Minimum Spanning Trees

Let’s prove something

- Prove: In an undirected graph, removing an edge from a cycle does not disconnect the graph (as in, the connected components remain identical)

Let's talk about trees

- Defn: A tree is an undirected acyclic graph
- Prove: A tree with n nodes has n-1 edges

Let's talk about trees

- Defn: A tree is an undirected acyclic graph
- Prove: A tree with n nodes has n-1 edges
- Prove: Any connected undirected graph $G = (V,E)$, with $|E| = |V| - 1$ is a tree

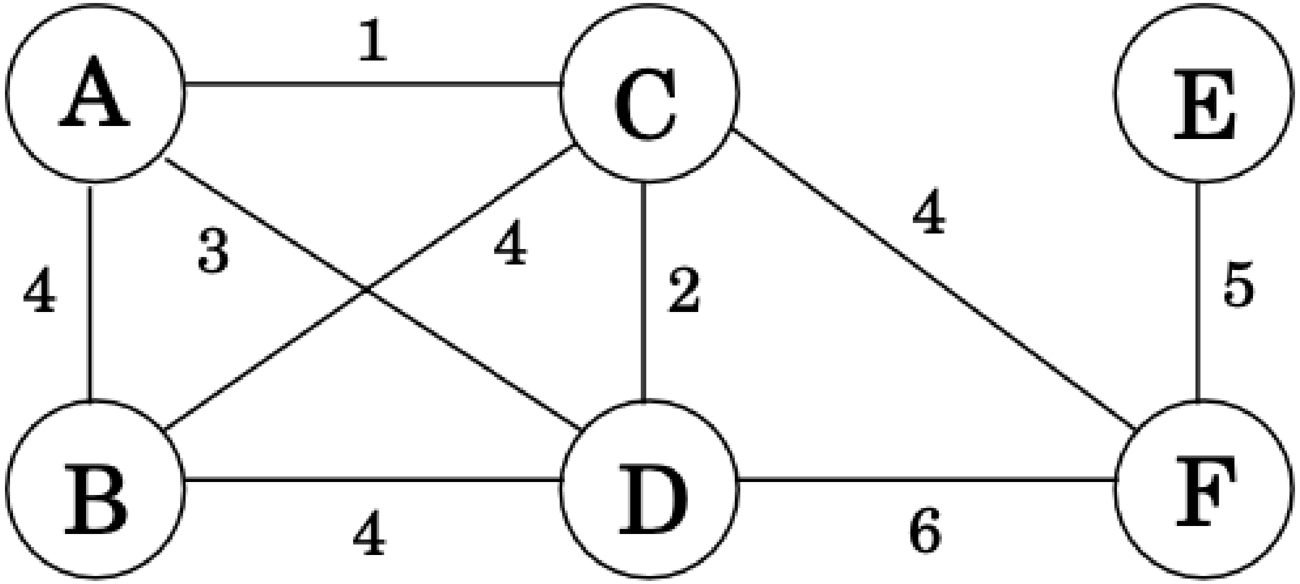
Let’s talk about trees

- Defn: A tree is an undirected acyclic graph
- Prove: A tree with n nodes has n-1 edges
- Prove: Any connected undirected graph G = (V,E), with |E| = |V| - 1 is a tree
- Prove: An undirected graph is a tree if and only if there is a unique path between any pair of nodes.

Minimum Spanning Trees

- Defn: A minimum spanning tree (MST) of a weighted undirected graph G is a tree that spans G and that has the minimum total weight of all spanning trees of G.

Problem: Find a minimum spanning tree



- Definition: A cut of a graph G is a partition of the vertices of G into two groups: S and V-S.

Cut property *Suppose edges X are part of a minimum spanning tree of $G = (V, E)$. Pick any subset of nodes S for which X does not cross between S and $V - S$, and let e be the lightest edge across this partition. Then $X \cup \{e\}$ is part of some MST.*

Cut property *Suppose edges X are part of a minimum spanning tree of $G = (V, E)$. Pick any subset of nodes S for which X does not cross between S and $V - S$, and let e be the lightest edge across this partition. Then $X \cup \{e\}$ is part of some MST.*

Figure 5.2 $T \cup \{e\}$. The addition of e (dotted) to T (solid lines) produces a cycle. This cycle must contain at least one other edge, shown here as e', across the cut $(S, V - S)$.

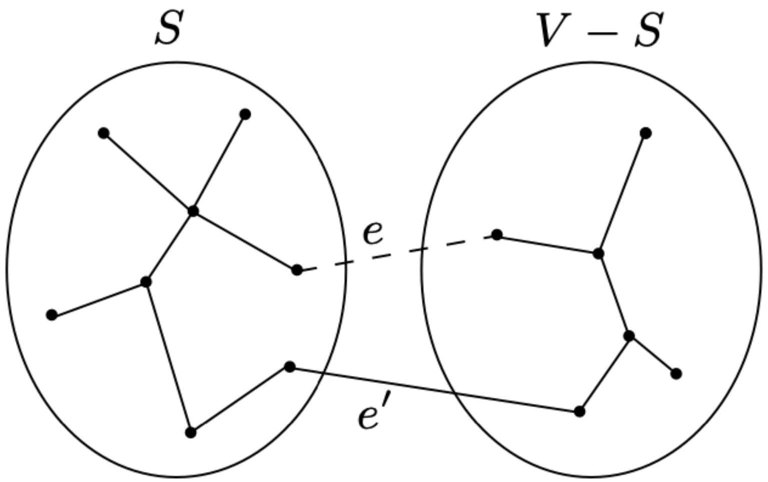


Figure 5.4 Kruskal's minimum spanning tree algorithm.

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

`makeset`(x): create a singleton set containing just x.

`find`(x): to which set does x belong?

`union`(x, y): merge the sets containing x and y.

Figure 5.4 Kruskal's minimum spanning tree algorithm.

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

So, what is the cost of this?

`makeset`(x): create a singleton set containing just x.

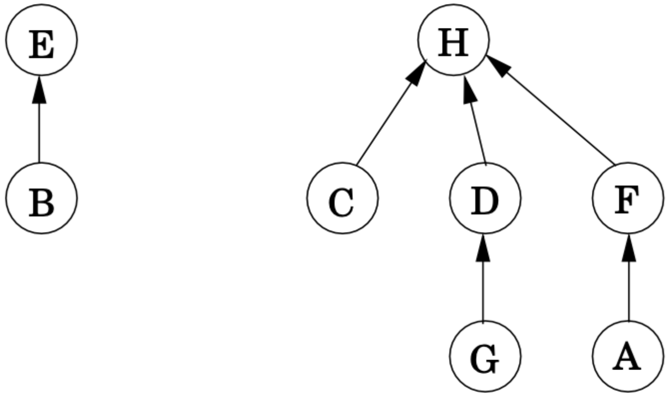
`find`(x): to which set does x belong?

`union`(x, y): merge the sets containing x and y.

Cost depends on the cost of each of these. And what does the cost of such things typically rely on?

```
procedure makeset(x)
π(x) = x
rank(x) = 0

function find(x)
while x ≠ π(x):  x = π(x)
```

Figure 5.5 A directed-tree representation of two sets $\{B, E\}$ and $\{A, C, D, F, G, H\}$.



```
procedure union(x, y)
r_x = find(x)
r_y = find(y)
if r_x = r_y:  return
if rank(r_x) > rank(r_y):
    π(r_y) = r_x
else:
    π(r_x) = r_y
    if rank(r_x) = rank(r_y):  rank(r_y) = rank(r_y) + 1
```

Figure 5.6 A sequence of disjoint-set operations. Superscripts denote rank.

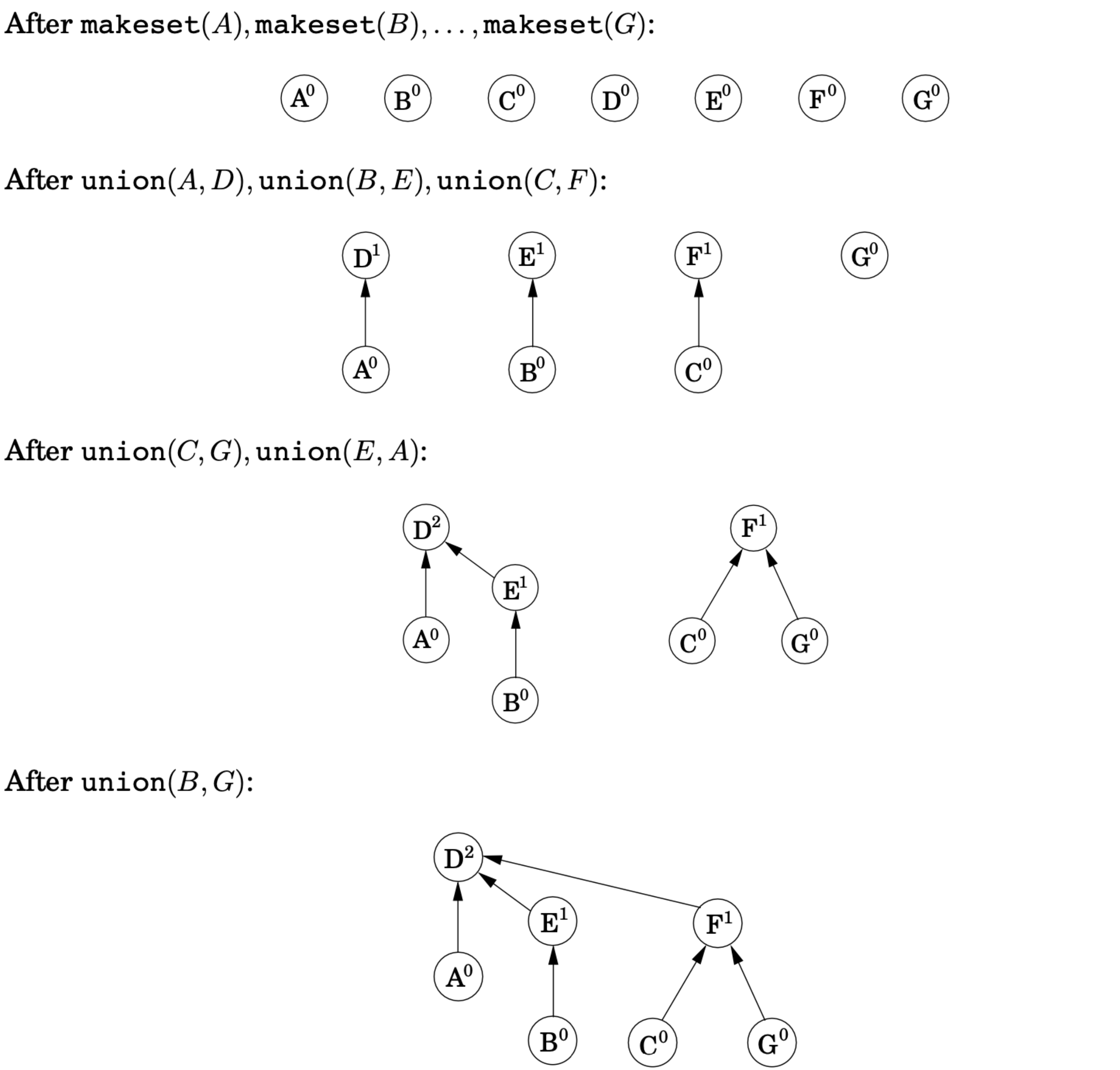


Figure 5.4 Kruskal's minimum spanning tree algorithm.

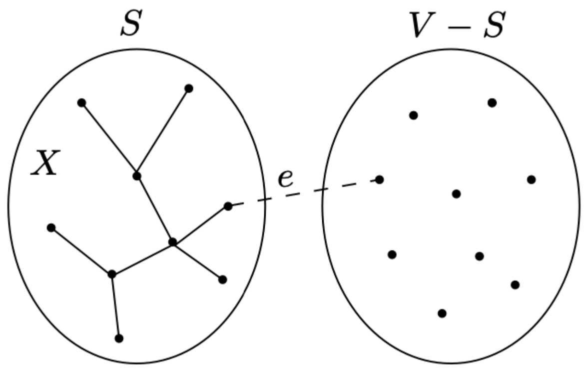
```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

So, what is the cost of this?

```
X = { } (edges picked so far)
repeat until |X| = |V| - 1:
  pick a set S ⊂ V for which X has no edges between S and V - S
  let e ∈ E be the minimum-weight edge between S and V - S
  X = X ∪ {e}
```

Cut property tells us that any algorithm that conforms to this greedy schema is guaranteed to work.

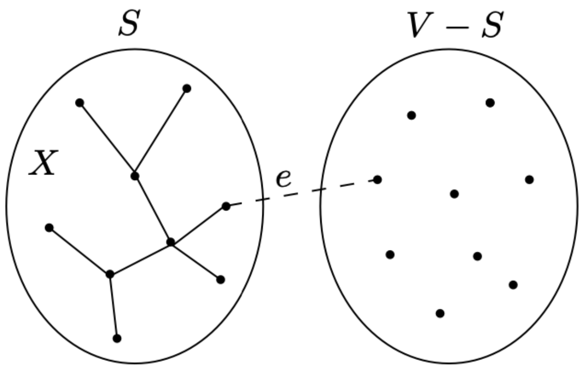
Figure 5.8 Prim's algorithm: the edges X form a tree, and S consists of its vertices.



growing to include the vertex $v \notin S$ of smallest `cost`:

$$\texttt{cost}(v) \quad = \quad \min_{u \in S} \; w(u, v).$$

Figure 5.8 Prim's algorithm: the edges X form a tree, and S consists of its vertices.

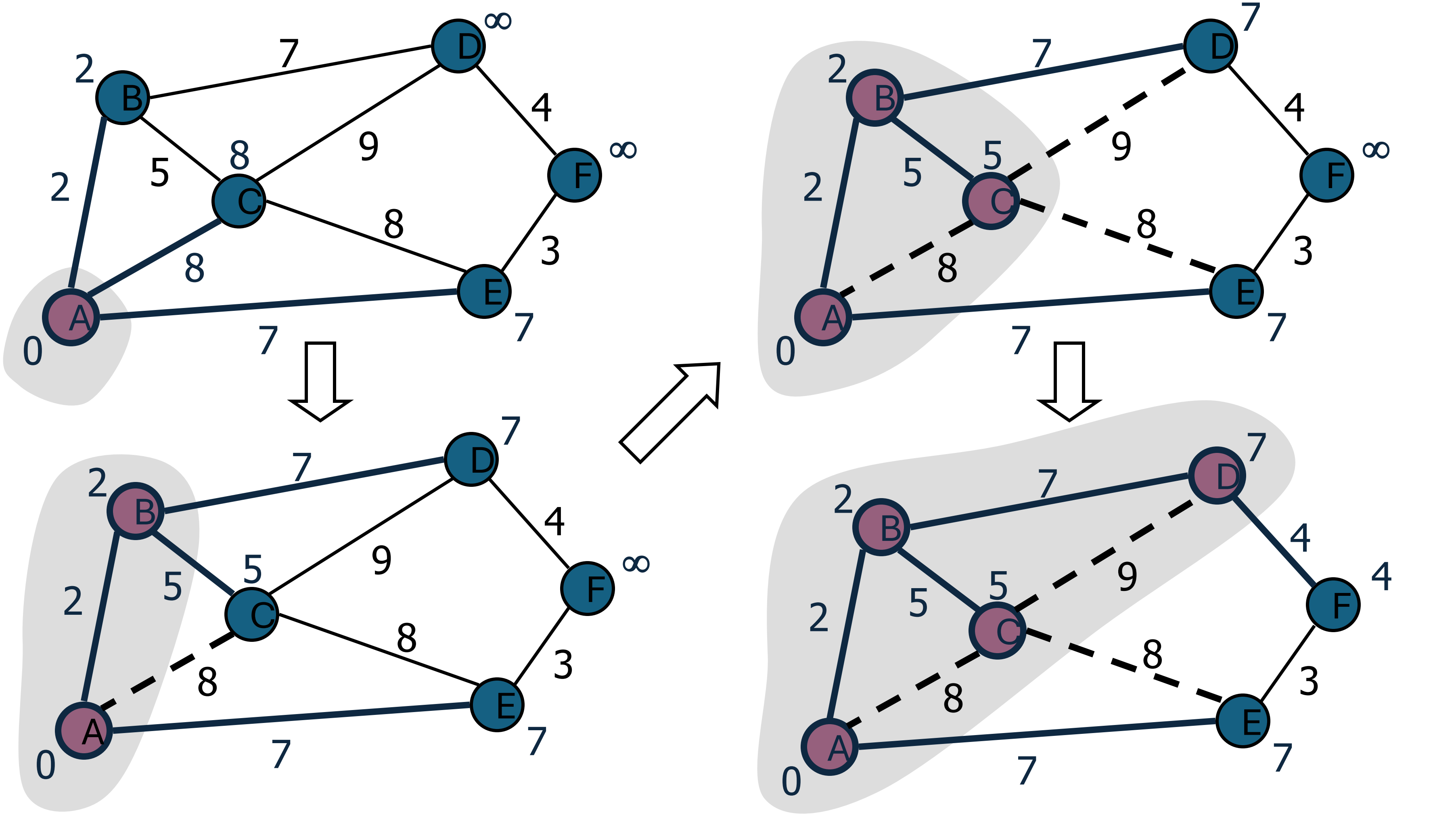


growing to include the vertex $v \notin S$ of smallest `cost`:

$$\texttt{cost}(v) \quad = \quad \min_{u \in S} w(u, v).$$

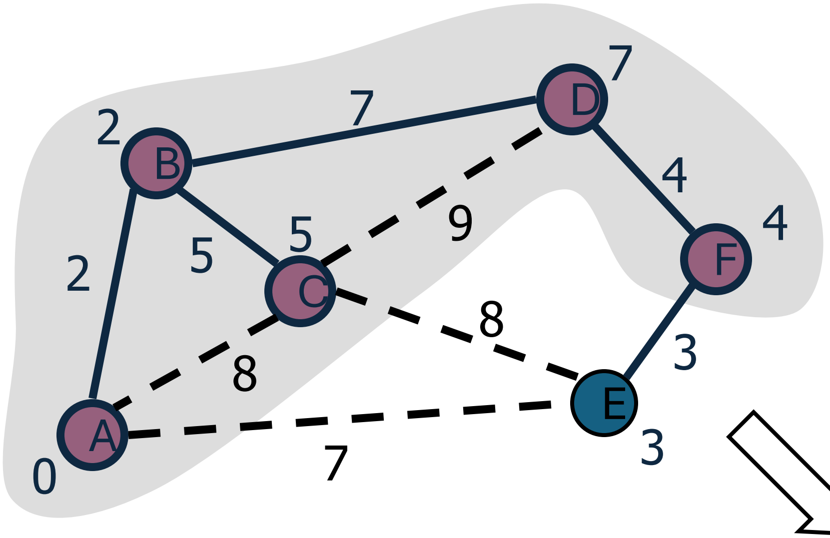
Does this look familiar?

Example



2
B
7
D
∞
4
2
5
8
C
9
F
∞
8
3
8
A
E
0
7
7
2
B
7
D
7
4
2
5
5
C
9
F
∞
8
3
8
A
E
0
7
7
2
B
7
D
7
4
2
5
5
C
9
F
∞
8
3
8
A
E
0
7
7
2
B
7
D
7
4
2
5
5
C
9
F
4
8
3
8
A
E
0
7
7

Example (contd.)



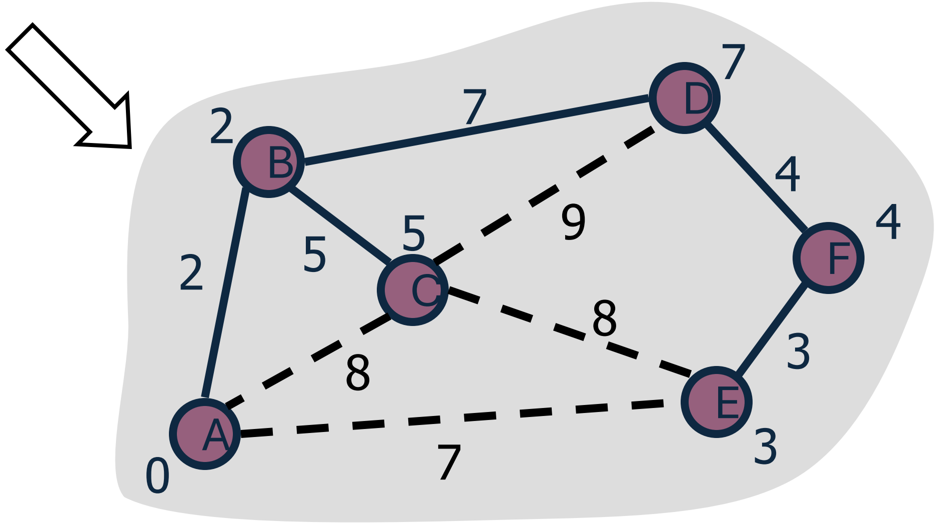


Figure 5.9 *Top:* Prim's minimum spanning tree algorithm. *Below:* An illustration of Prim's algorithm, starting at node A. Also shown are a table of `cost`/`prev` values, and the final MST.

```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
   cost(u) = ∞
   prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)    (priority queue, using cost-values as keys)
while H is not empty:
   v = deletemin(H)
   for each {v, z} ∈ E:
      if cost(z) > w(v, z):
         cost(z) = w(v, z)
         prev(z) = v
         decreasekey(H, z)
```