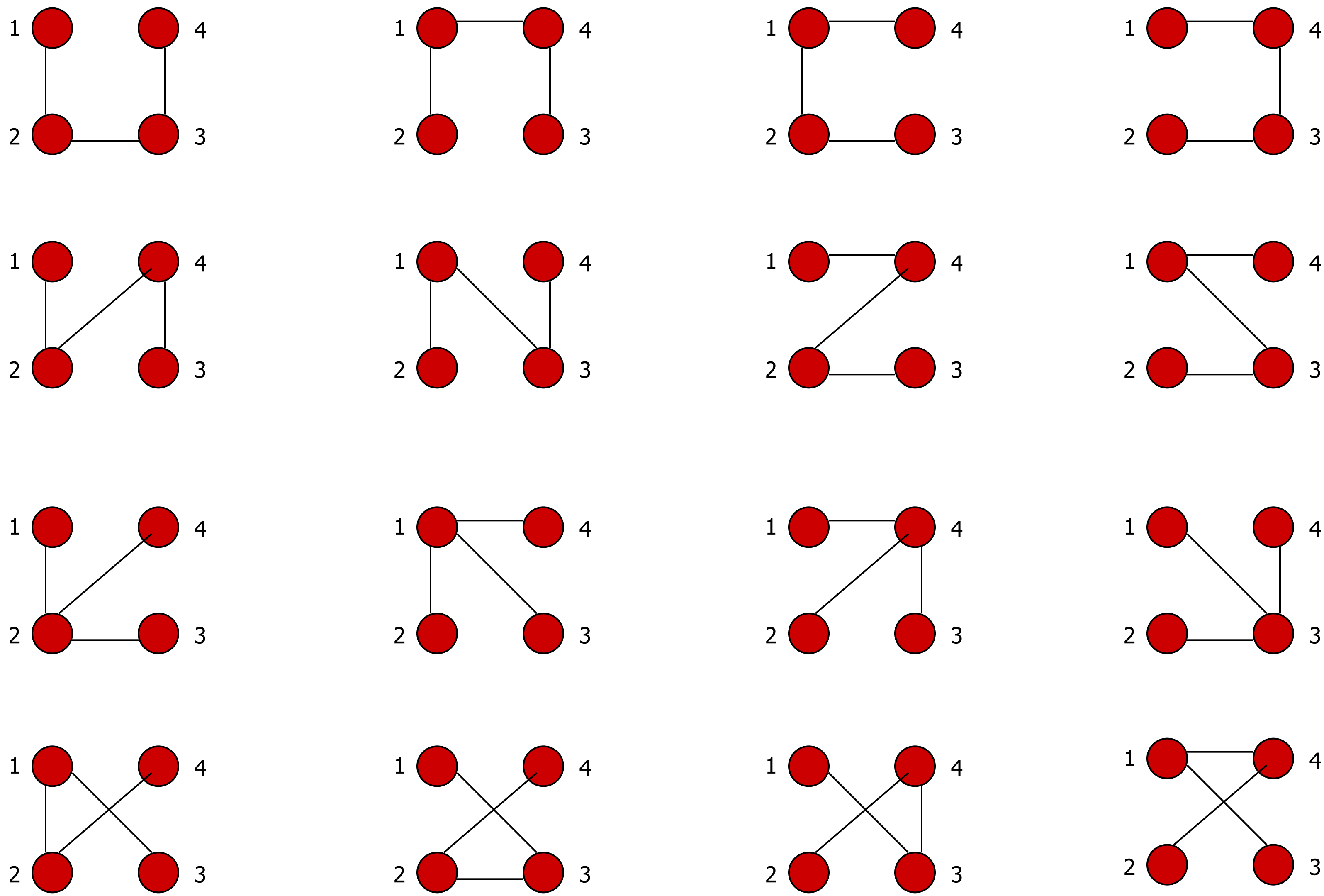
Greedy algorithm

A greedy algorithm always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

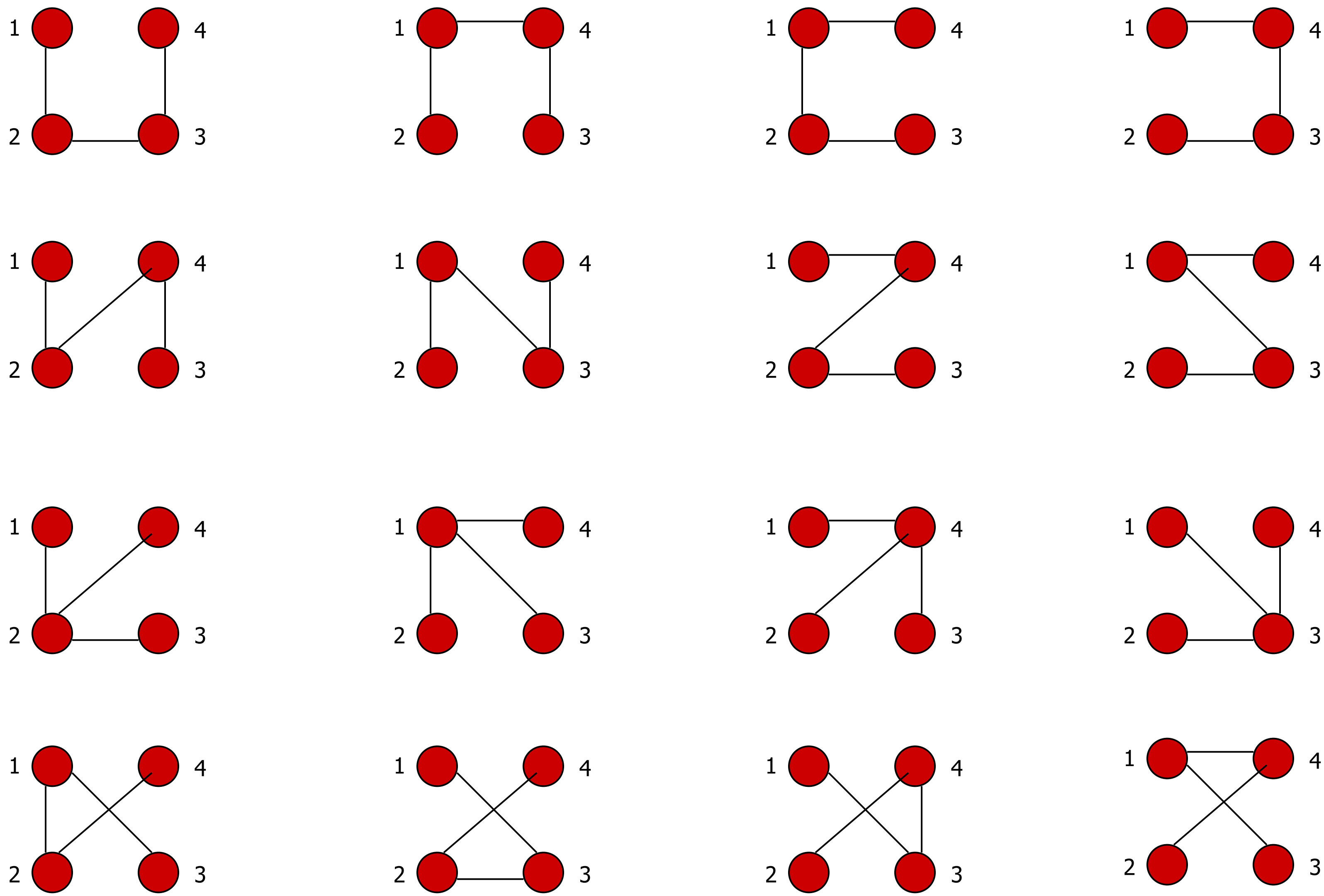
Spanning Tree

- Spanning tree of a graph is a tree that spans over all the nodes of the graph

- A spanning tree of minimum weight is called a minimum spanning tree.



For, 4 nodes: total number of possible spanning trees = 16



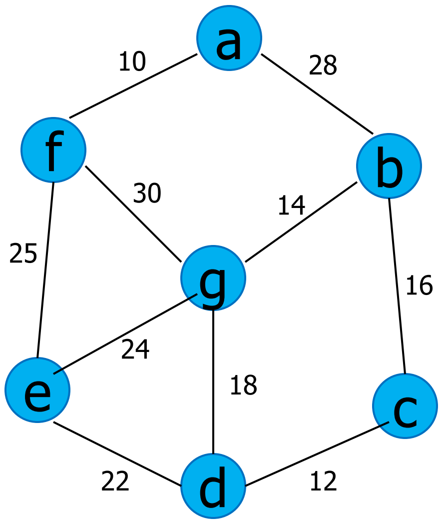
For, 4 nodes: total number of possible spanning trees = 16

In general, for n nodes: number of possible spanning trees = n^{n-2}

Minimum Spanning Tree (Kruskal's Algorithm)

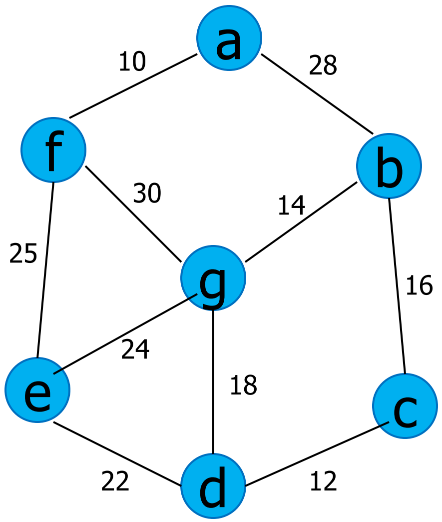
Input: Graph G = (V, E) and the weight(cost) on the edges E

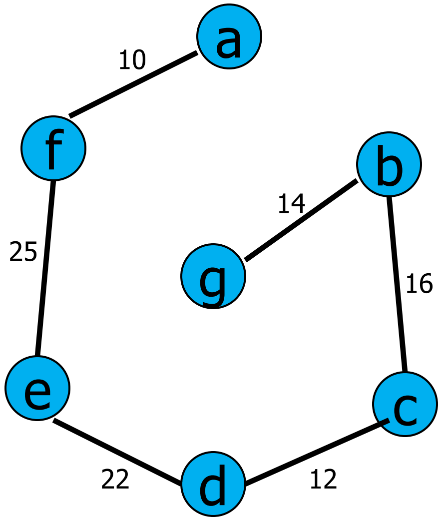
```
begin
   T = Ø
    while (T contains less than (n-1) edges and E is not empty do)
    begin
                    choose an edge (v, w) from E of lowest cost
                    delete (v, w) from E.
                    if (v, w) does not create a cycle in T then
                              add (v, w) to T
                    else
                              discard (v, w)
    end
end
```



Edges ordered by weight:

(a,f), (c,d), (b,g), (b,c),(d,g), (d,e), (e,g), (e,f), (a,b), (f,g)





T

Edges ordered by weight:

(a,f), (c,d), (b,g), (b,c),(d,g), (d,e), (e,g), (e,f), (a,b), (f,g)

Edges selected in the spanning tree T:

e1 = (a, f)
e2 = (c, d)
e3 = (b, g)
e4 = (b, c)
e5 = (d, e)
e6 = (e, f)

Minimum Spanning Tree

Theorem:
Any spanning tree $T = \{e_1, e_2, \ldots, e_{n-1}\}$ constructed by Kruskal's algorithm is an optimal tree.

Proof (by contradiction): Let T be the Spanning tree constructed by Kruskal's algorithm. By contradiction, assume that T is not an optimal tree and let S be an optimal tree such that $W(T) > W(S)$.

1. Let e be the edge with smallest weight in T that is not in S.
2. Add edge e to S. This will create a cycle C, and C contains e.

   Cycle C contains an edge e' , where e' is not in T

   If we remove e' from S and add edge e to that, we get a spanning tree S' where

      (i) $W(e) \leq W(e')$ otherwise kruskal's algorithm would have chosen e instead of e' to create T.

      (ii) S' is now one edge closer to T than S.
3. $W(S') = W(S) + W(e) - W(e')$ therefore $W(S') \leq W(S)$.
4. We can now repeat from step 1 until S' = T
5. Process terminates with S' = T and $W(T) \leq W(S)$. This contradicts our initial assumption, that there can be another MST, S with less weight than T.

Prim's algorithm

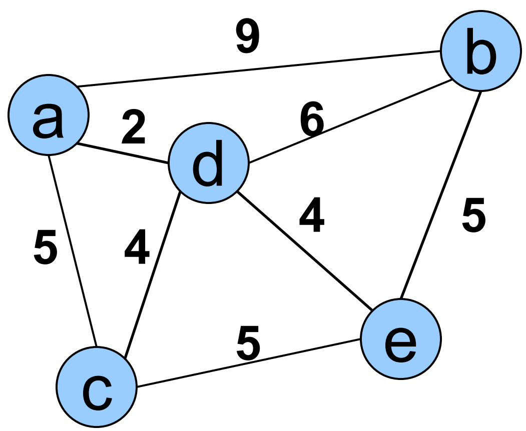
Informal description of the algorithm:

1. Start by selecting an arbitrary vertex, include it into the current MST.
2. Grow the current MST by the following approach:
   Pick the node closest to one of the nodes already in the current MST and Insert it into the current MST.
3. Repeat step 2 (until all vertices are in the MST tree).

Prim's Algorithm

Initialization

a. Pick a vertex r to be the root
b. Set D(r) = 0, parent(r) = null
c. For all vertices $v \in V$, $v \neq r$, set $D(v) = \infty$
d. Insert all vertices into priority queue Q, using distances as the keys



e	a	b	c	d
0	∞	∞	∞	∞

Vertex	Parent
e	-

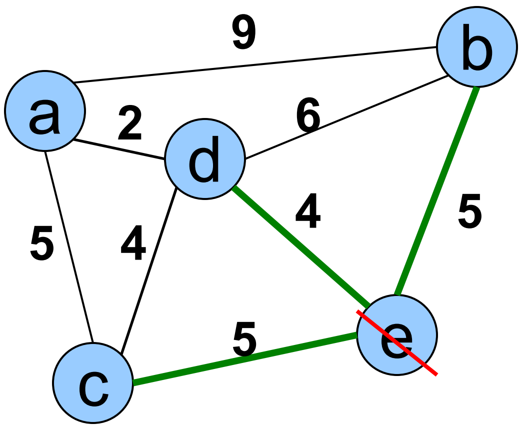
Prim's Algorithm (cont…)

While Q is not empty:

1. Select the next vertex u to add to the tree
   u = Q.deleteMin()

2. Update the weight of each vertex w adjacent to u which is not in the tree (i.e., $w \in Q$)
   If weight(u,w) < D(w),
   a. parent(w) = u
   b. D(w) = weight(u,w)
   c. Update the priority queue to reflect new distance for w

Prim's algorithm



e	d	b	c	a
0	∞	∞	∞	∞

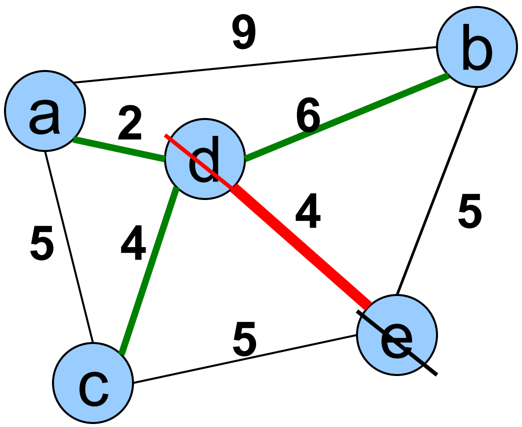
Vertex	Parent
e	-
b	-
c	-
d	-

d	b	c	a
4	**5**	**5**	∞

Vertex	Parent
e	-
b	e
c	e
d	e

The MST initially consists of the vertex e, and we update the distances and parent for its adjacent vertices

Prim's algorithm



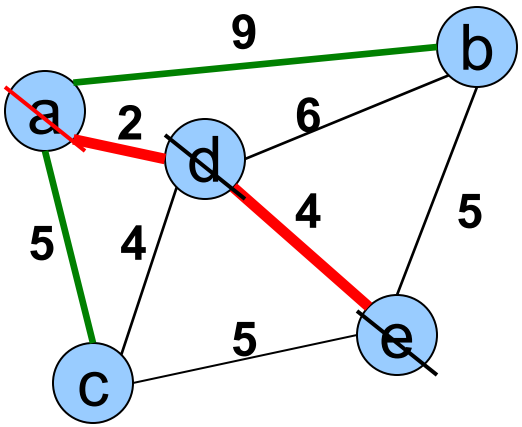
d	b	c	a
4	**5**	**5**	∞

Vertex	Parent
e	-
b	e
c	e
d	e

a	c	b
2	**4**	**5**

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

Prim's algorithm



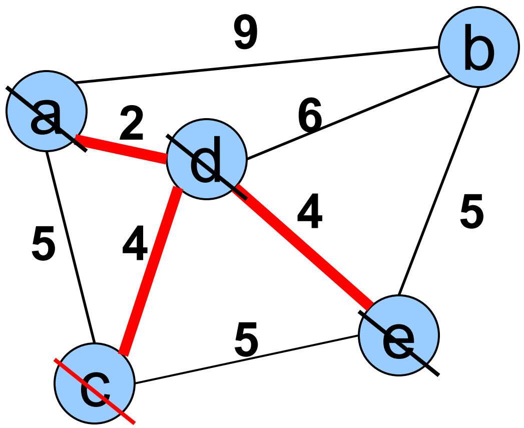
a	c	b
2	**4**	**5**

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

c	b
4	**5**

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

Prim's algorithm



c	b
4	**5**

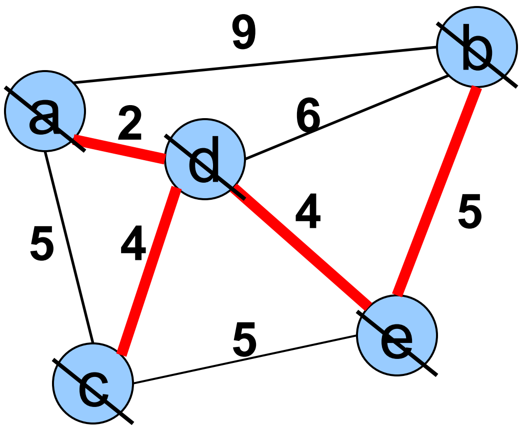
Vertex	Parent
e	-
b	e
c	d
d	e
a	d

---

b
5

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

Prim's algorithm



b
5

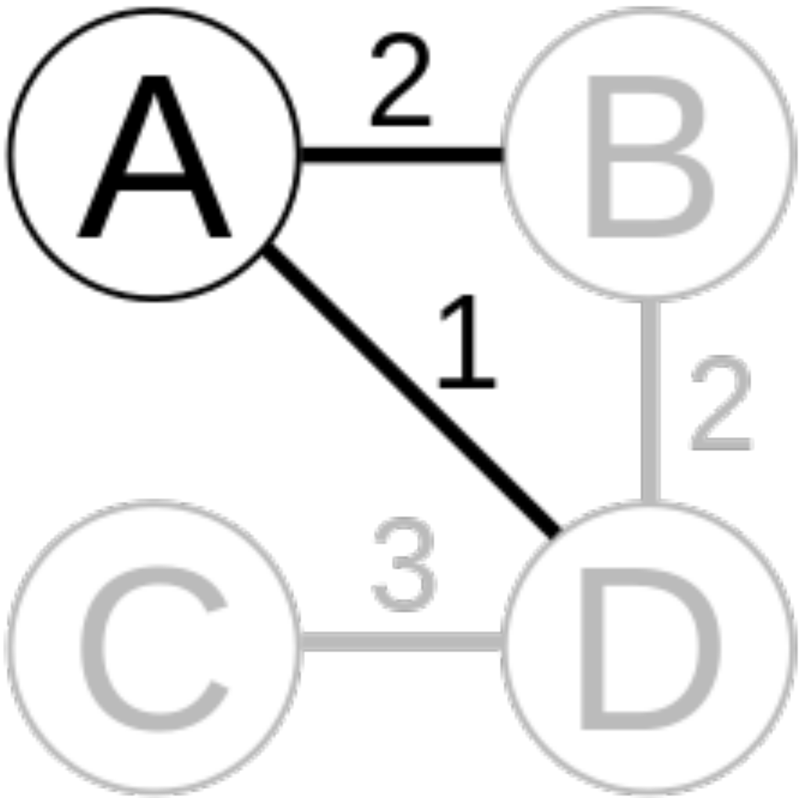
Vertex	Parent
e	-
b	e
c	d
d	e
a	d

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

The final minimum spanning tree

Prim's Algorithm: example

We run prim's algorithm starting at vertex A:
A is connected to nodes B and D. The edge with smallest cost is (A,D).



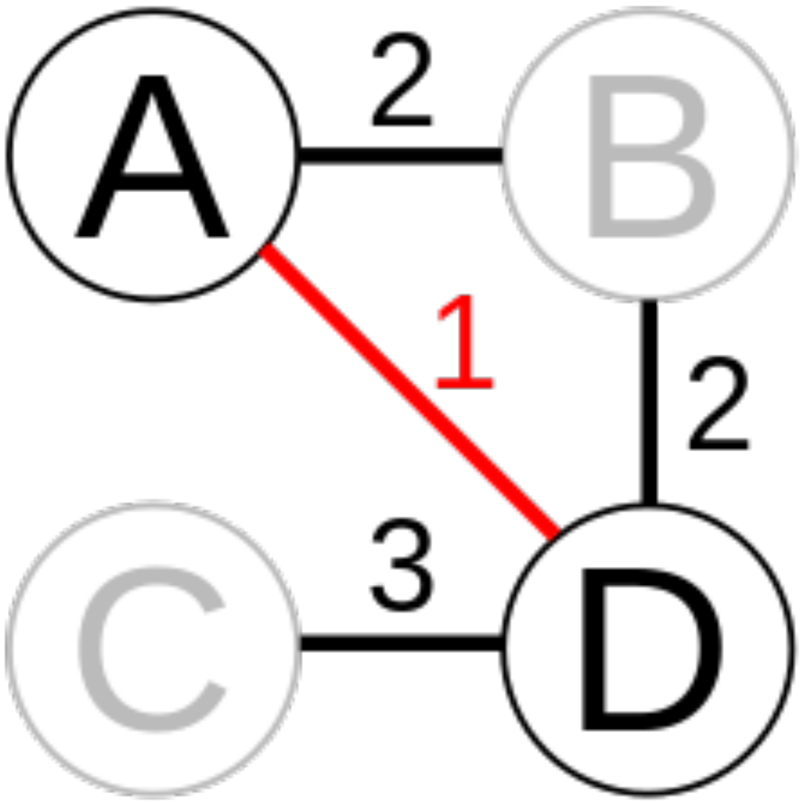
Example from Wikipedia: prim's algorithm

Prim's Algorithm: example

We run prim's algorithm starting at vertex A:

Now:

BD and BA have the same weight 2, so BD is chosen arbitrarily.

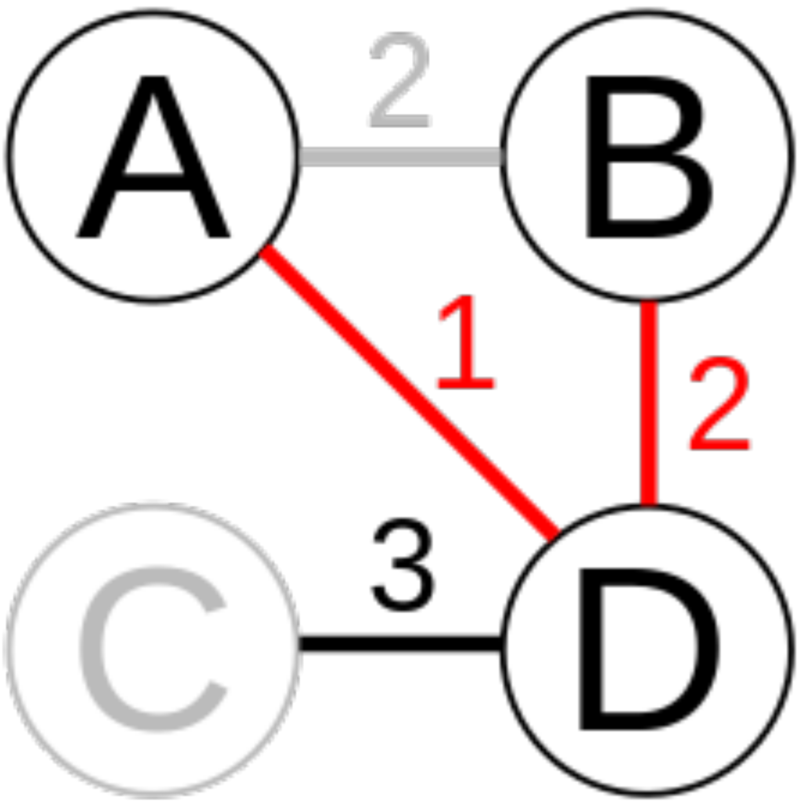


Prim's Algorithm: example

Let's run prim's algorithm starting at vertex A:

Now:

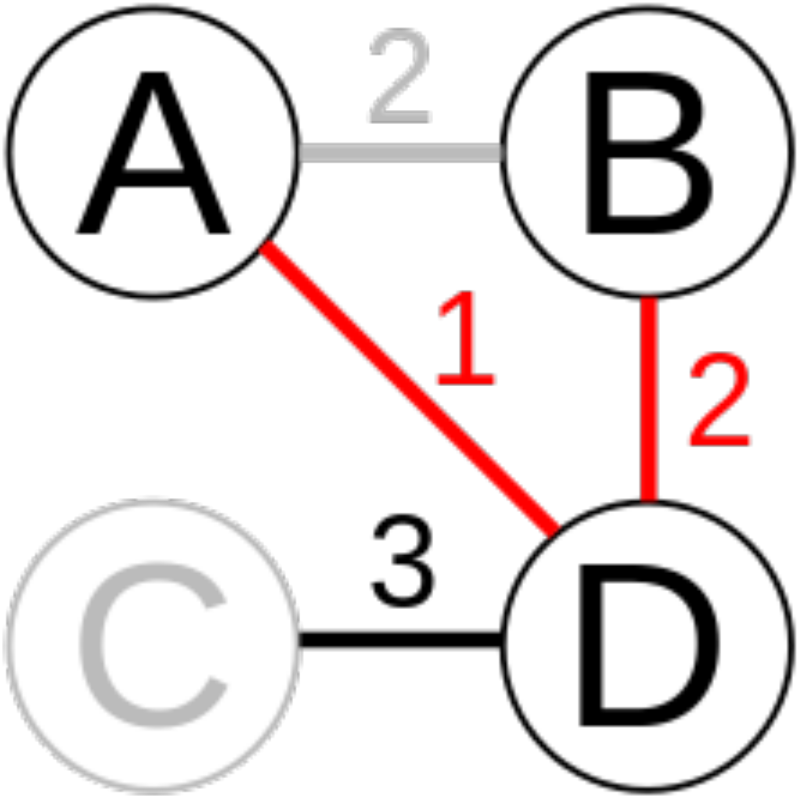
BD and BA have the same weight 2, so BD is chosen arbitrarily.



Prim's Algorithm: example

Let's run prim's algorithm starting at vertex A:

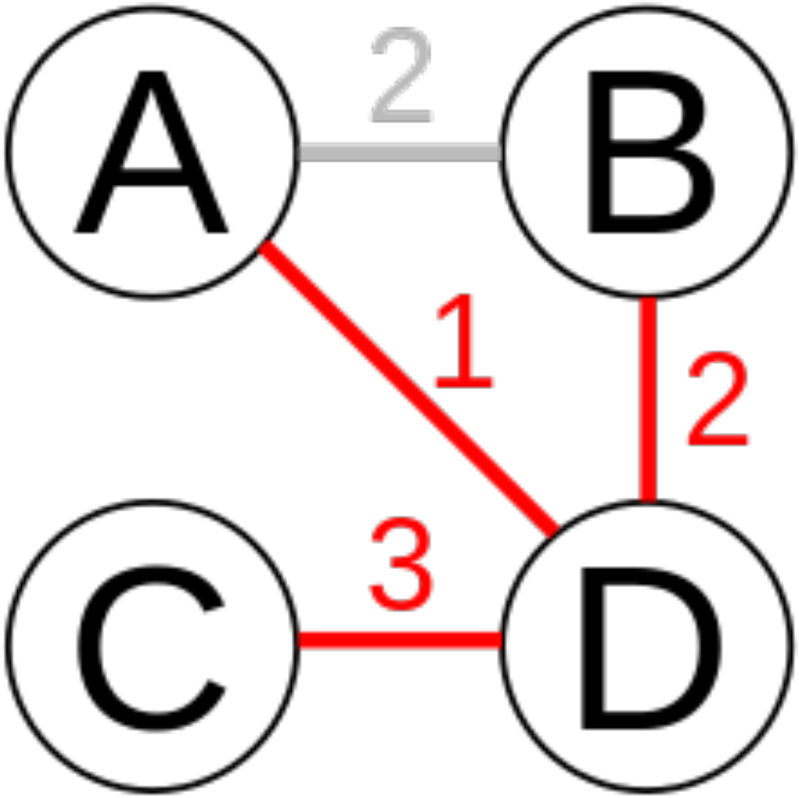
Finally: DC is the next edge to be added to the tree.



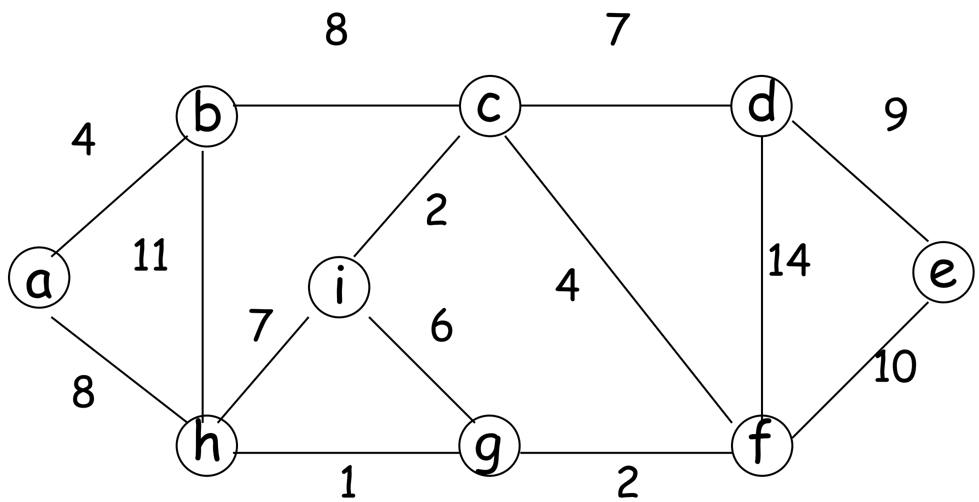
Prim's Algorithm: example

We run prim's algorithm starting at vertex A:

And we have the resulted minimum spanning tree as:



Prim's algorithm (Example)



Let a be the root of the tree. Find MST of the above graph using prim's algorithm.