

CSCI 256 Lecture 10:

Minimum Spanning Trees

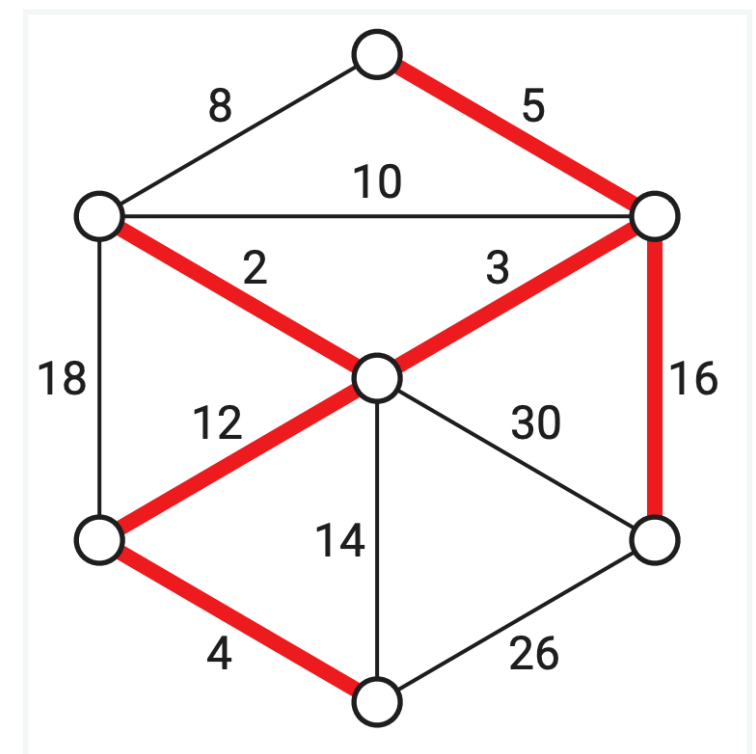
Announcements

- Hand in **Exercise #8**, pick up **Exercise #9**
- No mountain day today 🙄
- No assignment due next week
- First in-class test is on **Mon, Oct 5**
 - **50-mins closed book in-class test**
 - All content up to but not including Dijkstra's
 - Practice test posted on GLOW

Minimum Spanning Trees

Problem. Given a connected, undirected graph $G = (V, E)$ with edge weights w_e , output a minimum spanning tree T , i.e., set of edges $T \subseteq E$ s.t.

- T is a spanning tree of G : tree T connects all vertices
- T is a spanning tree of minimum weight: for any other spanning tree $\hat{T}$ of G , we have $\sum_{e \in T} w_e \leq \sum_{e \in \hat{T}} w_e$
- **Question.** How would we compute a min spanning tree in a graph with all edge weights equal to 1?

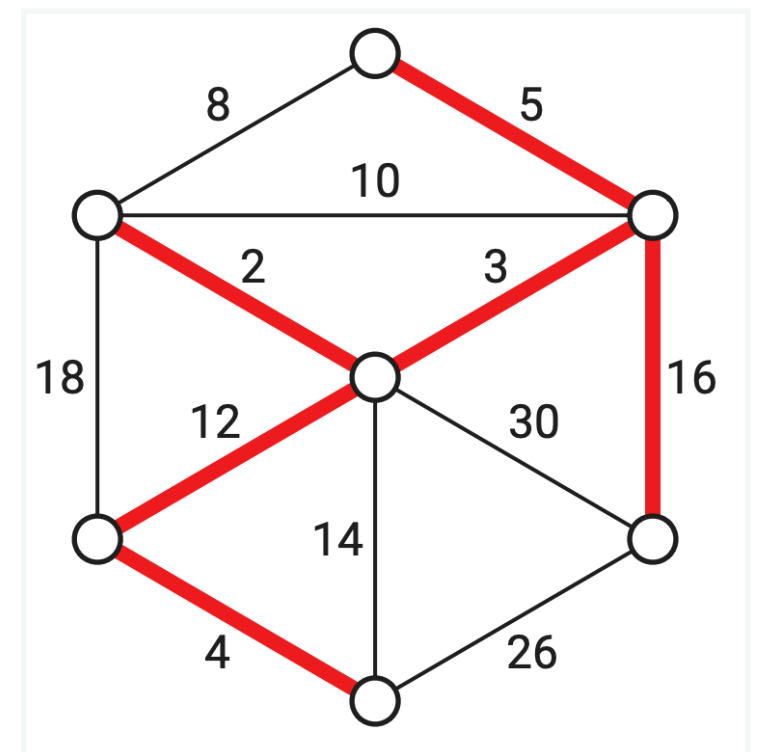


MST: Applications

- Many applications!
 - Classic application:
 - Connectivity in telecom or road network
 - Efficient broadcasting on a computer network
 - Approximate solutions to harder problems, such as TSP
 - Real-time face verification
 - Clustering and data analysis

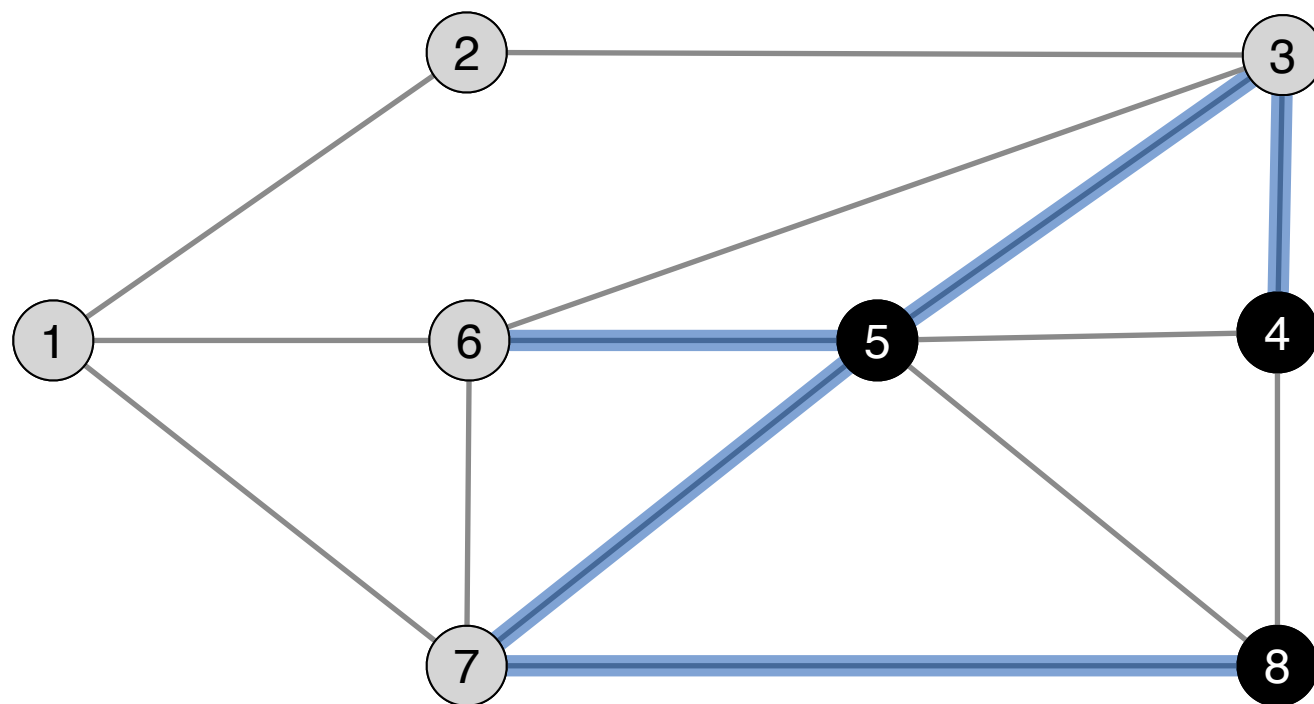
Ideas of Algorithms?

- How to construct a spanning tree?
- Constraints:
 - Need to ensure graph is not disconnected
 - Need to do so with lowest cost edges possible



Spanning Trees and Cuts

A cut is a partition of the vertices into two nonempty subsets S and $V - S$.
A cut edge of a cut S is an edge with one end point in S and another in $V - S$.



cut $S = \{ 4, 5, 8 \}$

Cut edges = $\{ (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) \}$

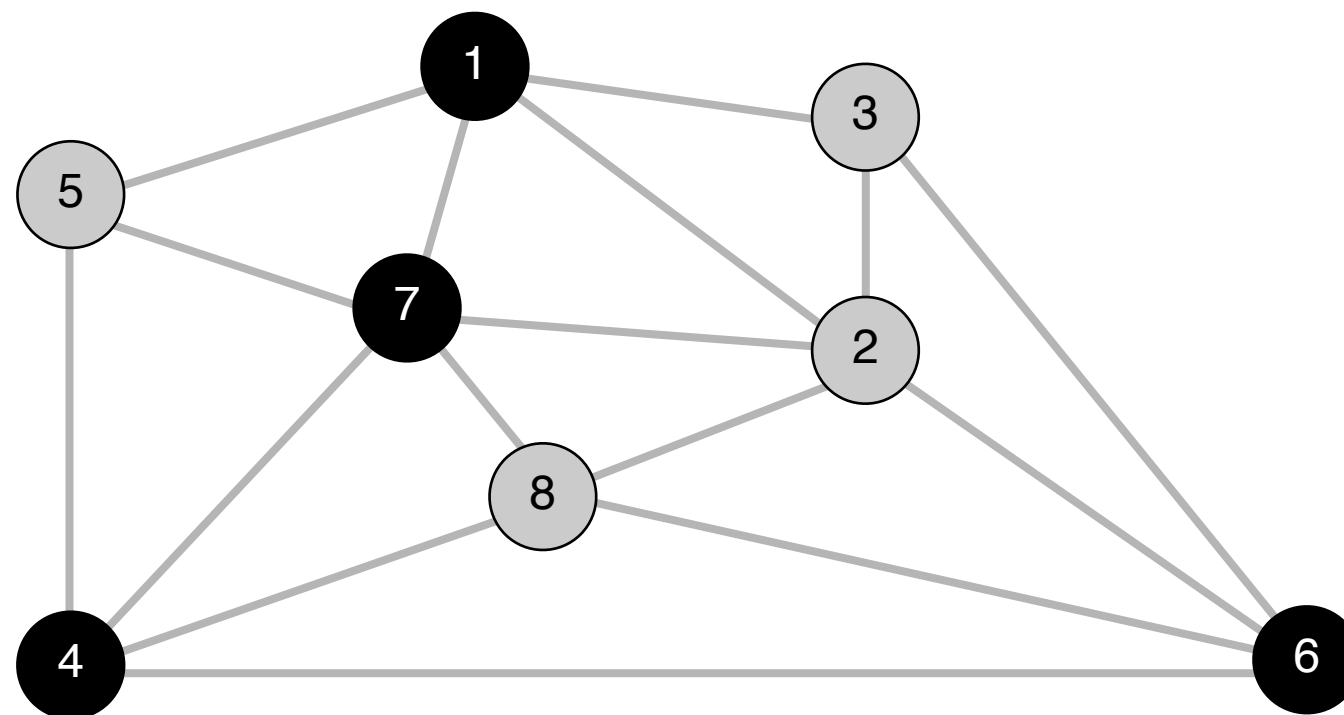
Spanning Trees and Cuts

Question. Consider the cut $S = \{1,4,6,7\}$. Which of the following edges are cut edges with respect to this cut?

A. (1, 7)

B. (5, 7)

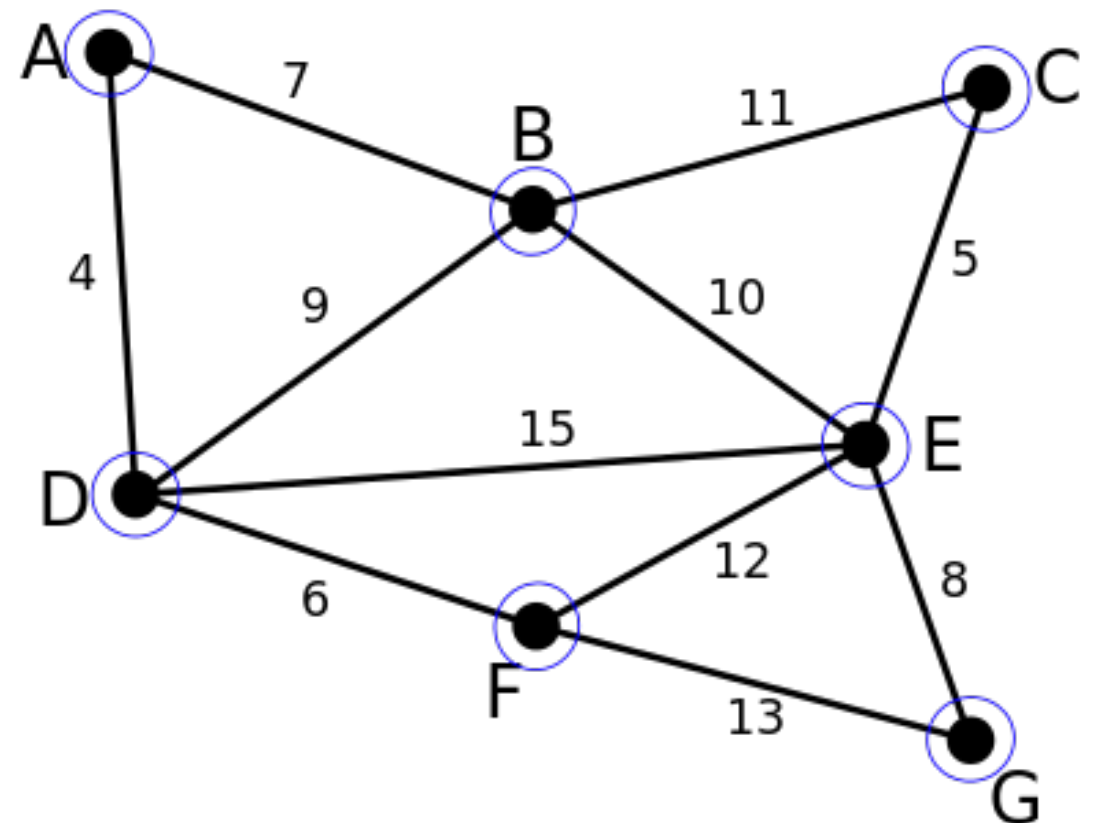
C. (2, 3)

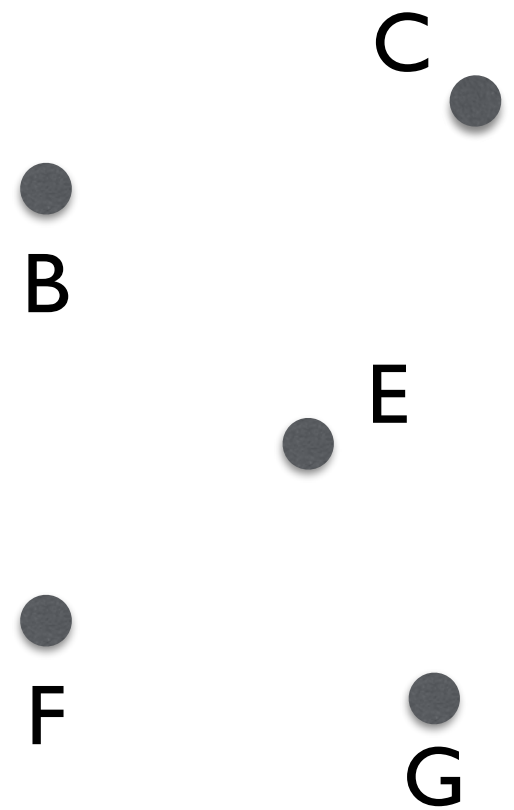
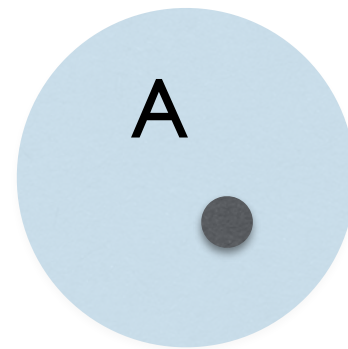
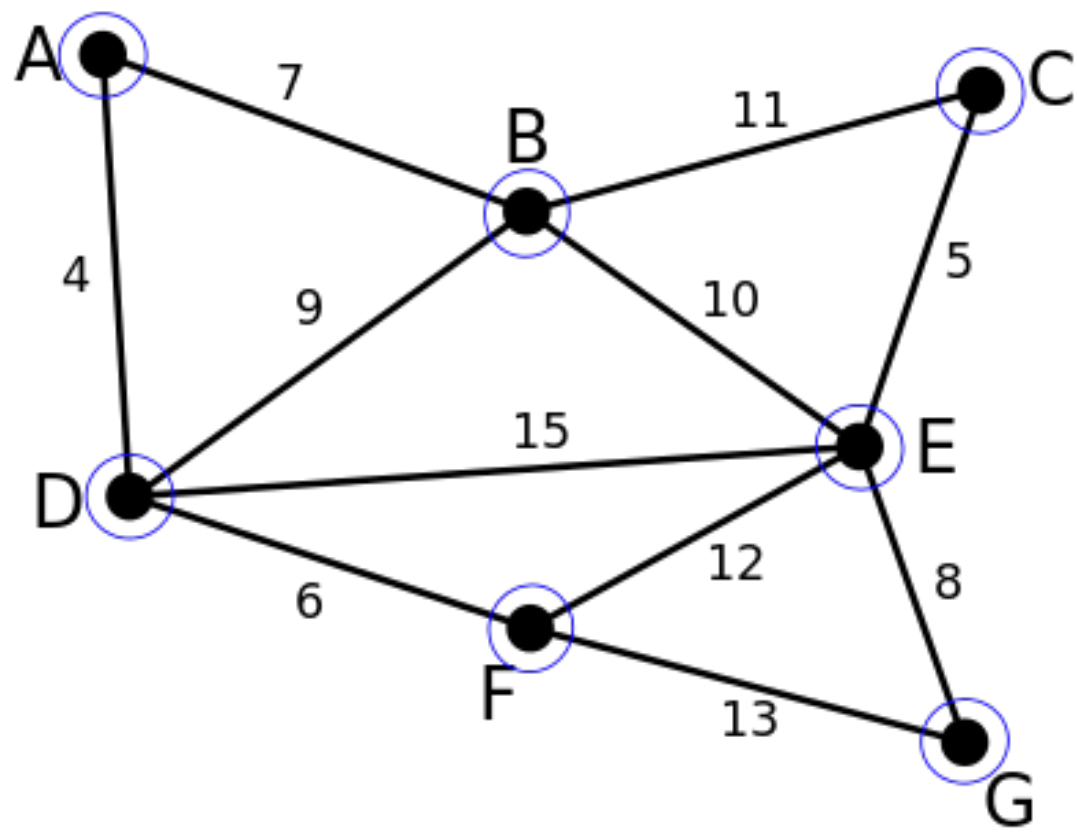


“Prims” Algorithm

Jarník's ("Prims Algorithm")

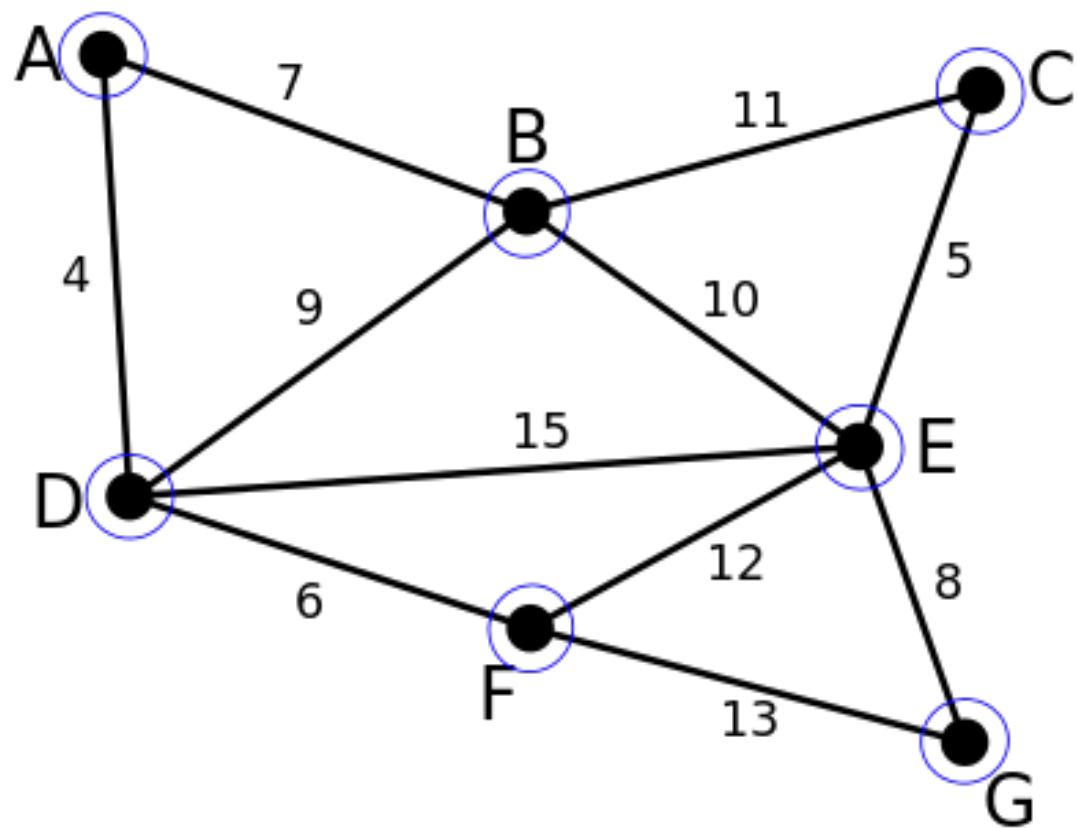
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$



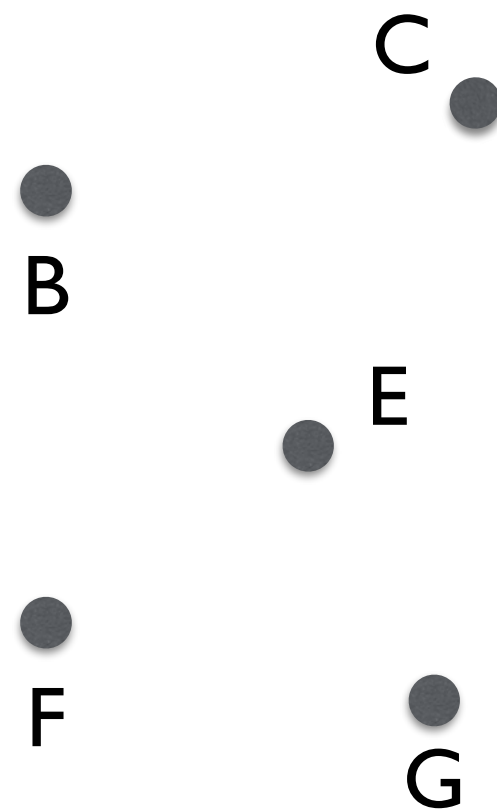
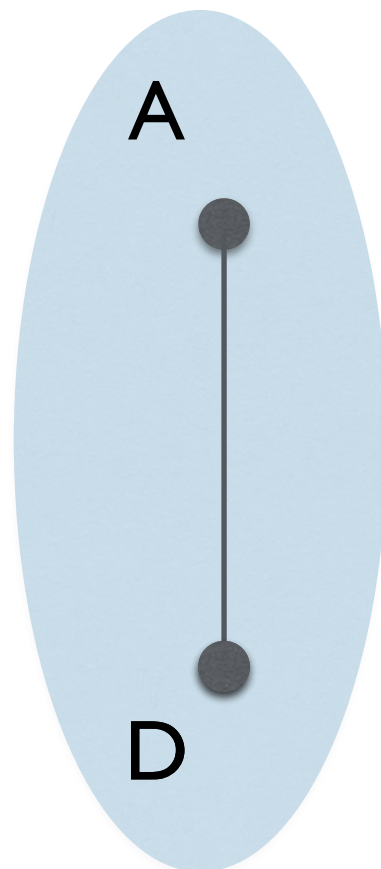


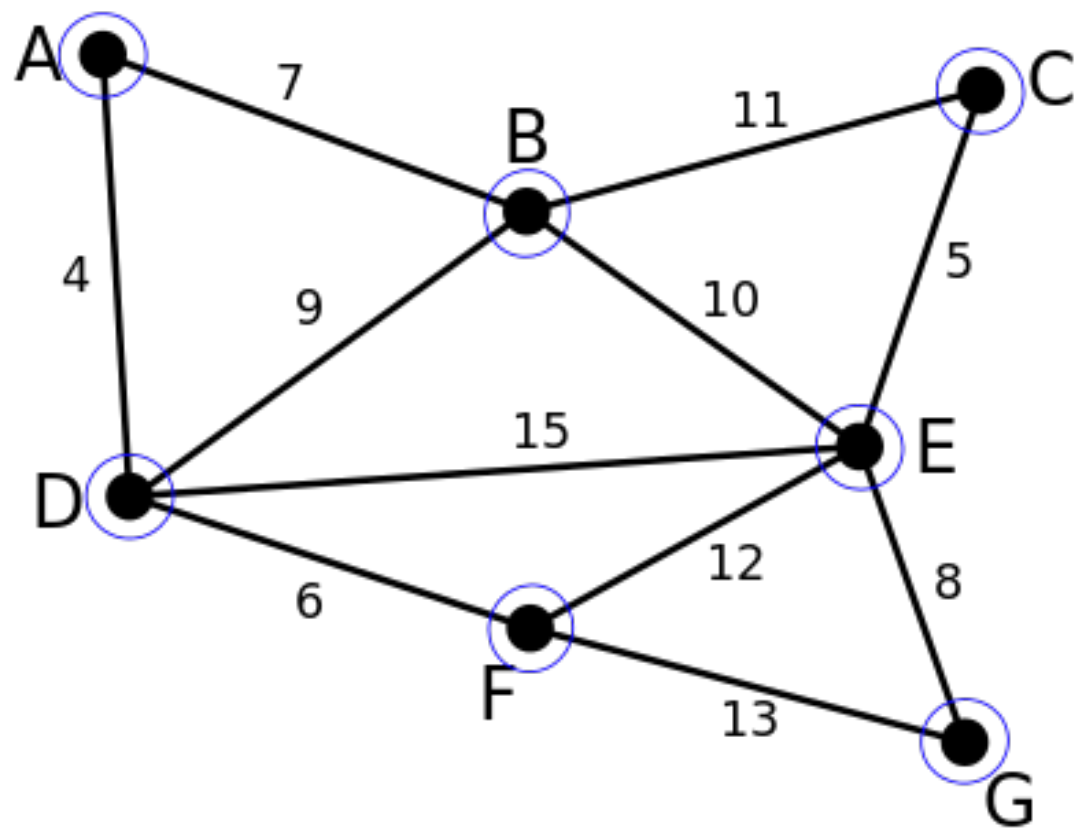
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$

D

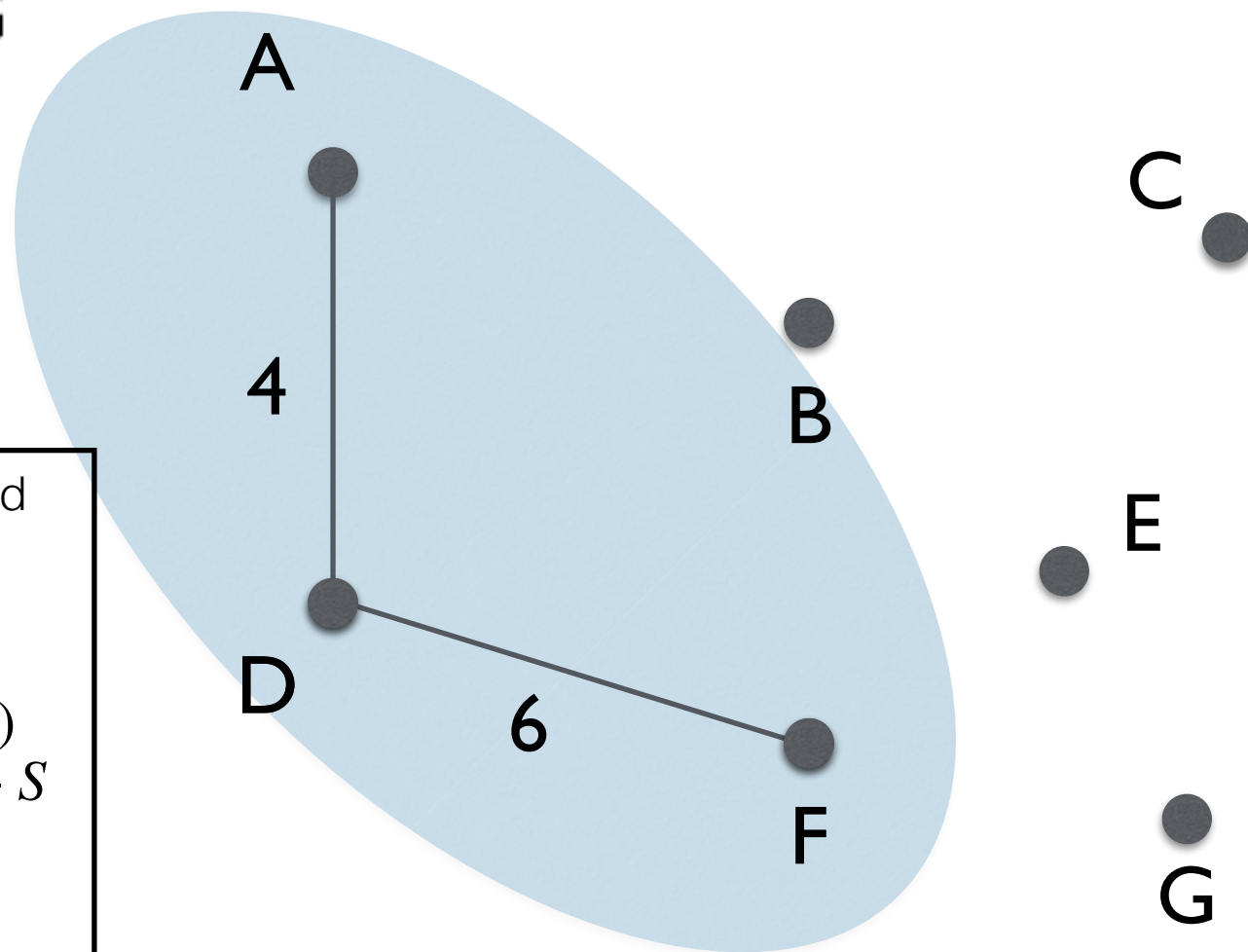


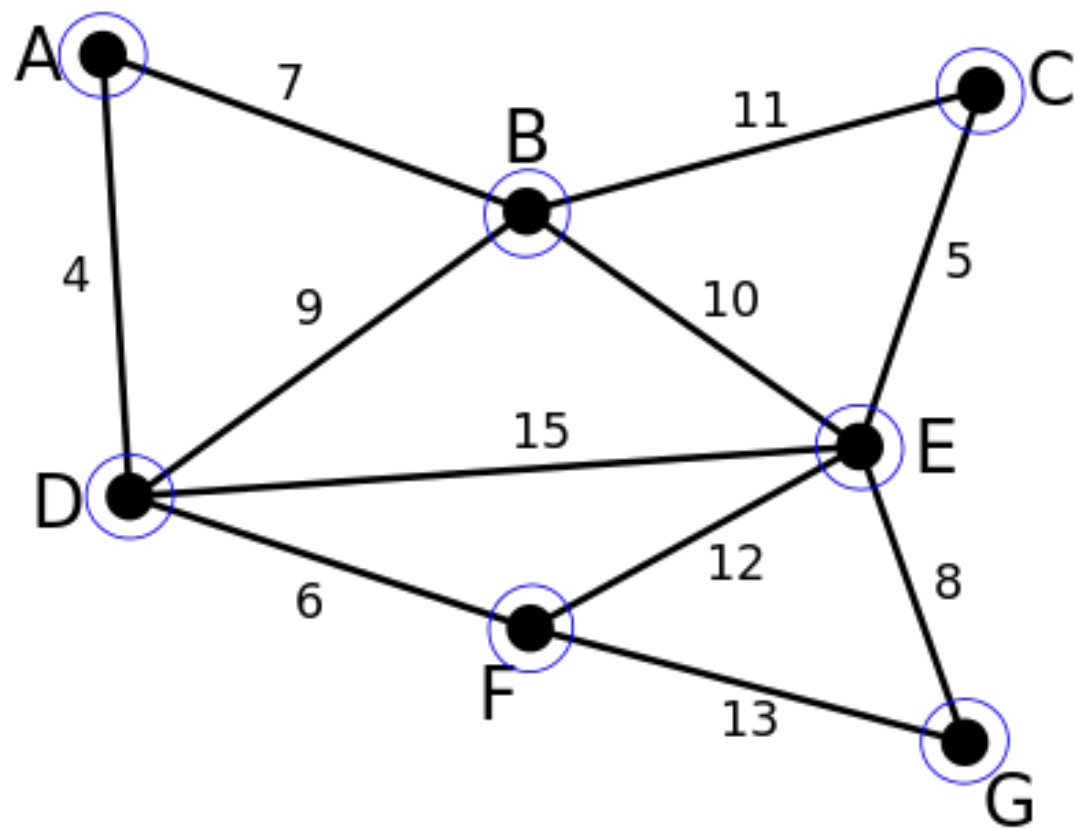
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$



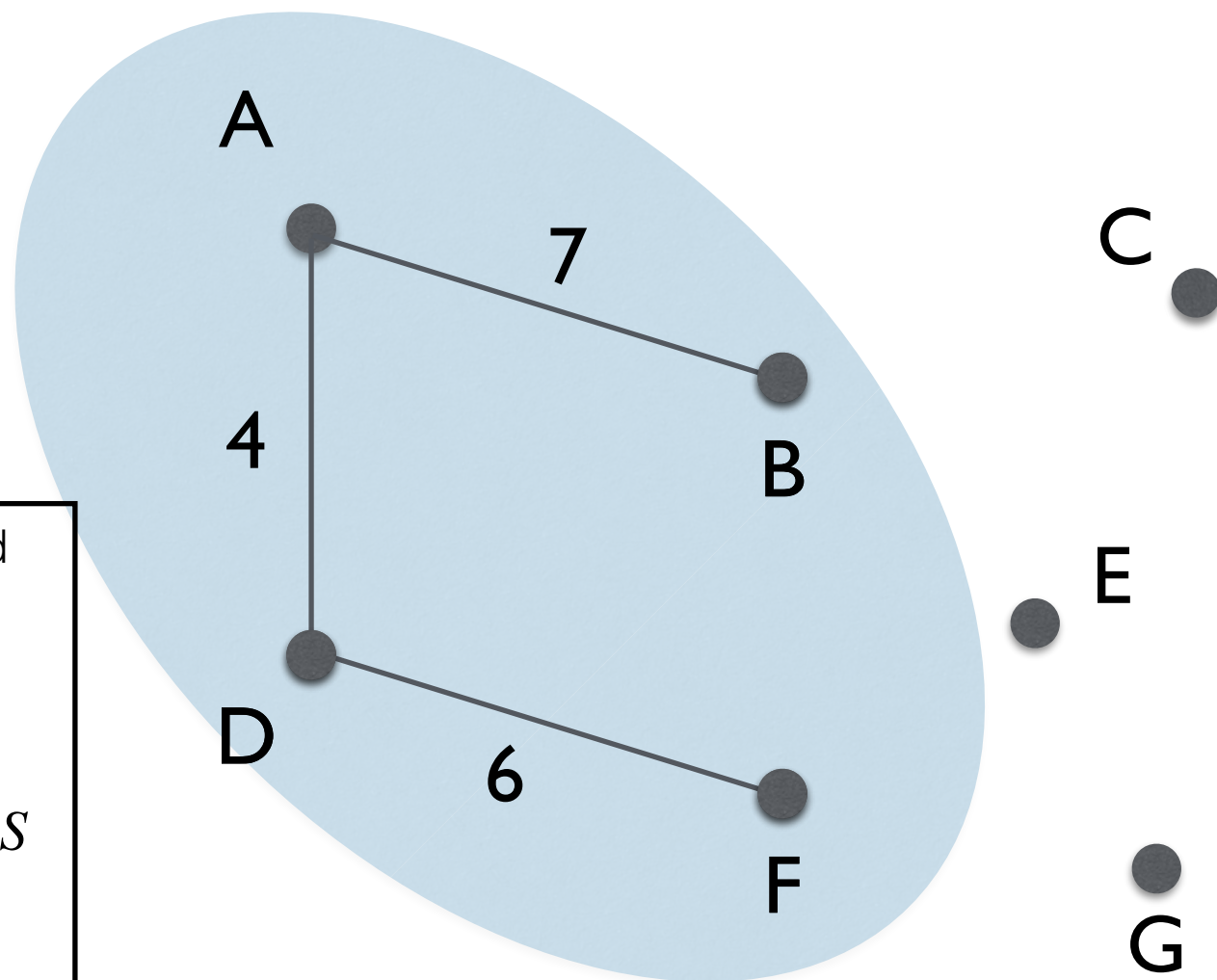


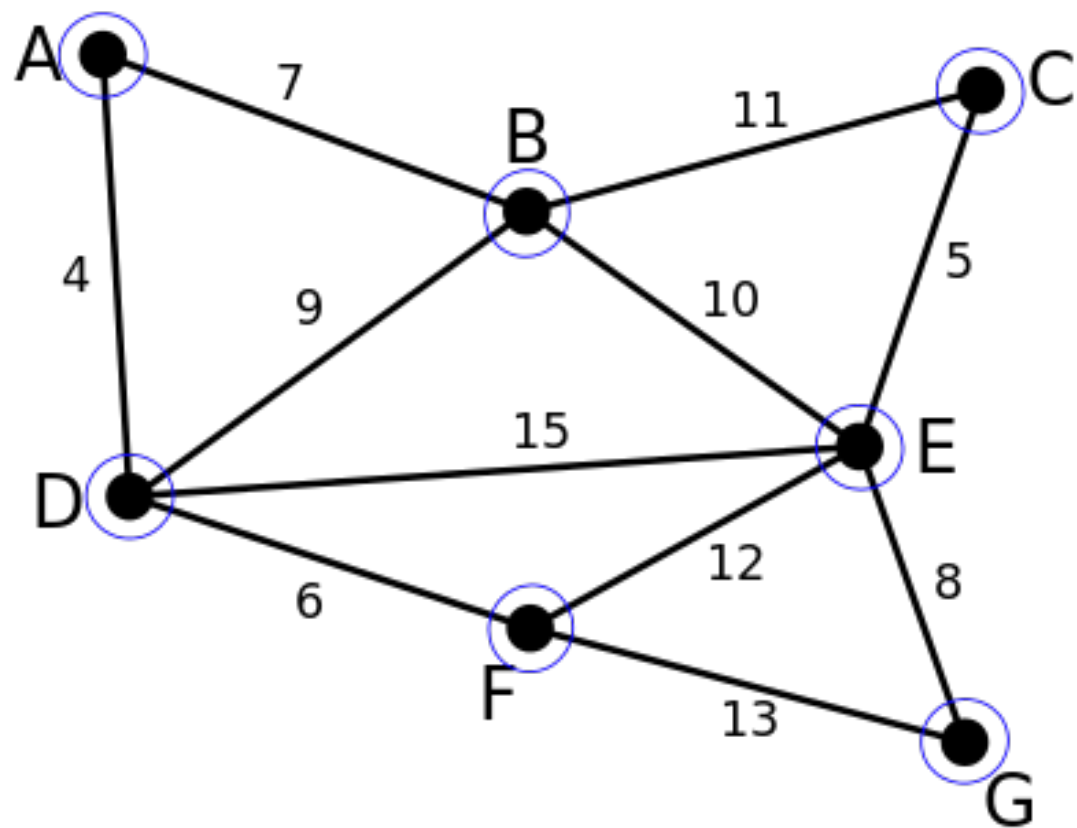
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$



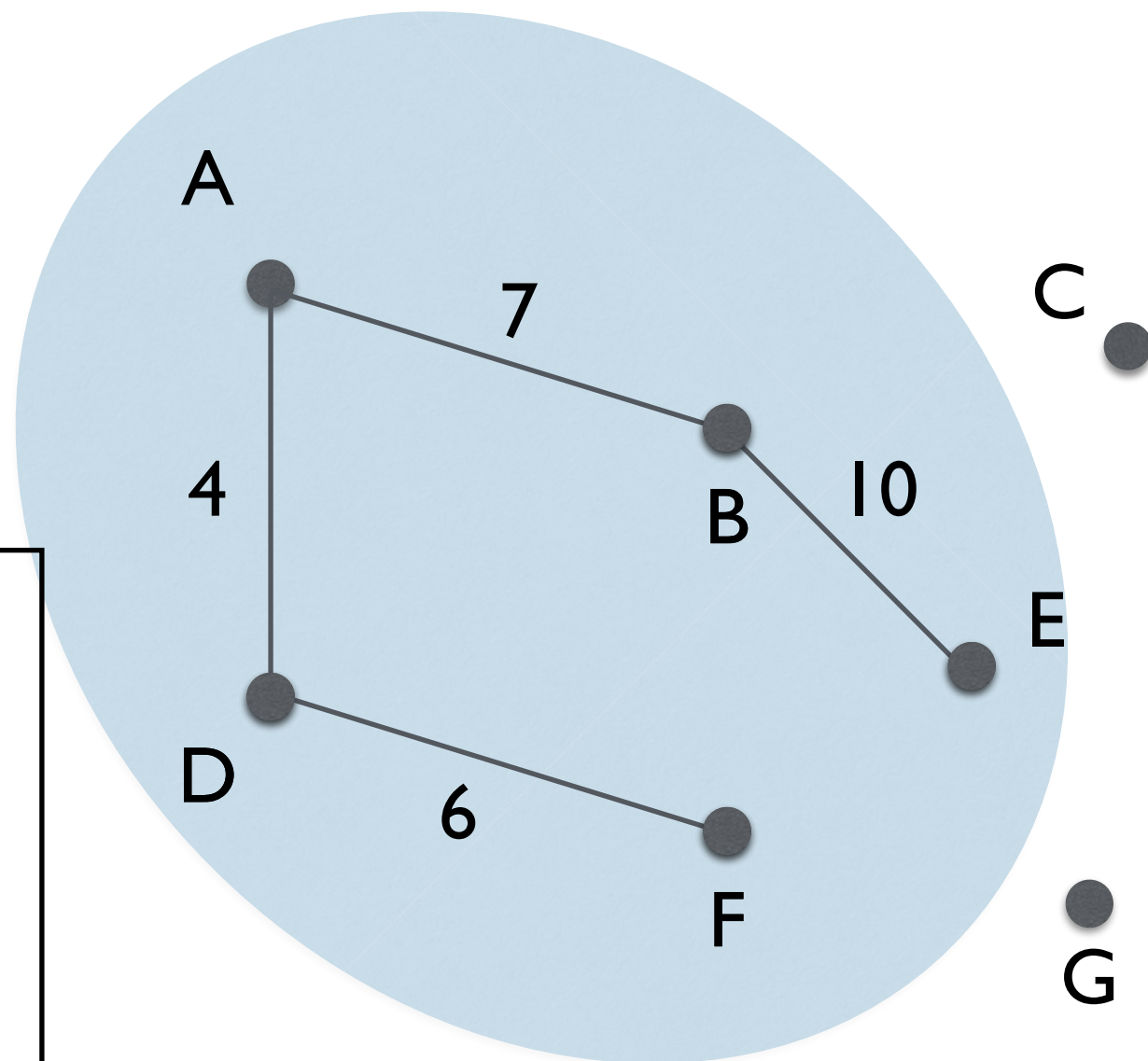


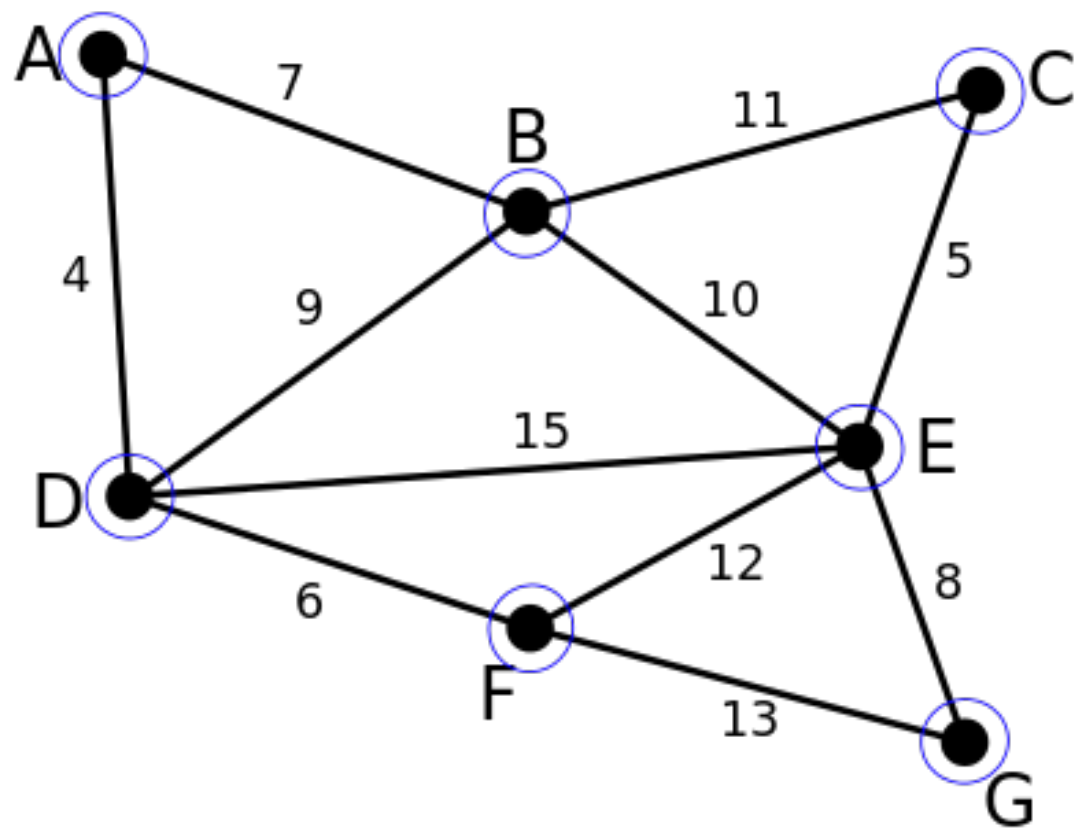
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$



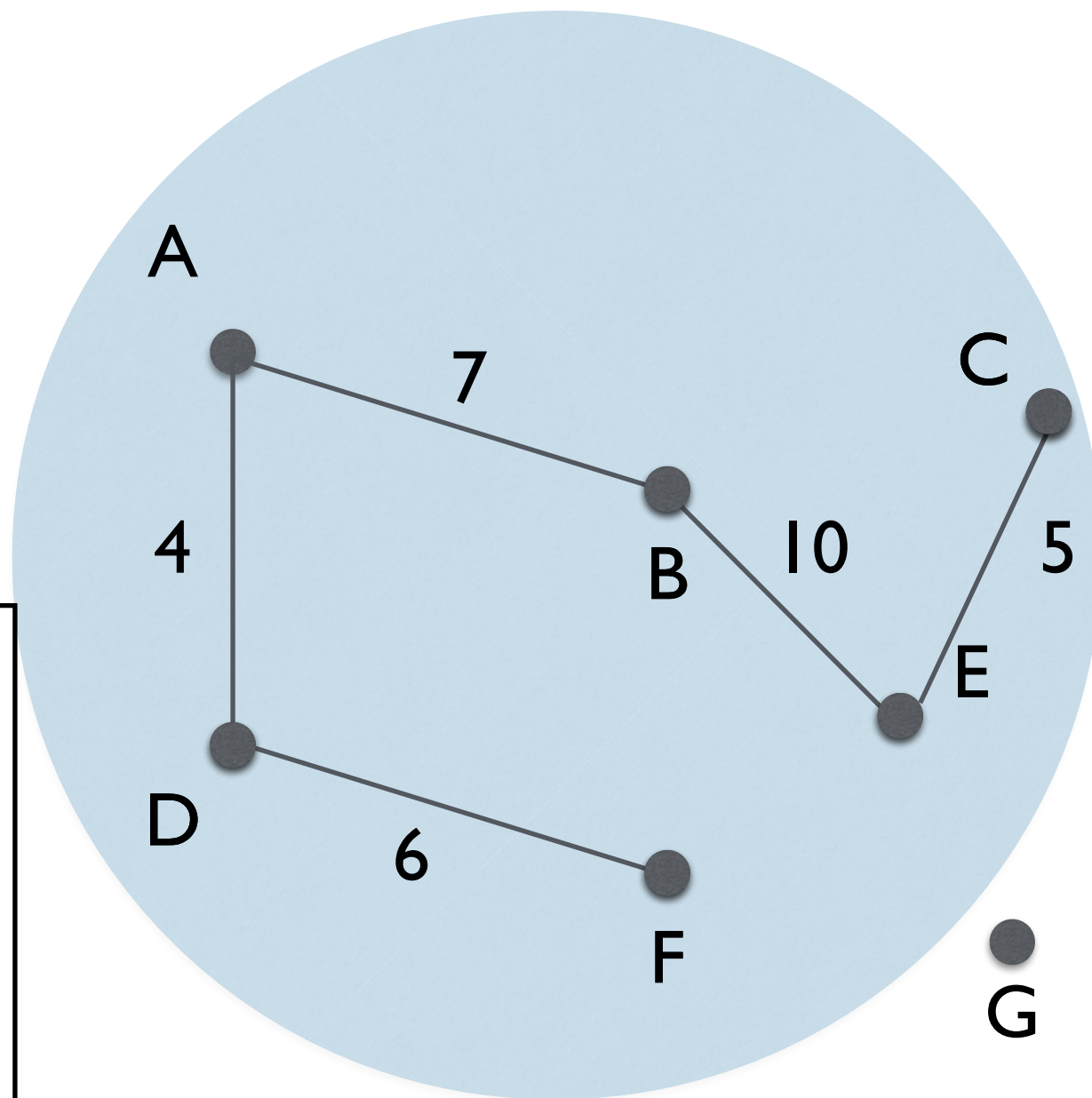


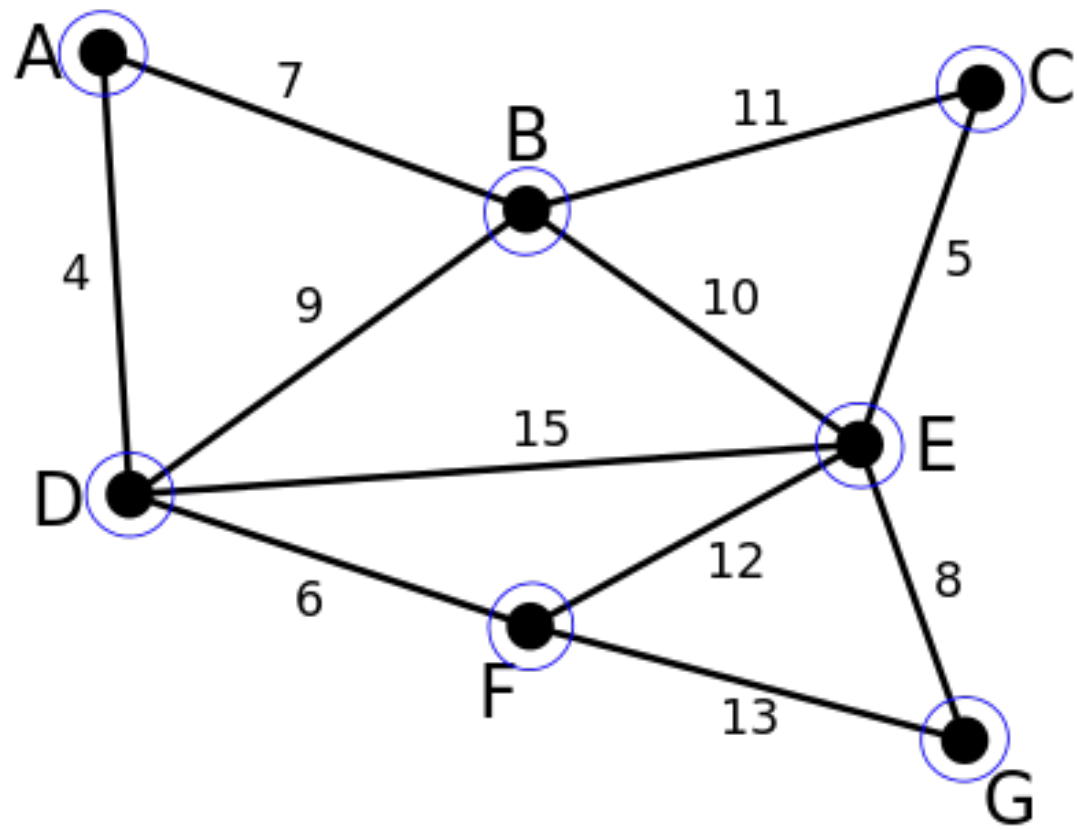
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$





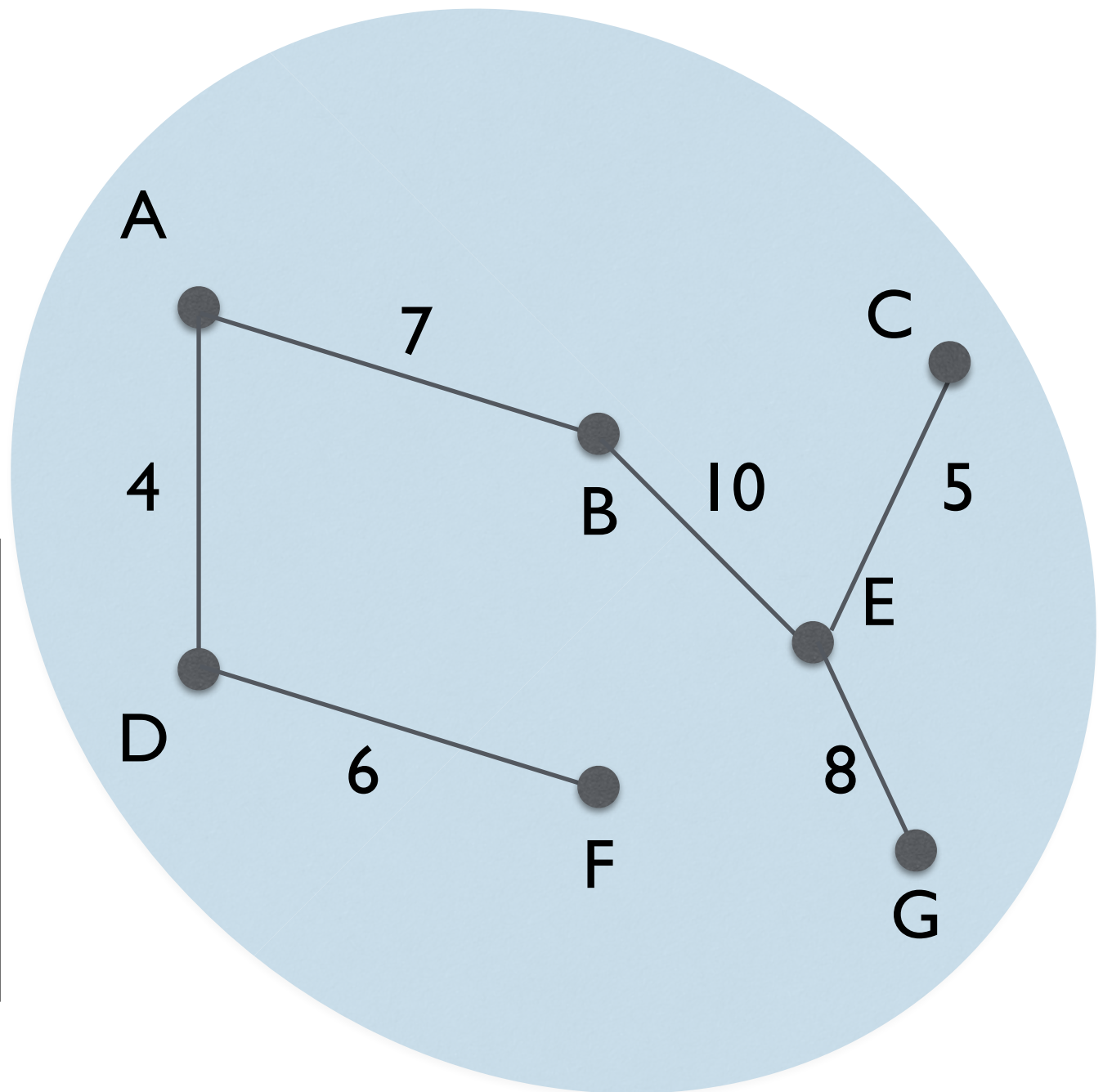
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$





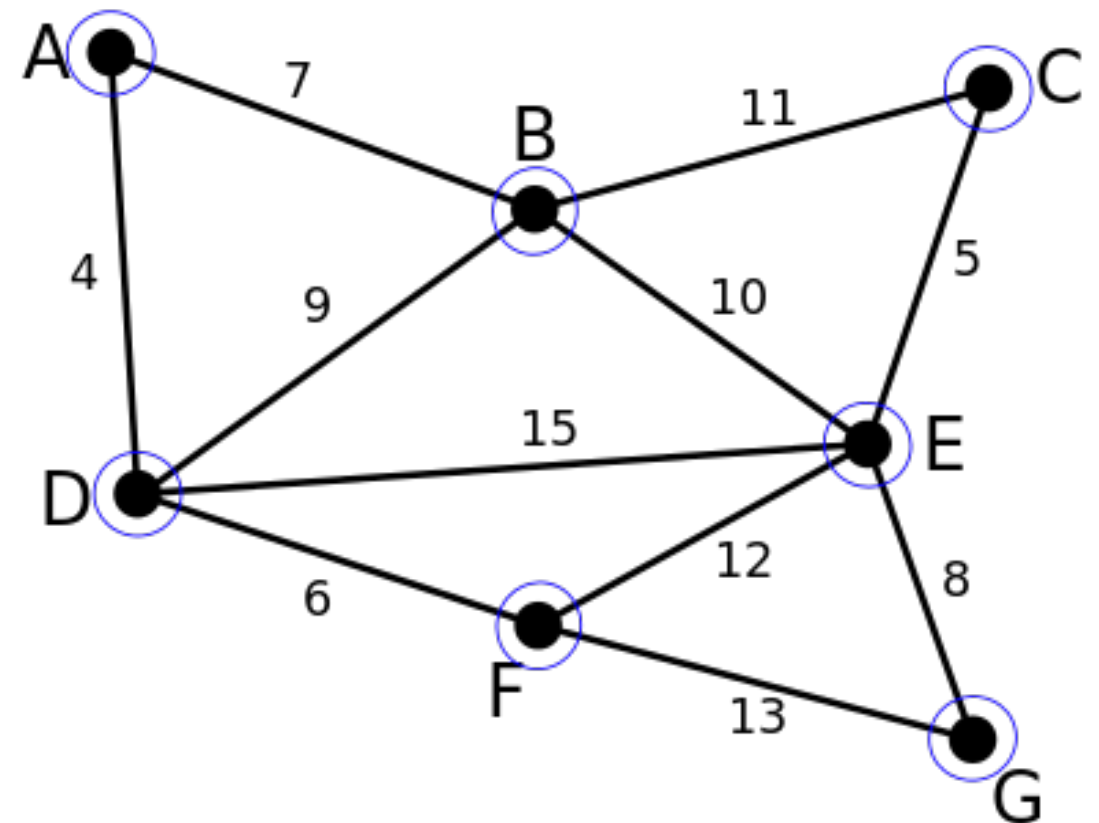
Total weight: 40

- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| < n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$



Jarník's ("Prims Algorithm")

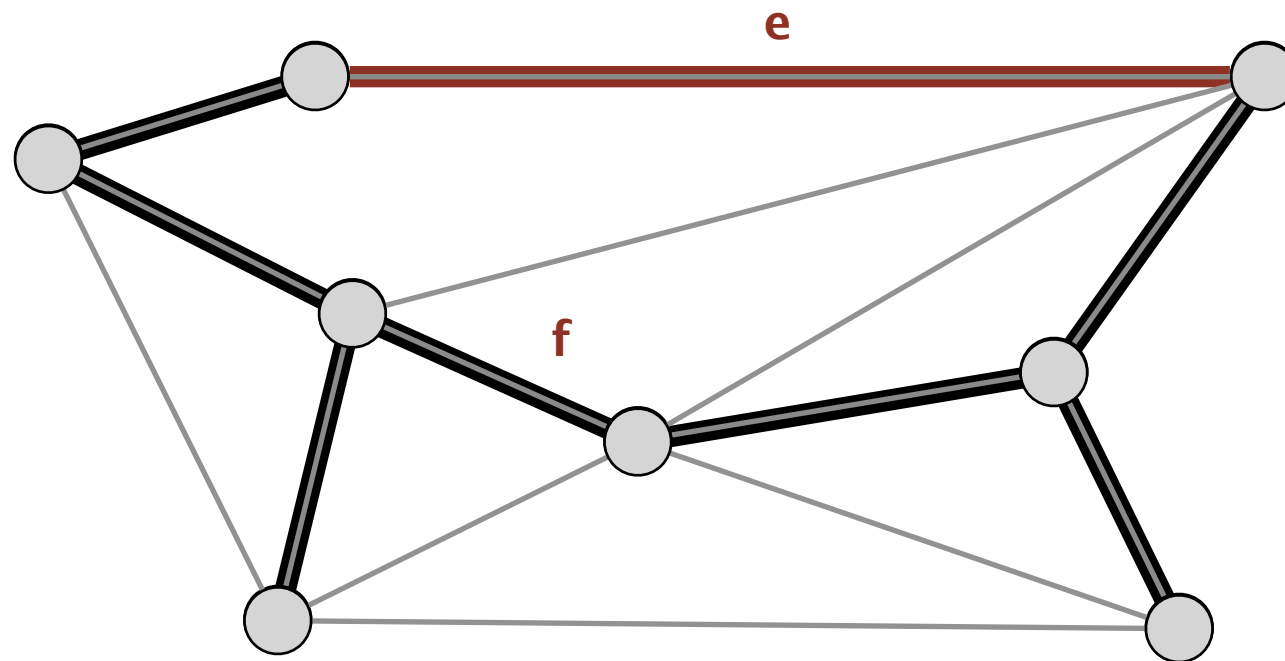
- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| \leq n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$
 - $S \leftarrow S \cup \{v\}$
- Correctness:
 - Why is T a MST?



Fundamental Cycle

Let T be a spanning tree of G .

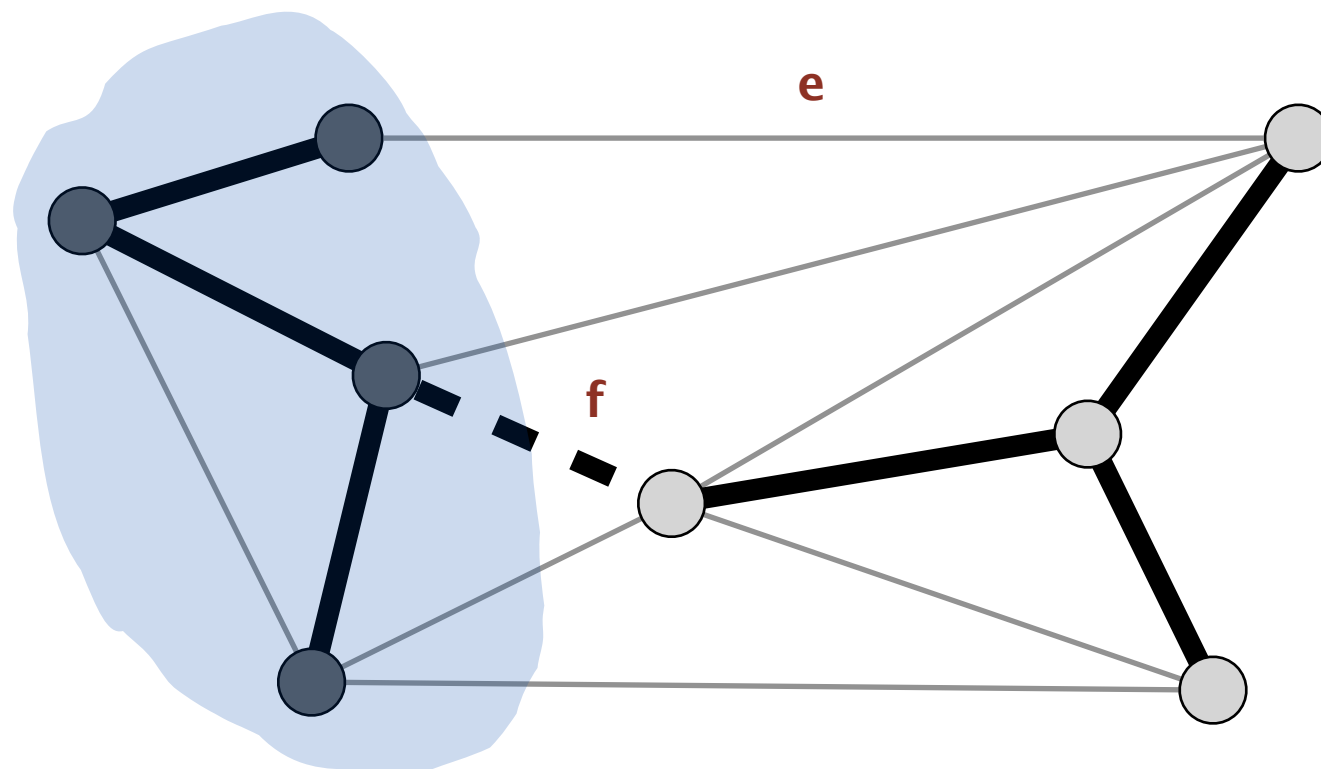
- For any edge $e \notin T$, $T \cup \{e\}$ creates a unique cycle C
- For any edge $f \in C : T \cup \{e\} - \{f\}$ is a spanning tree



Fundamental Cut

Let T be a spanning tree of G .

- For any edge $f \in T$, $T - \{f\}$ breaks the graph into two connected components
- For any edge e crossing this cut, $T - \{f\} \cup \{e\}$ is a spanning tree



Spanning Trees and Cuts

(Assume edge costs are distinct for simplicity)

Lemma (Cut Property). For any cut $S \subset V$, let e be the minimum weight edge connecting any vertex in S to a vertex in $V - S$, then every minimum spanning tree must include e .

Proof. (By contradiction)

Suppose T is a spanning tree that does not contain $e = (v, w)$.

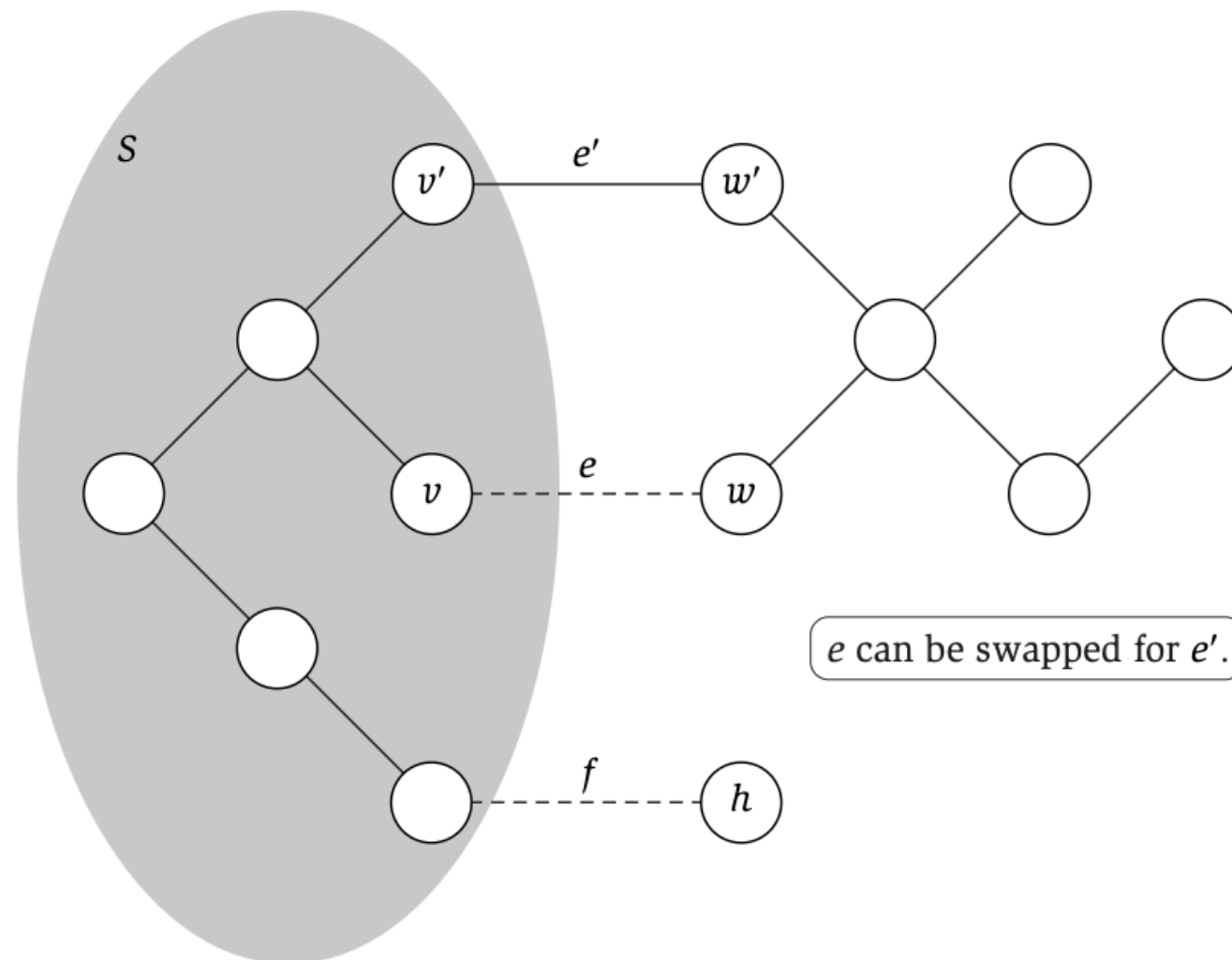
Main Idea: We will construct another spanning tree $T' = T \cup e - e'$ with weight less than T ($\Rightarrow \Leftarrow$)

How to find such an edge e' ?

Proof by "exchange argument"

Suppose T is a spanning tree that does not contain $e = (v, w)$ where $v \in S$ and $w \in V - S$.

Find an $e' \in T$ and exchange e and e' to obtain a "lighter" tree



Cut Property: MST

Proof (Cut Property). Suppose T is a minimum spanning tree that does not contain $e = (v, w)$.

- T is a spanning tree so it must have a path P from v to w
- Starting from v , follow the nodes on P until you reach the first node w' that is not in S
 - Let $e' = (v', w')$ be the first edge in P leaving S (i.e. $v' \in S, w' \notin S$)
- Is $T' = T - \{e'\} \cup \{e\}$ a spanning tree?
 - Why?
- Since e is min-cost weight crossing the cut, $w_e < w_{e'}$ and thus $w(T') < w(T)$ which is a contradiction.

Cycle Property: MST

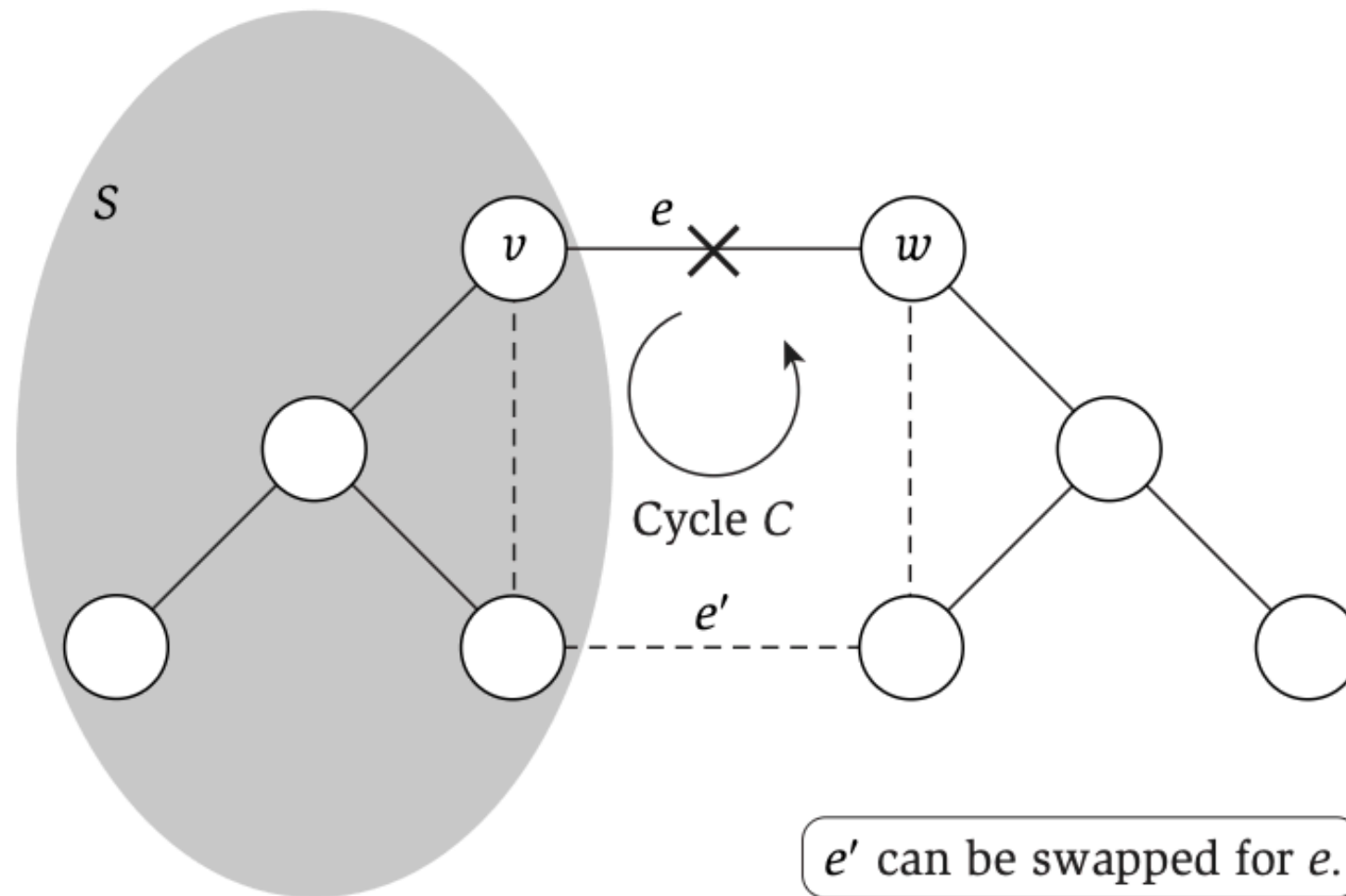
Lemma (Cycle Property). For any cycle C in G , its highest cost edge e is in no MST of G .

Proof. (By contradiction)

Suppose a MST T contains e .

- Main Idea: Construct another spanning tree $T' = T - \{e\} \cup \{e'\}$ with weight less than T ($\Rightarrow \Leftarrow$)
- How to find such an e' ?

Cycle Property: MST



Jarník's ("Prims Algorithm")

- Initialize $S = \{u\}$ for any vertex $u \in V$ and $T = \emptyset$
- While $|T| \leq n - 1$:
 - Find the min-cost edge $e = (u, v)$ with one end $u \in S$ and $v \in V - S$
 - $T \leftarrow T \cup \{e\}$, $S \leftarrow S \cup \{v\}$
- **Implementation.** Find and add min-cost edge for the cut $(S, V - S)$ and add it to the tree in each iteration, update cut edges
- How to implement? Naive implementation may take $O(nm)$
 - Need to maintain set of edges adjacent to nodes in T and extract min-cost cut edge from it each time
 - Which data structure can we use?

“Prims” Implementation

- Using **Binary heaps**: create a priority queue initially holding all edges incident to u .
- At each step,
 - Dequeue edges from the priority queue until we find an edge (x, y) where $x \in S$ and $y \notin S$.
 - Add (x, y) to T .
 - Add to PQ all edges incident to y whose endpoints aren't in S .
- **Analysis**. Each edge is enqueued and dequeued at most once
- Total runtime: $O(m \log m)$
 - In any graph, $m = O(n^2)$
 - So $O(m \log m) = O(m \log n)$

“Prims” Implementation

- Implementation using **Binary heaps**
 - Total runtime: $O(m \log n)$
- If a **Fibonacci heap** is used instead of binary heap:
 - Runs in $O(m + n \log n)$ “**amortized time**”
 - Supports amortized $O(1)$ -amortized decrease-key, $O(\log n)$ worst-case time extract min

Definition. If k operations take total time $O(t \cdot k)$, then the amortized time per operation is $O(t)$.

