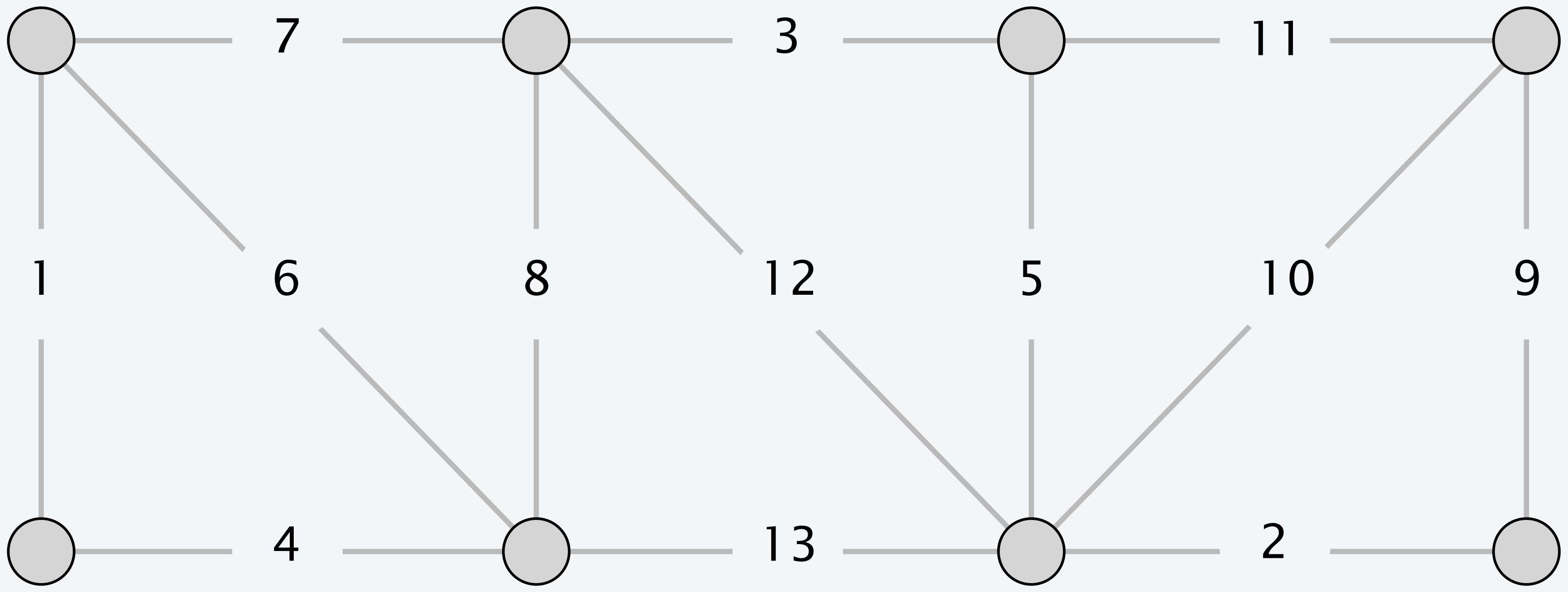
Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

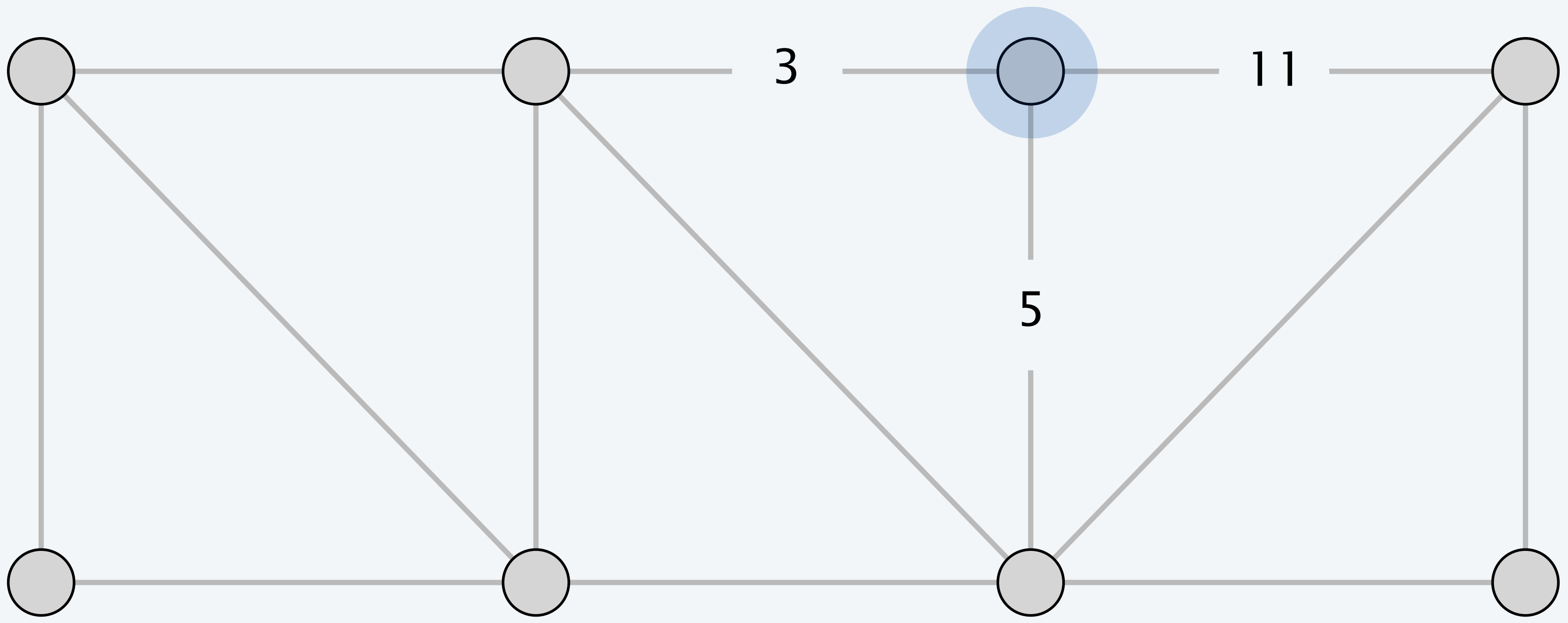


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

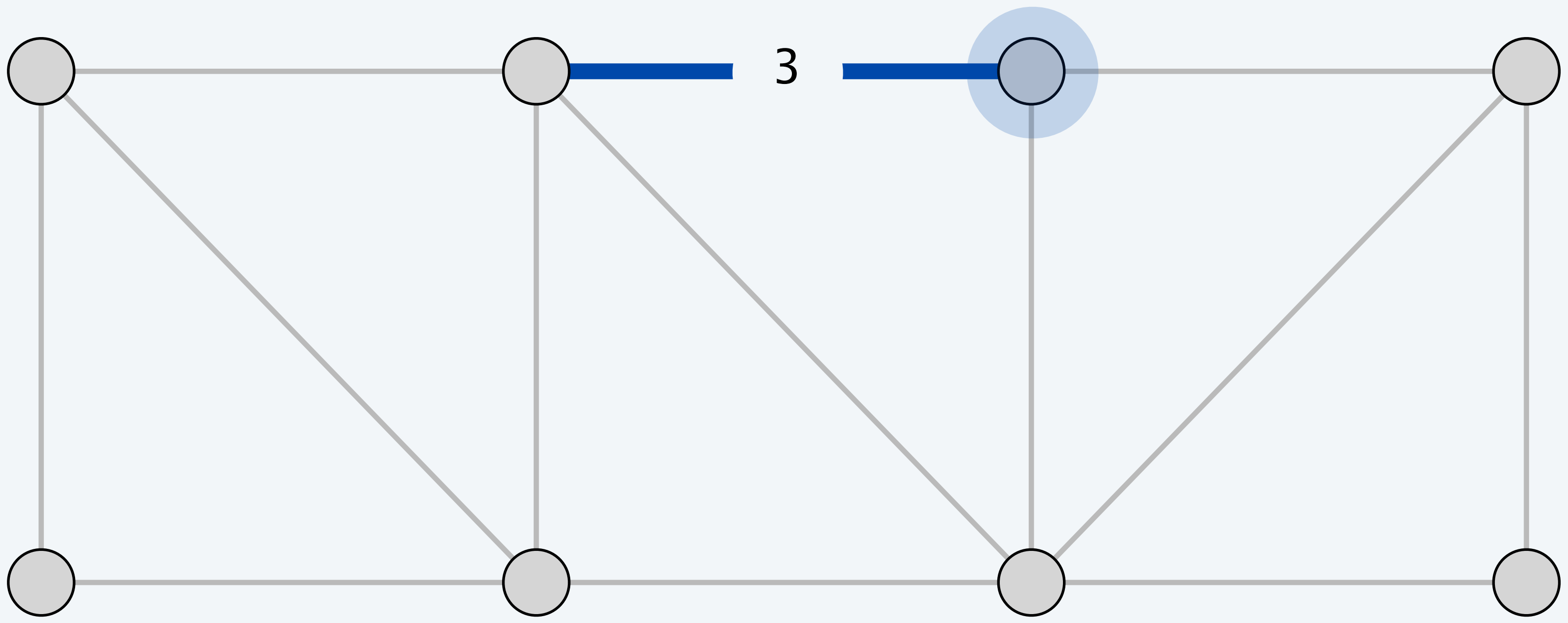


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

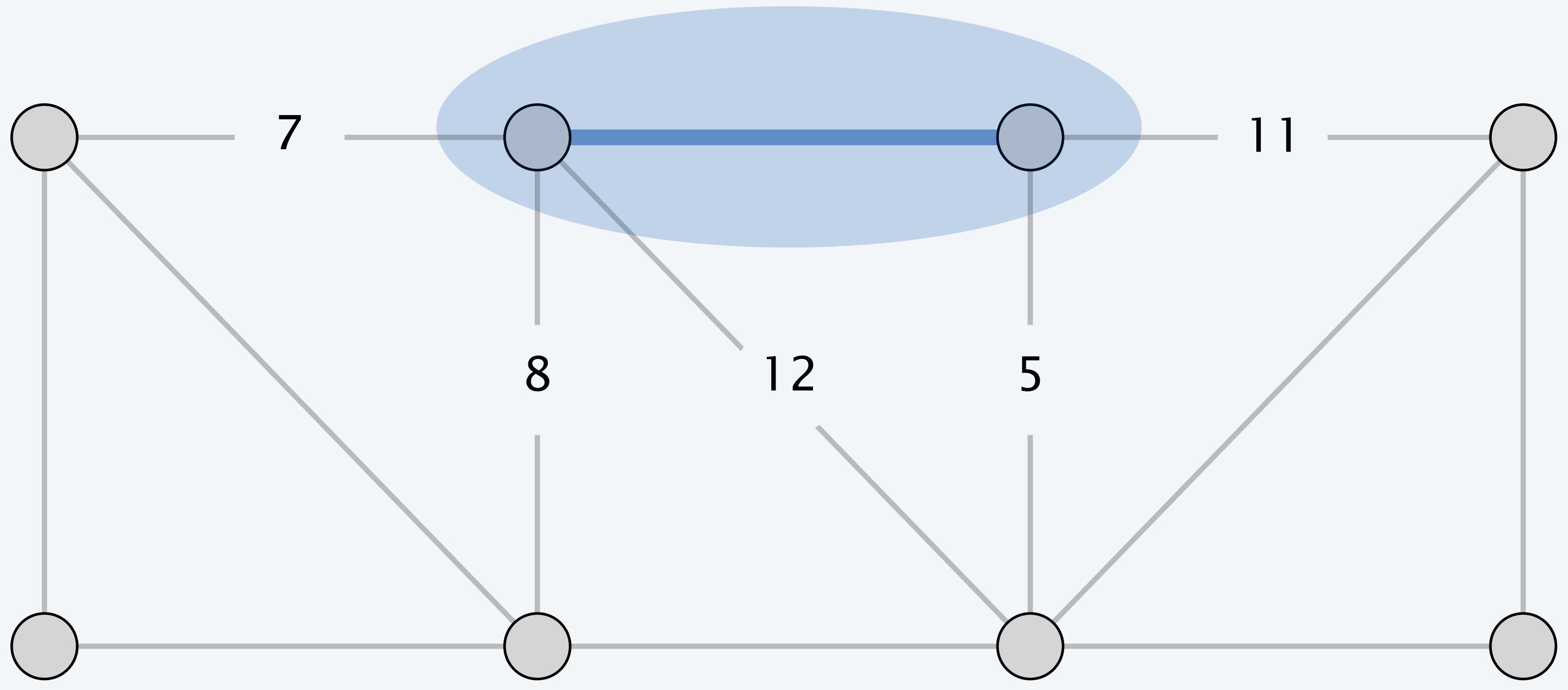


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

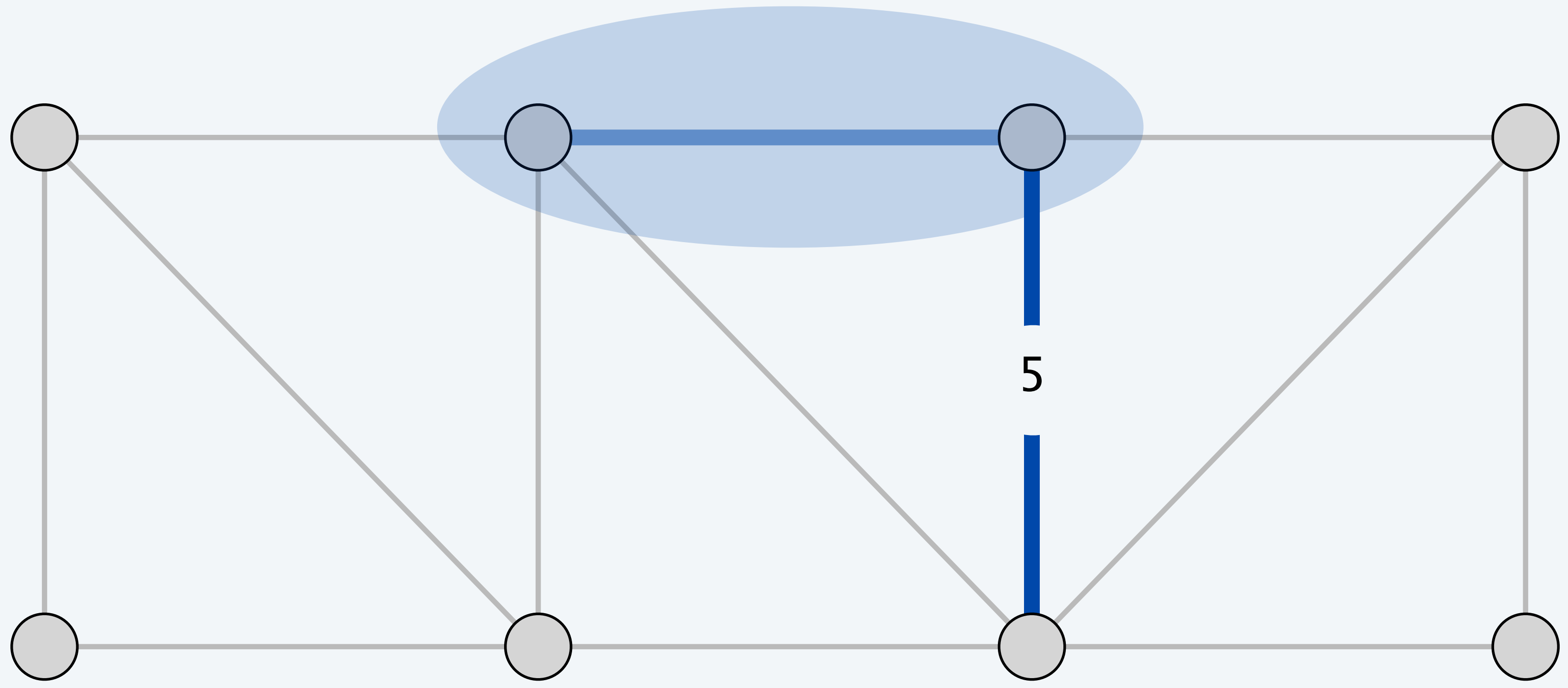


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

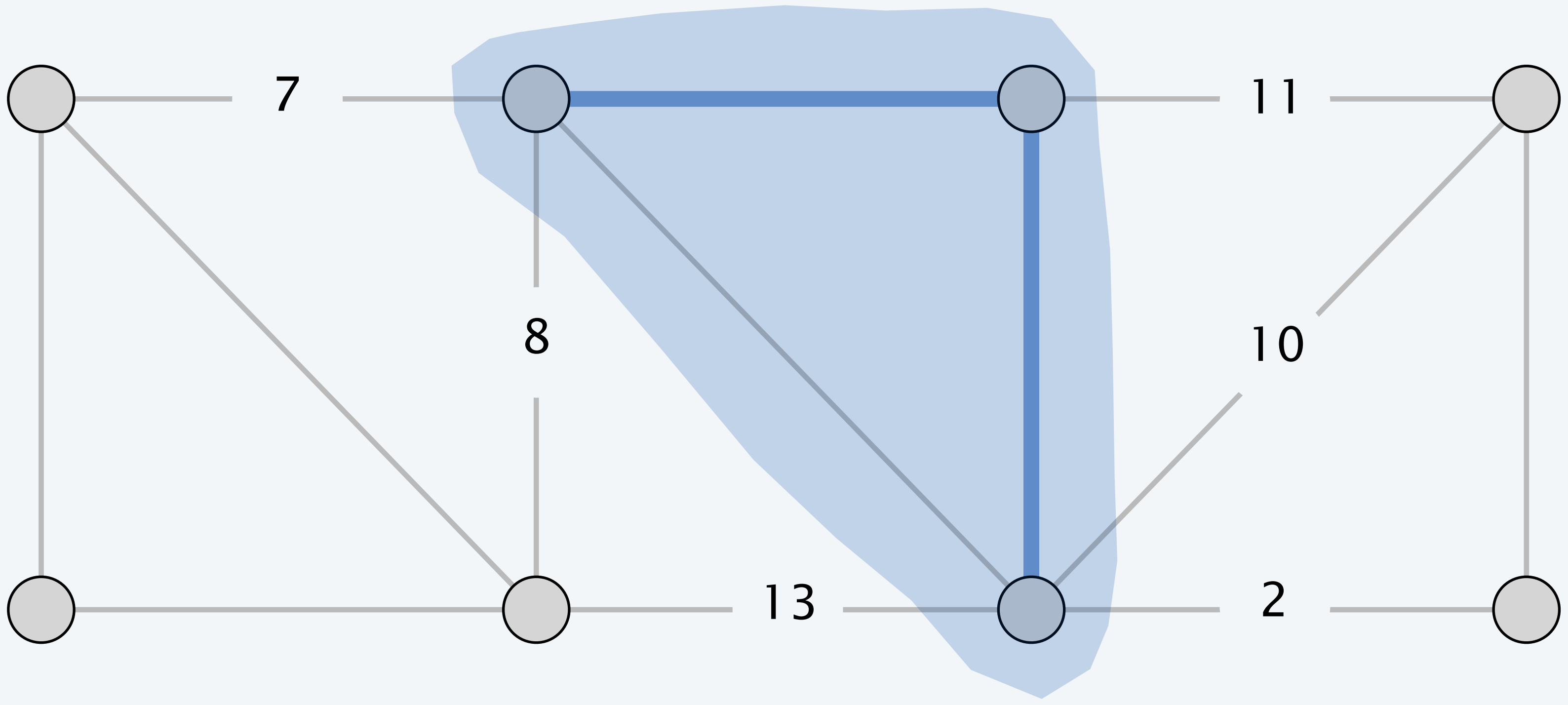


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

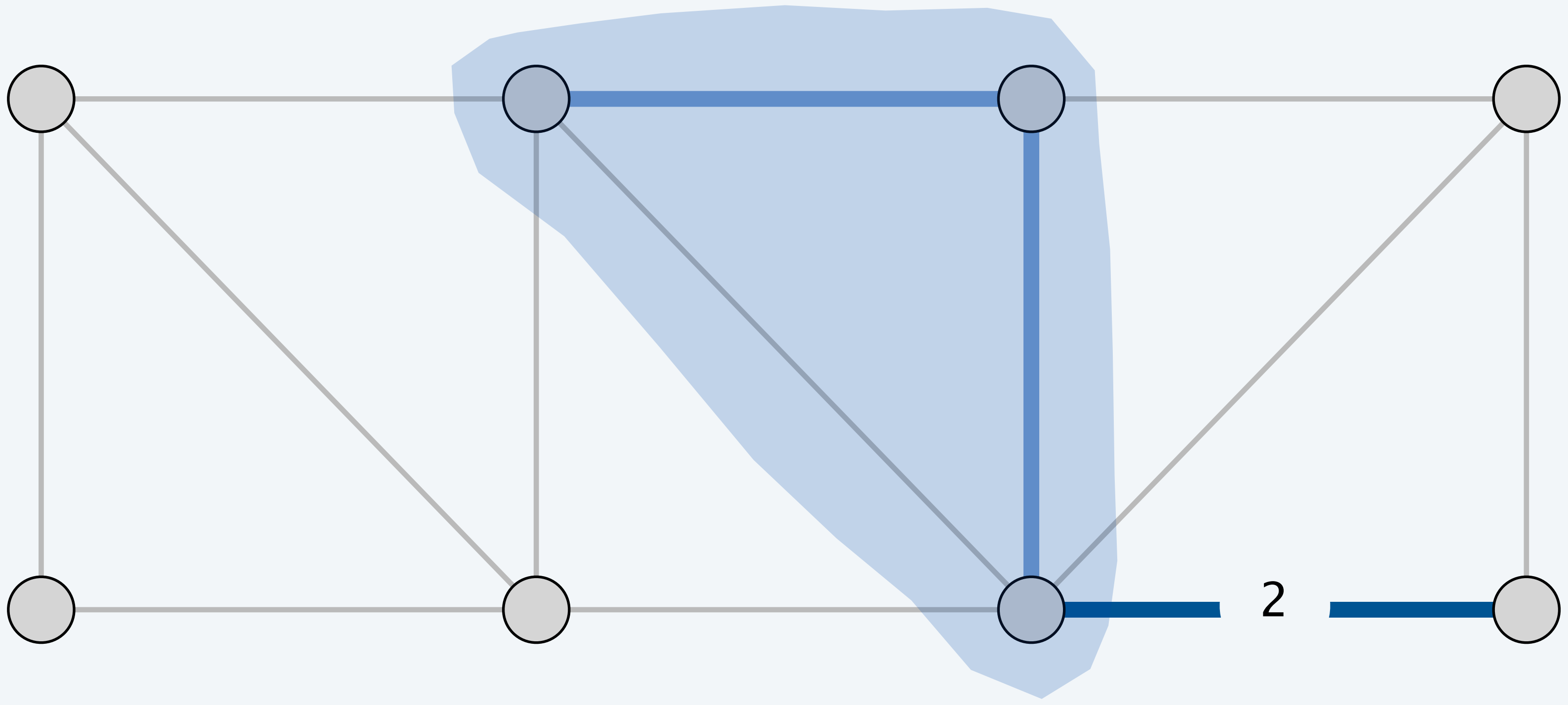


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

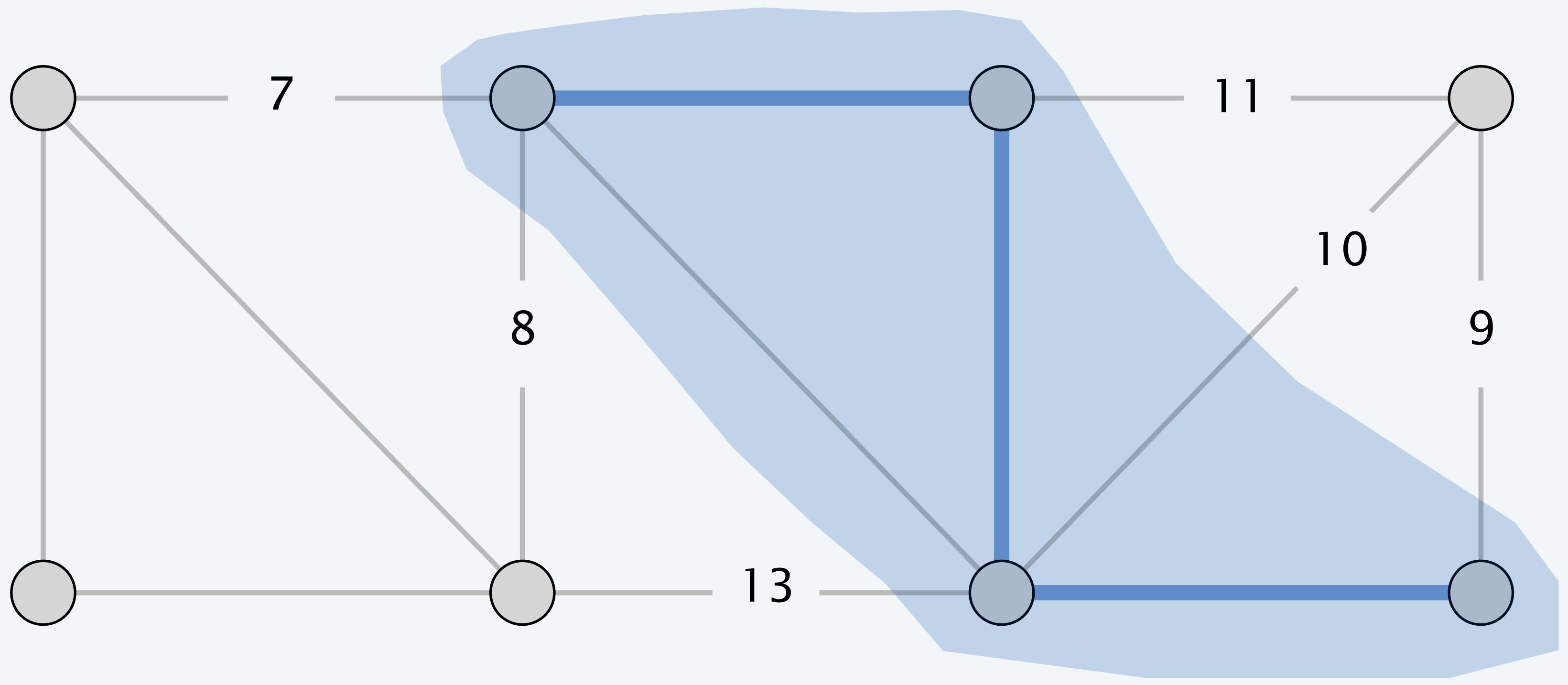


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

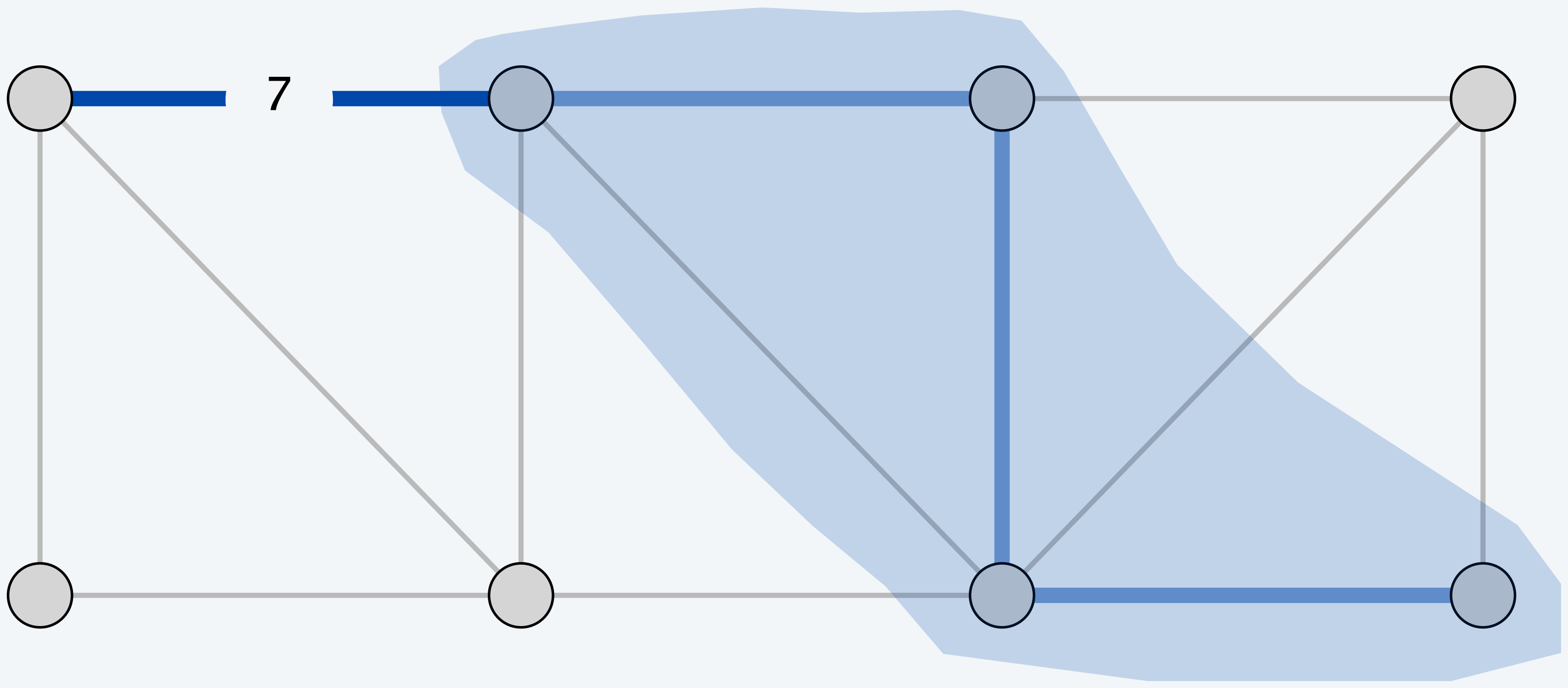


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

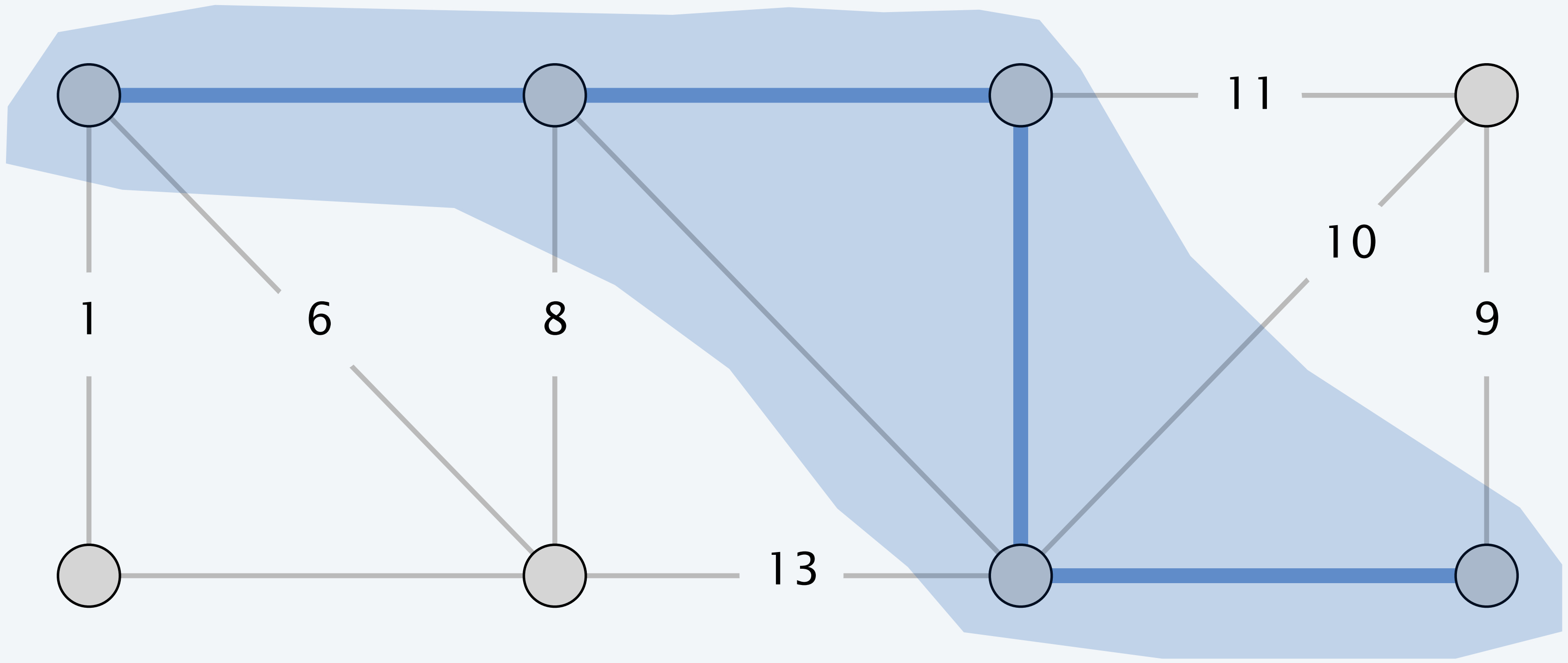


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

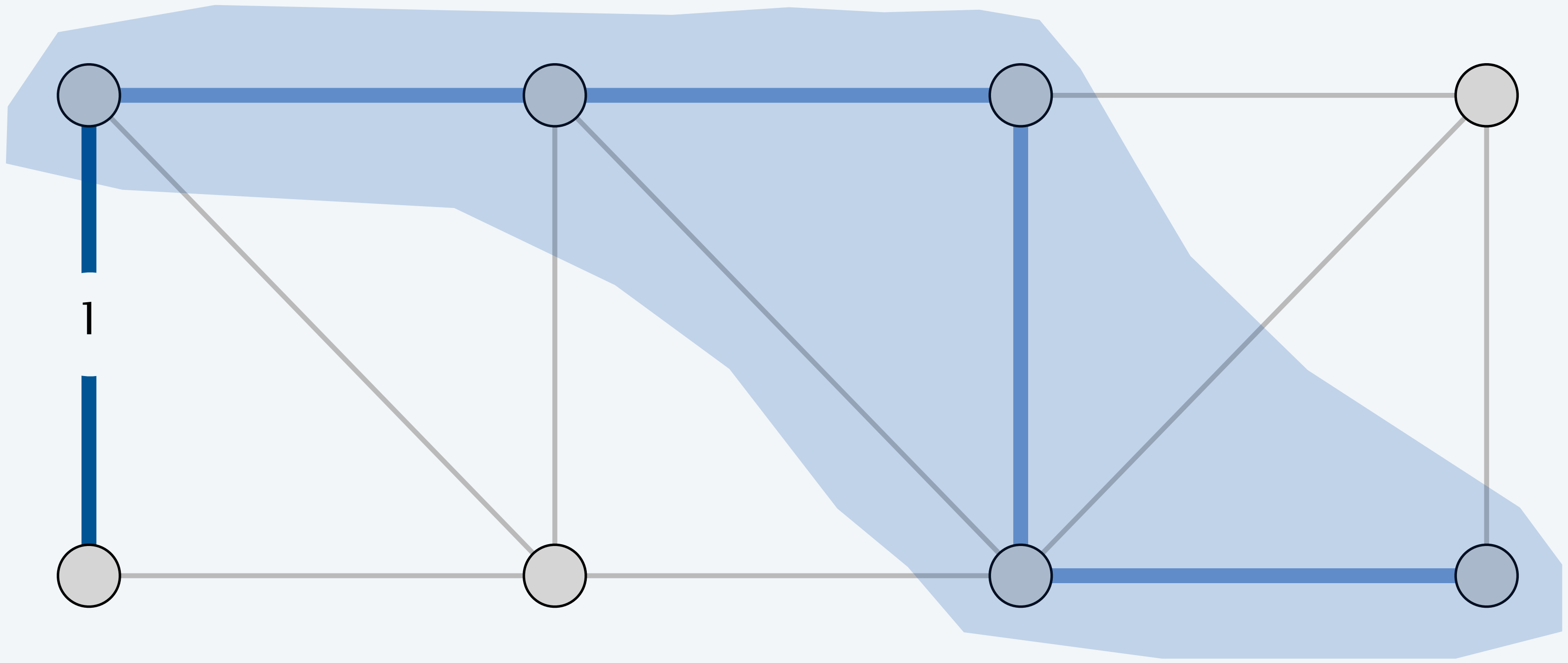


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

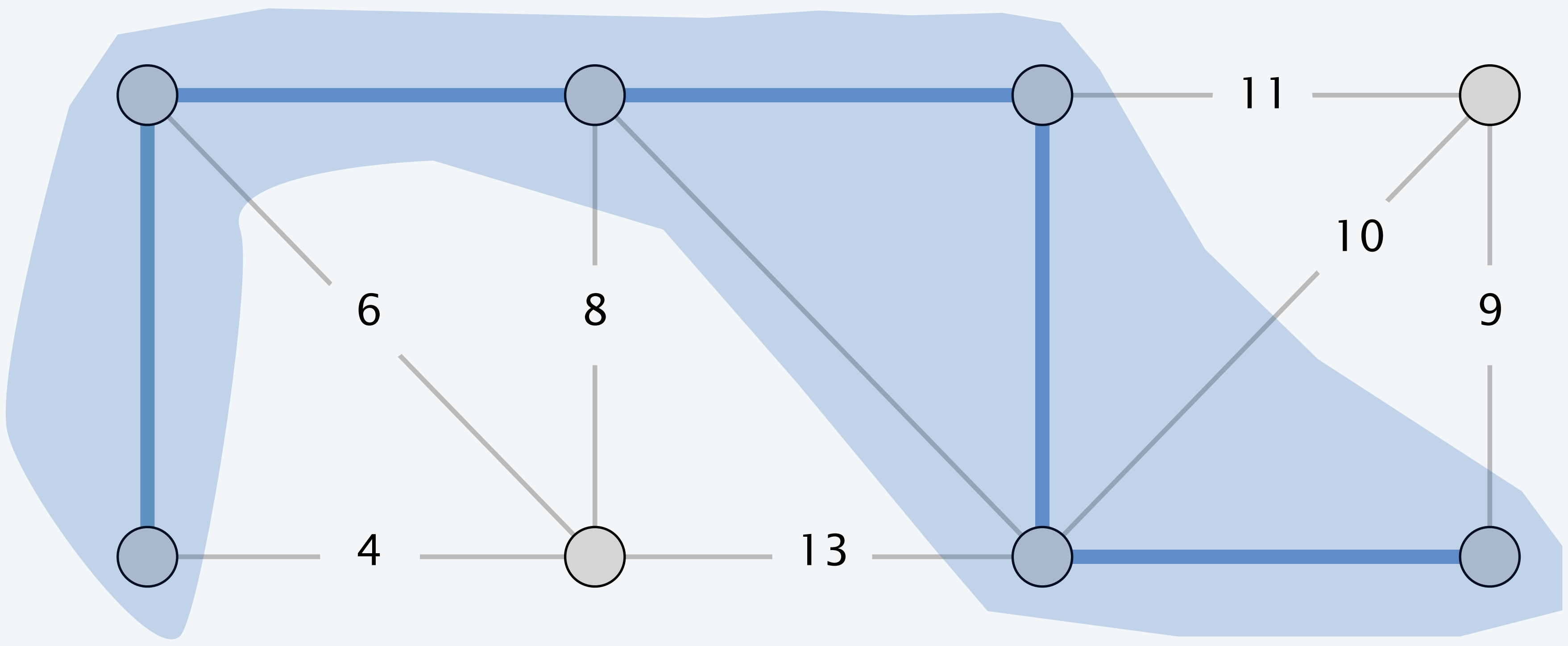


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

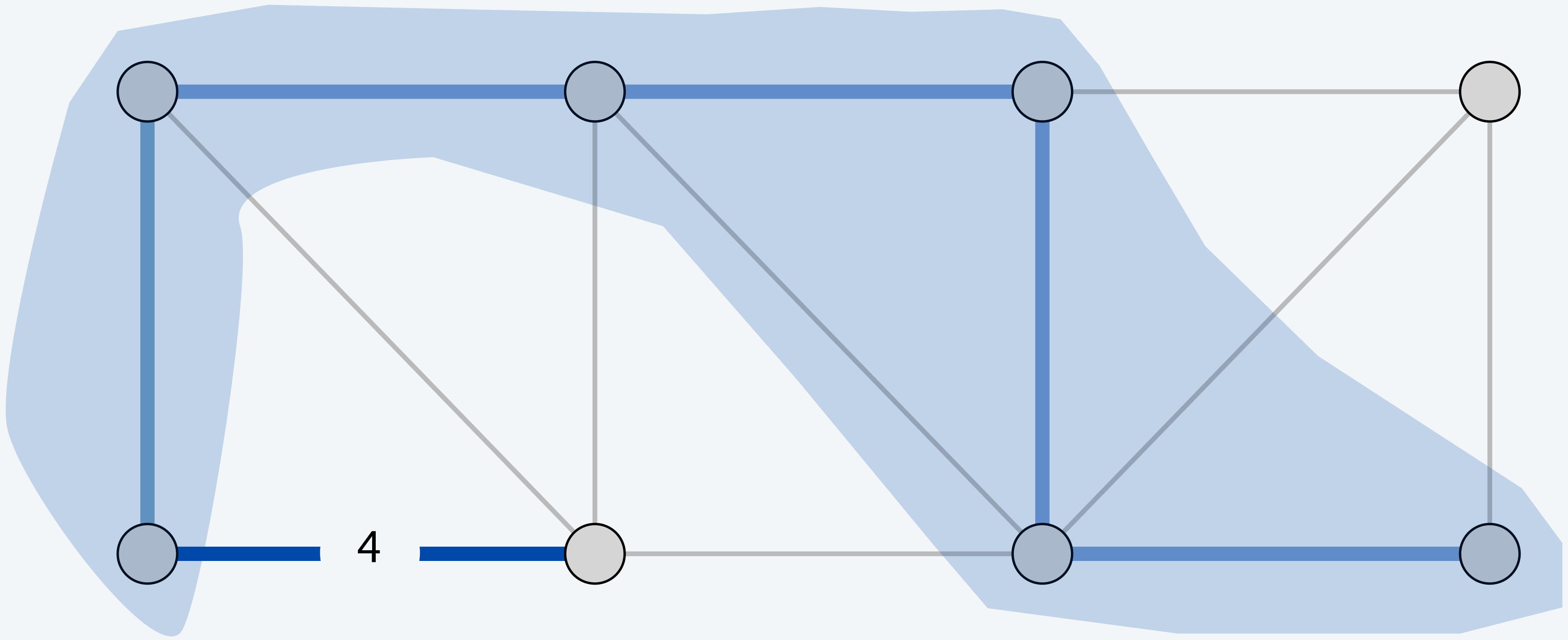


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

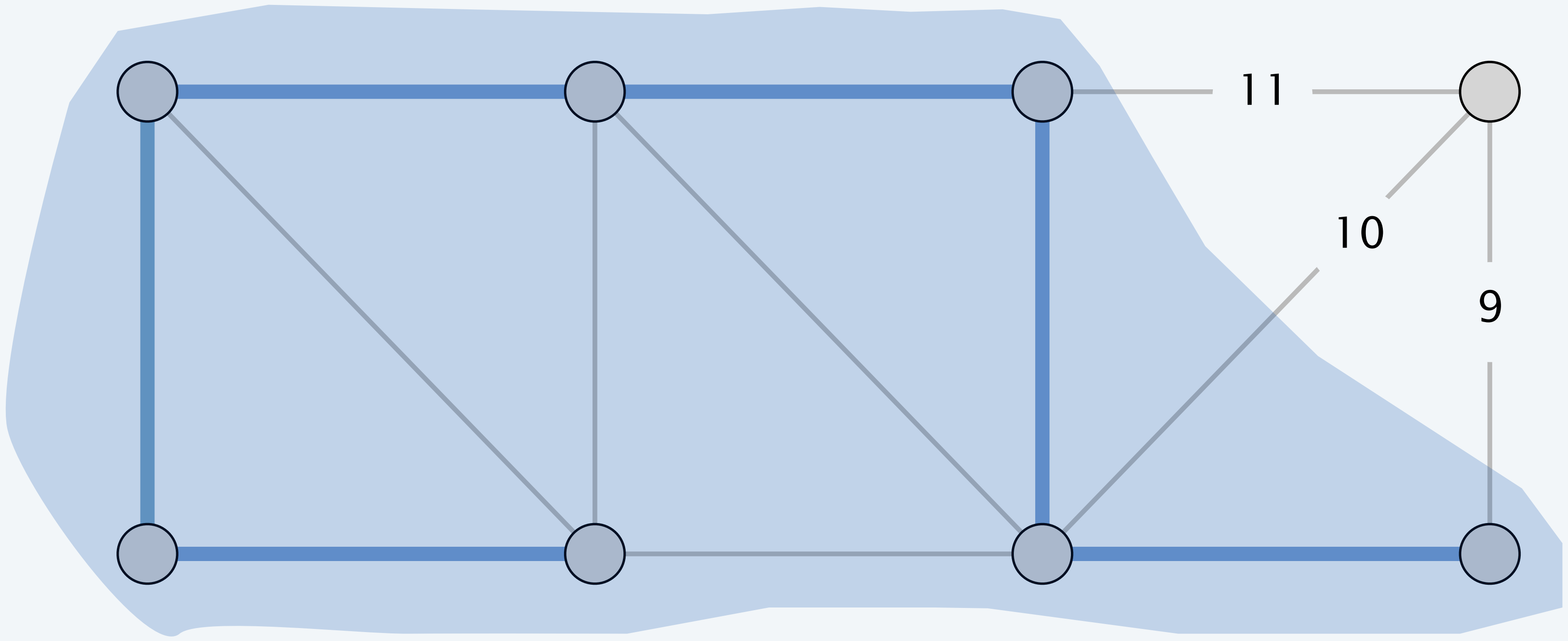


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

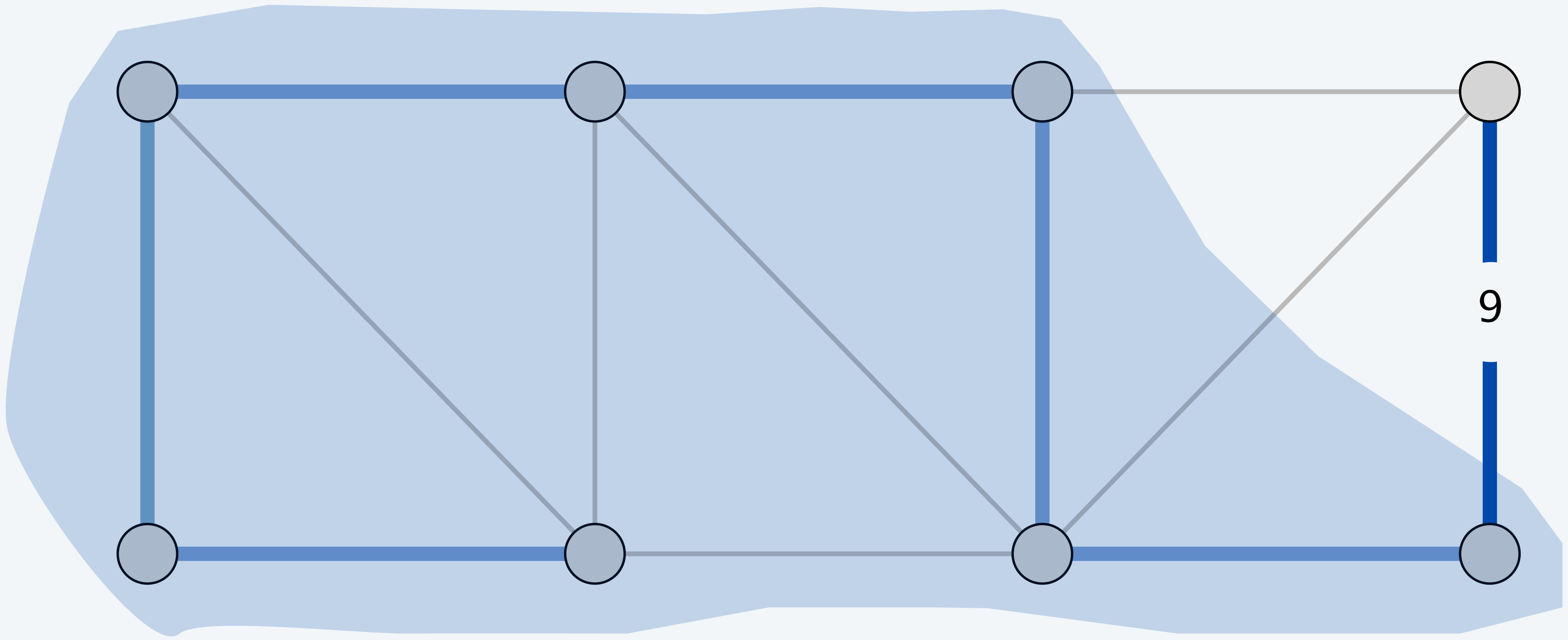


Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

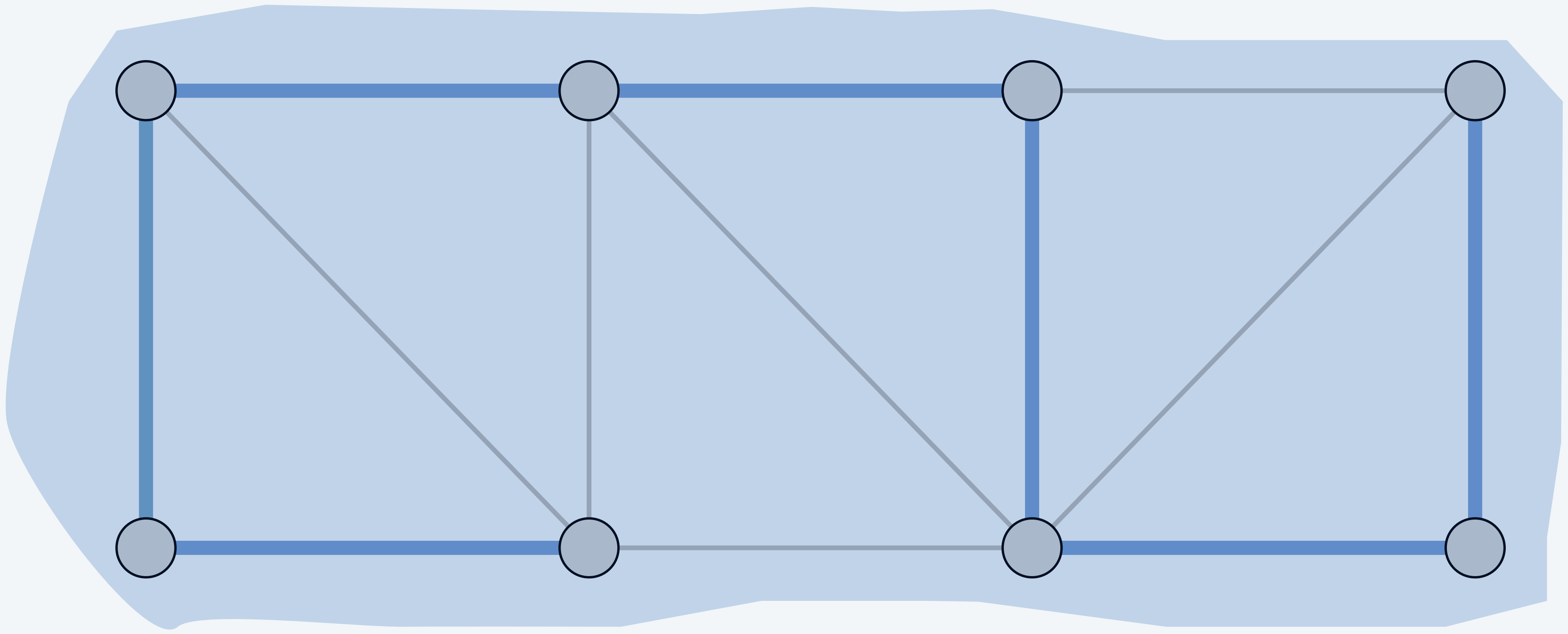


Prim's algorithm demo

Initialize S = any node, $T = \emptyset$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.

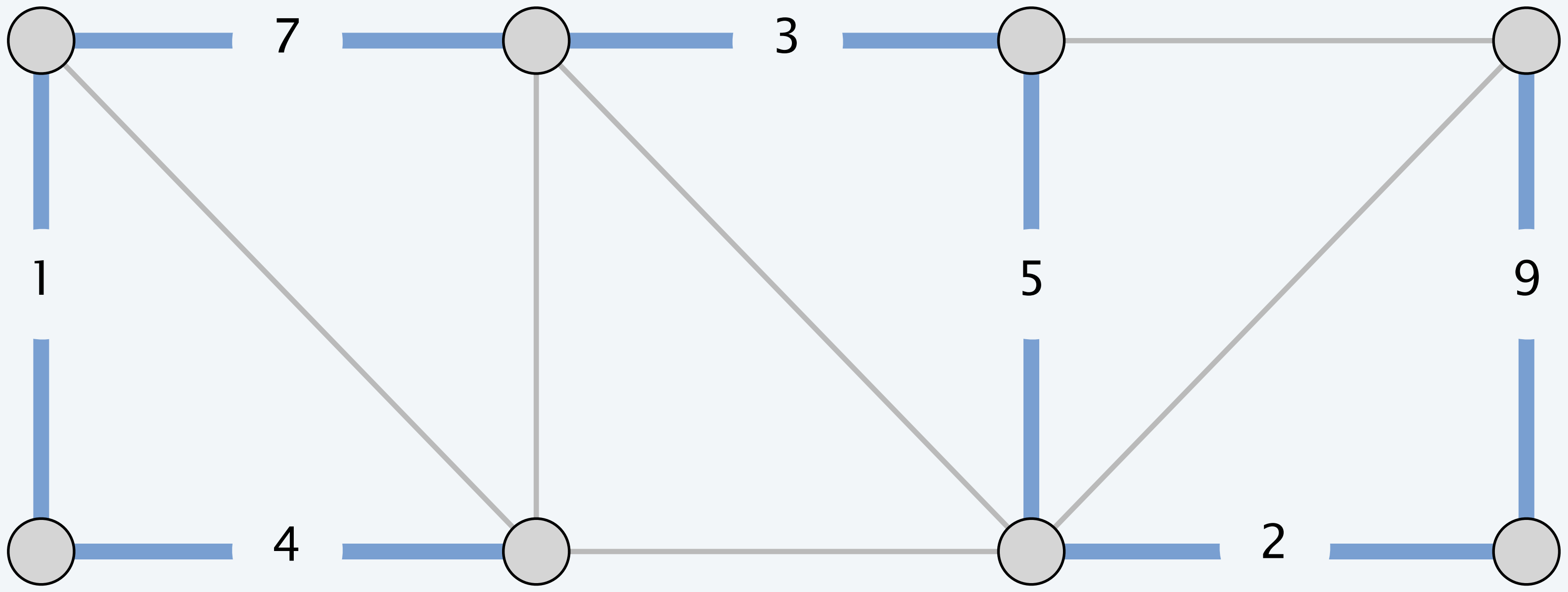


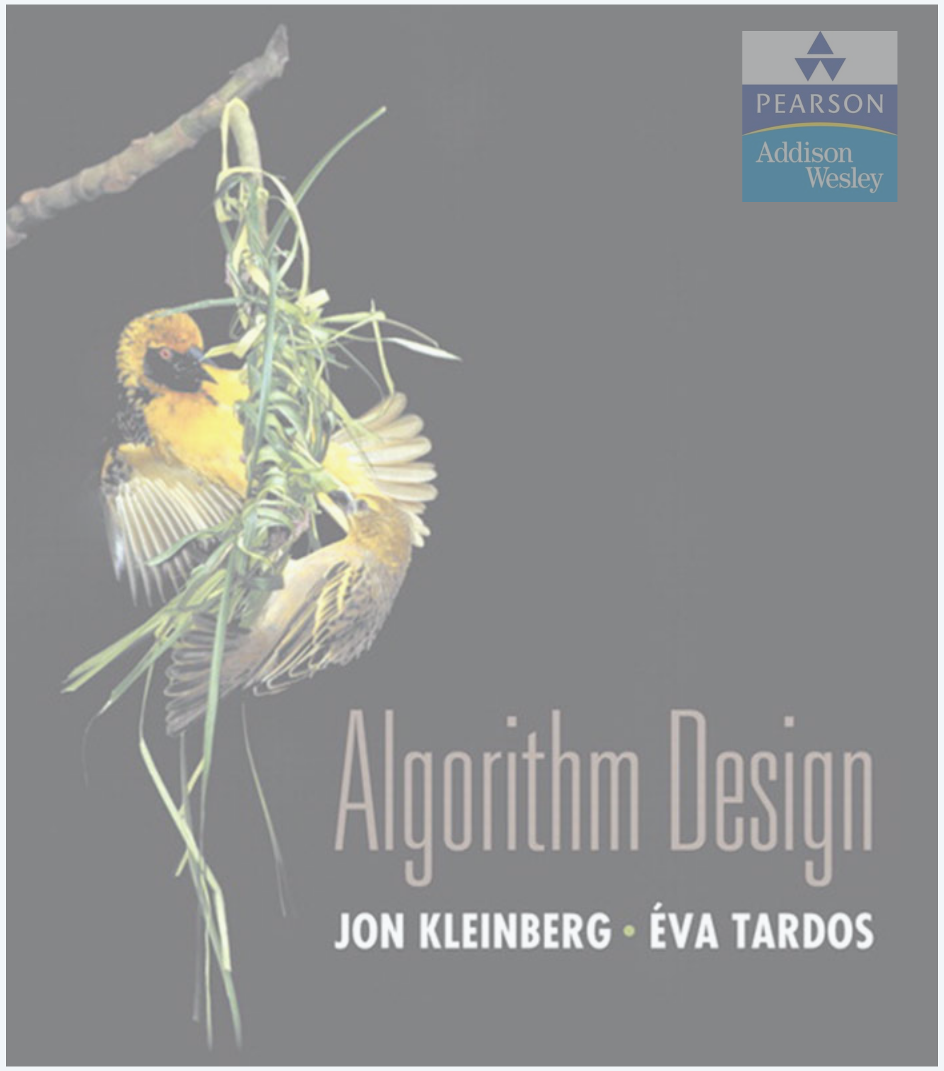
Prim's algorithm demo

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node to S.





SECTION 4.5

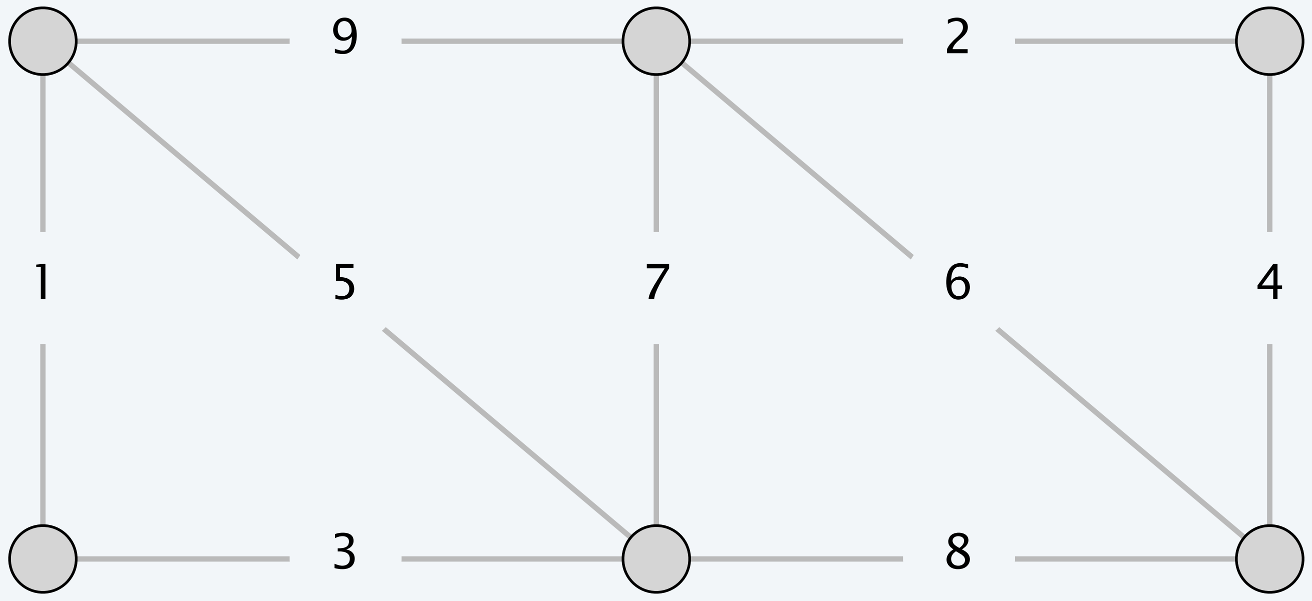
4. GREEDY ALGORITHMS II

- red-rule blue-rule demo
- Prim's algorithm demo
- *Kruskal's algorithm demo*
- reverse-delete algorithm demo
- Boruvka's algorithm demo

Kruskal's algorithm demo

Consider edges in ascending order of weight:

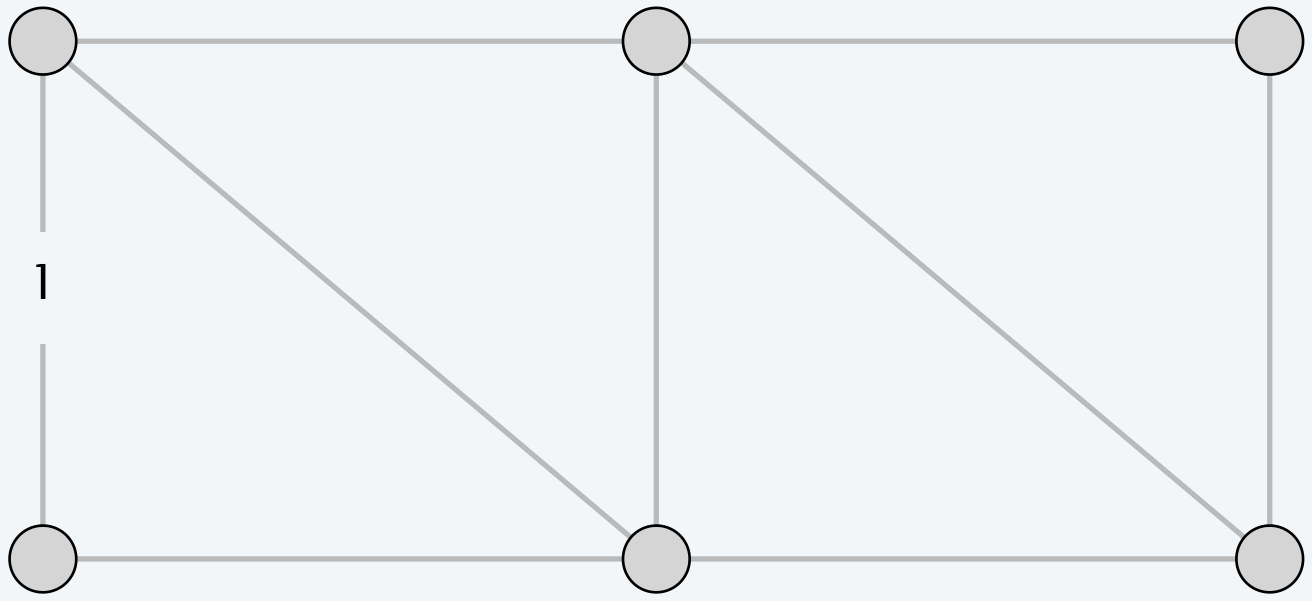
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

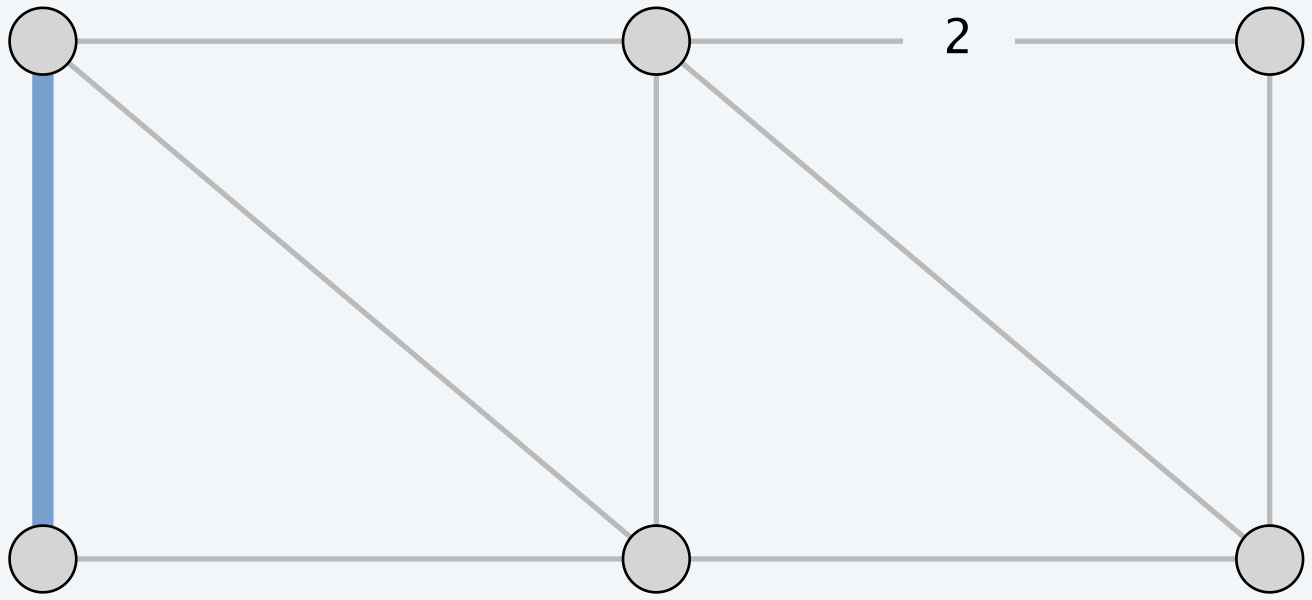
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

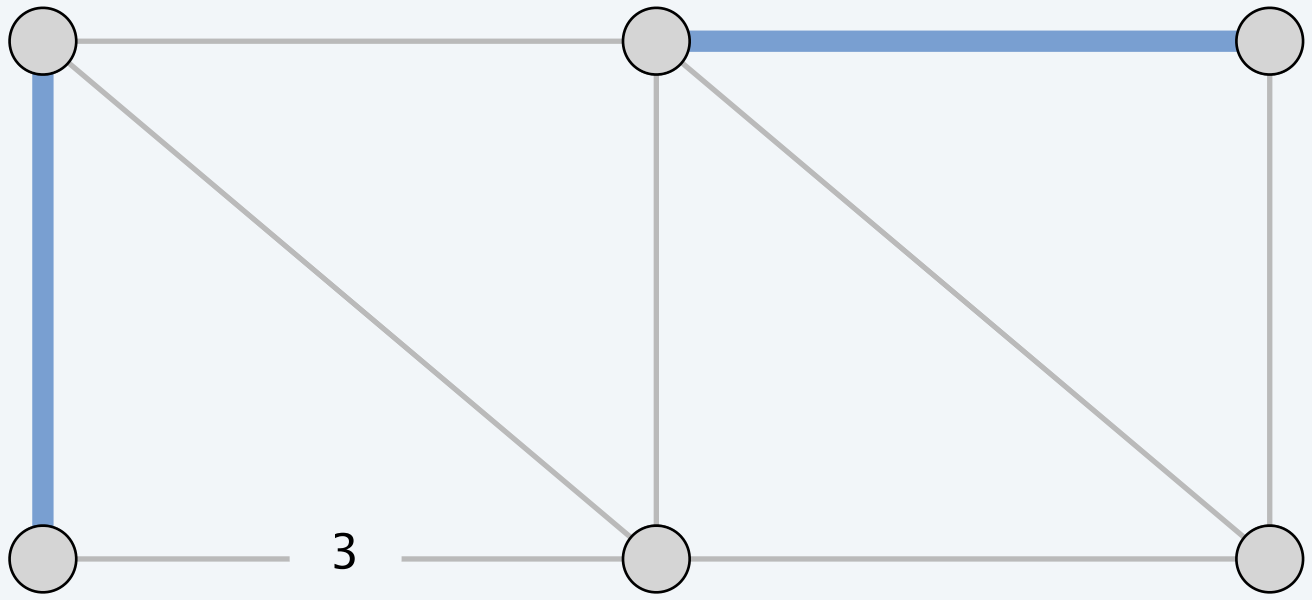
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

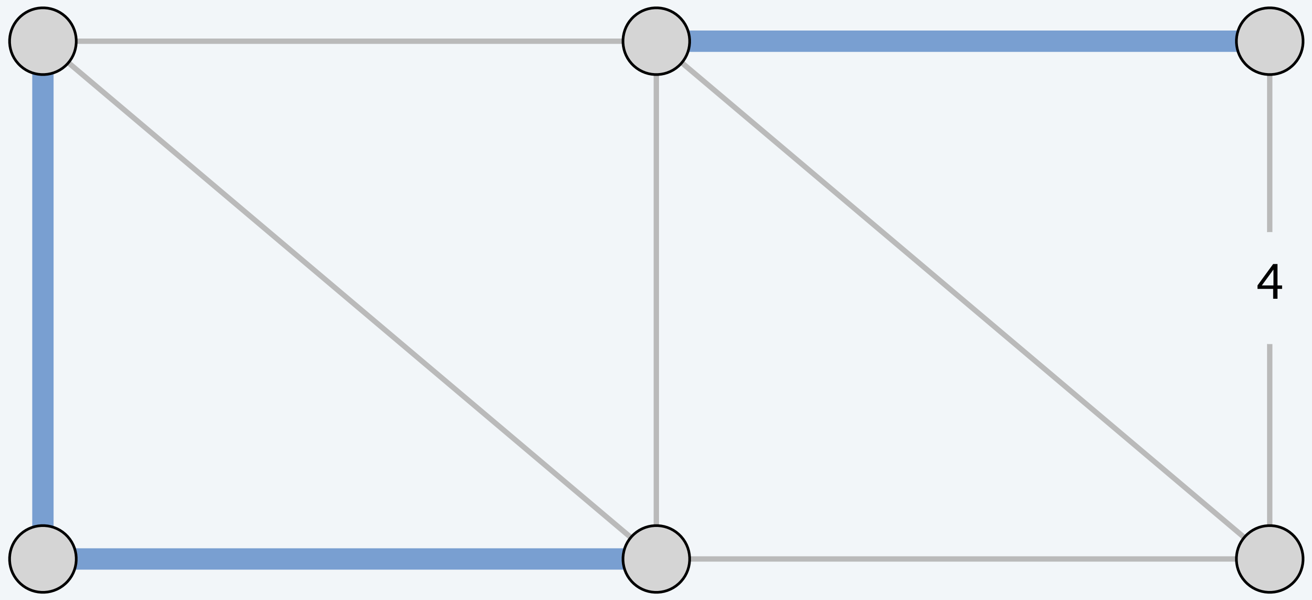
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

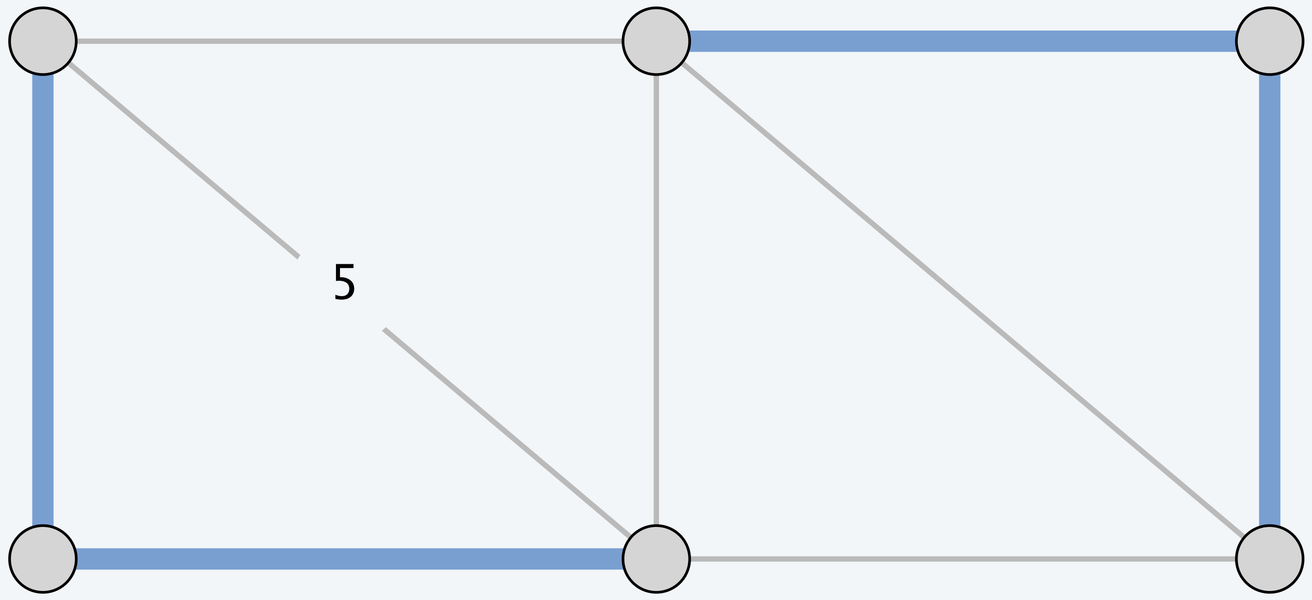
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

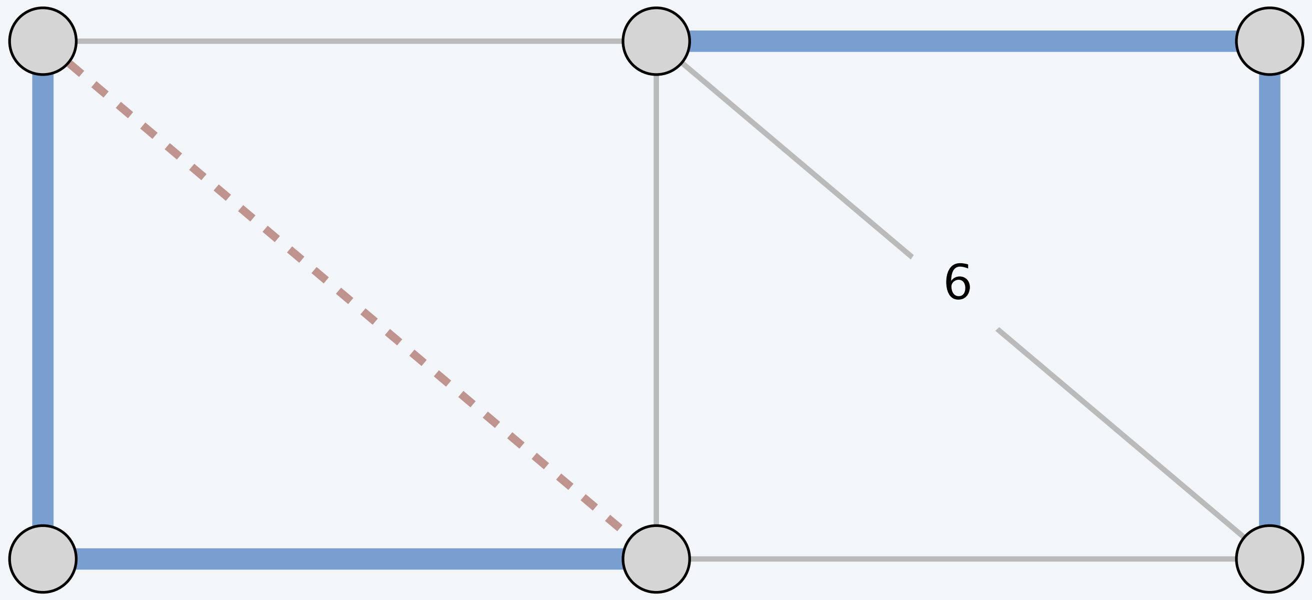
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

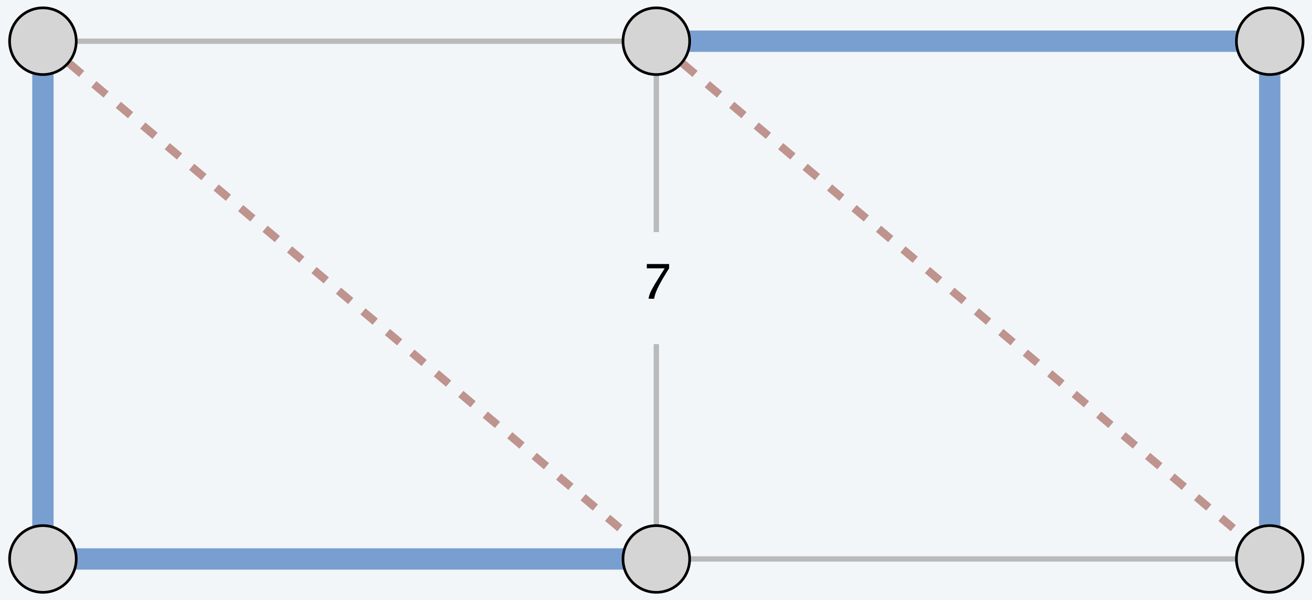
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

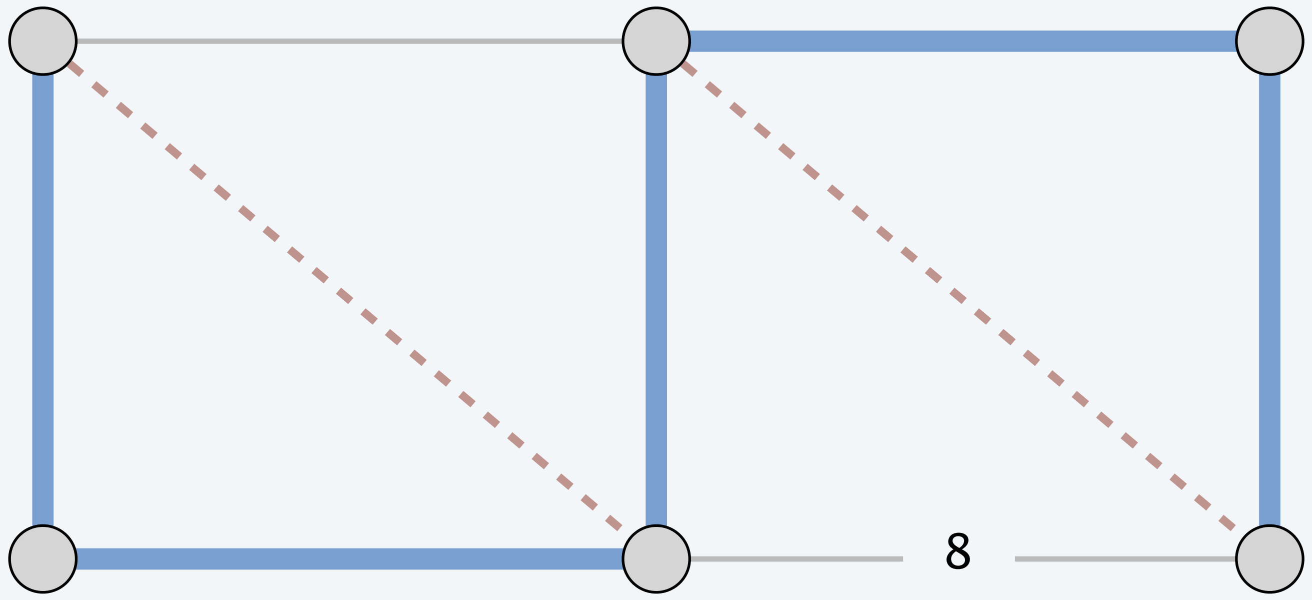
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

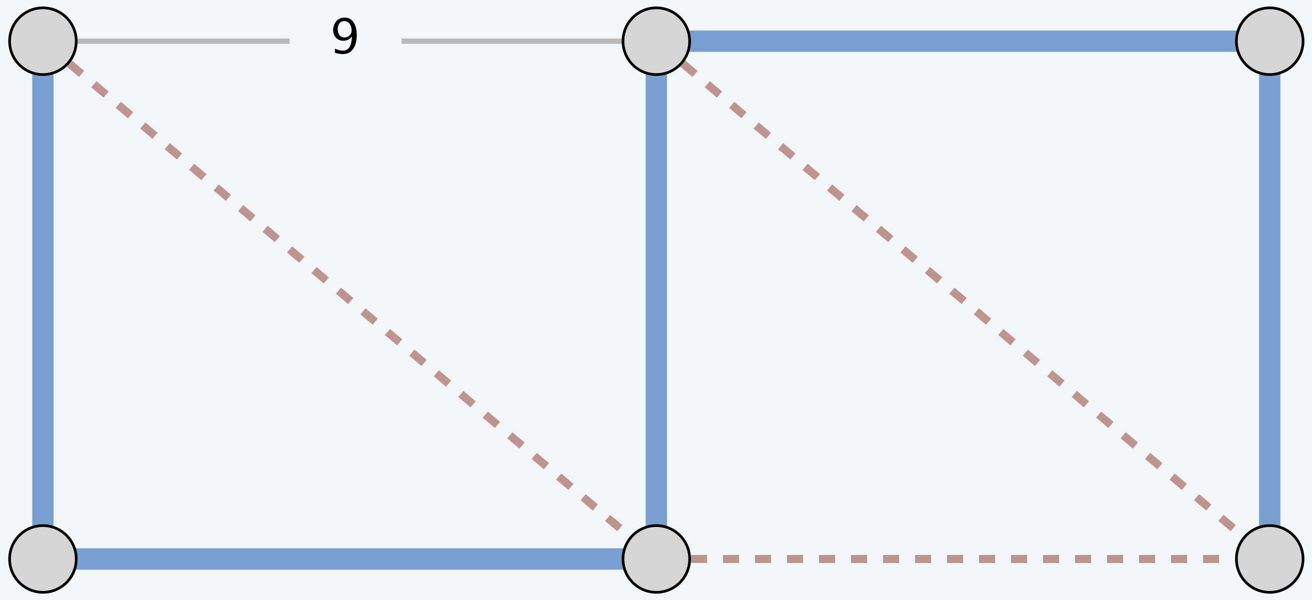
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

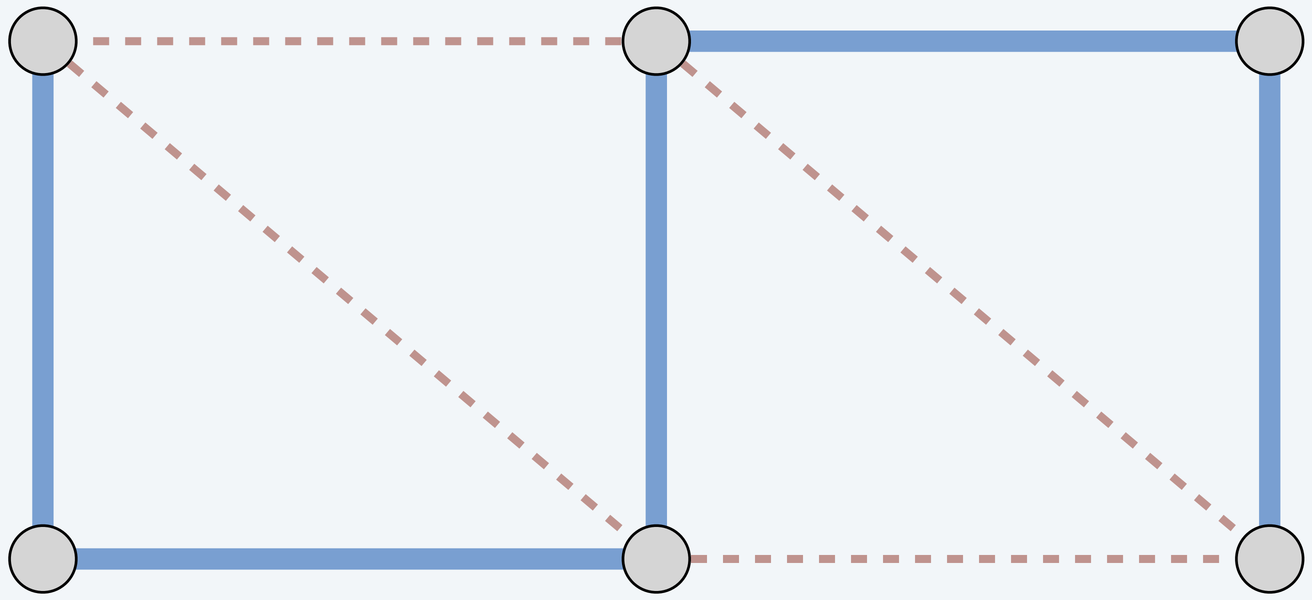
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

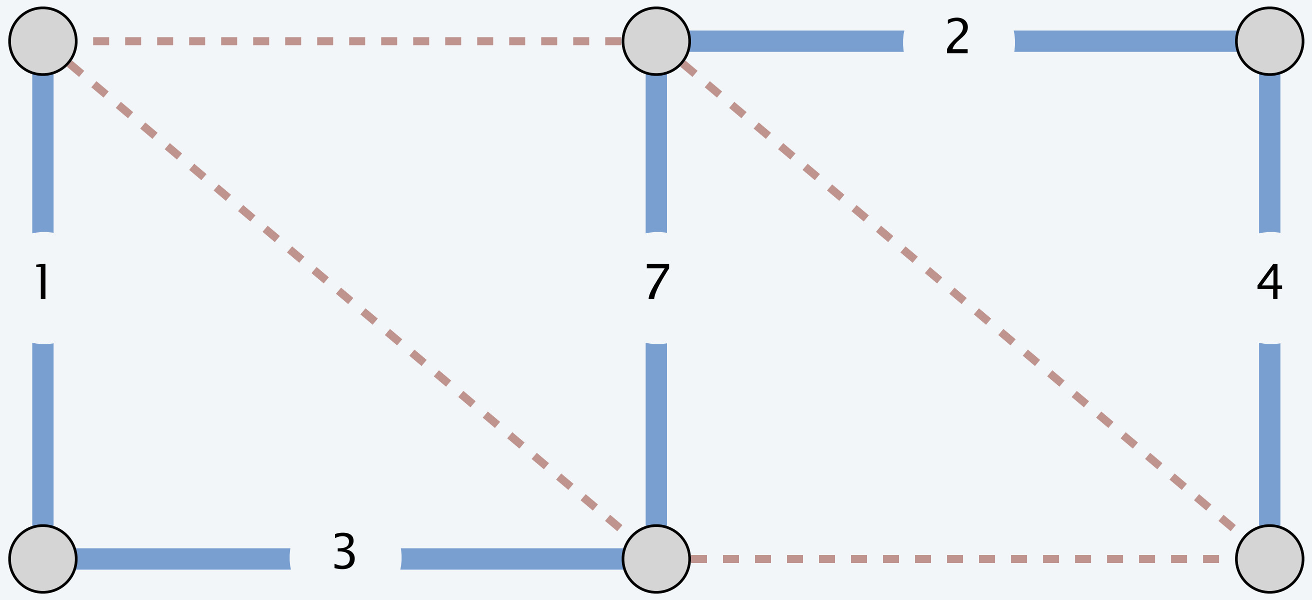
- Add to T unless it would create a cycle.

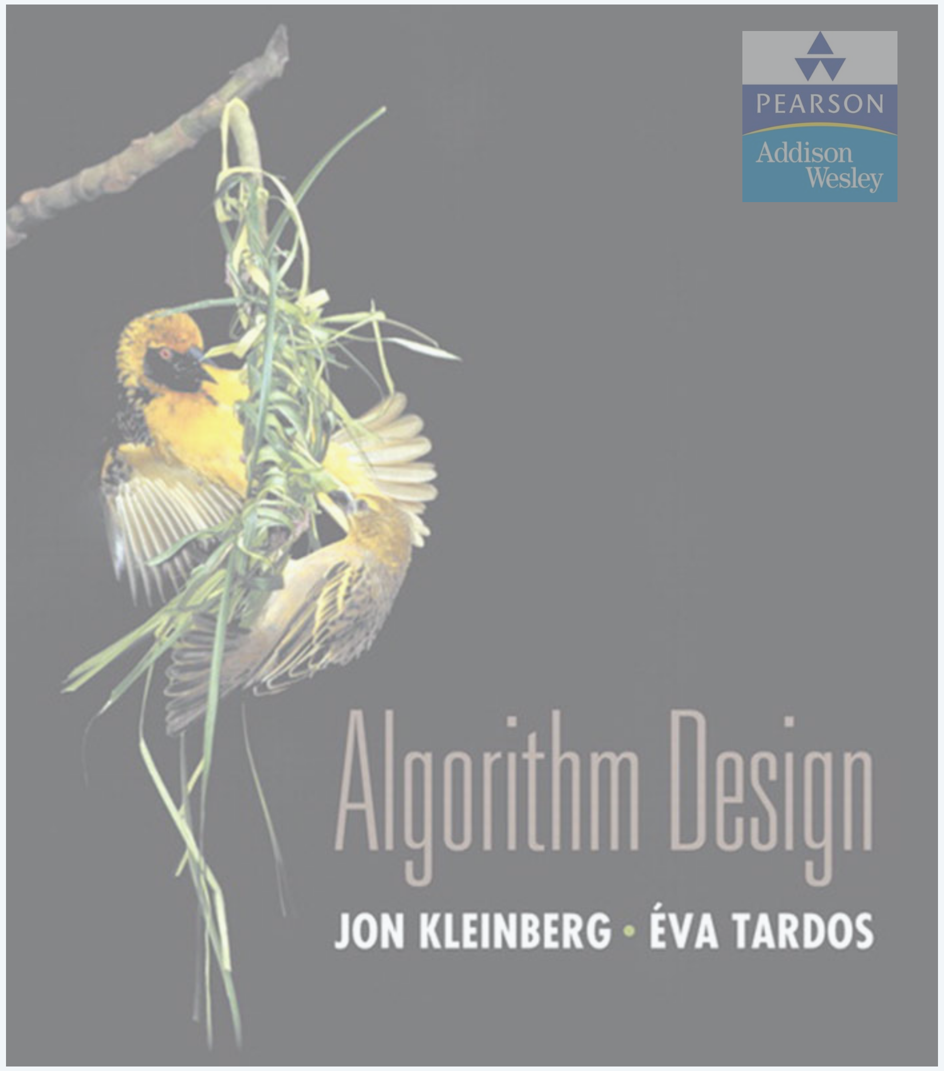


Kruskal's algorithm demo

Consider edges in ascending order of weight:

- Add to T unless it would create a cycle.





SECTION 4.5

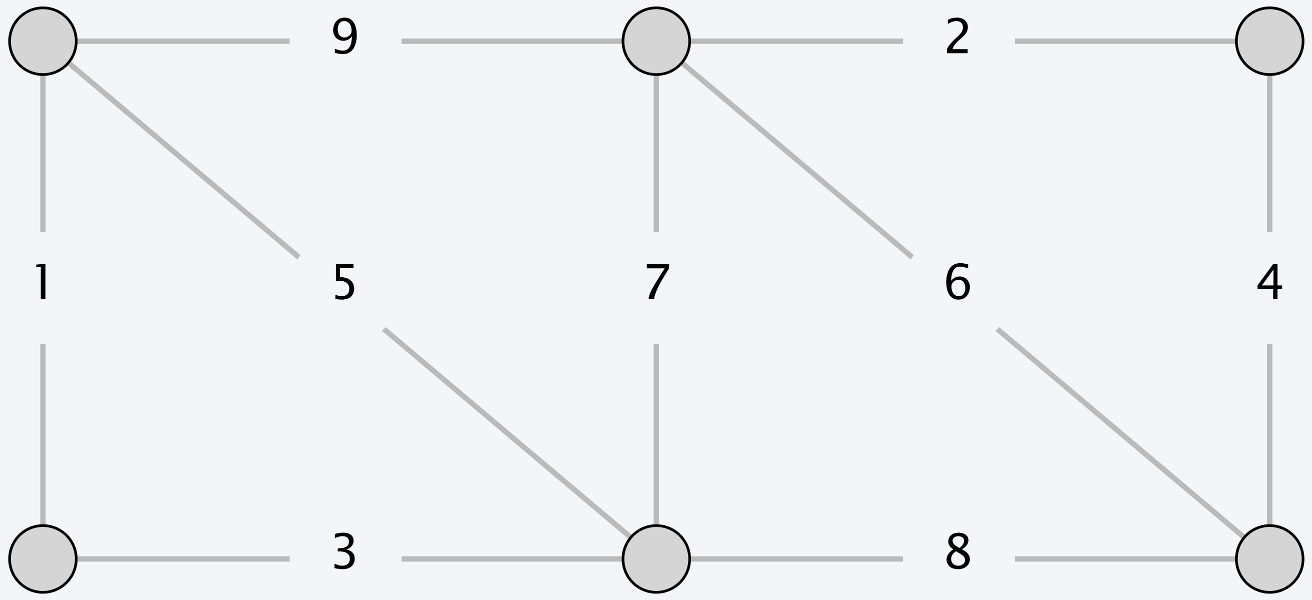
4. GREEDY ALGORITHMS II

- *red-rule blue-rule demo*
- *Prim's algorithm demo*
- *Kruskal's algorithm demo*
- *reverse-delete algorithm demo*
- *Boruvka's algorithm demo*

Reverse-delete algorithm demo

Start with all edges in T and consider them in descending order of weight:

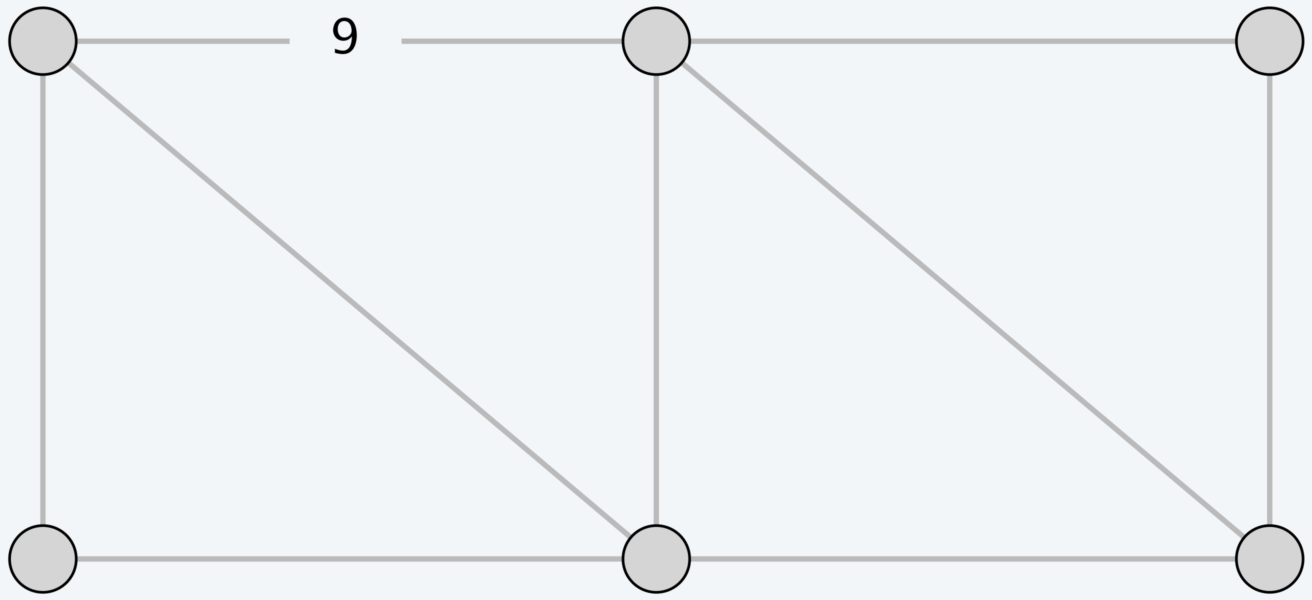
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

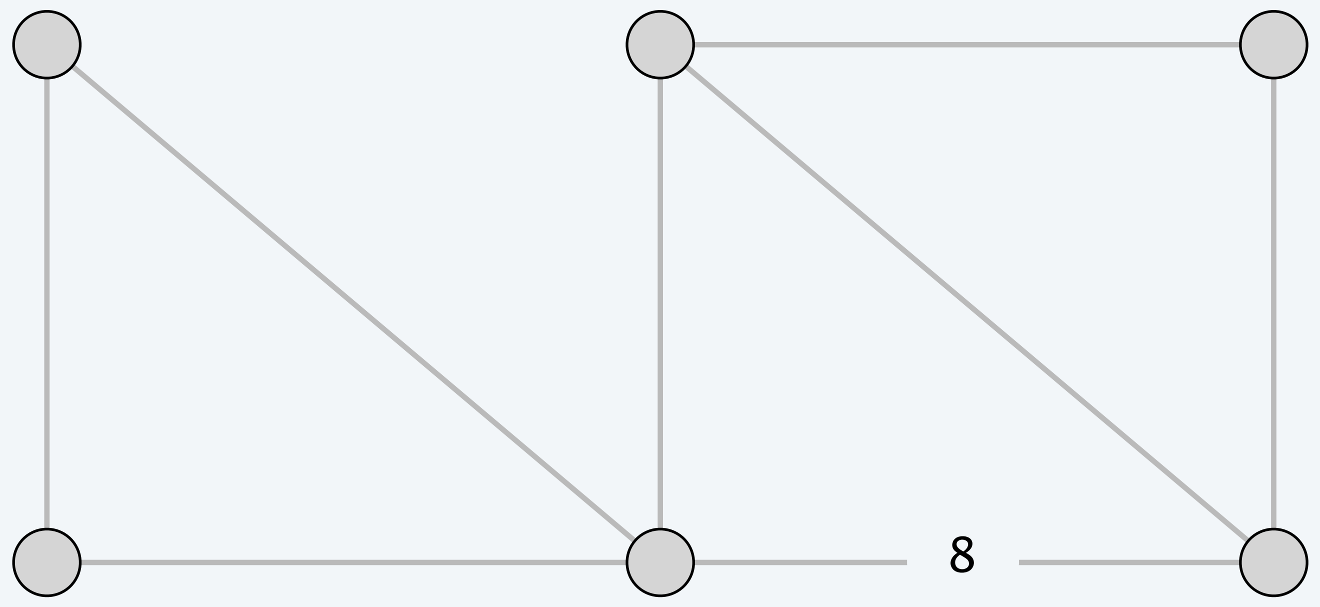
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

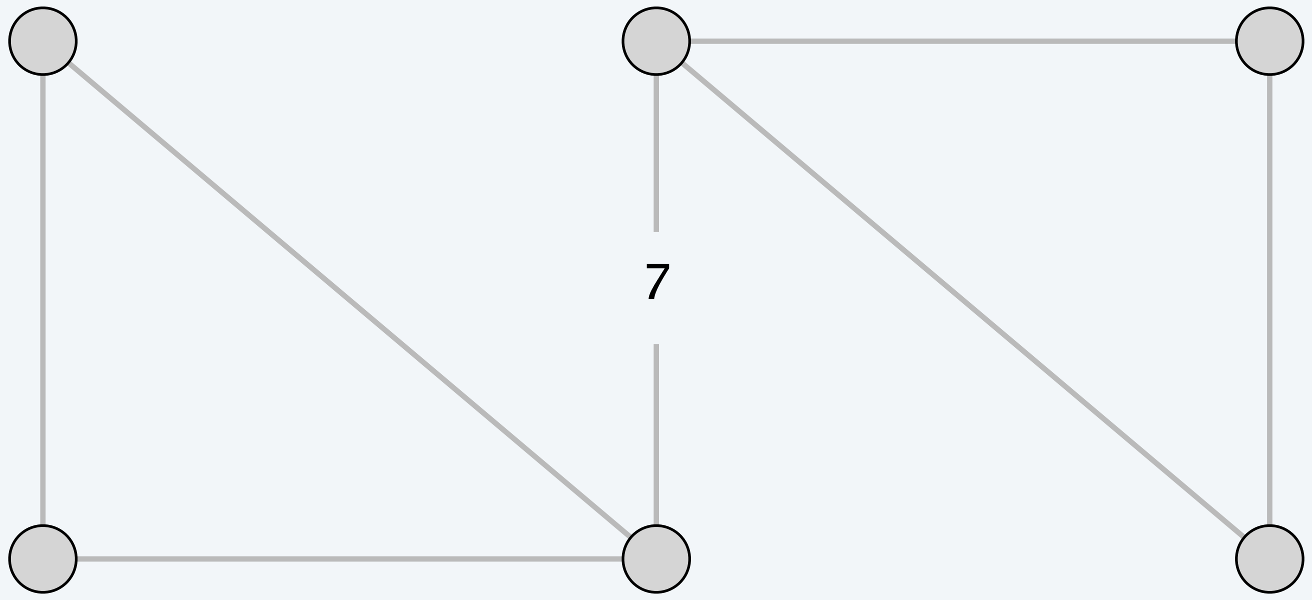
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

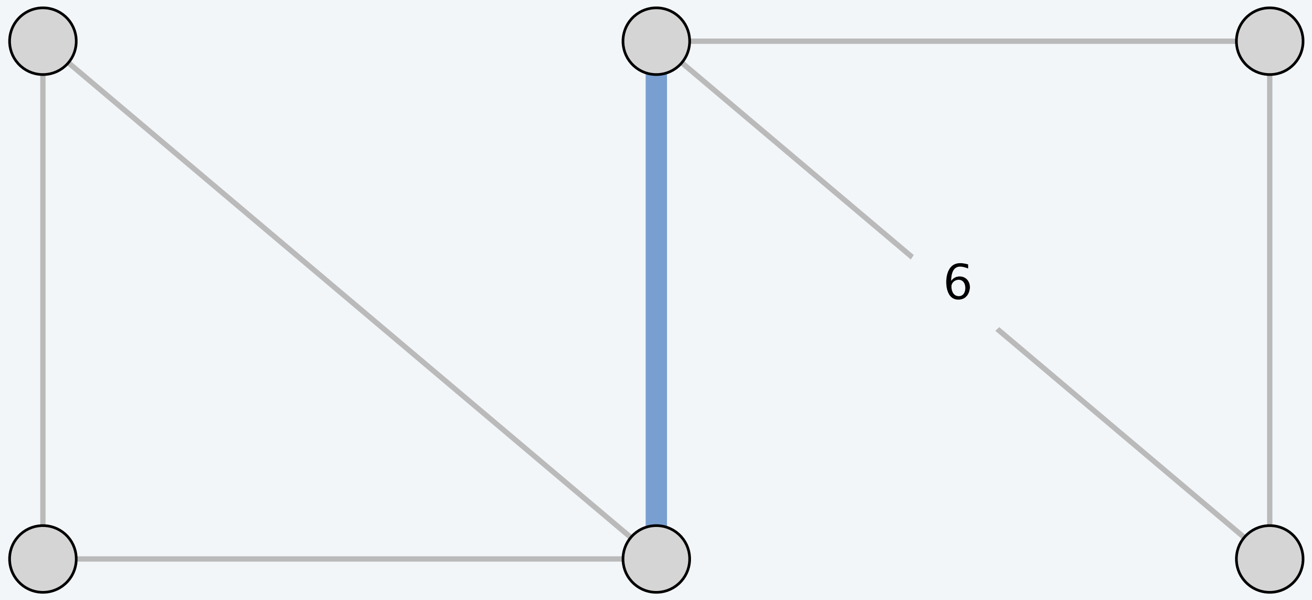
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

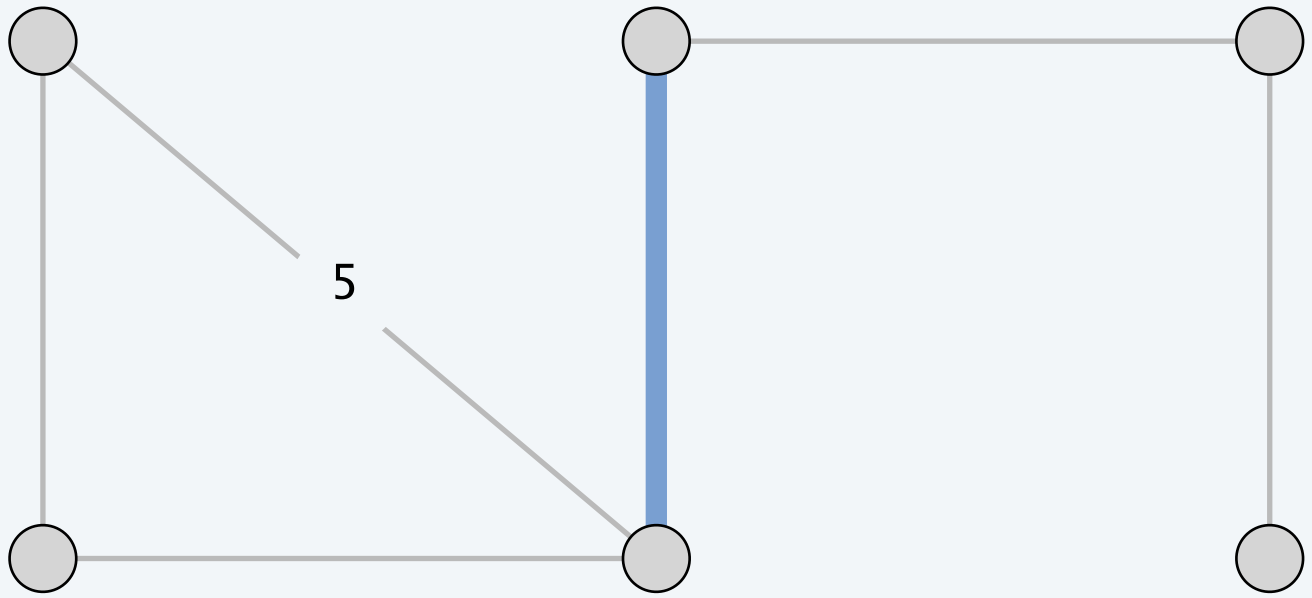
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

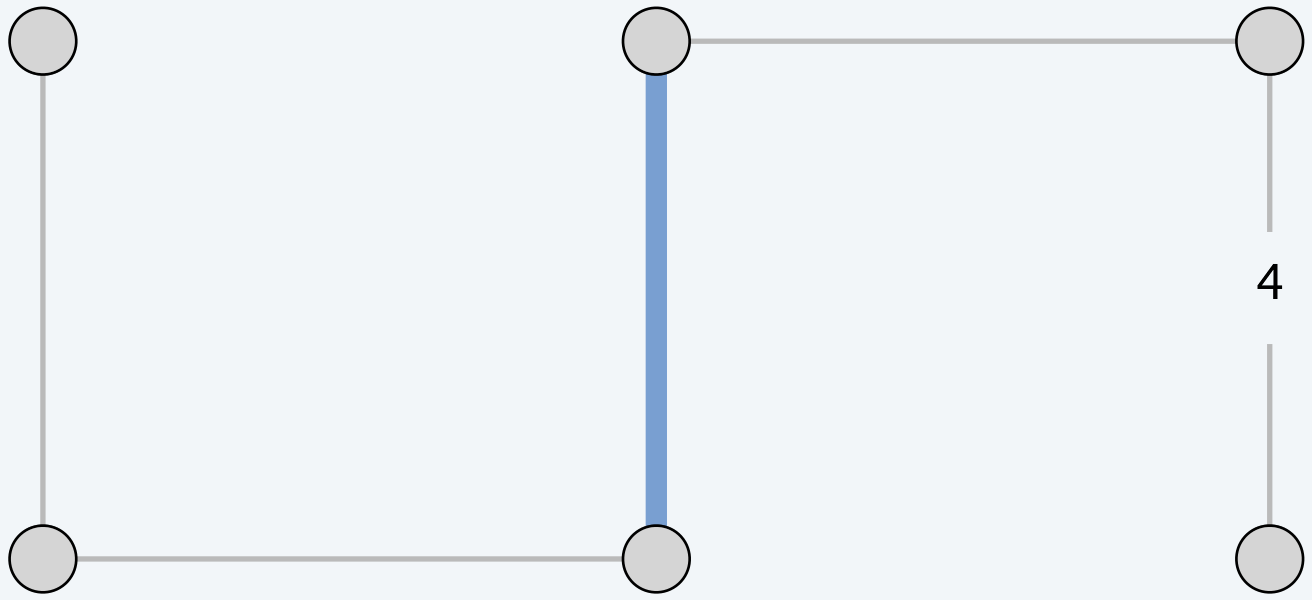
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

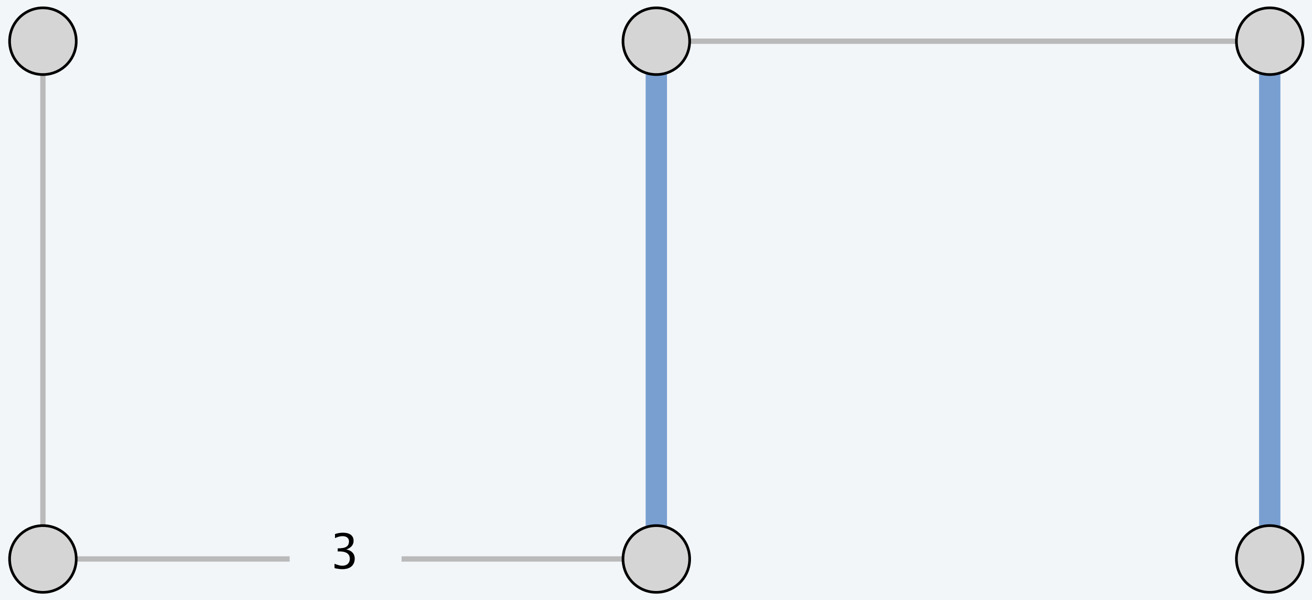
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

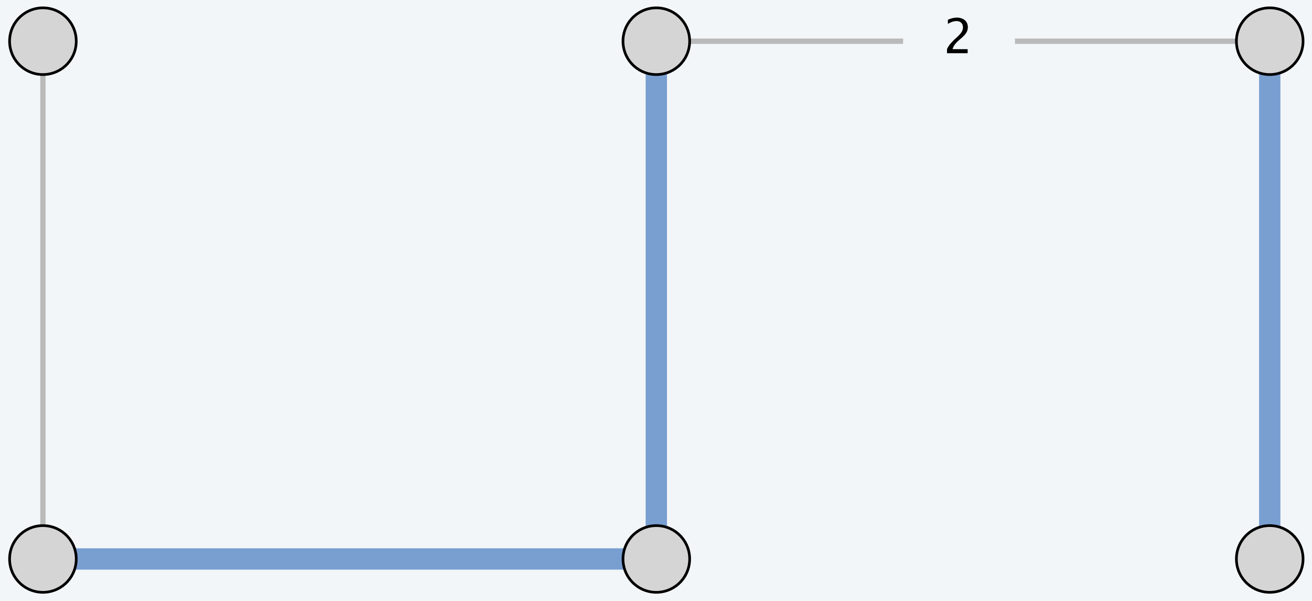
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

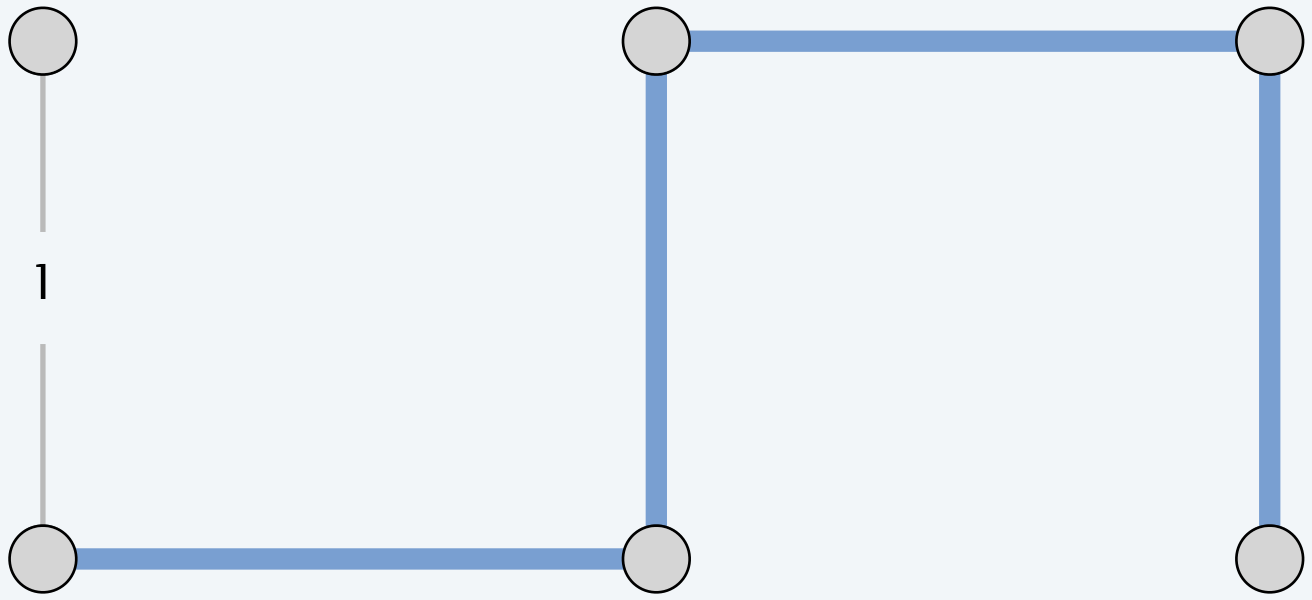
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

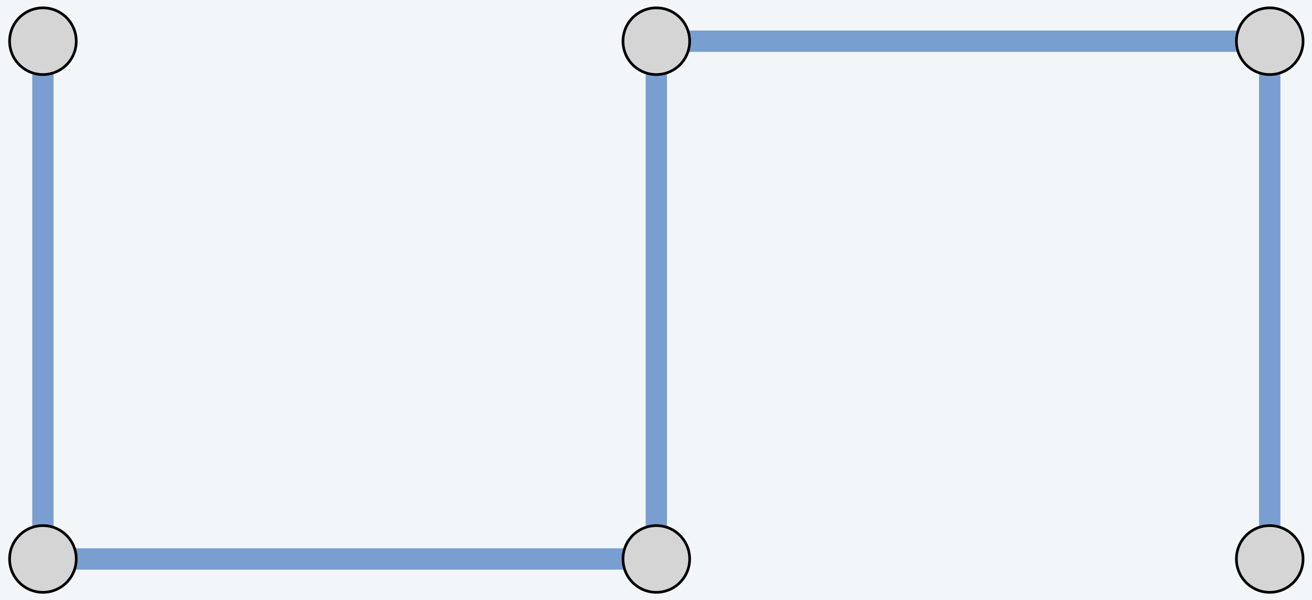
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

- Delete edge from T unless it would disconnect T.