

Analysis of Algorithms

Minimum Spanning Trees

Minimum Spanning Trees (MST)

Input:

- An undirected and connected graph $G = (V, E)$ with n vertices and m edges.
- A **weight** function $w : E \rightarrow \mathfrak{R}$ on the edges.

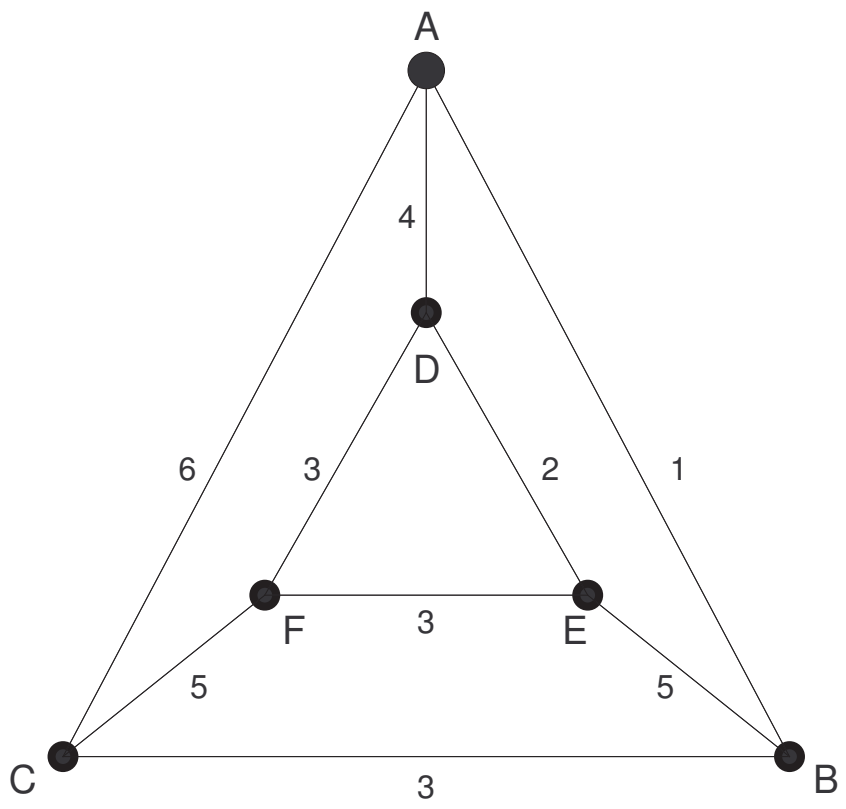
A spanning tree: A **connected sub-graph** with $n - 1$ edges.

A minimum spanning tree (MST): A spanning tree for which the sum of the weights of the $n - 1$ edges is **minimum**.

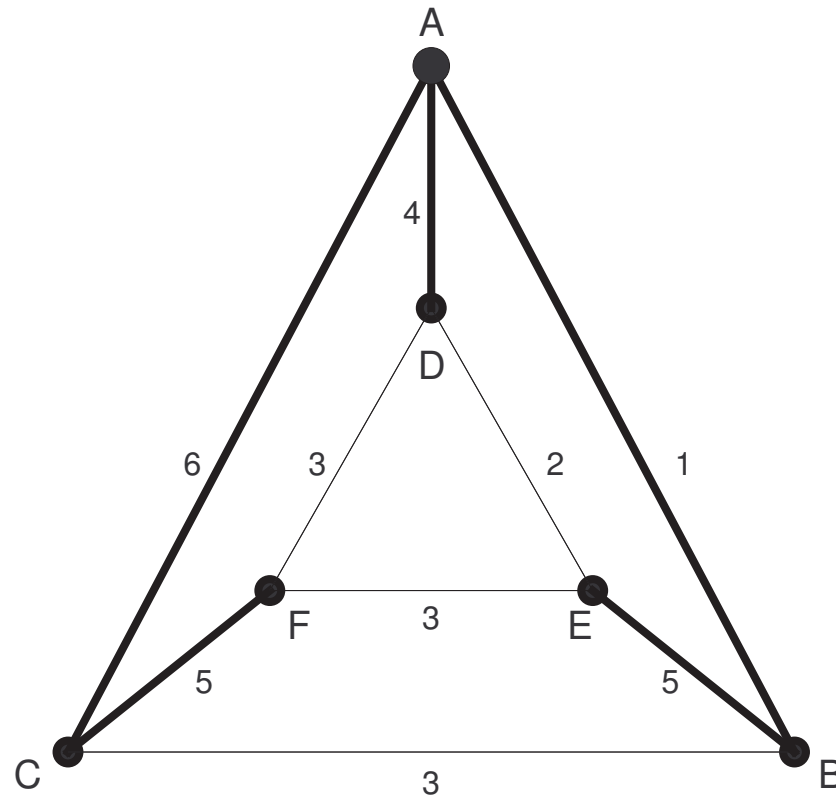
Two greedy algorithms:

- Kruskal - $O(m \log m)$ -time, a **distributed** algorithm.
- Prim - $O(m + n \log n)$ -time, a **centralized** algorithm.

Example: A weighted Graph

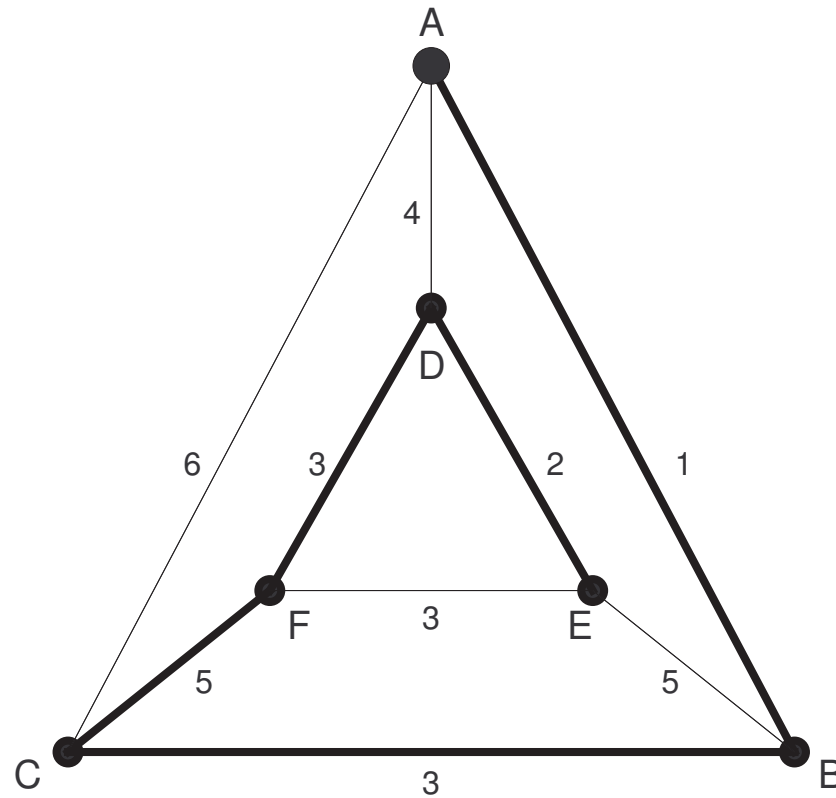


Example: A BFS Spanning Tree



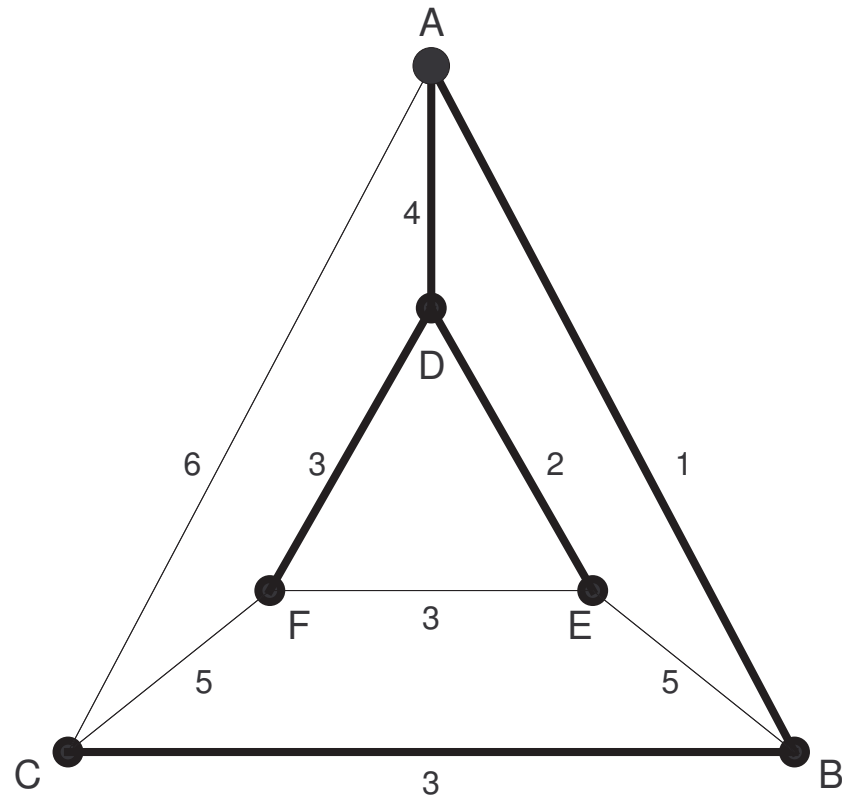
$$W(T) = 21$$

Example: A DFS Spanning Tree



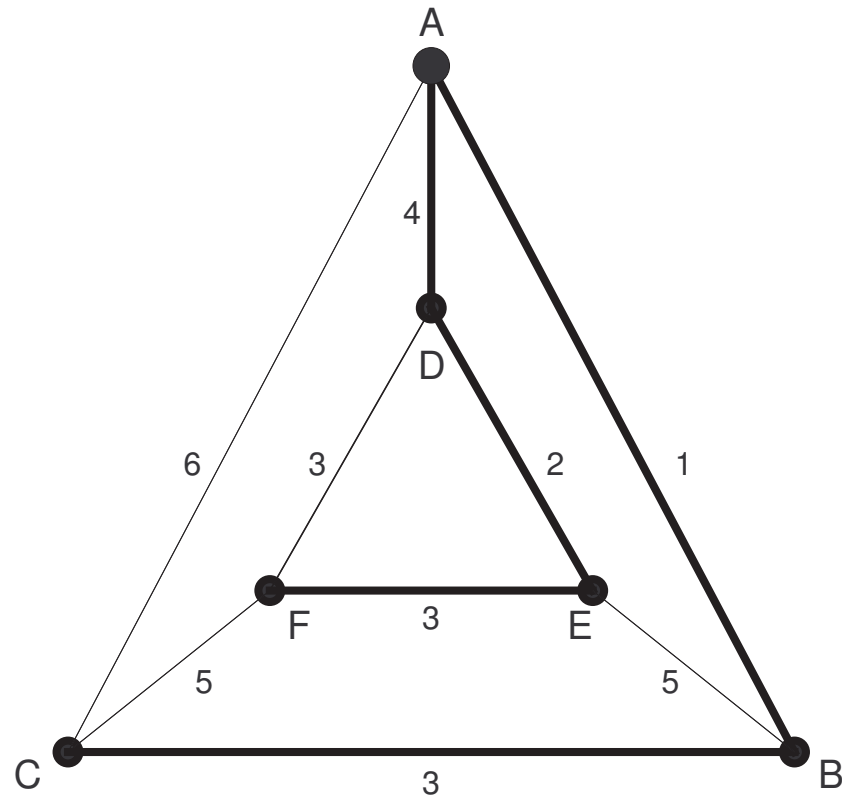
$$W(T) = 14$$

Example: An MST



$$W(T) = 13$$

Example: Another MST

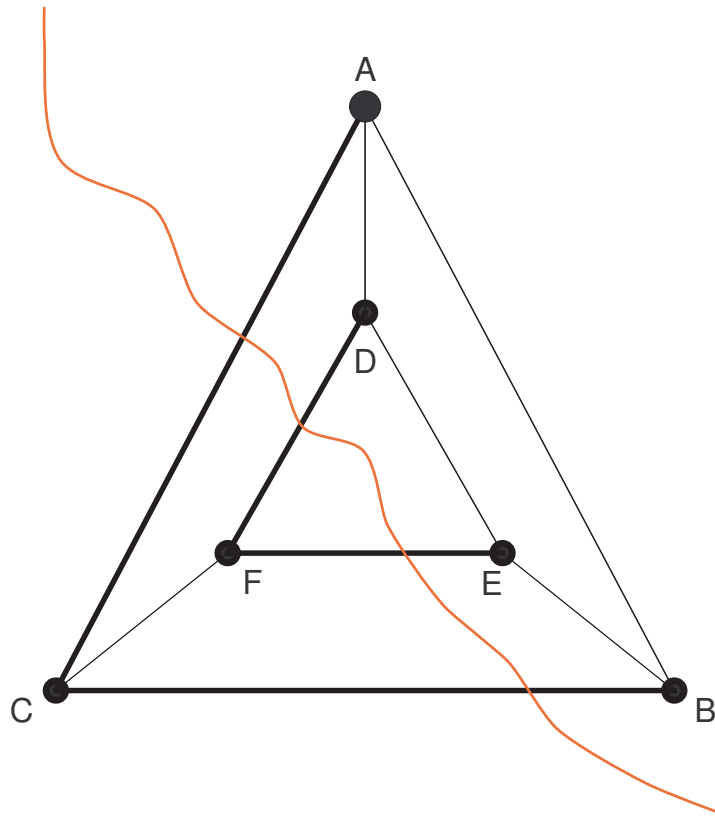


$$W(T) = 13$$

Cuts

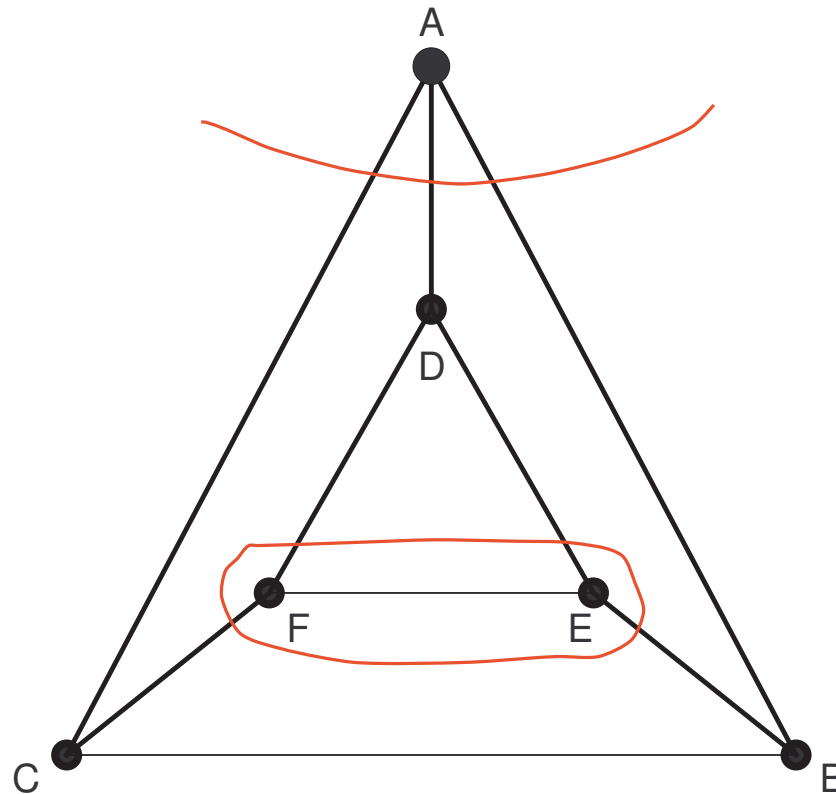
- ★ A **cut** $\langle S, (V - S) \rangle$ in a graph is a partition of the set of vertices V into two disjoint sets: $V = S \cup (V - S)$.
- ★ An edge (u, v) **crosses** a cut $\langle S, (V - S) \rangle$ if $(u \in S \ \& \ v \in (V - S))$ or $(v \in S \ \& \ u \in (V - S))$.
- ★ An edge (u, v) is **contained** in a cut $\langle S, (V - S) \rangle$ if $(u, v) \in S$ or $(u, v) \in (V - S)$.
- ★ A set A of edges is **contained** in a cut $\langle S, (V - S) \rangle$ if all of the edges of A are contained in the cut.

Example: A Cut



An $\langle \{ABDE\}, \{CF\} \rangle$ cut

Example: Another Cut



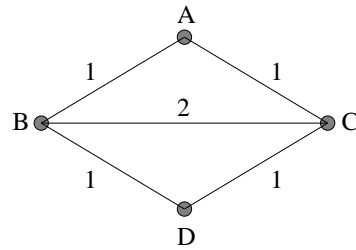
An $\langle \{AEF\}, \{BCD\} \rangle$ cut

Minimal Forests

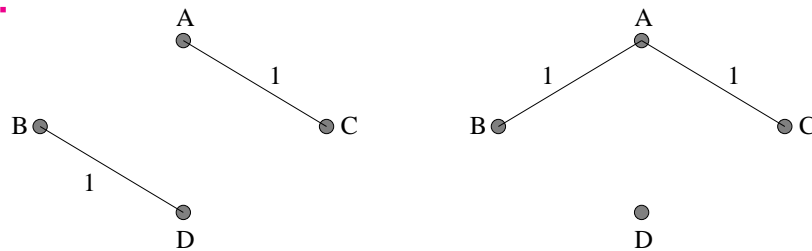
- ★ A set A of edges is a **minimal forest** if it is possible to add edges to A to become a **minimal spanning tree**.
- ★ An **empty** set is in particular a minimal forest.
- ★ A **minimum spanning tree** is in particular a minimal forest.

A Minimal Forest

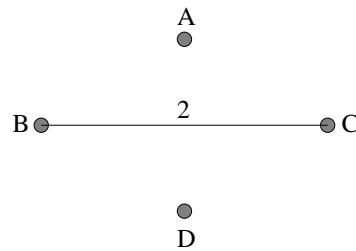
A weighted graph:



Minimal forests:



Not a minimal forest:

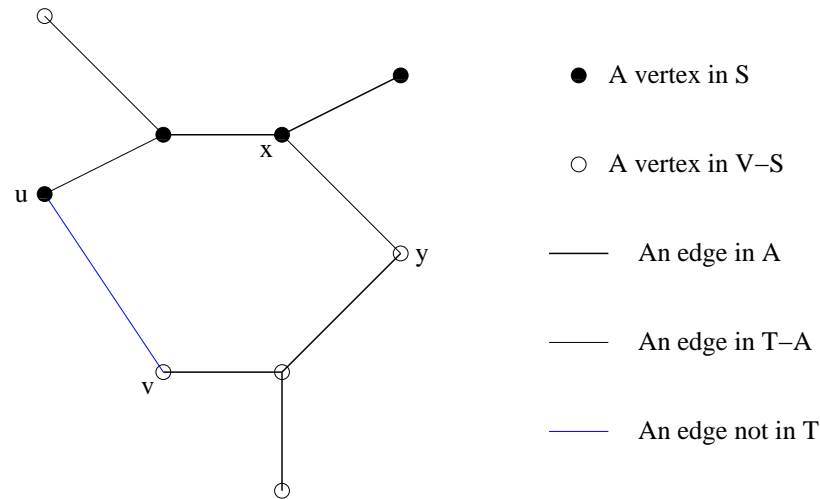


The Greedy Lemma

Lemma:

- ★ A weighted connected **graph** $G = (V, E)$.
 - ★ A **minimal forest** $A \subset E$.
 - ★ A **cut** $\langle S, (V - S) \rangle$ that contains A .
 - ★ An edge (u, v) **crossing** $\langle S, (V - S) \rangle$ with **minimum** weight.
- $\implies A \cup \{(u, v)\}$ is also a **minimal forest**.

Proof



- ★ An MST T that **contains** A and $(u, v) \notin T$.
- ★ $(x, y) \in T$ **crossing** $\langle S, (V - S) \rangle \Rightarrow w(u, v) \leq w(x, y)$.
- ★ $T' = T - \{(x, y)\} \cup \{(u, v)\} \Rightarrow T'$ is also an MST.
- ★ $\Rightarrow A \cup \{(u, v)\}$ is a **minimal forest**.

A Schematic Greedy Algorithm

- (1) $A = \emptyset$
- (2) **while** $|A| < n - 1$ **do**
- (3) **find** (u, v) s.t. $A \cup \{(u, v)\}$ is a minimal forest
- (4) $A = A \cup \{(u, v)\}$
- (5) **return**(A)

A Schematic Greedy Algorithm

Correctness:

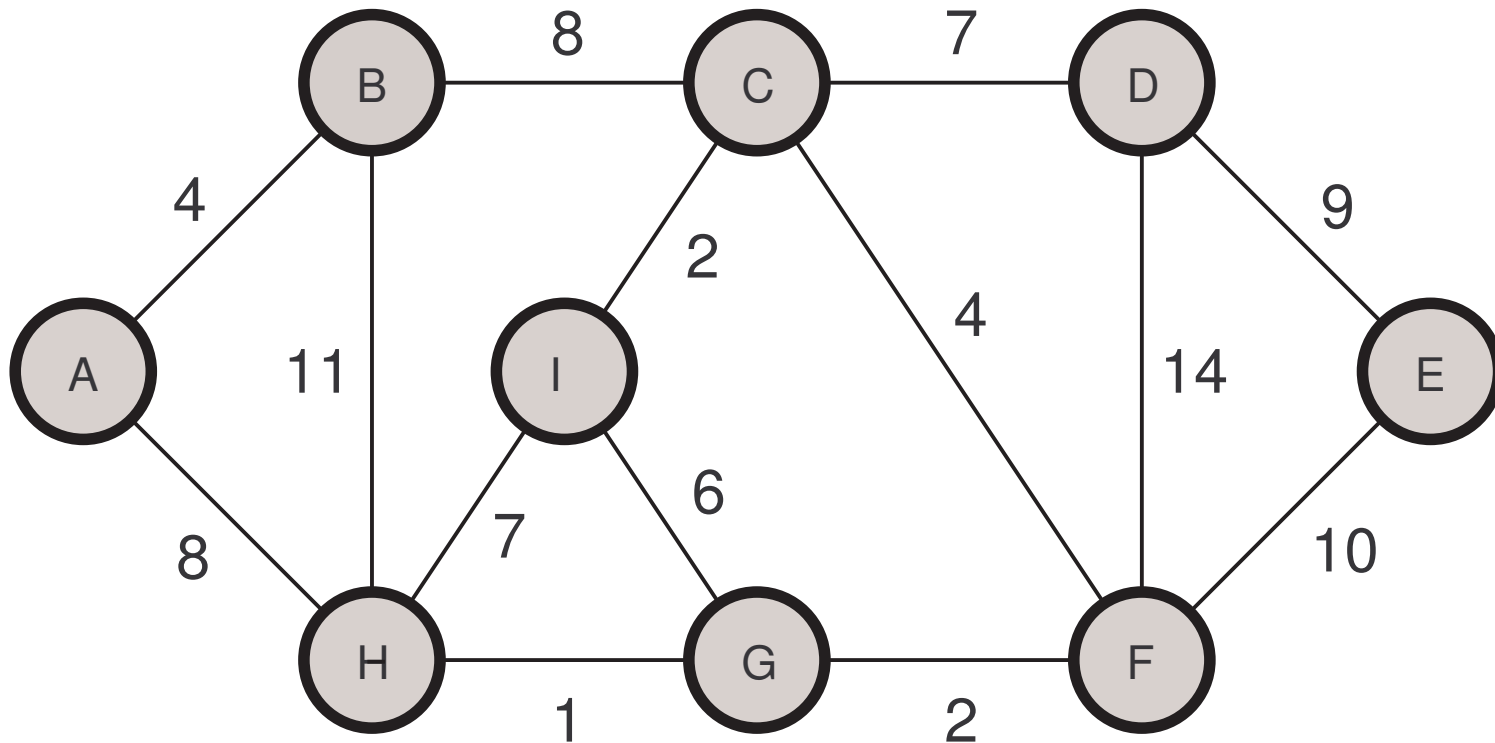
- ★ Due to the lemma, step (3) is **always** possible.
- ★ By definition, A is **always** a minimal forest.
- ★ At the end A has $n - 1$ edges.
- ★ A forest with $n - 1$ edges is a tree.

Complexity: Depends on the **implementation** of step (3).

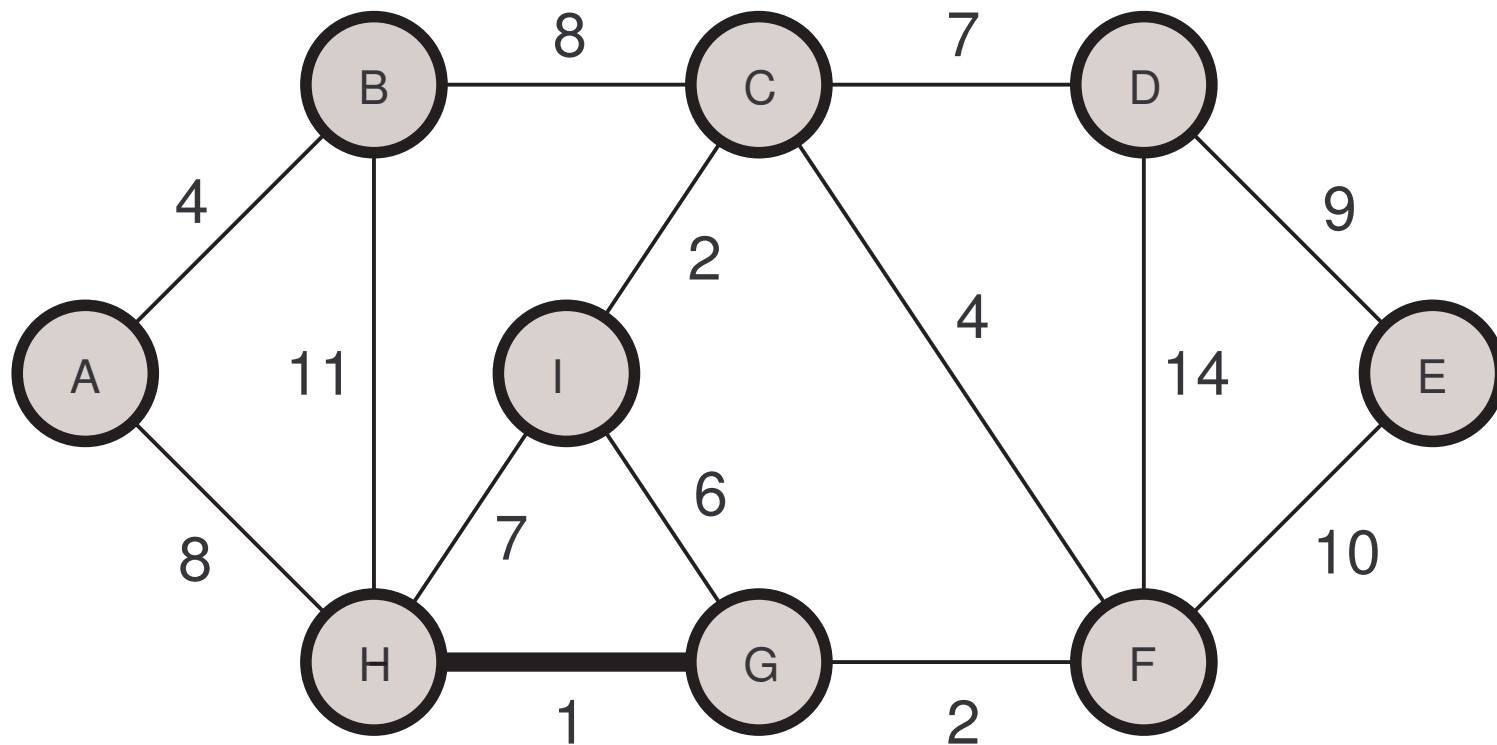
Kruskal Algorithm

- ★ Sort the edges from the lightest to the heaviest.
- ★ Consider the edges following this order.
- ★ Start with an empty minimal forest.
- ★ Add an edge to the minimal forest if it doesn't close a cycle.
- ★ Terminate when there are $n - 1$ edges in the minimal forest.

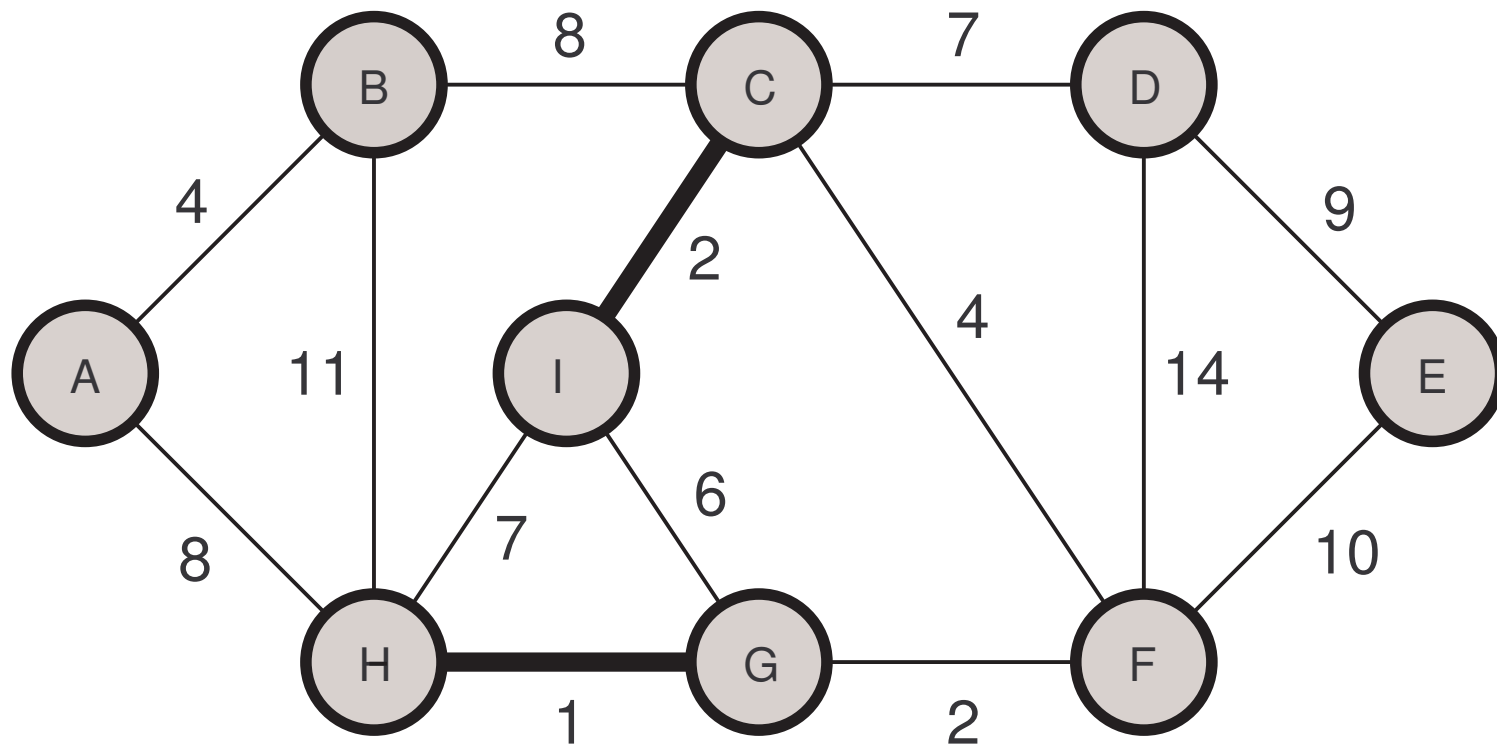
Example: Kruskal Algorithm



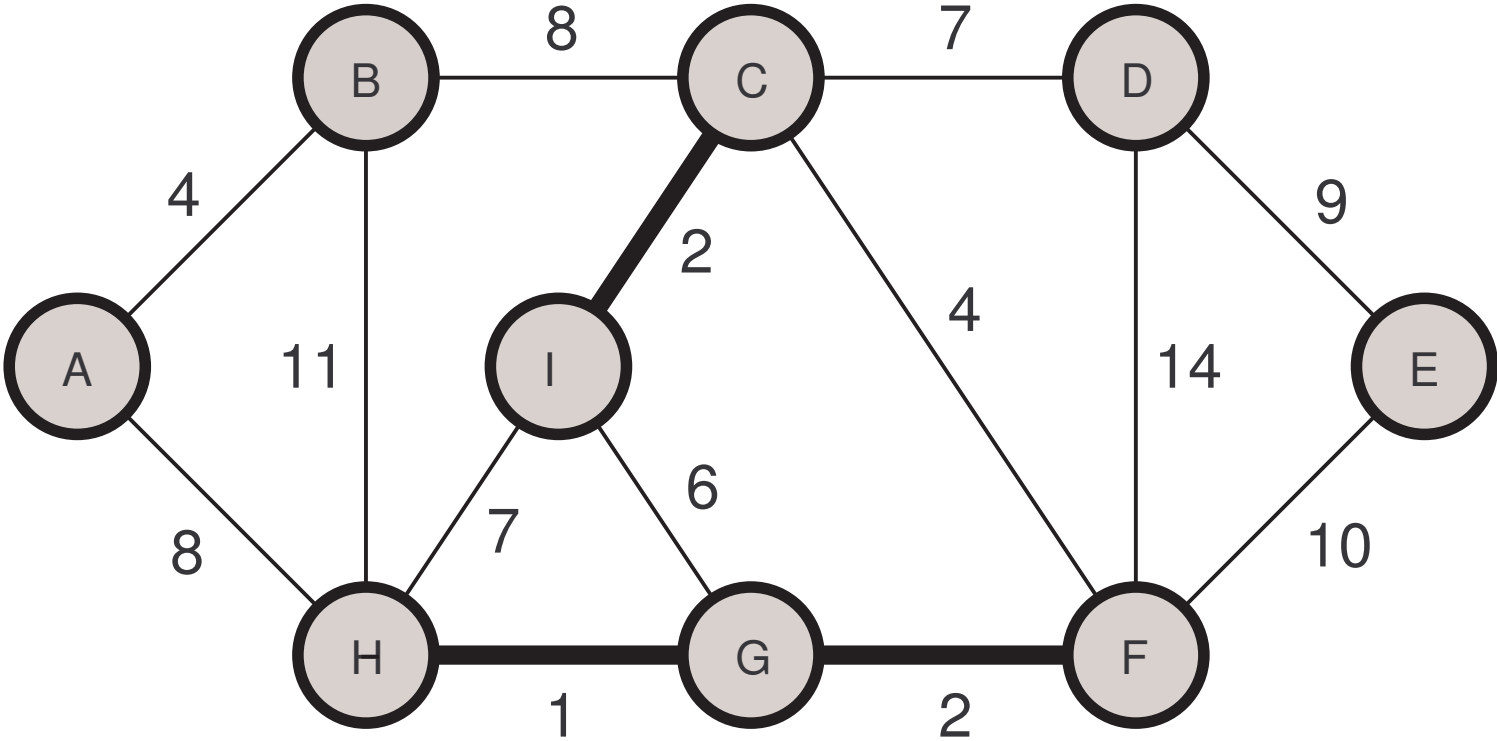
Example: Kruskal Algorithm



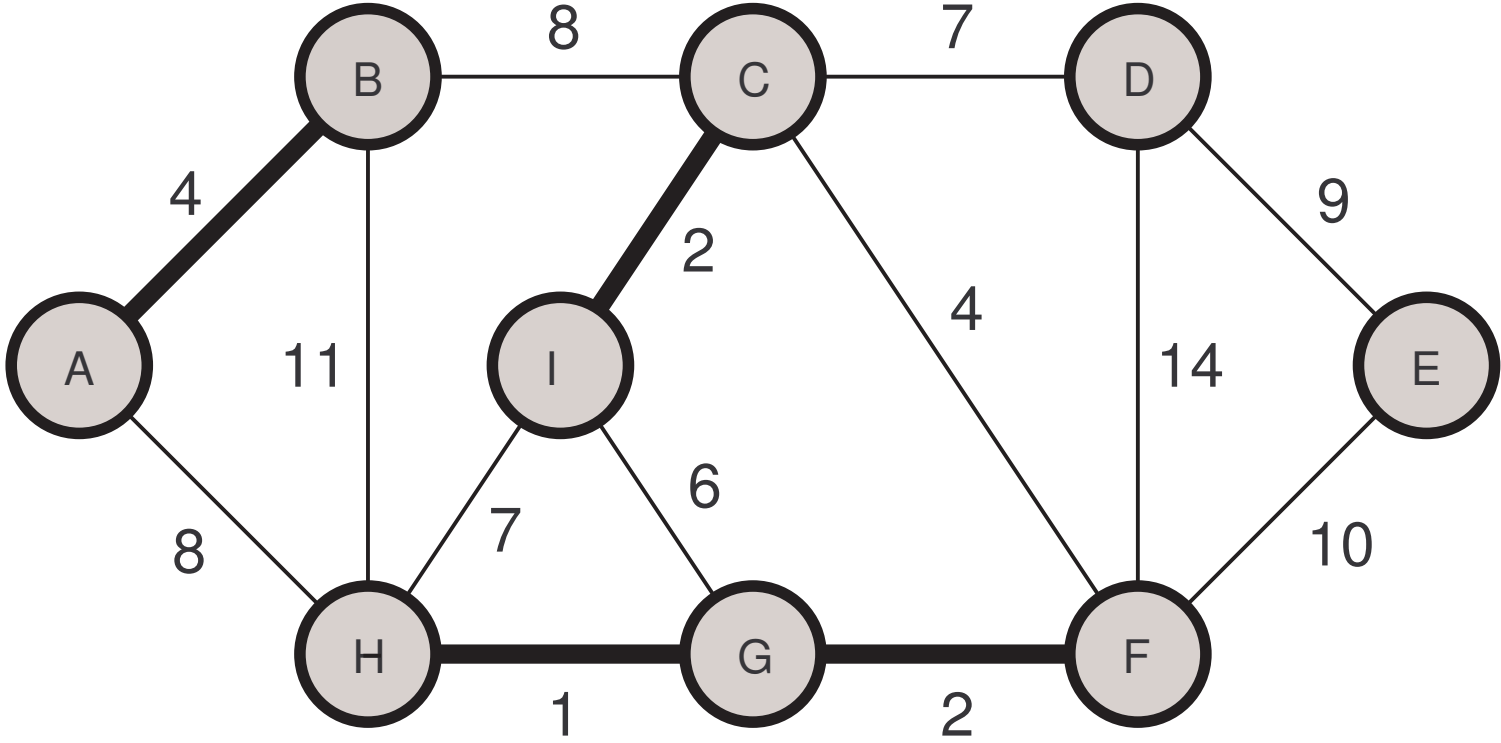
Example: Kruskal Algorithm



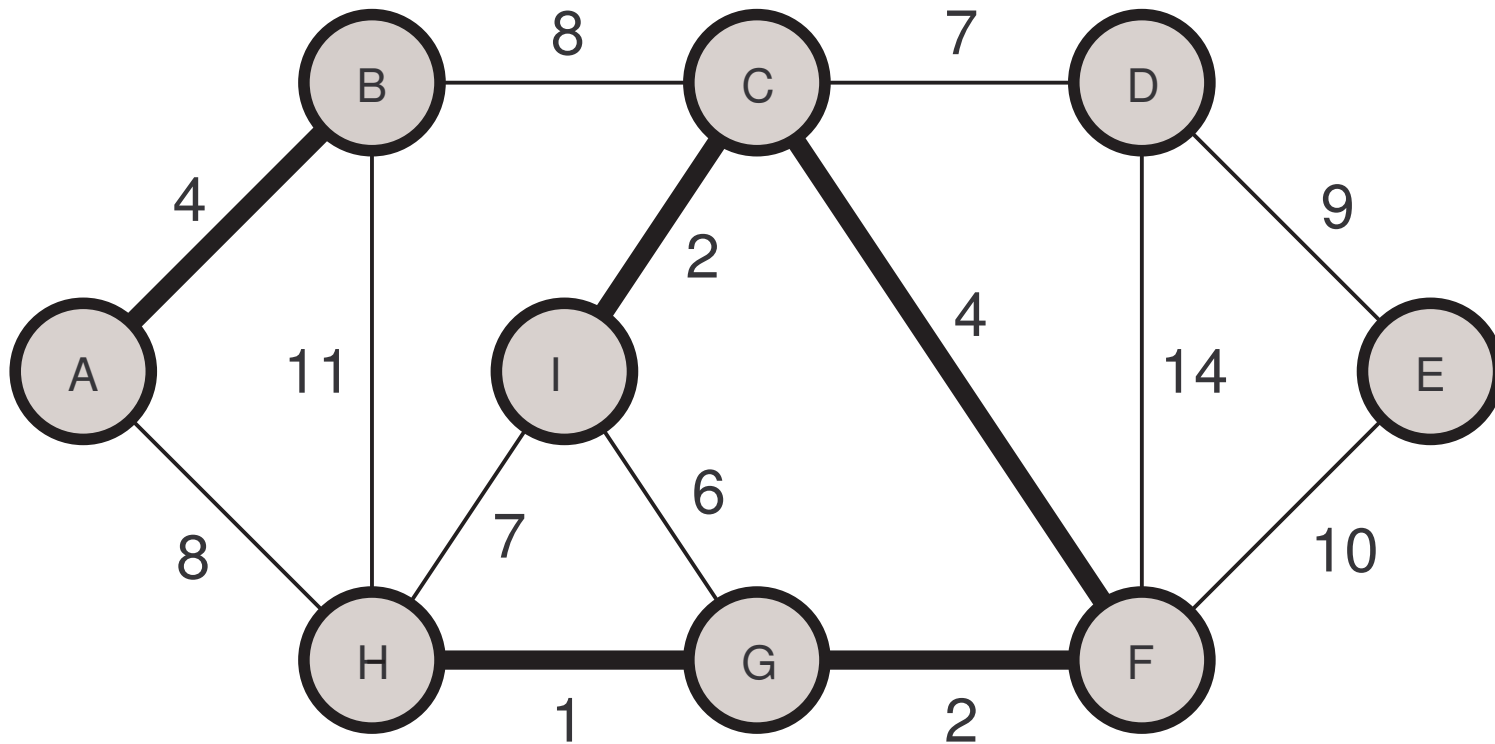
Example: Kruskal Algorithm



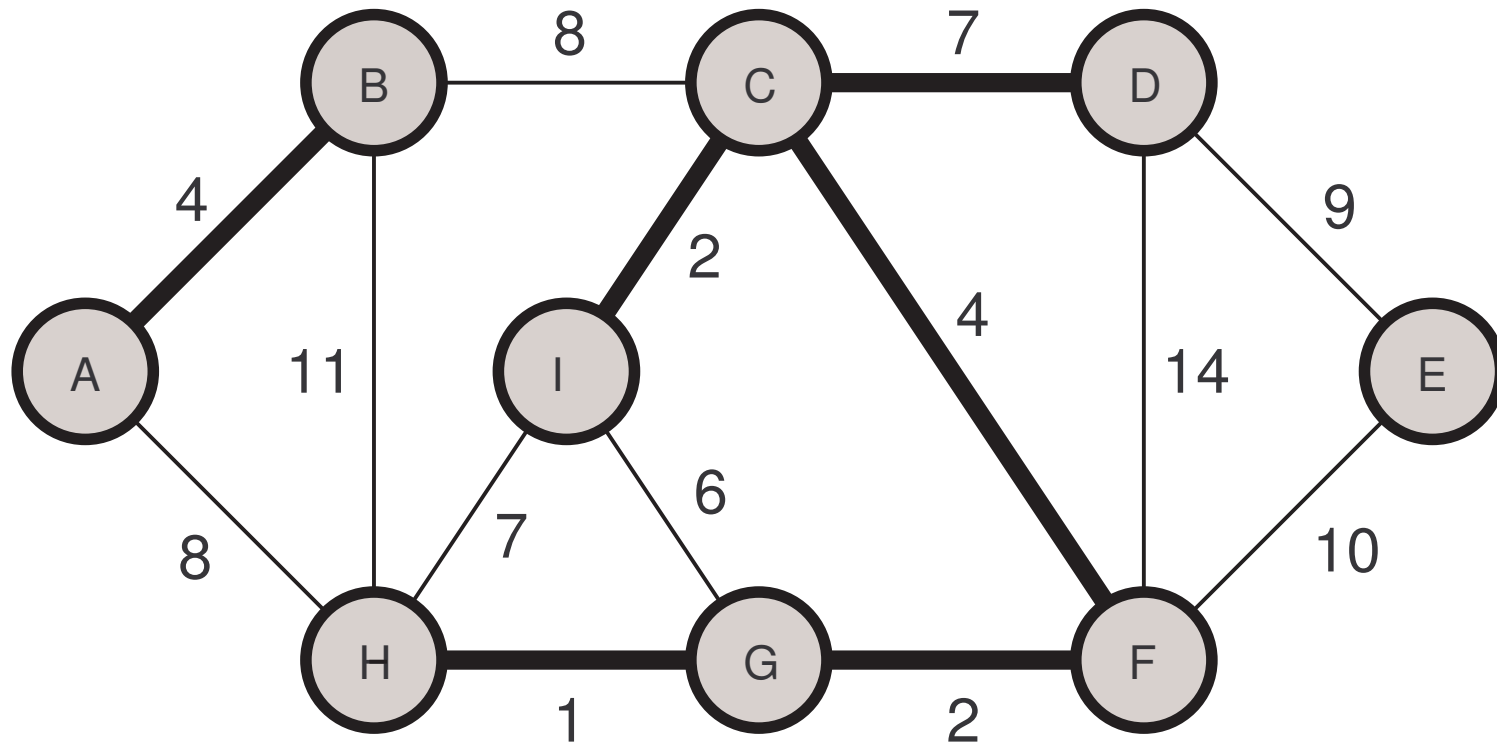
Example: Kruskal Algorithm



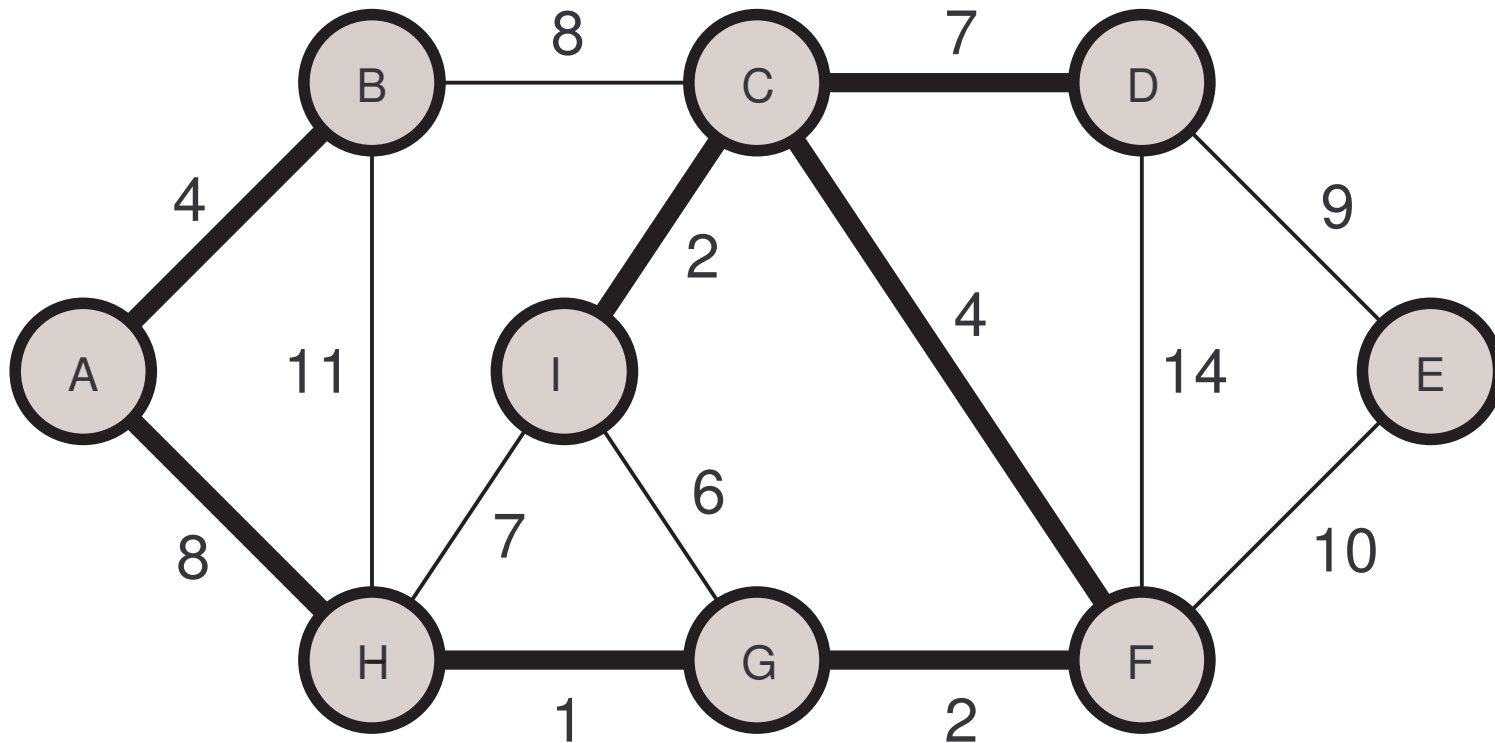
Example: Kruskal Algorithm



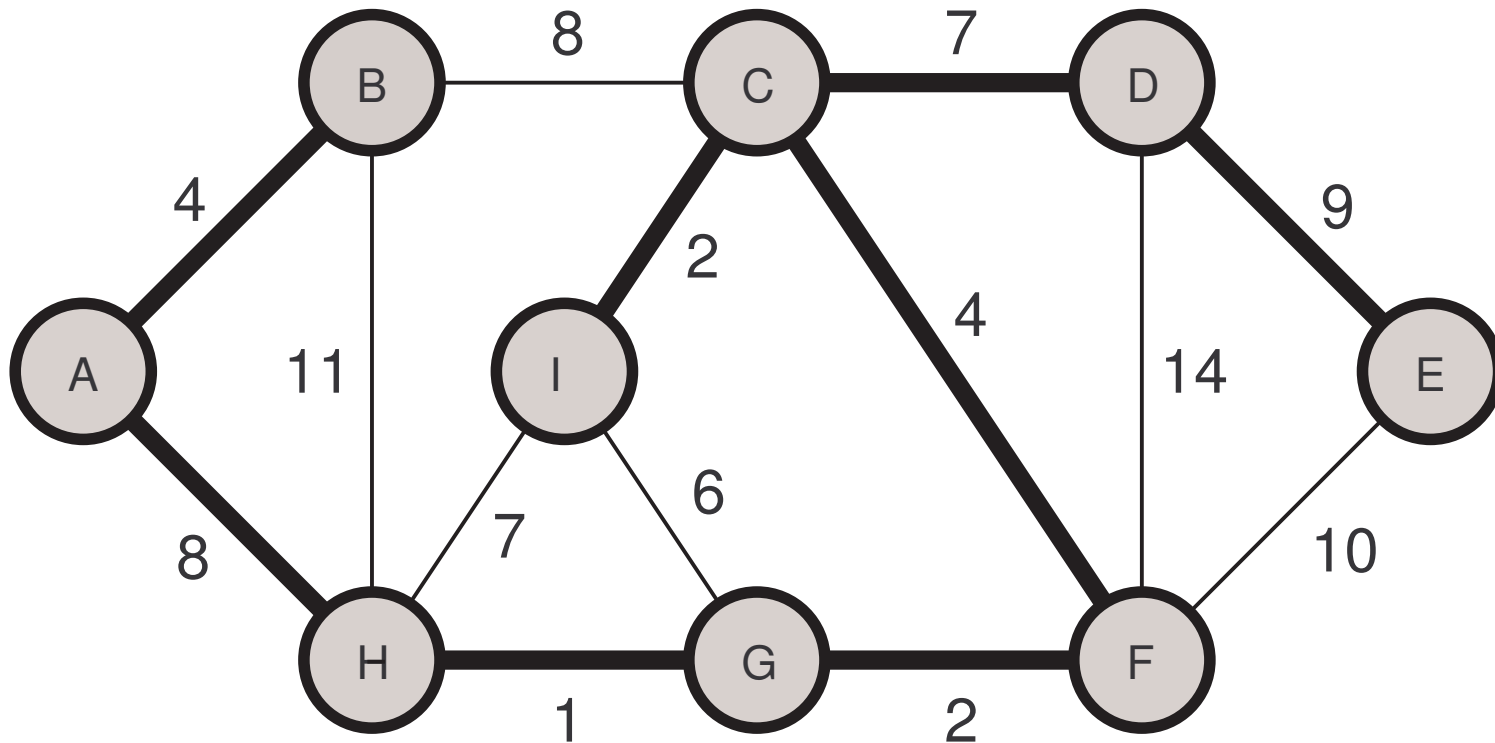
Example: Kruskal Algorithm



Example: Kruskal Algorithm



Example: Kruskal Algorithm



Kruskal Algorithm – Data Structure

- ★ A collection \mathcal{S} of **disjoint** sets of vertices: $\{S_1, S_2, \dots, S_k\}$.
 - $S_i \cap S_j = \emptyset$ for all $1 \leq i \neq j \leq k$.
- ★ **Initially**, the collection is empty: $\mathcal{S} = \emptyset$.
- ★ **At the end** \mathcal{S} contains one set of all the vertices: $\mathcal{S} = \{V\}$.
- ★ **Make-Set(x)**: Creates a new set containing only x and adds it to the collection \mathcal{S} : $\mathcal{S} = \mathcal{S} \cup \{\{x\}\}$.
- ★ **Find-Set(x)**: Finds the set in the collection \mathcal{S} that contains x : $S_i \in \mathcal{S}$ such that $x \in S_i$.
- ★ **Union(S_i, S_j)**: replaces in the collection \mathcal{S} the two sets S_i and S_j with their union: $\mathcal{S} = \mathcal{S} - \{S_i, S_j\} \cup \{S_i \cup S_j\}$.

Kruskal Algorithm – Code

Variables:

- ★ A : A minimal forest.
- ★ Each tree in A is represented by a set of vertices.

Algorithm:

- (1) $A = \emptyset$
- (2) **for all** $v \in V$ **do** **Make-Set**(v)
- (3) **Sort**(E) all edges from *min* to *max*
- (4) **for each** edge (u, v) in sorted order **do**
- (5) **if** **Find-Set**(u) \neq **Find-Set**(v) **then**
- (6) $A = A \cup \{(u, v)\}$
- (7) **Union**(**Find-Set**(u), **Find-Set**(v))
- (8) **return** (A)

Kruskal Algorithm – Correctness

- * Assume to the **contrary** that the output set A is not an MST.
- * Let (u, v) be the **first** edge that was added to A such that $A \cup \{(u, v)\}$ is not a **minimal forest**.
 - In particular, at that time, A is a **minimal forest**.
- * Let $\mathcal{C} = \langle S, (V - S) \rangle$ be a cut for the set $u \in S$.
- * (u, v) crosses \mathcal{C} since v belongs to another set.

Kruskal Algorithm – Correctness

- * Any **lighter** edge (x, y) is contained in \mathcal{C} :
 - If $(x, y) \notin A$, then both x and y belong to the same set **before** (x, y) was examined.
 - If $(x, y) \in A$, then both x and y belong to the same set **after** (x, y) was examined.
- * Sorting $\Rightarrow (u, v)$ is a **minimal crossing edge** for the cut \mathcal{C} .
- * Greedy lemma $\Rightarrow A \cup \{(u, v)\}$ is a **minimal forest**.
- * A **contradiction**.

Kruskal Algorithm – Complexity

- ★ **Sorting complexity:** $O(m \log m)$.
- ★ **Set operations complexity:**
 - There are n **Make-Set** operations.
 - There are $O(m)$ **Find-Set** operations.
 - There are $n - 1$ **Union** operations.
 - Possible to **implement** in time: $O(m \cdot \alpha(m, n))$.
 - * $\alpha(m, n)$ is the inverse of the Ackerman's function.
 - * A function that **grows** very slowly.
 - * For example, $m, n \approx 10^{80} \Rightarrow \alpha(m, n) \leq 4$.
- ★ **Overall complexity:** $O(m \log m)$.

The Ackerman's Function

$$A_k(n) = \begin{cases} 2n & \text{for } k = 0 \text{ and } n \geq 0 \\ A_{k-1}(1) & \text{for } k \geq 1 \text{ and } n = 1 \\ A_{k-1}(A_k(n-1)) & \text{for } k \geq 1 \text{ and } n \geq 2 \end{cases}$$

$A_0(n) = 2 + \dots + 2 = 2n$ – The **multiply-by-2** function.

$A_1(n) = 2 \times \dots \times 2 = 2^n$ – The **power-of-2** function.

$A_2(n) = 2^{2^{\ddots^2}}$ – With n 2's, the **tower-of-2** function.

The Ackerman's Function

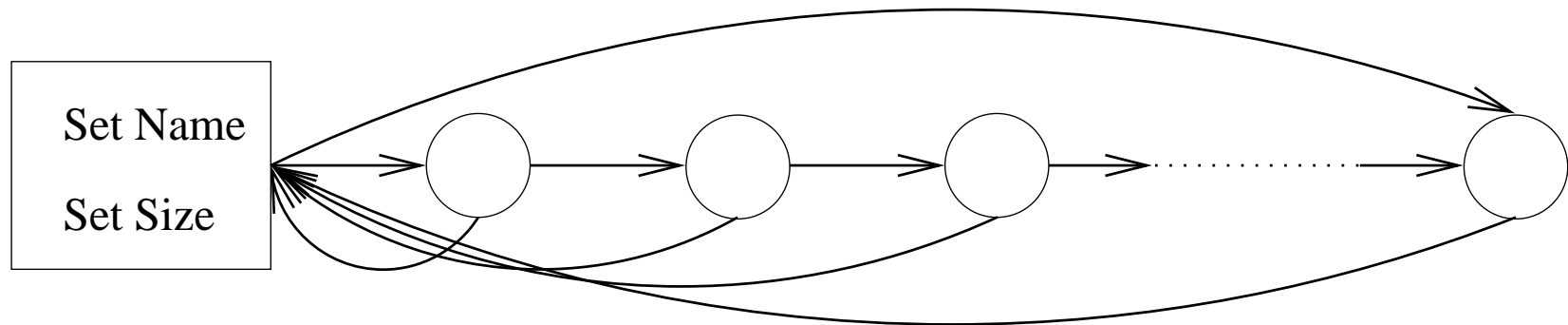
- ★ $A_2(4) = 2^{2^{2^2}} = 2^{16} = 65536$.
- ★ Each recursive level, k , does the operation from the previous level n times.
- ★ What is $A_3(4)$?
- ★ Therefore, $A_4(4)$ must be **extremely** large!

Very Slow Growing Functions

- ★ $\log^* n$ – the inverse of the **tower-of-2** function – is the least x such that $2^{2^{\cdot^{\cdot^2}}}$ x times is greater or equal to n .
 - $\log^*(2^{65536}) = 5$.
- ★ $\alpha(n)$ – the inverse Ackerman's function – is the least x such that $A_x(x) \geq n$.
 - $\alpha(n)$ is **much** slower than $\log^* n$.

Kruskal Algorithm – A Simple Implementation

A **set** is represented by the following **linked list**:

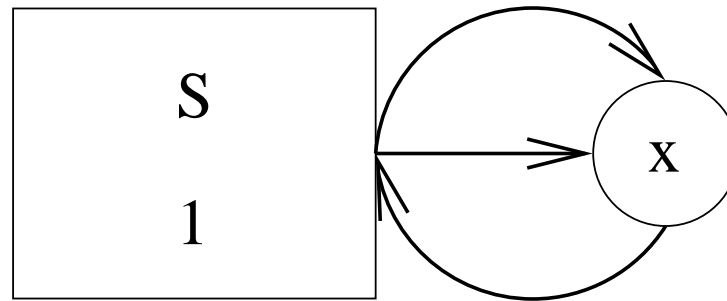


Kruskal Algorithm – A Simple Implementation

- ★ The **head of the set** contains two **fields**:
 - The **name** of the set.
 - The **size** of the set.
- ★ The **head of the set** has two **pointers**:
 - To a **linked list** of **vertices**.
 - To the **last vertex** in the **linked list**.
- ★ The **vertices** are stored in an **array** of size n . Each **vertex** has two **pointers**:
 - To the **head of the set**.
 - To the **next vertex** in the **linked list**.

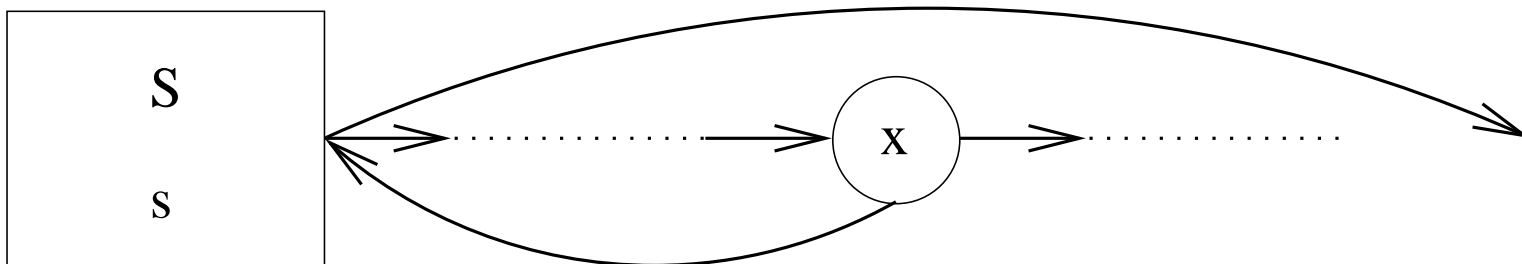
Kruskal Algorithm – A Simple Implementation

Make-Set(x) $\Rightarrow O(1)$ complexity.



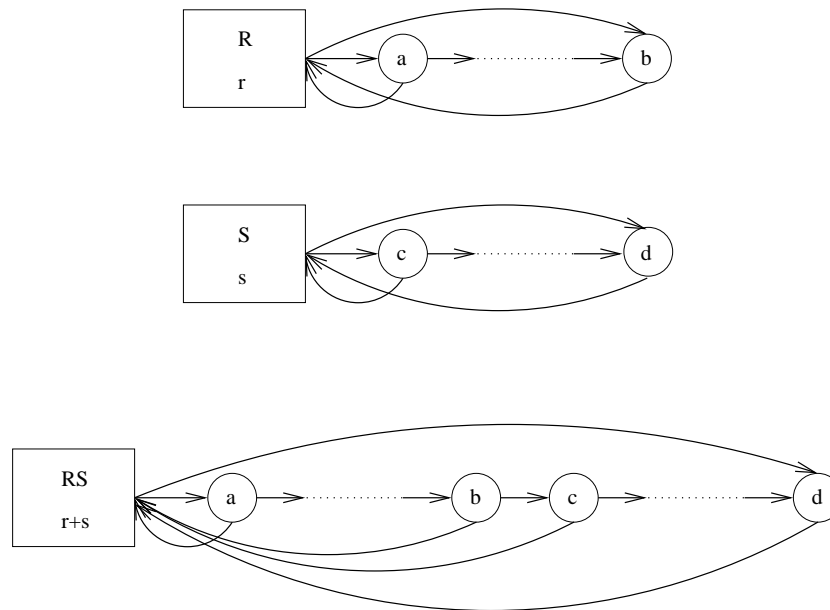
Kruskal Algorithm – A Simple Implementation

Find-Set(x) $\Rightarrow O(1)$ complexity.



Kruskal Algorithm – A Simple Implementation

$\text{Union}(R, S) \Rightarrow O(s)$ complexity.



Kruskal Algorithm – Simple Implementation Complexity

Worst case complexity for the $n - 1$ Union operations:

Union(S_{n-1}, S_n), Union(S_{n-2}, S_{n-1}), \dots , Union(S_1, S_2)

$$\Rightarrow \Omega(1) + \Omega(2) + \dots + \Omega(n - 1) = \Omega(n^2).$$

Kruskal algorithm – Simple Implementation Complexity

A better implementation for the $n - 1$ **Union** operations:

- Connect the **smaller** set to the **larger** set.
- The pointer of each vertex is **changed** at most $\log n$ times, since after each **Union** operation the pointer points to a set whose size is at least **twice** the size of the previous set.
- All together, for **all** vertices, $O(n \log n)$ complexity.

Complexity n **Make-Set** operations: $n \cdot \Theta(1) = \Theta(n)$.

Complexity $O(m)$ **Find-Set** operations: $O(m) \cdot \Theta(1) = O(m)$.

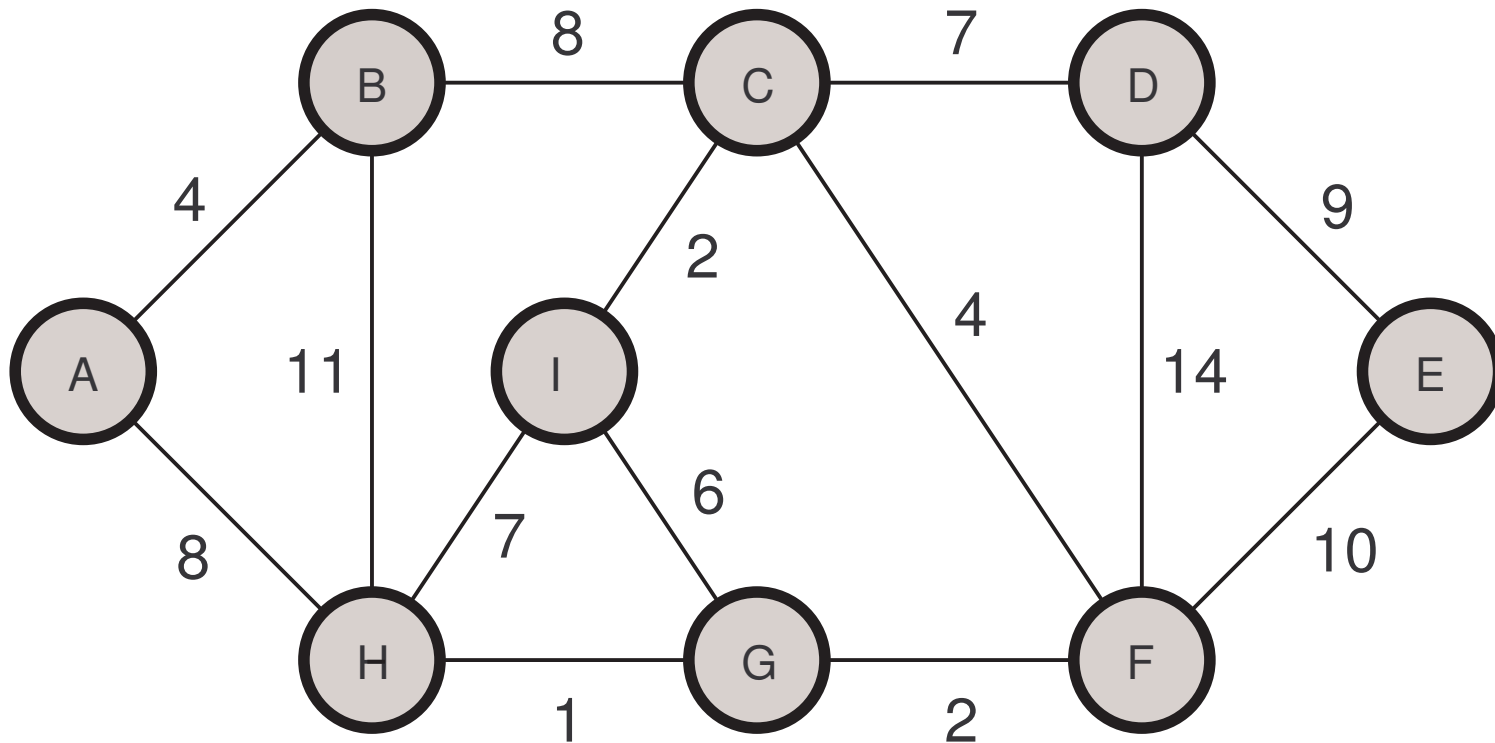
Sorting complexity: $\Theta(m \log m)$.

Kruskal algorithm complexity: $\Theta(m \log m)$.

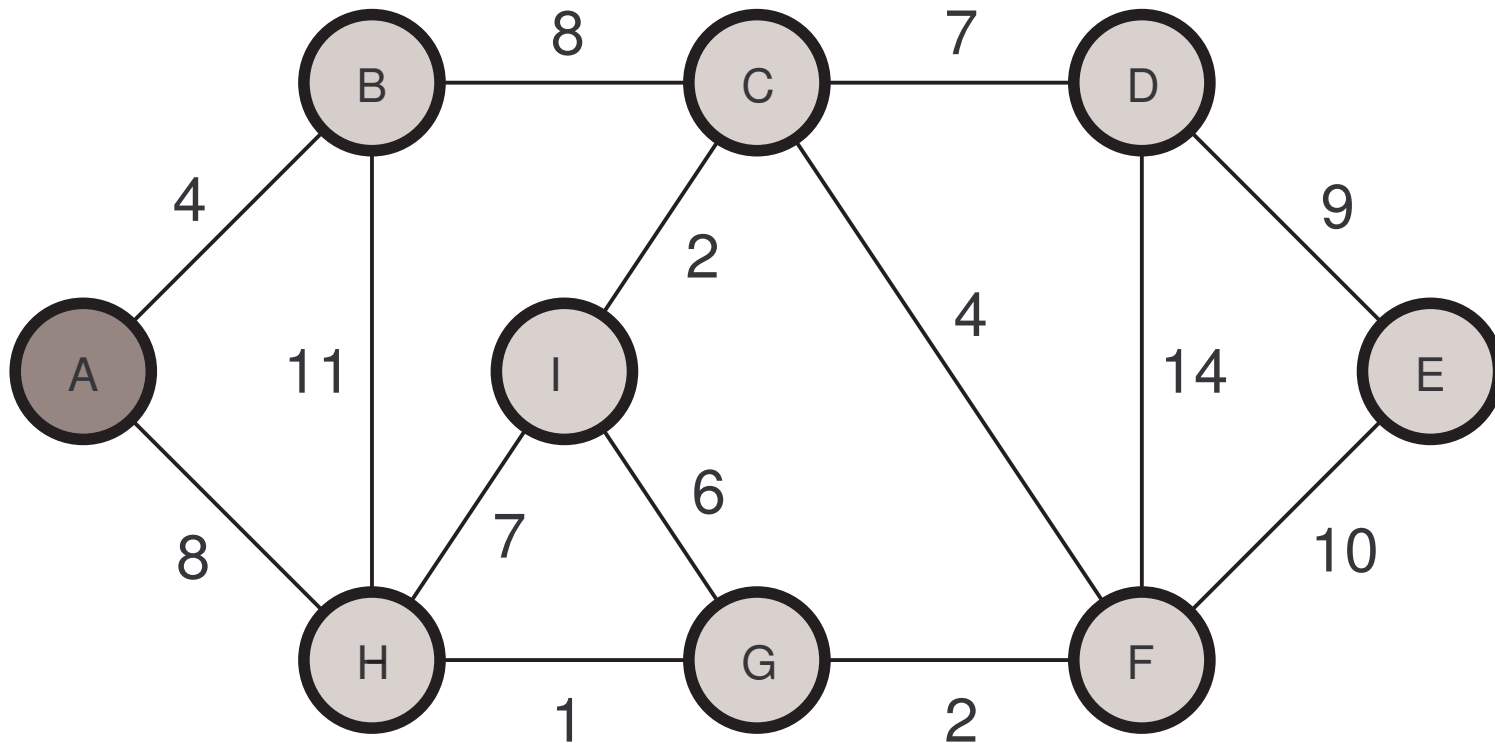
Prim Algorithm

- ★ Start with an arbitrary vertex as a singleton tree.
- ★ Maintain a minimal forest initially set to be this tree.
- ★ Find the **closest** vertex to the minimal forest.
- ★ Add this vertex and its **closest** edge to the minimal forest.
- ★ Continue until all vertices are in the minimal forest.

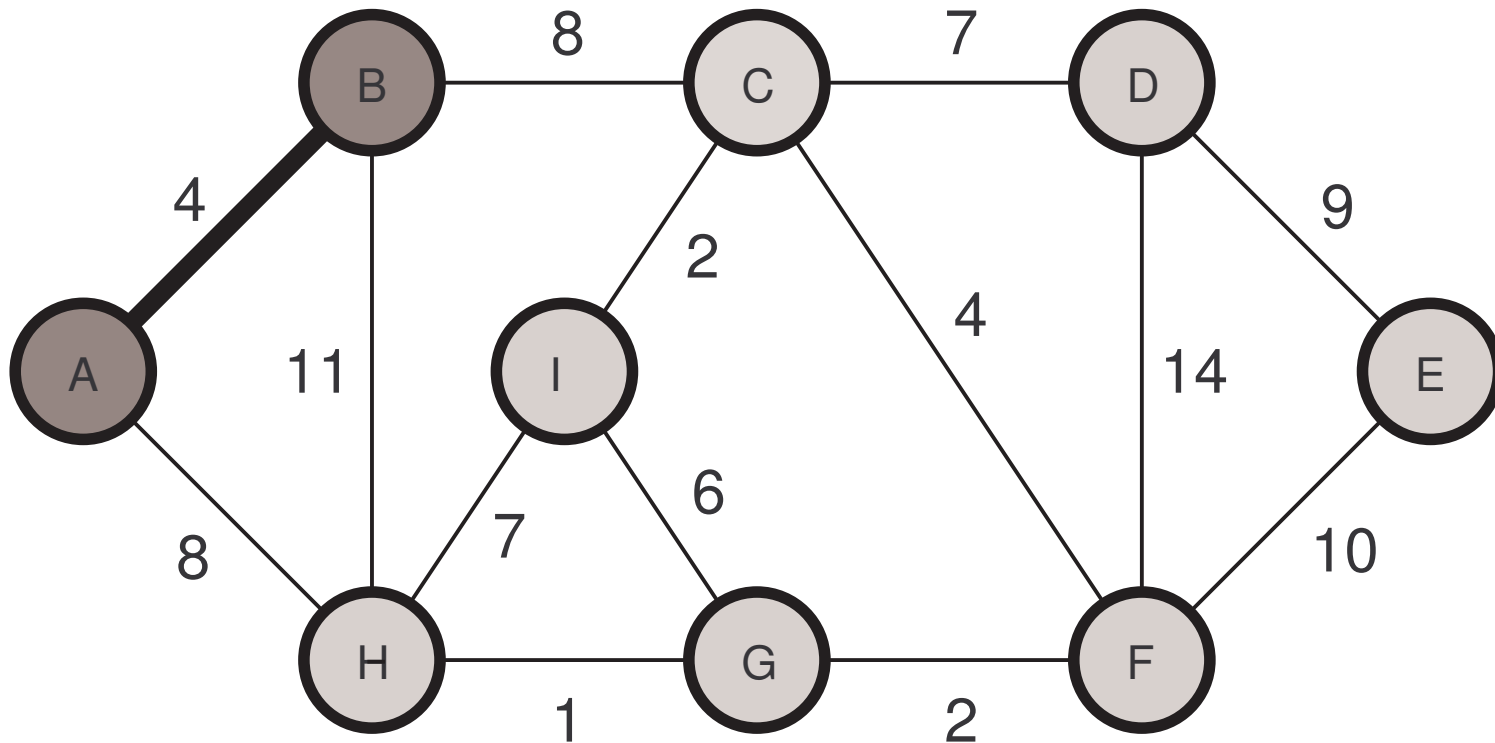
Example: Prim Algorithm



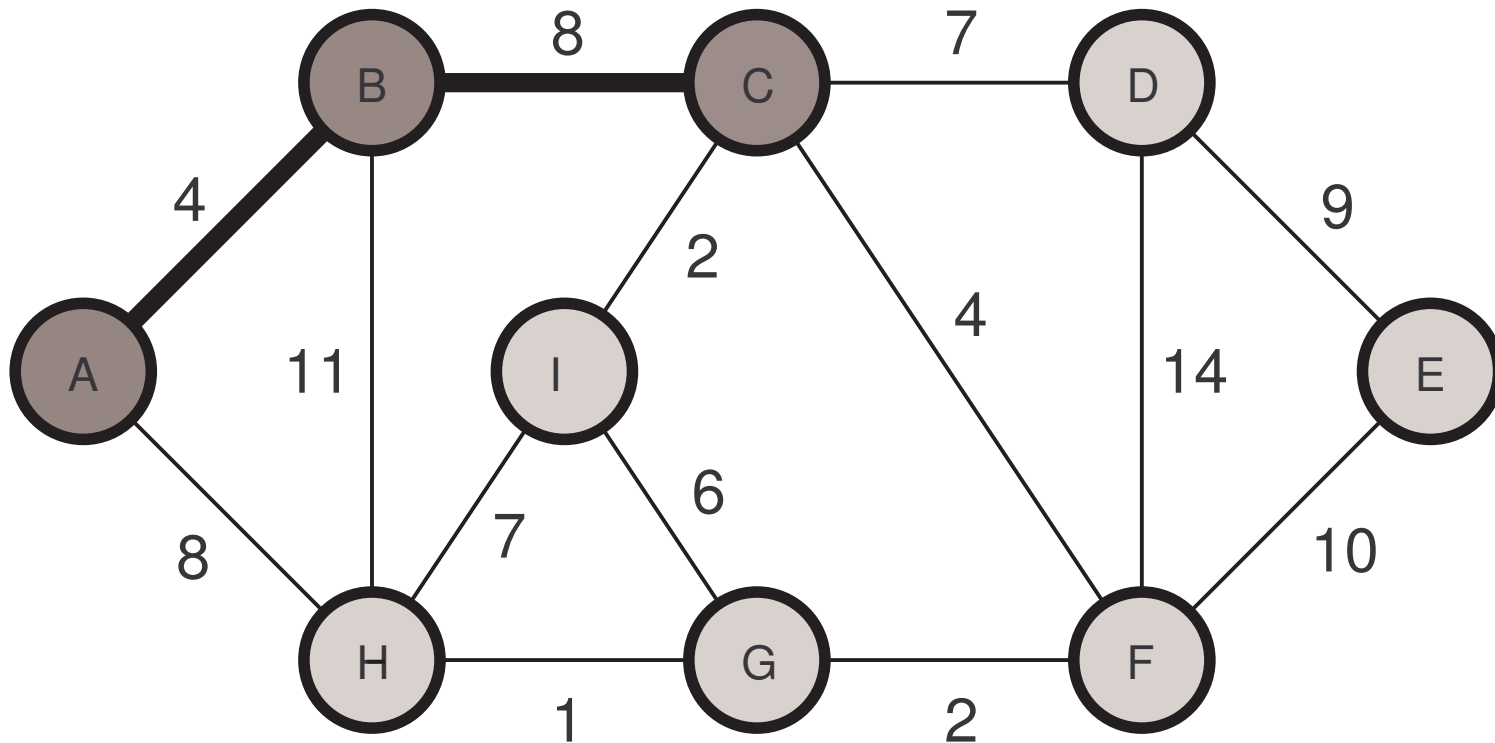
Example: Prim Algorithm



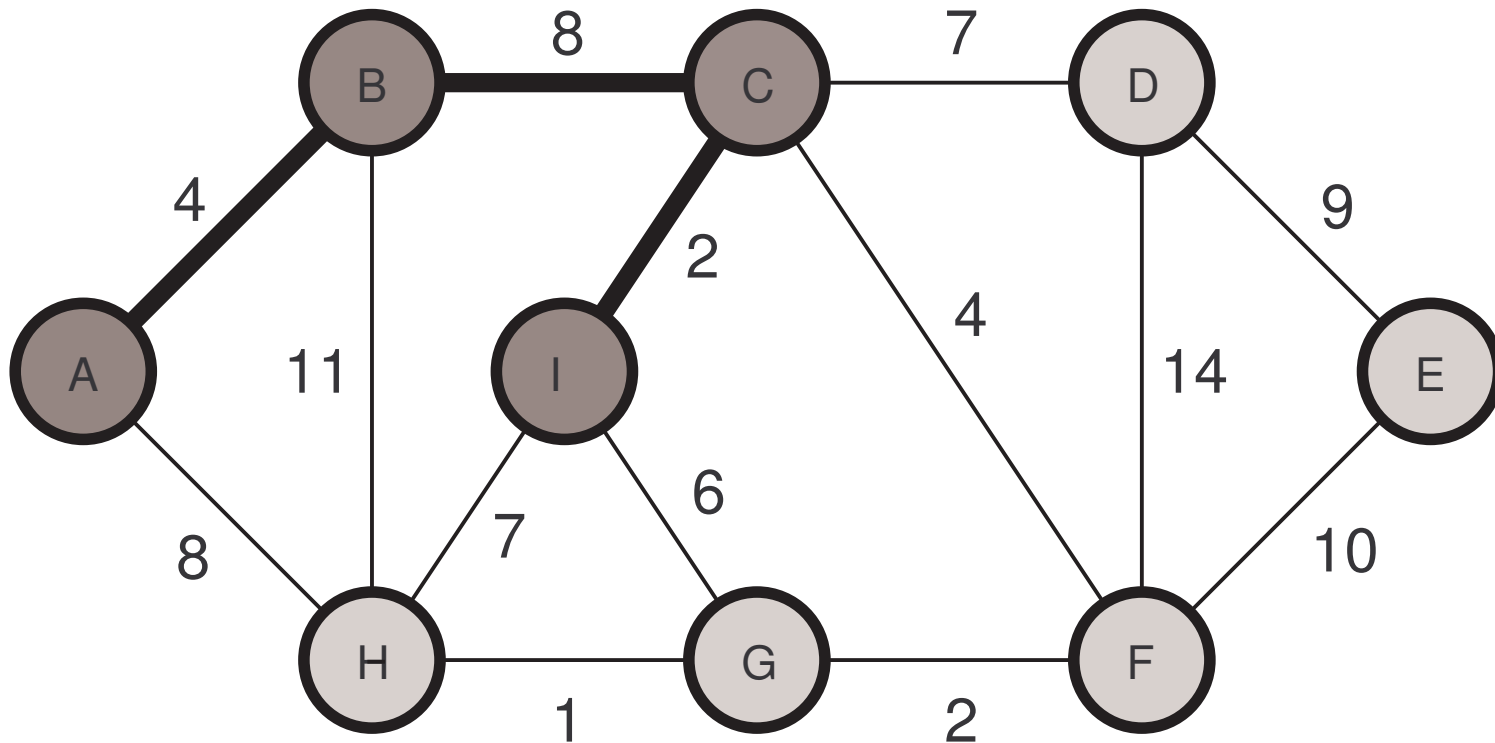
Example: Prim Algorithm



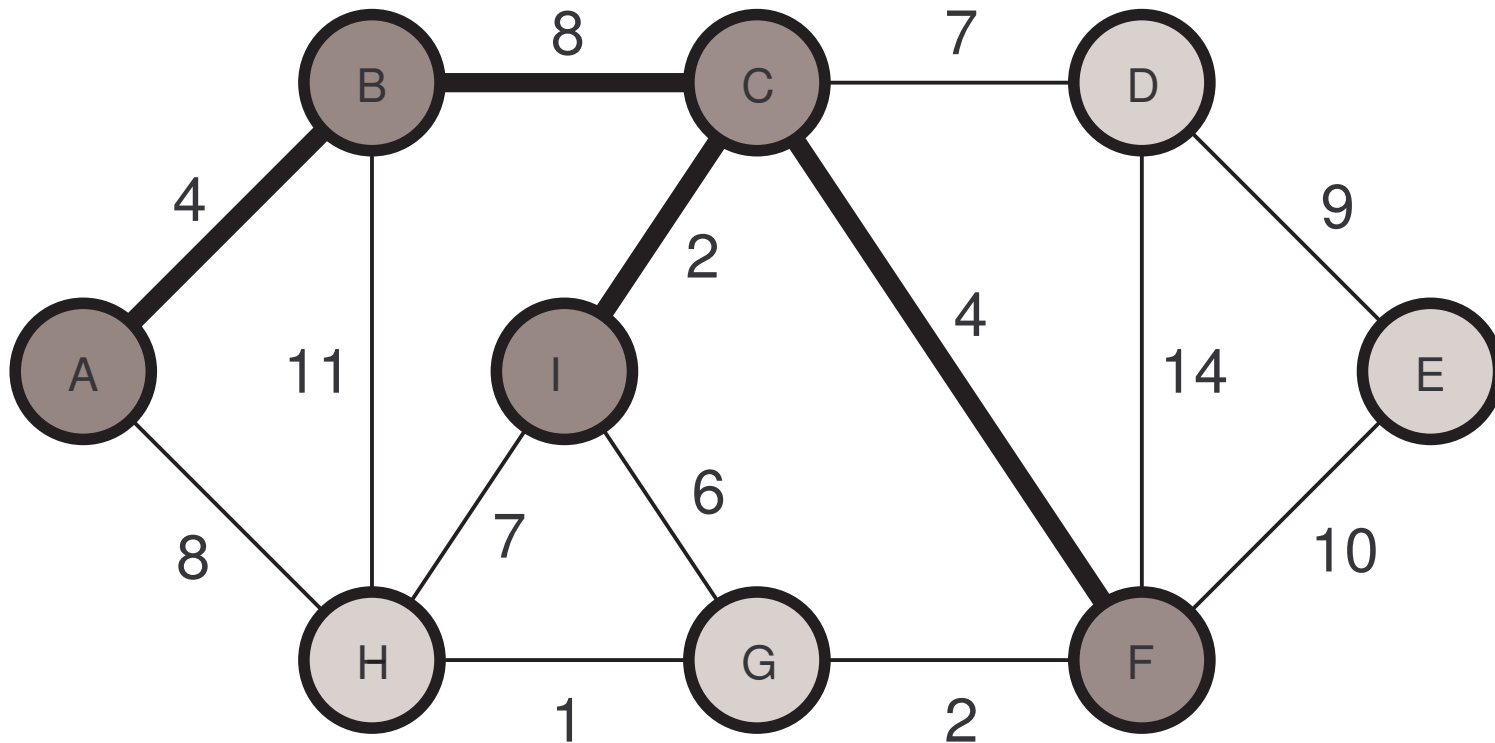
Example: Prim Algorithm



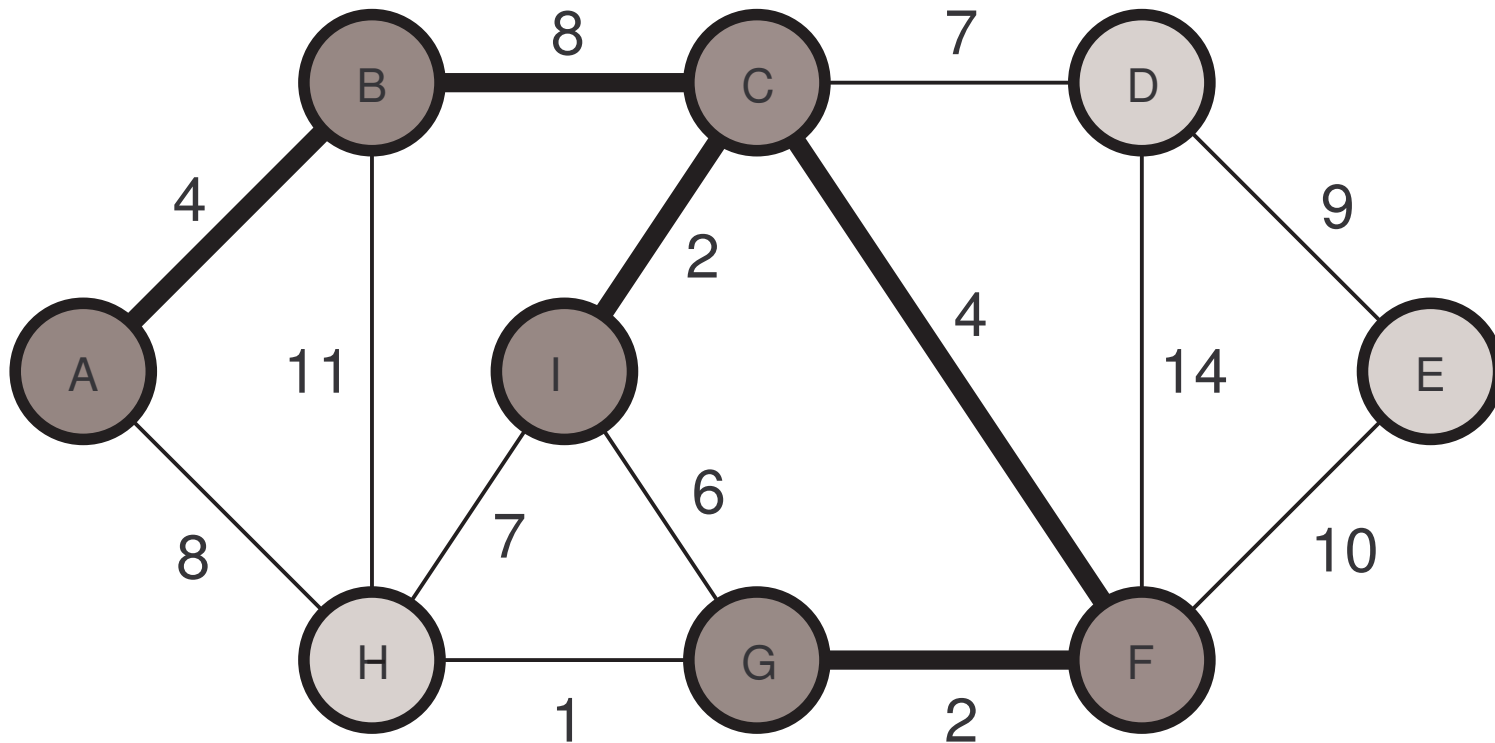
Example: Prim Algorithm



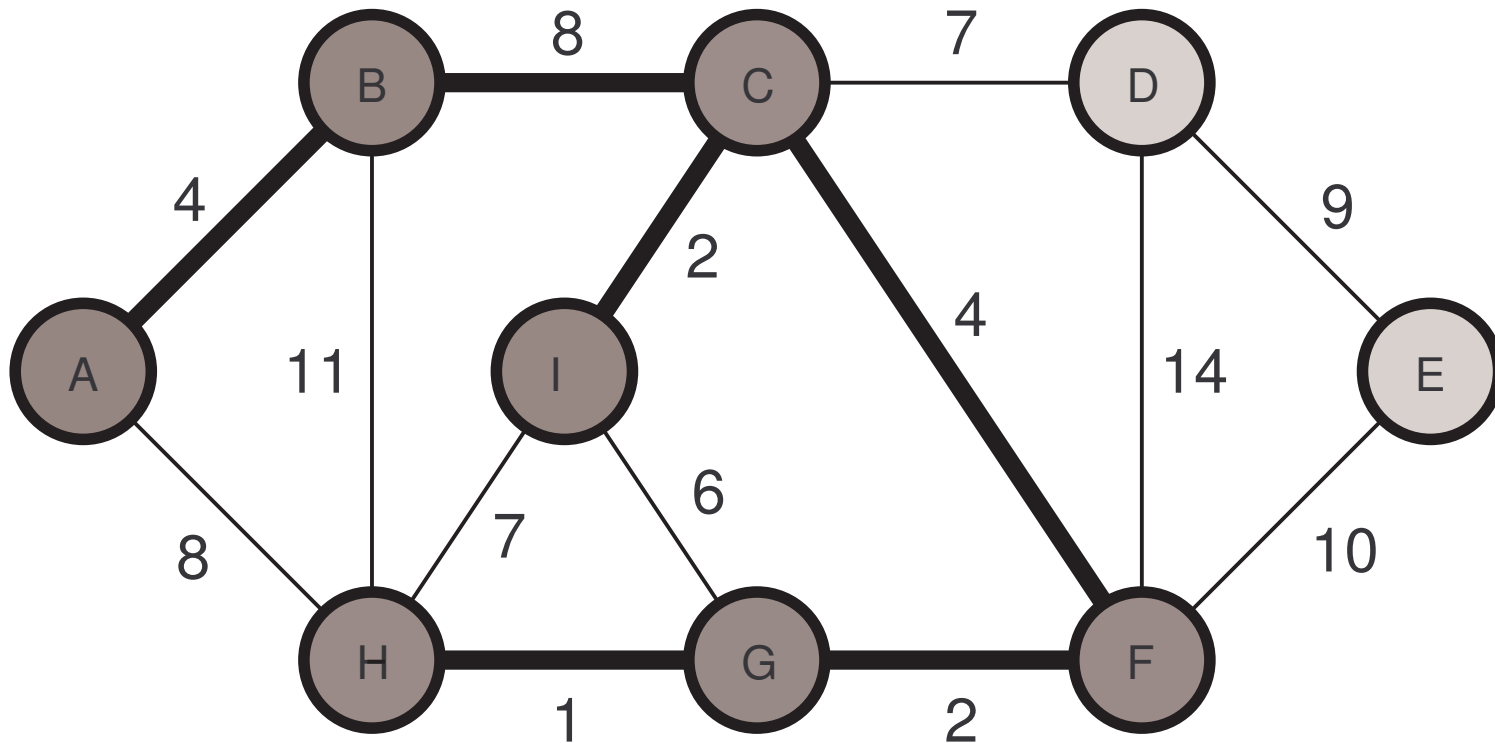
Example: Prim Algorithm



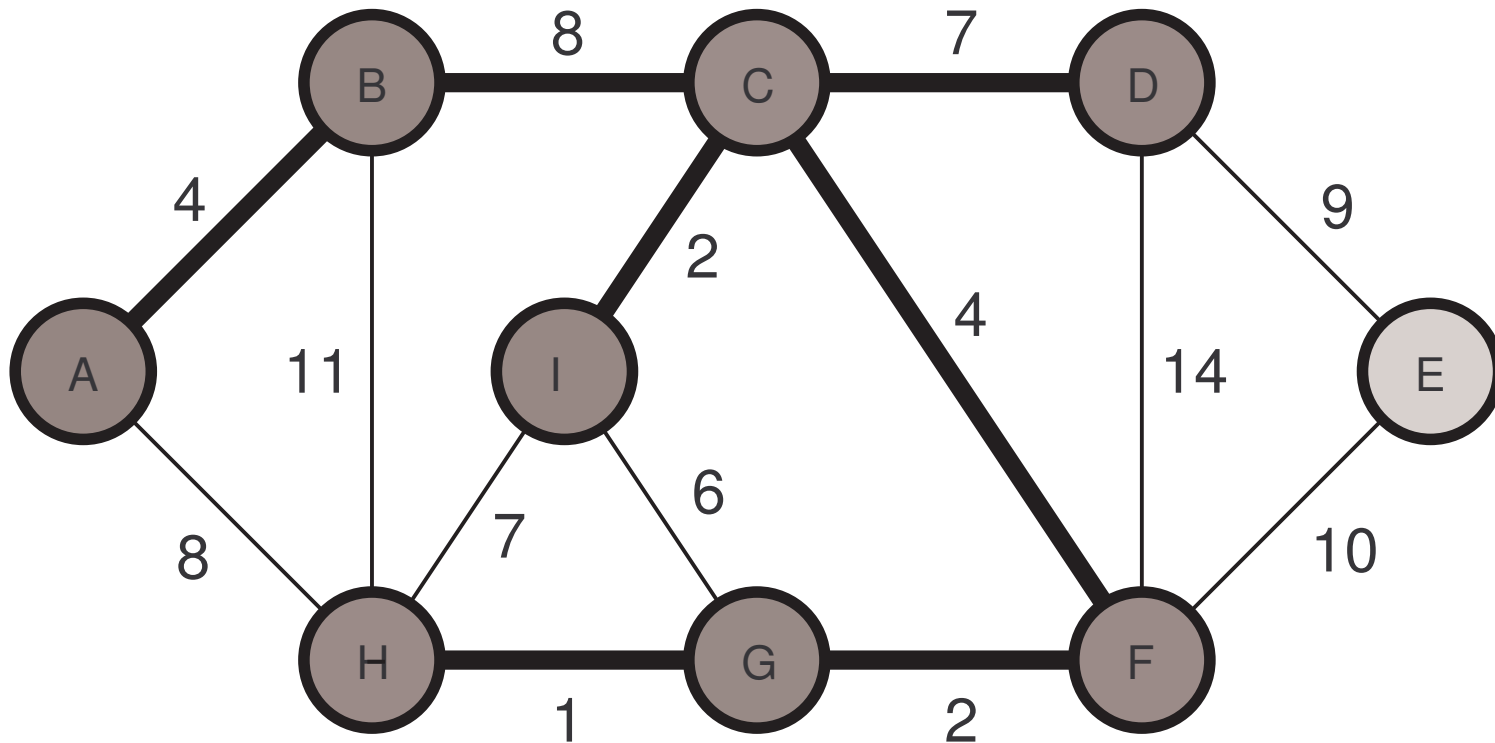
Example: Prim Algorithm



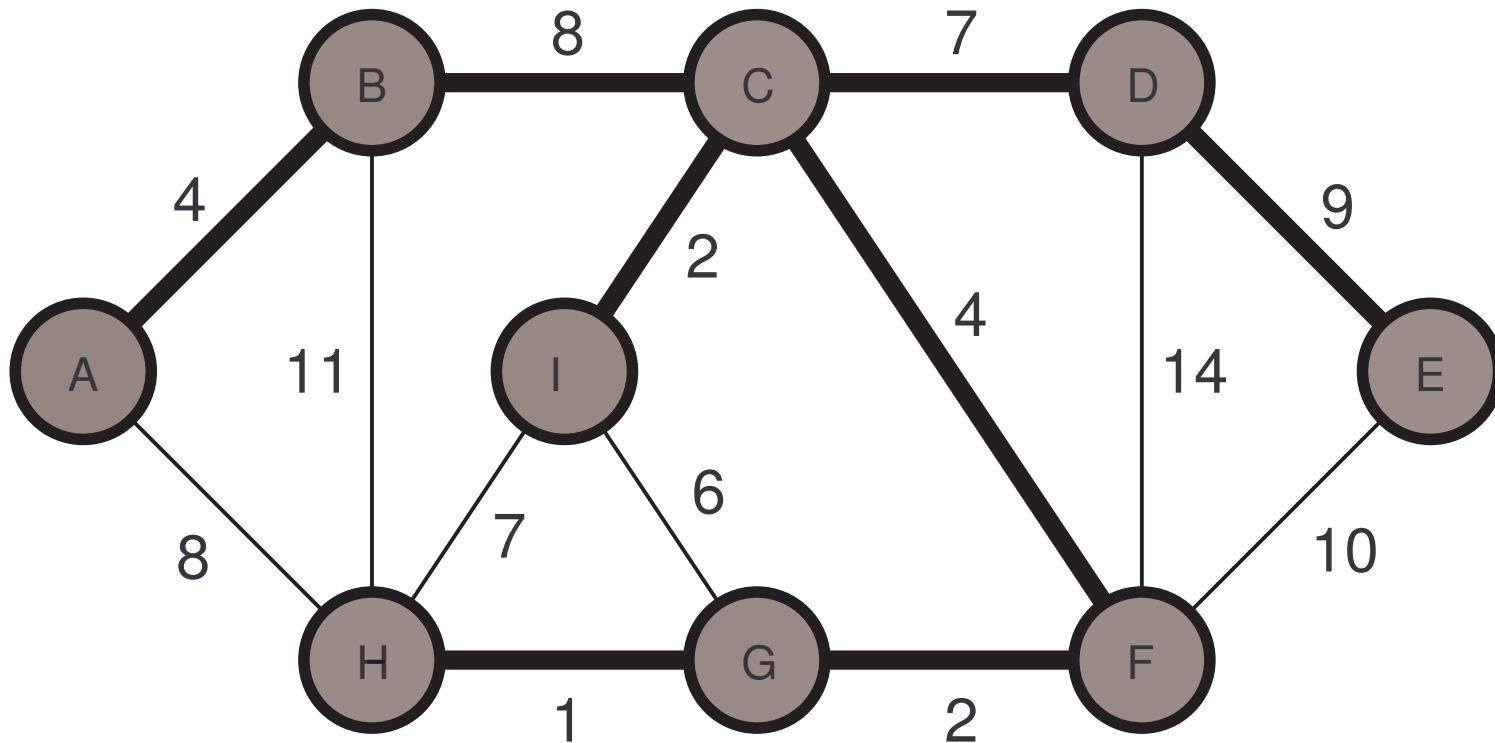
Example: Prim Algorithm



Example: Prim Algorithm



Example: Prim Algorithm



Prim Algorithm – Data Structure

- A **minimal tree (forest)** A that will be the MST.
- A starting vertex r that is the **root** of the MST.
- A **priority queue** Q for vertices not **yet** in A .
- A **distance** $key(v)$ from A for vertices in Q .
- A **candidate edge** $(v, \Pi(v))$ for any vertex v in Q .

Prim Algorithm – Implementing The Greedy Idea

- ★ **Repeat** adding to A the edge $(u, \Pi(u))$ for the vertex u that has the **minimum** value for $key(\cdot)$ in Q .
- ★ **Update**, if necessary, the distance $key(v)$ and the candidate edge $(v, \Pi(v))$ for each neighbor v of u that is still in Q .

Prim Algorithm – Code

- (1) initialize $\Pi(r) = nil$; $A = \emptyset$; $Q = V - \{r\}$
- (2) for all $u \in Q$ do $key(u) = \infty$
- (3) $u = r$
- (4) Repeat
- (5) for each neighbor v of u do
- (6) if $(v \in Q)$ and $w(u, v) < key(v)$ then
- (7) $key(v) = w(u, v)$; $\Pi(v) = u$
- (8) $u = \text{Extract-Min}(Q)$
- (9) $A = A \cup \{(u, \Pi(u))\}$
- (10) until $Q = \emptyset$
- (11) return (A)

Prim Algorithm – Correctness

- * Assume to the **contrary** that the output set A is not an MST.
- * Let (u, v) be the **first** edge that was added to A such that $A \cup \{(u, v)\}$ is not a **minimal forest**.
 - In particular, at that time, A is a **minimal forest**.
- * Let $\mathcal{C} = \langle Q, (V - Q) \rangle$ be a cut.

Prim Algorithm – Correctness

- * The algorithm guarantees that \mathcal{C} contains A .
- * The priority queue implies that (u, v) is a **minimal crossing edge** for the cut \mathcal{C} .
- * Greedy lemma $\Rightarrow A \cup \{(u, v)\}$ is a **minimal forest**.
- * A **contradiction**.

Prim Algorithm – Complexity

Queue operations:

- ★ One time **building** a priority queue.
- ★ $n - 1$ times the operation **Extract-Min**.
- ★ At most m times **updating** the function *key*.

Prim Algorithm – Complexity

An implementation with an unsorted array:

- ★ $\Theta(n)$ to build a queue.
- ★ $\Theta(n)$ for the **Extract-Min** operation.
- ★ $\Theta(1)$ for the **Update-Queue** operation.
- ★ All together, $\Theta(n^2)$ complexity.

An implementation with a sorted array:

- ★ $\Theta(n)$ to build a queue.
- ★ $\Theta(1)$ for the **Extract-Min** operation.
- ★ $\Theta(n)$ for the **Update-Queue** operation.
- ★ All together, $\Theta(nm)$ complexity.

Prim Algorithm – Complexity

An implementation with a heap:

- ★ $\Theta(n)$ to build a queue.
- ★ $\Theta(\log n)$ for the **Extract-Min** operation.
- ★ $\Theta(\log n)$ for the **Update-Queue** operation.
- ★ All together, $\Theta(m \log n)$ complexity.

An implementation with a Fibonacci heap:

- ★ $\Theta(n)$ to build a queue.
- ★ $\Theta(\log n)$ for the **Extract-Min** operation.
- ★ $\Theta(1)$ (**amortized**) for the **Update-Queue** operation.
- ★ All together, $\Theta(m + n \log n)$ complexity.

Kruskal vs. Prim

Complexity:

- ★ Kruskal is an $O(m \log m)$ algorithm.
- ★ Prim is an $O(m + n \log n)$ algorithm.

Implementation:

- ★ Kruskal is a **distributed** algorithm.
- ★ Prim is a **centralized** algorithm.