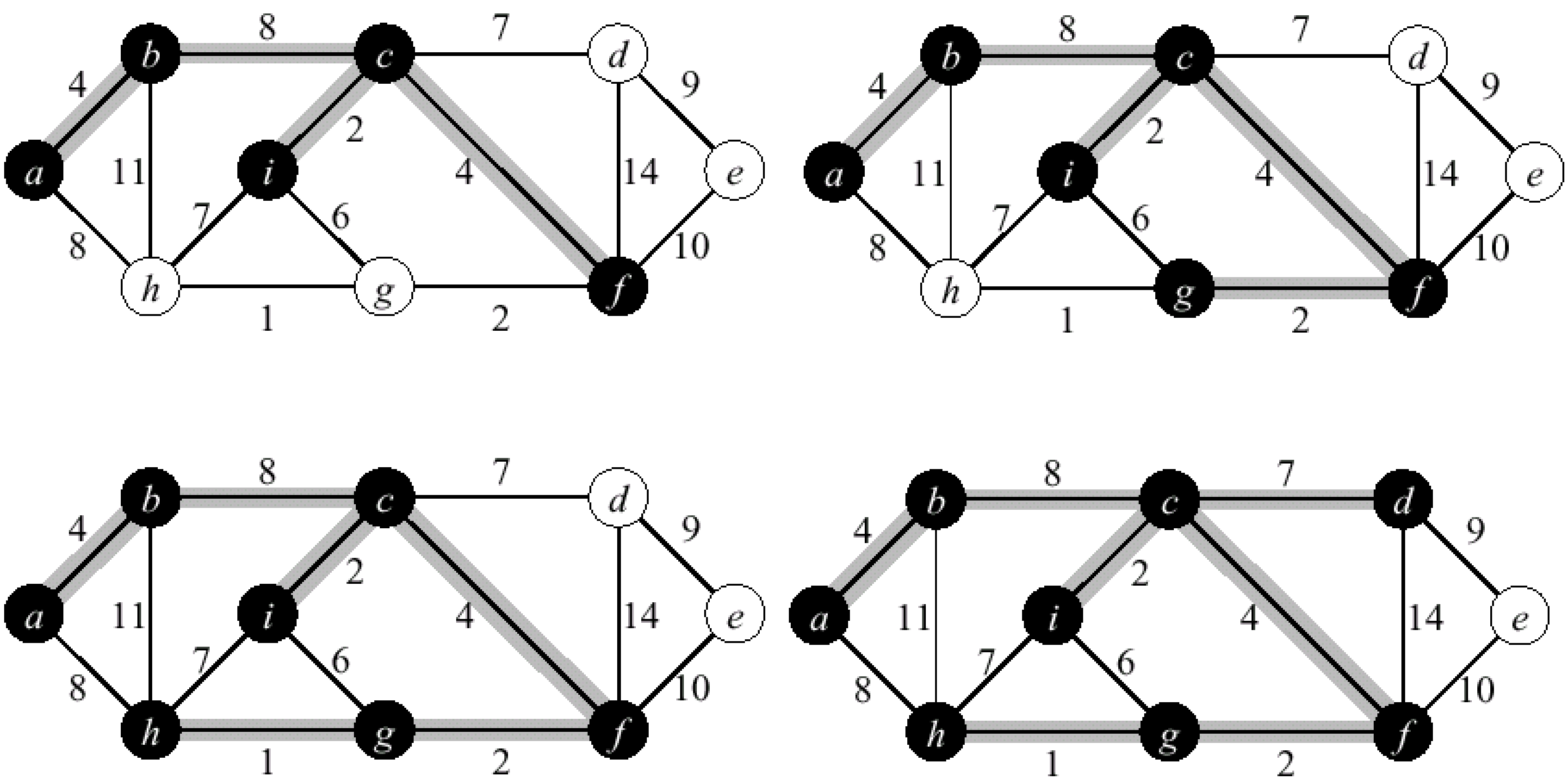
Prim-Jarnik Algorithm (2)

```
MST-Prim(G,w,r)
Q ← V[G]  // Q - vertices out of T
for each u ∈ Q
   key[u] ← ∞
key[r] ← 0
π[r] ← NIL
while Q ≠ ∅ do
  u ← ExtractMin(Q)  // making u part of T
     for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
           π[v] ← u
           key[v] ← w(u,v)
```

Prim Example



Prim Example (2)



Prim Example (3)