20.3
The Algorithms for computing MST
Greedy Template

Initially $E$ is the set of all edges in $G$
$T$ is empty (* $T$ will store edges of a MST *)

while $E$ is not empty do
    choose $e \in E$
    remove $e$ from $E$
    if ($e$ satisfies condition)
        add $e$ to $T$

return the set $T$

Main Task: In what order should edges be processed? When should we add edge to spanning tree?
Kruskal’s Algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s Algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s Algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s Algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s Algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s Algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s Algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

![Graph showing Prim's algorithm animation](attachment:image.png)
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

![Diagram of Prim's Algorithm](image)
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

![Diagram of Prim's Algorithm Animation]
Prim’s Algorithm: Animation

The maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

![Diagram of Prim's Algorithm]

1. Start with node 1.
2. In the first iteration, add edge (1, 2) with cost 20.
3. In the second iteration, add edge (2, 3) with cost 15.
4. In the third iteration, add edge (3, 4) with cost 3.
5. In the fourth iteration, add edge (4, 5) with cost 17.
6. In the fifth iteration, add edge (5, 6) with cost 28.
7. In the sixth iteration, add edge (6, 7) with cost 23.
8. In the seventh iteration, add edge (7, 1) with cost 9.
9. In the eighth iteration, add edge (1, 4) with cost 25.
10. In the ninth iteration, add edge (4, 3) with cost 16.
11. The final tree includes all nodes and has a total cost of 107.
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

![Diagram showing the process of Prim's Algorithm]

1. Start with node 1.
2. Add edge (1, 2) with cost 20.
3. Add edge (2, 7) with cost 15.
4. Add edge (7, 3) with cost 9.
5. Add edge (3, 6) with cost 16.
6. Add edge (6, 5) with cost 25.
7. Add edge (5, 4) with cost 3.
8. Add edge (4, 7) with cost 28.
9. Add edge (7, 2) with cost 4.
10. Add edge (2, 1) with cost 20.
11. Add edge (1, 6) with cost 23.
12. Add edge (6, 3) with cost 15.
13. Add edge (3, 4) with cost 3.
14. Add edge (4, 5) with cost 17.
15. Add edge (5, 7) with cost 25.
16. Add edge (7, 3) with cost 9.

The final tree $T$ connects all nodes with the minimum possible total edge cost.
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

1. Start with node 1.
2. Pick edge with least cost to 1, which is 20, connecting node 2.
3. Pick edge with least cost to node 2, which is 15, connecting node 3.
4. Pick edge with least cost to node 3, which is 17, connecting node 4.
5. Pick edge with least cost to node 4, which is 28, connecting node 5.
6. Pick edge with least cost to node 5, which is 25, connecting node 6.
7. Pick edge with least cost to node 6, which is 9, connecting node 7.
8. Pick edge with least cost to node 7, which is 16, connecting node 3.
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

![Diagram of Prim's Algorithm](image-url)
Prim’s Algorithm: Animation

$T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$. 

![Diagram of Prim's Algorithm](image-url)
Reverse Delete Algorithm

Initially $Z$ is the set of all edges in $G$

$T \leftarrow Z$ (* $T$ will store edges of a MST *)

while $Z$ is not empty do

choose $e \in Z$ of largest cost

remove $e$ from $Z$

if removing $e$ does not disconnect $T$ then

remove $e$ from $T$

return the set $T$

Returns a minimum spanning tree.
Borůvka’s Algorithm

Simplest to implement. See notes.
Assume $G$ is a connected graph.

\[
\begin{align*}
T & \text{ is } \emptyset \text{ (* } T \text{ will store edges of a MST *)} \\
\text{while } T \text{ is not spanning do} \\
& \quad X \leftarrow \emptyset \\
& \quad \text{for each connected component } S \text{ of } T \text{ do} \\
& \quad \quad \text{add to } X \text{ the cheapest edge between } S \text{ and } V \setminus S \\
& \quad \text{Add edges in } X \text{ to } T \\
\text{return the set } T
\end{align*}
\]
Borůvka’s Algorithm
Borůvka’s Algorithm
Borůvka’s Algorithm
Borůvka’s Algorithm
Borůvka’s Algorithm
Borůvka’s Algorithm
Borůvka’s Algorithm
Borůvka’s Algorithm

1, 2, 6, 7

3, 4, 5
Borůvka’s Algorithm
Borůvka’s Algorithm
THE END

... (for now)