

Lecture 13: Minimal Spanning Trees

Course Learning Outcome

- Use fundamental **graph algorithms**, like traversal, shortest path and **spanning tree** in the solution of real-life problems

Dr. Khalil Yousef

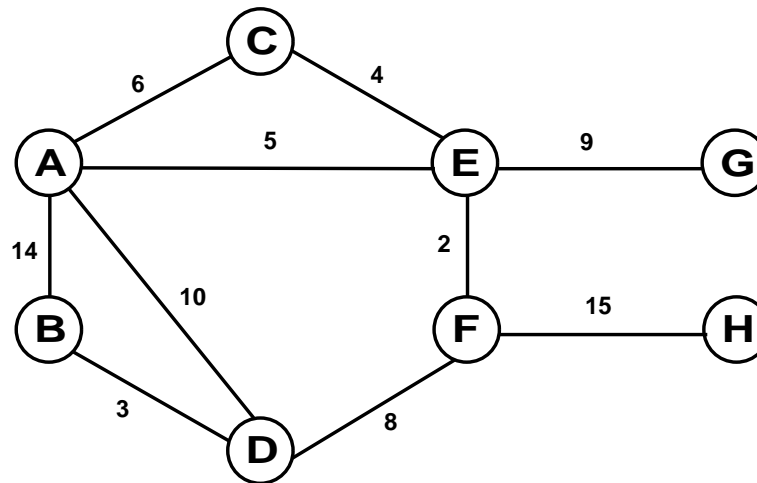
Adopted from the Slides of the ECE 608 Computational Models and Methods Course at Purdue University

Read Sections 21.1-21.3 and Chapter 23 of *Introduction to Algorithms*

Spanning Trees

A **spanning tree**, $S = (V, T)$, for a connected undirected graph $G = (V, E)$ is an undirected tree (i.e., connected and acyclic) that connects all vertices V with $T \subseteq E$.

A **minimum spanning tree** (or MST) for a connected undirected graph $G = (V, E)$, given a weight function $w : E \rightarrow R$, is the spanning tree $S = (V, T)$ for which $w(T) = \sum_{(u,v) \in T} w(u,v)$ is minimized. For example:



Applications: Minimize the cost of wiring electronic circuits.

Optimal Substructure Property of MST

An MST has the optimal substructure property in that an optimal MST contains optimal subtrees.

- To demonstrate this, consider an MST for the connected undirected graph $G = (V, E)$, T , containing an edge (u, v) .
- If we remove the edge, T is partitioned into two subtrees T_1 and T_2 .
- T_1 must be an MST of $G_1 = (V_1, E_1)$, the subgraph of G induced by the vertices of T_1 (i.e., V_1 are the vertices of T_1 and $E_1 = \{(x, y) \in E : x, y \in V_1\}$).
- Similarly, T_2 must be an MST of G_2 .
- Because $w(T) = w(u, v) + w(T_1) + w(T_2)$, there cannot be a more optimal tree than T_1 or T_2 ; otherwise, T would be suboptimal.

GENERIC-MST

GENERIC-MST(G, w)

1. $A \leftarrow \emptyset$
2. **while** A does not form a spanning tree
3. **do** find an edge (u, v) that is safe for A
4. $A \leftarrow A \cup \{(u, v)\}$
5. **return** A

A is the set of edges added so far to the MST being constructed; hence, it is a subgraph of an MST.

Each edge (u, v) added to A must be a **safe edge**, that is, it can be added without violating the fact that $A \cup \{(u, v)\}$ is a subset of an MST.

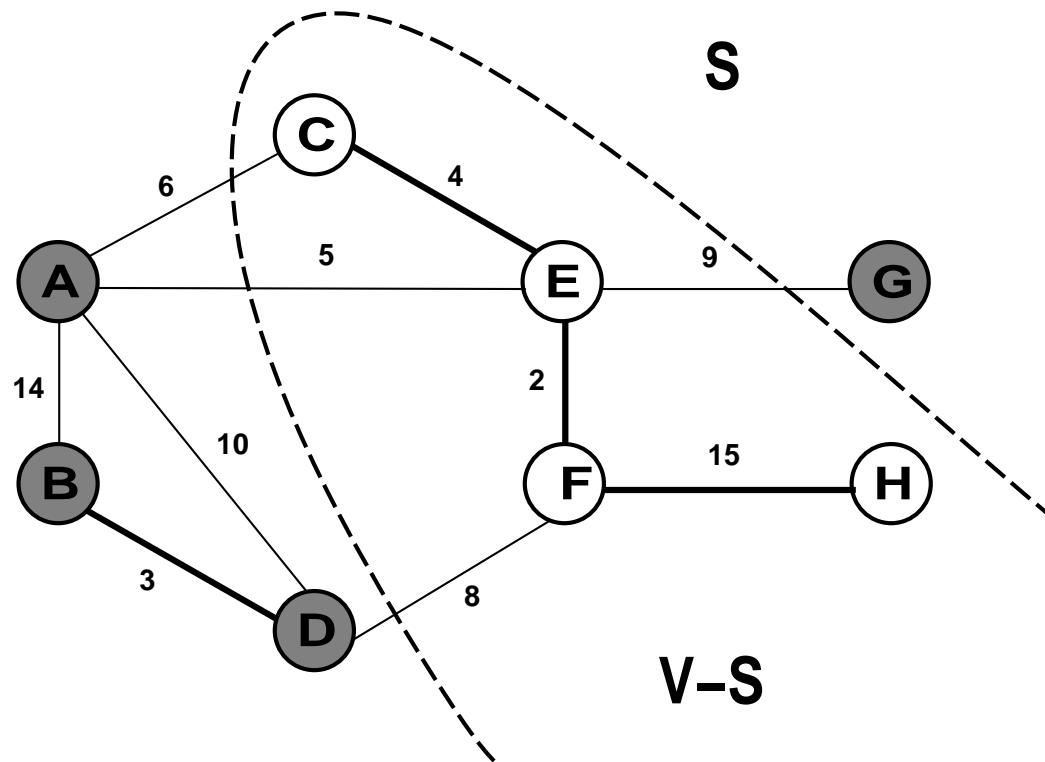
The loop in lines 2-4 is executed $|V| - 1$ times as each of the $|V| - 1$ edges of the MST is determined. Initially, there are $|V|$ trees, with each iteration reducing the number by 1. When there is a single MST, the algorithm terminates.

Terminology for Safe Edge Selection

We need some definitions to help us to understand GENERIC-MST:

- 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 endpoint is in S and the other is in $V - S$.
- A cut **respects** the set A if no edge in A crosses the cut.
- A **light edge** is the the minimum-weight edge crossing the cut.
- A **safe edge** is an edge that can be added to A , a subset of the MST, so that is is still a subset of the MST.

Example Illustrating Terminology



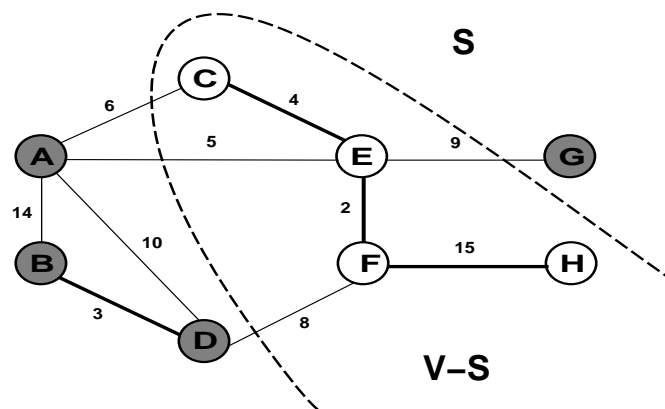
$$A = \{(B,D), (C,E), (E,F), (F,H)\}$$

MST: Important Notes:

Distinct weights in the a graph guarantee that the minimum spanning tree of the graph is unique. Without this condition, there may be several different minimum spanning trees. For example, if all the edges have weight 1, then every spanning tree is a minimum spanning tree with weight $V - 1$.

MST: Important Notes:

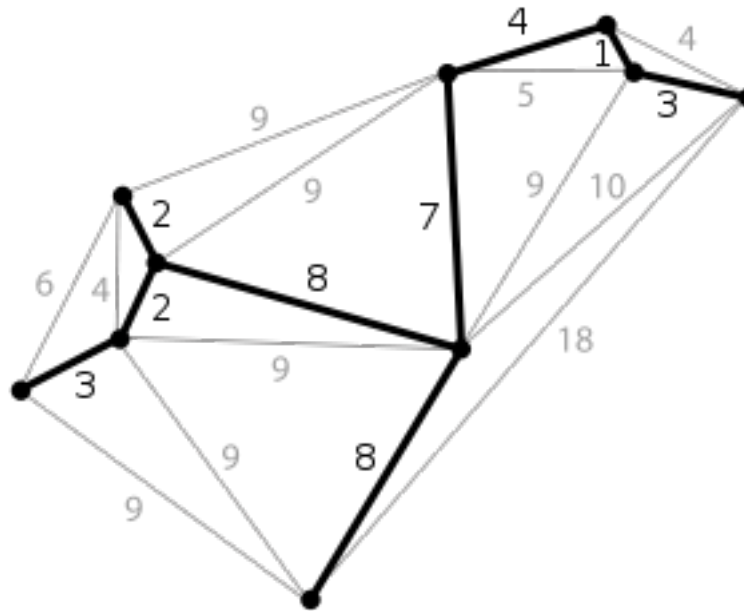
- For Greedy algorithms, there are efficient means to identify safe choices of the edges; Prim and Kruskal Algorithms.
- For MST, we find a safe edge using the lemma: Given a set of edges A that is part of MST A and a cut $(S, V - S)$ respecting A , then the light edge crossing the cut is safe.
- Given a set of edges A that is part of MST, the set of these edges is not necessary connected. For example the edges in the MST shown below is disconnected.



$A = \{(B,D), (C,E), (E,F), (F,H)\}$

MST: Important Notes:

- The MST actual algorithm doesn't developed yet; because how to find a cut and how to find a lightest edge. The following algorithms describe the safe edge but use the same Generic MST.
 - Kruskal's MST Algorithm
 - Prim's MST Algorithm



Kruskal's MST Algorithm

Repeatidly:

Add (Update partition (tree) by UNION (u, v) i.e. $A = A \cup \{(u, v)\}$)

the lightest (lowest weight edge)

legal edge to A (doesn't cause a cycle to A; Question: how to find this edge; This edge should join two different blocks of the partitions (trees) satisfying the condition that $\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$).

Note that UNION (u, v) , FIND-SET(u), FIND-SET(v), and others are operations on a disjoint data structure. We will now describe some of these operations.

Kruskal's MST Algorithm

Disjoint Set Data structure operations:

- **MAKE-SET**(x): adds x to the partition, extending the domain.
- **FIND-SET**(x): returns the representative element (name of the set) of the partition block containing x .
- **UNION**(x, y): combine the partition blocks for x and y and choose a new representative element for the combined resulting block.

Running time cost: A sequence of m operations on a disjoint set data structure runs in $O(m\alpha(m))$ time, where $\alpha(m) \leq 4$ (extremely a low growing function)

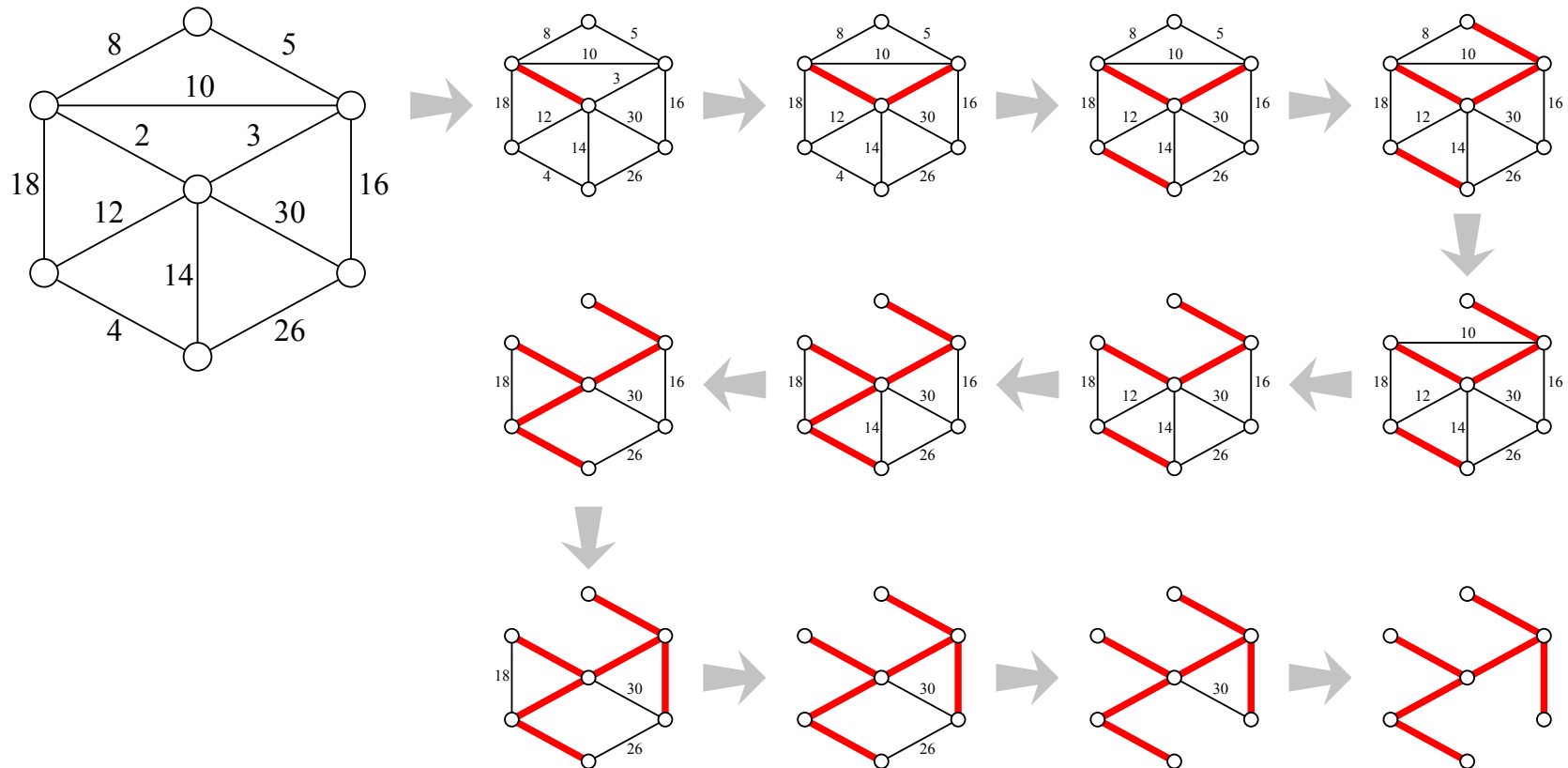
Kruskal's MST Algorithm

This algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees, the edge (u, v) of least weight. Let T_1 and T_2 denote two trees connected by edge (u, v) . Since (u, v) must be a light edge connecting T_1 to another tree, Corollary 24.2 implies that (u, v) is a safe edge.

MST-KRUSKAL(G, w)

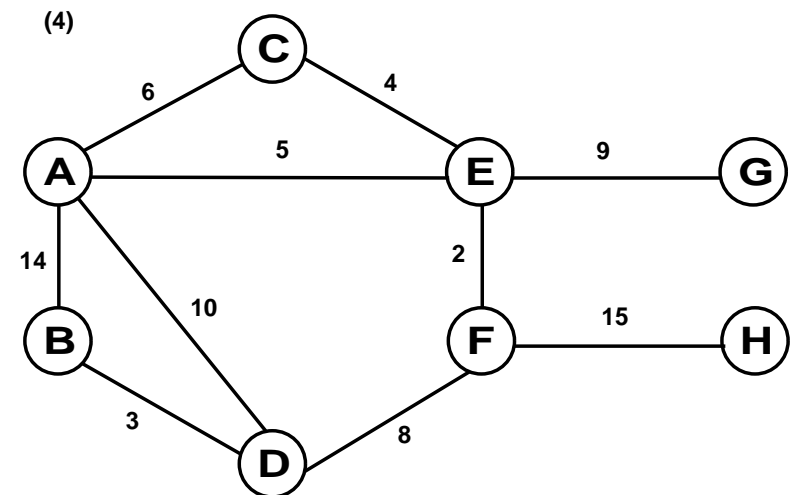
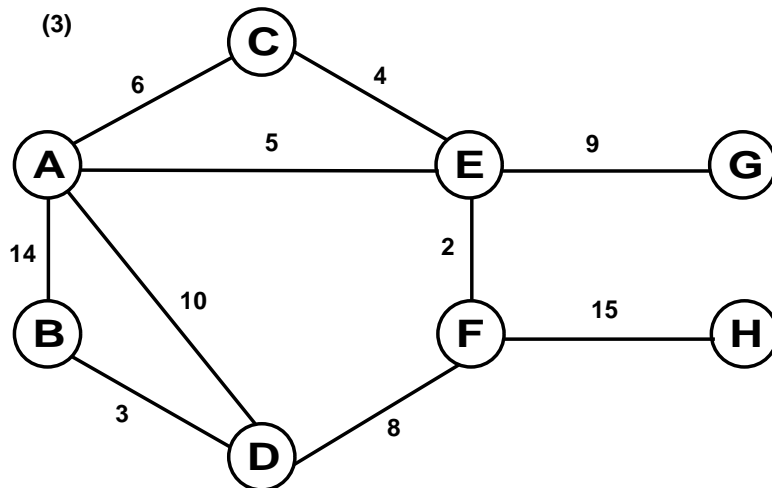
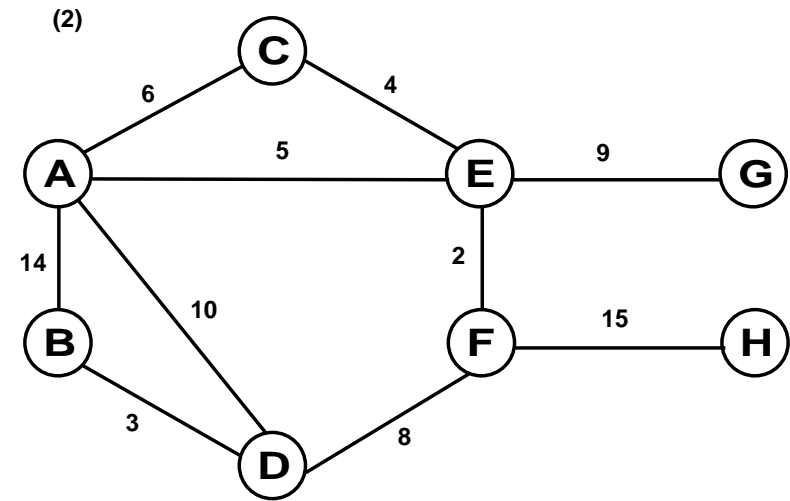
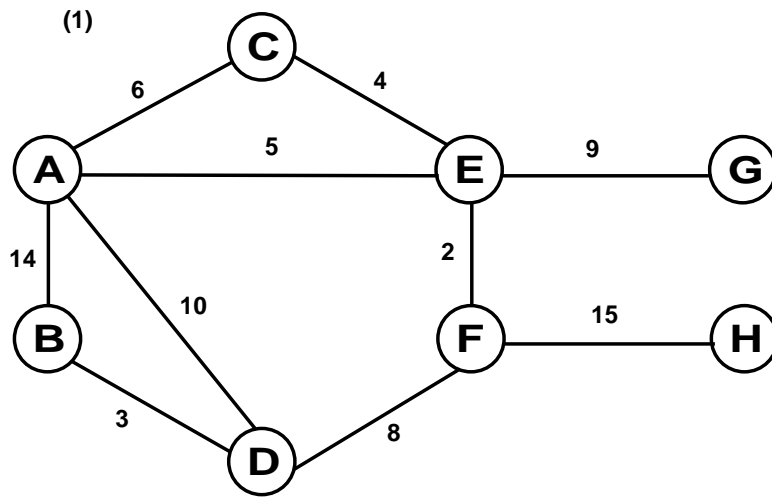
1. $A \leftarrow \emptyset$
2. **for** each vertex $v \in V[G]$
3. **do** MAKE-SET(v)
4. sort the edges of E by non-decreasing weight w
5. **for** each edge $(u, v) \in E$, in order of w
6. **do if** FIND-SET(u) \neq FIND-SET(v)
7. **then** $A \leftarrow A \cup \{(u, v)\}$
8. UNION(u, v)
9. **return** A

MST-KRUSKAL Example

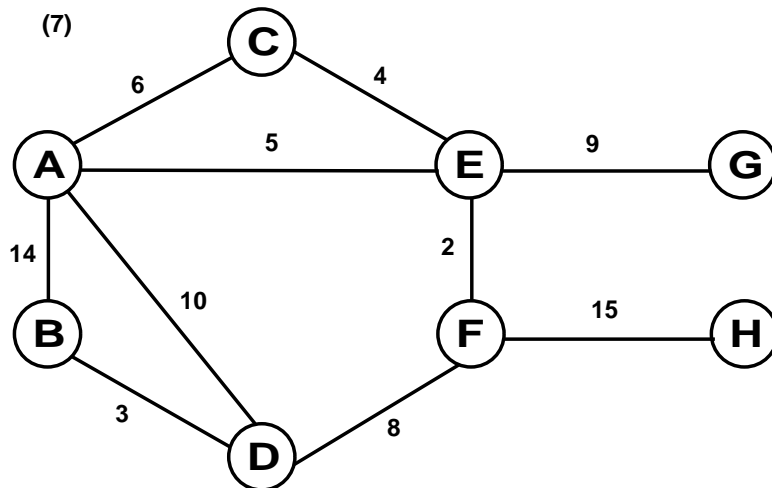
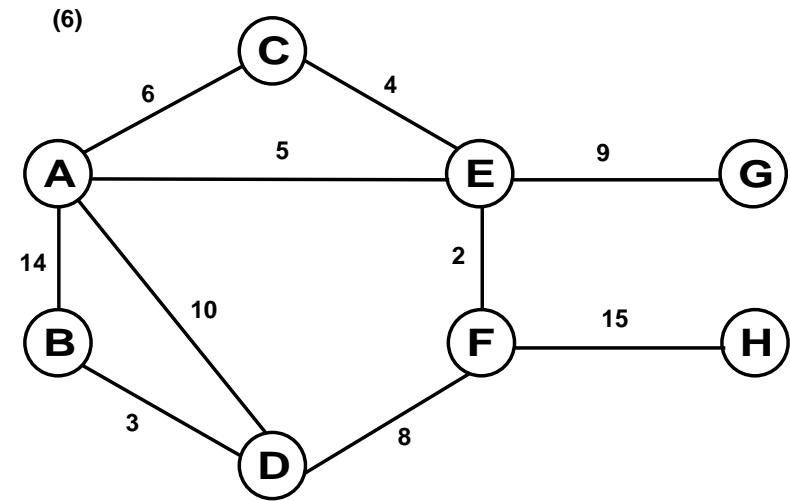
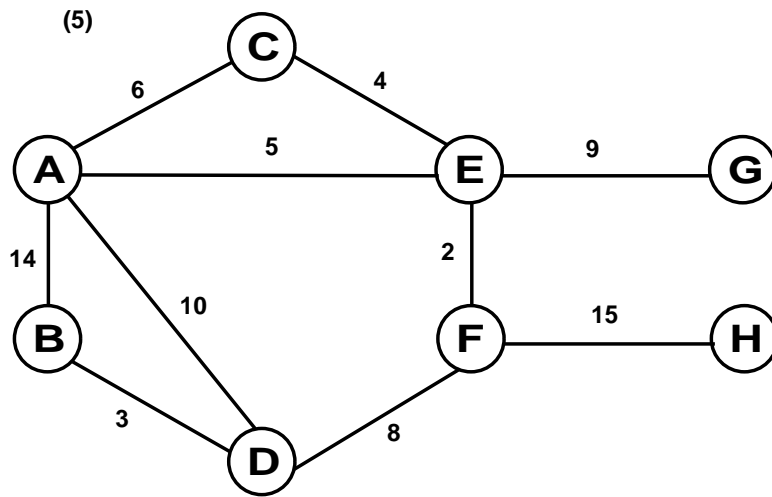


Kruskal's algorithm run on the example graph. Thick edges are in A.

MST-KRUSKAL Example



MST-KRUSKAL Example *continued*



MST-KRUSKAL Running Time

If we use union-by-rank and path compression in our disjoint set implementation, the time to execute MST-KRUSKAL is calculated as follows:

- **Initialization:** $\Theta(V)$
- **Sort E edges:** $O(E \lg E)$
- **Perform $O(E)$ operations on the disjoint set forest:** $O(E \alpha(E, V))$
- **Total:** $O(E \lg E)$ (because G is connected, $|E| \geq |V| - 1$)

Note that $\alpha(m, n)$ is the inverse of Ackerman's function (a fast growing function), and as such it is a very slow growing function. Note that $\alpha(E, V) \leq 4$ for all practical purposes (covers numbers as high as the number of atoms observable in the universe), and $O(E \lg^* V)$ is only slightly weaker than $O(E \alpha(E, V))$.

Prim's MST Algorithm

The algorithm uses a queue, Q , to select an edge to add to A . For each vertex v , $key[v]$ keeps the minimum weight of any edge connecting to v from vertices in the tree, and $\pi[v]$ points to the parent node in the tree. Initially, $key[v] = \infty$.

MST-PRIM(G, w, r)

1. $Q \leftarrow V[G]$
2. **for** each vertex $u \in Q$
3. **do** $key[u] \leftarrow \infty$
4. $key[r] \leftarrow 0$
5. $\pi[r] \leftarrow NIL$
6. **while** $Q \neq \emptyset$
7. **do** $u \leftarrow \text{EXTRACT-MIN}(Q)$
8. **for** each $v \in Adj[u]$
9. **do if** $v \in Q$ and $w(u, v) < key[v]$
10. **then** $\pi[v] \leftarrow u$
11. $key[v] \leftarrow w(u, v)$

MST-PRIM Discussion

During the course of the algorithm:

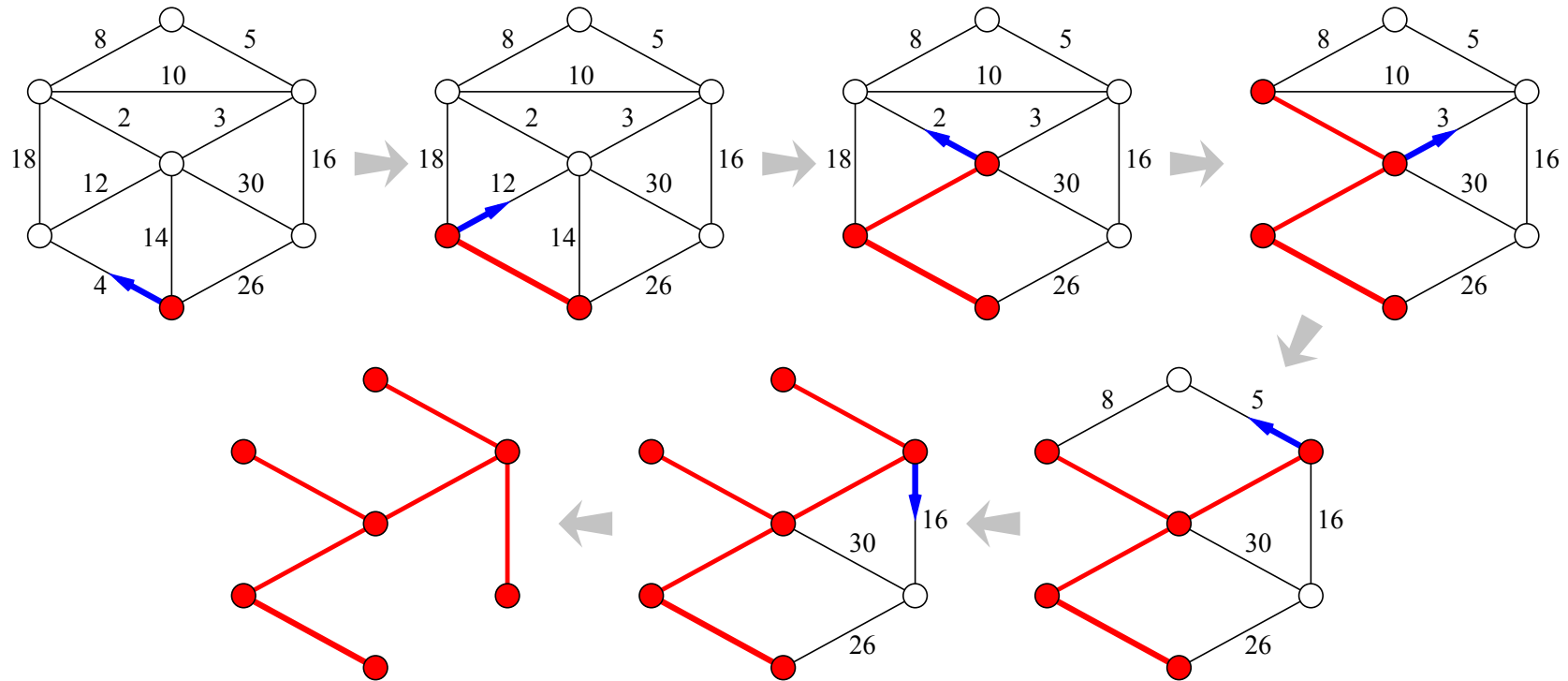
$$A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$$

At the end:

$$A = \{(v, \pi[v]) : v \in V - \{r\}\}$$

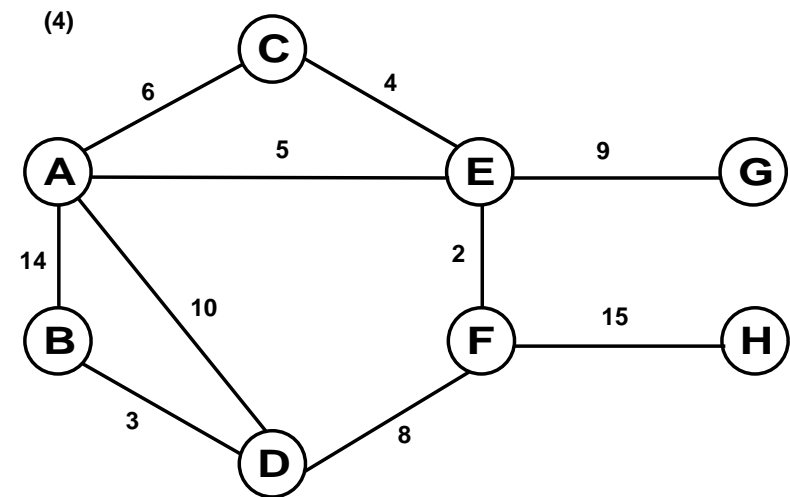
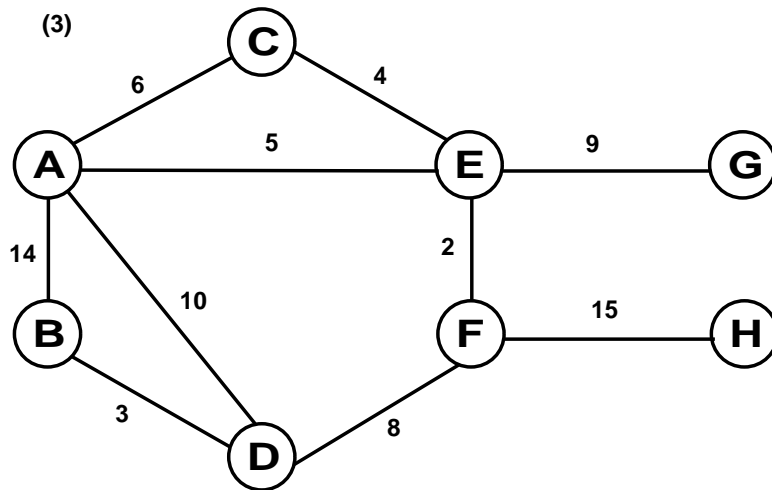
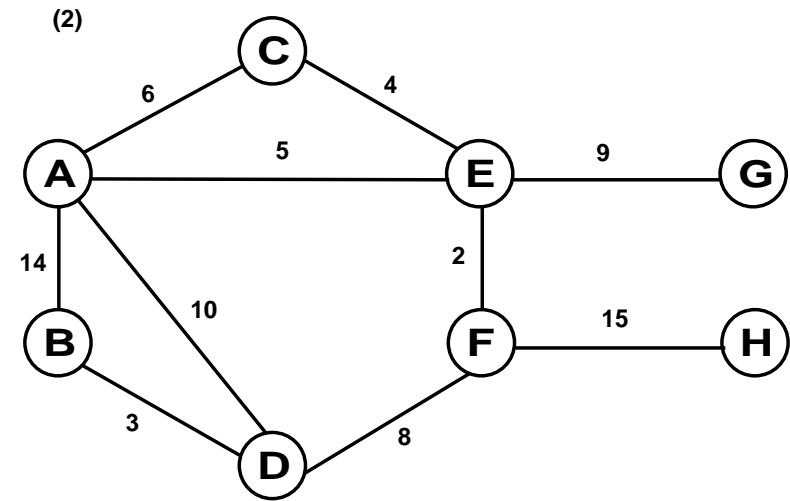
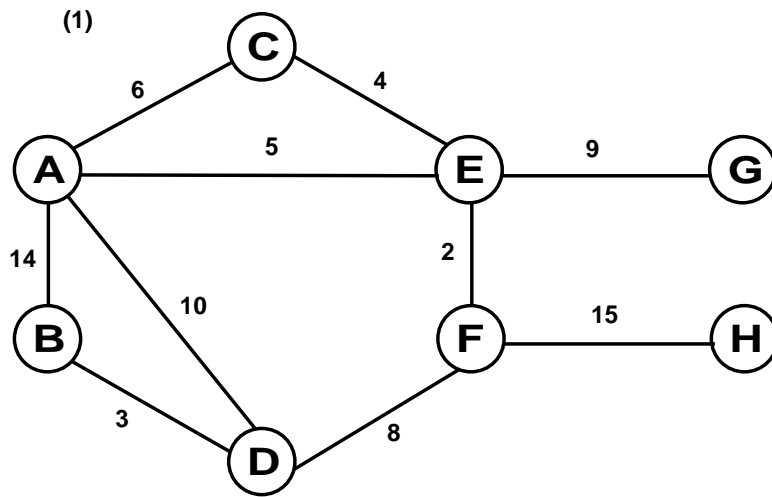
MST-PRIM starts with an arbitrary root r and it loops until it constructs a tree spanning the vertices of V . At each step, a light edge connecting a vertex in A to a vertex in $V - A$ is added to the tree. By corollary 24.2, the algorithm will add only edges that are safe for A , so the resulting tree will be an MST. The strategy is clearly a greedy one, since at each step, the tree is augmented with an edge that contributes a minimum amount to the tree's weight.

MST-PRIM Example

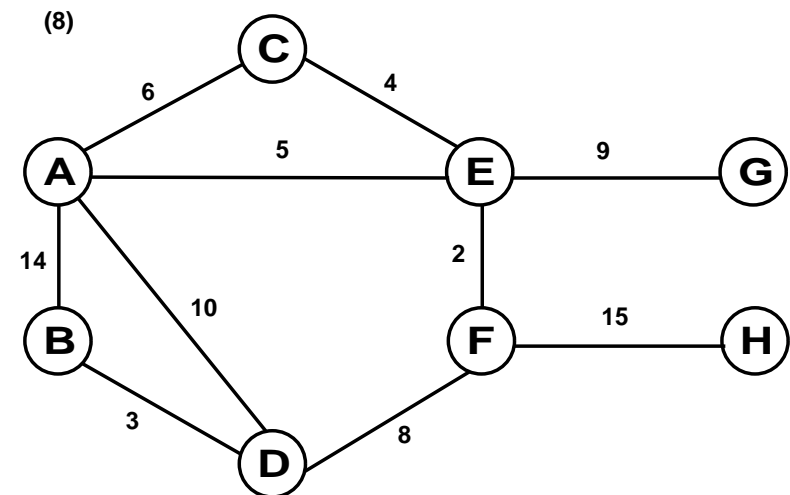
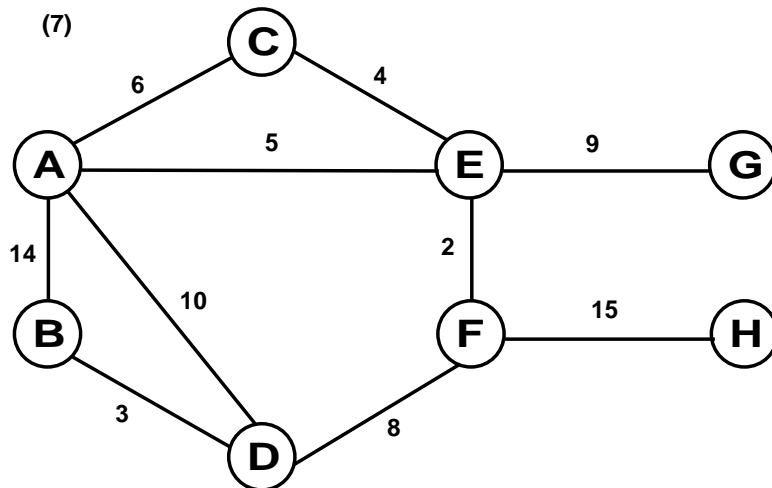
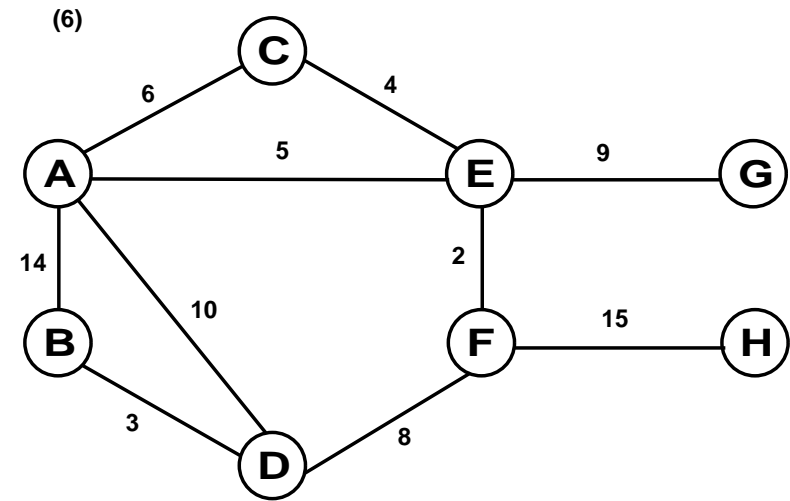
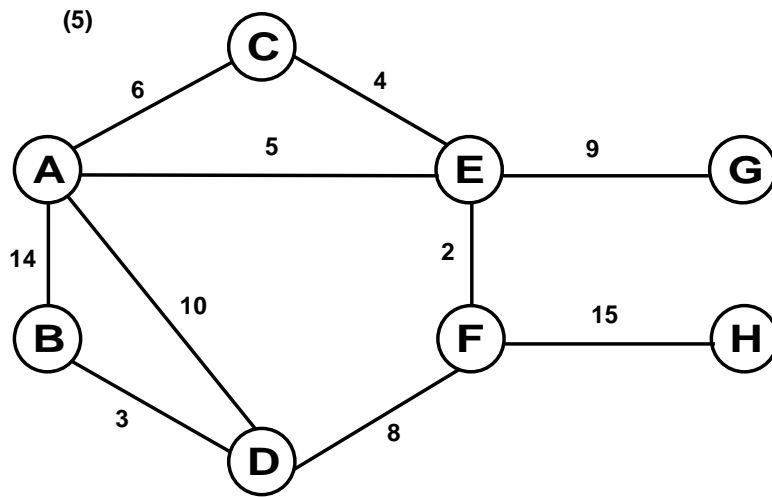


PRIM's algorithm run on the example graph. Thick edges are in A.

MST-PRIM Example



MST-PRIM Example *continued*



MST-PRIM Complexity

The running time of MST-PRIM depends on the implementation of the priority queue selected.

Heap: DECREASE-KEY runs in $O(\lg n)$ time.

- Initialization: $O(V)$
- EXTRACT-MIN Operations: $O(V \lg V)$
- DECREASE-KEY Operations: $O(E \lg V)$
- **Total:** $O((V + E) \lg V)$

Fibonacci Heap (see Chapter 21): DECREASE-KEY runs in $O(1)$ time.

- Initialization: $O(V)$
- EXTRACT-MIN Operations: $O(V \lg V)$
- DECREASE-KEY Operations: $O(E)$
- **Total:** $O(E + V \lg V)$