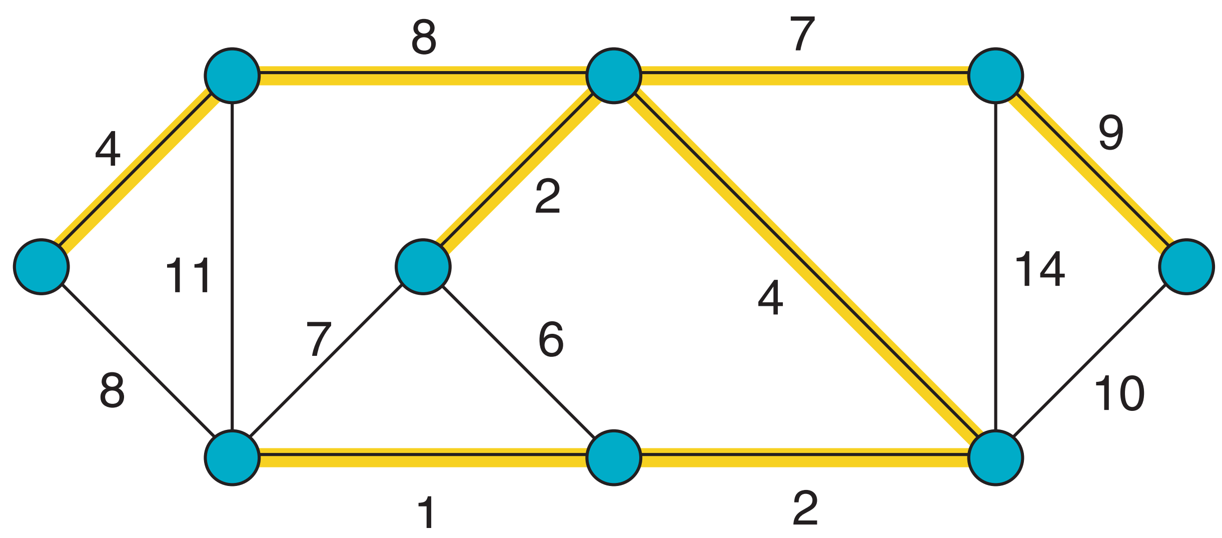
Examples:

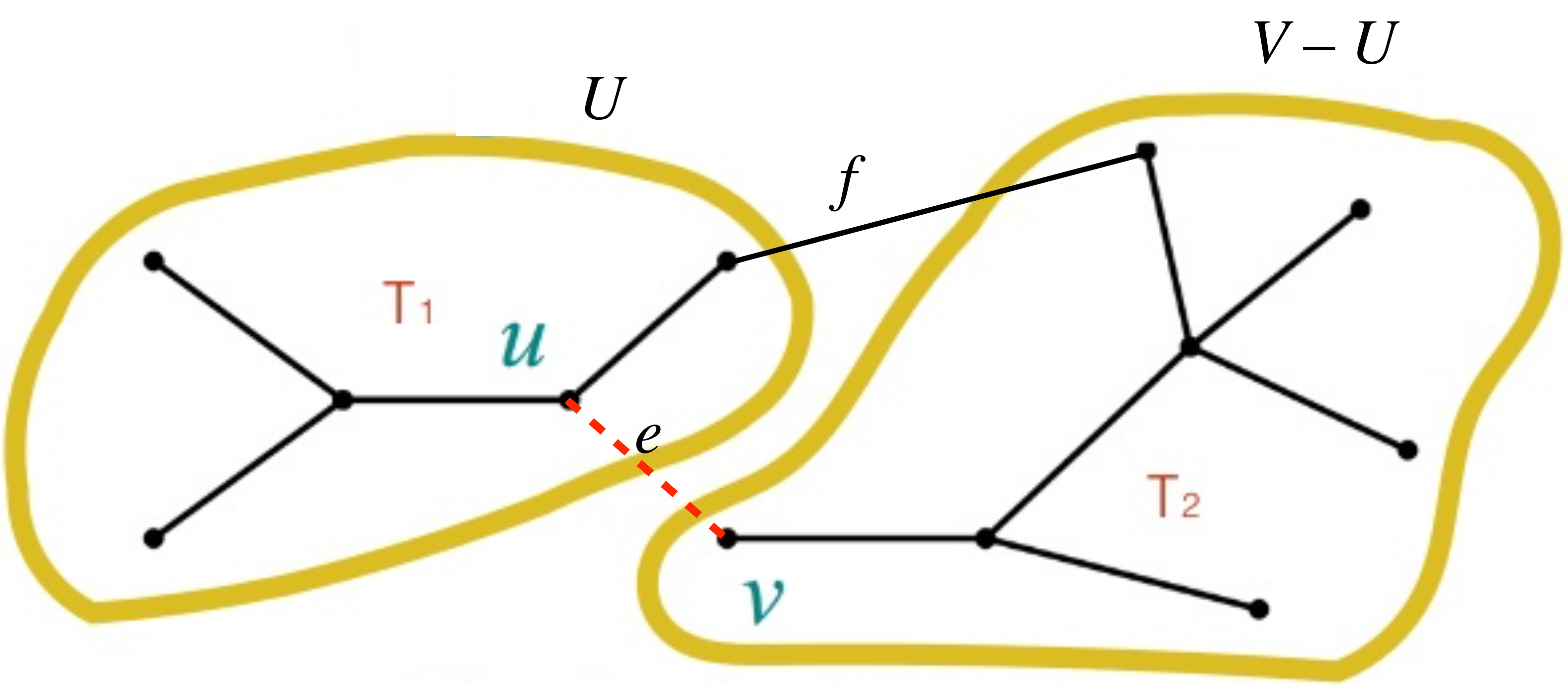
Minimum Spanning Tree (MST):

find a subgraph that connects all vertices in the graph (for example a spanning subgraph) and whose edges have minimum total weight.



Lemma: Let $U \subset V$ be any subset of the vertices of $G=(V, E)$, and let e be the edge with the smallest weight of all edges connecting U and $V - U$. Then e is part of the MST.

Proof (by contradiction): Suppose T is an MST not containing e. Let $e=(u,v)$, with $u \in U$ and $v \in V - U$. Then, because T is a spanning tree, it contains a unique path from u to v that together with e forms a cycle in G. This path must include another edge f connecting U and $V - U$. Now $T + e - f$ is another spanning tree with less total weight than T. So T was not an MST.



This lemma lets an MST grow edge by edge!

Prim's algorithm (this is not the only one, as we have seen)

Prim(G)
Input: weighted graph $G(V, E)$

Output: minimum spanning tree $T \subseteq G$

begin

Let T be a single vertex v from G

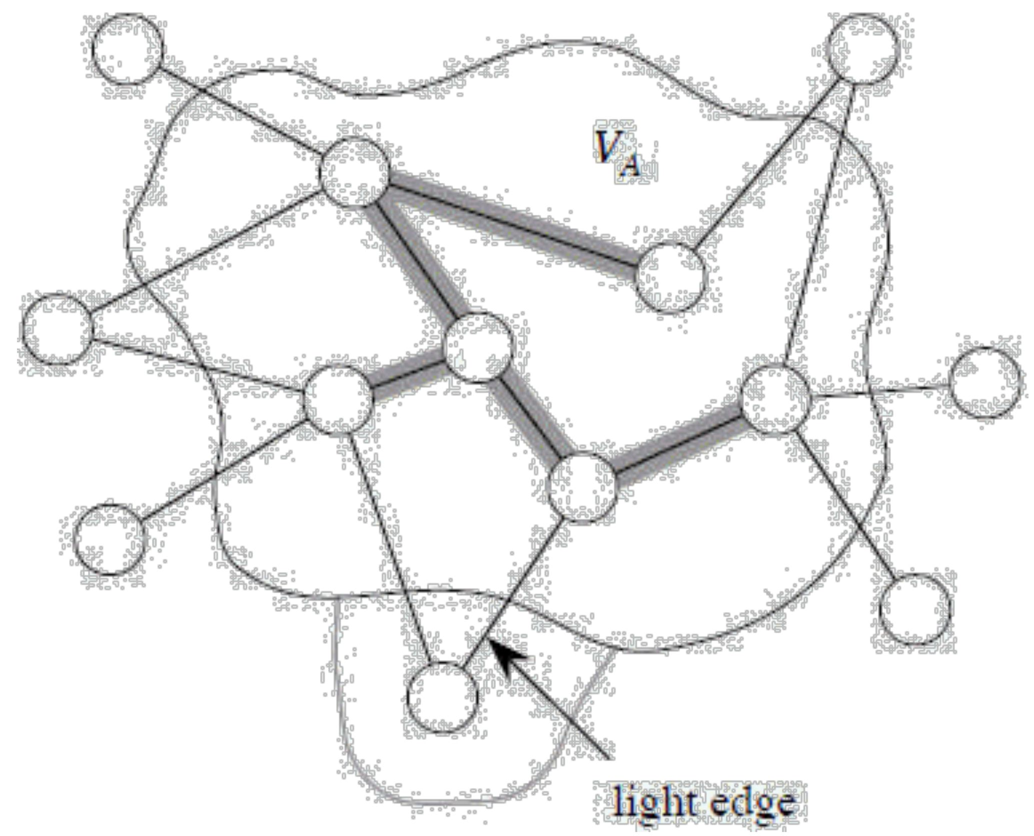
while T has less than n vertices

find the minimum edge connecting T to $G - T$

add it to T

end

end

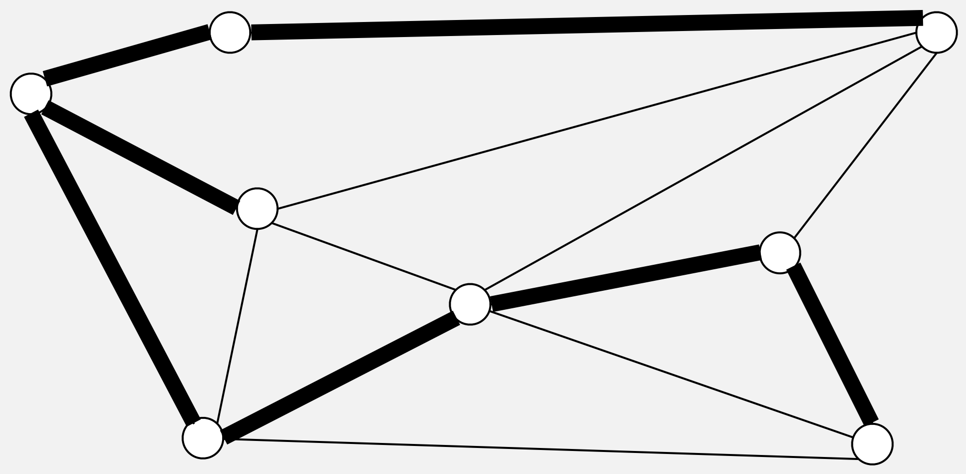


Computational cost: $T(n) = O(n^2 \log n)$. Solvable in Polynomial time, class P

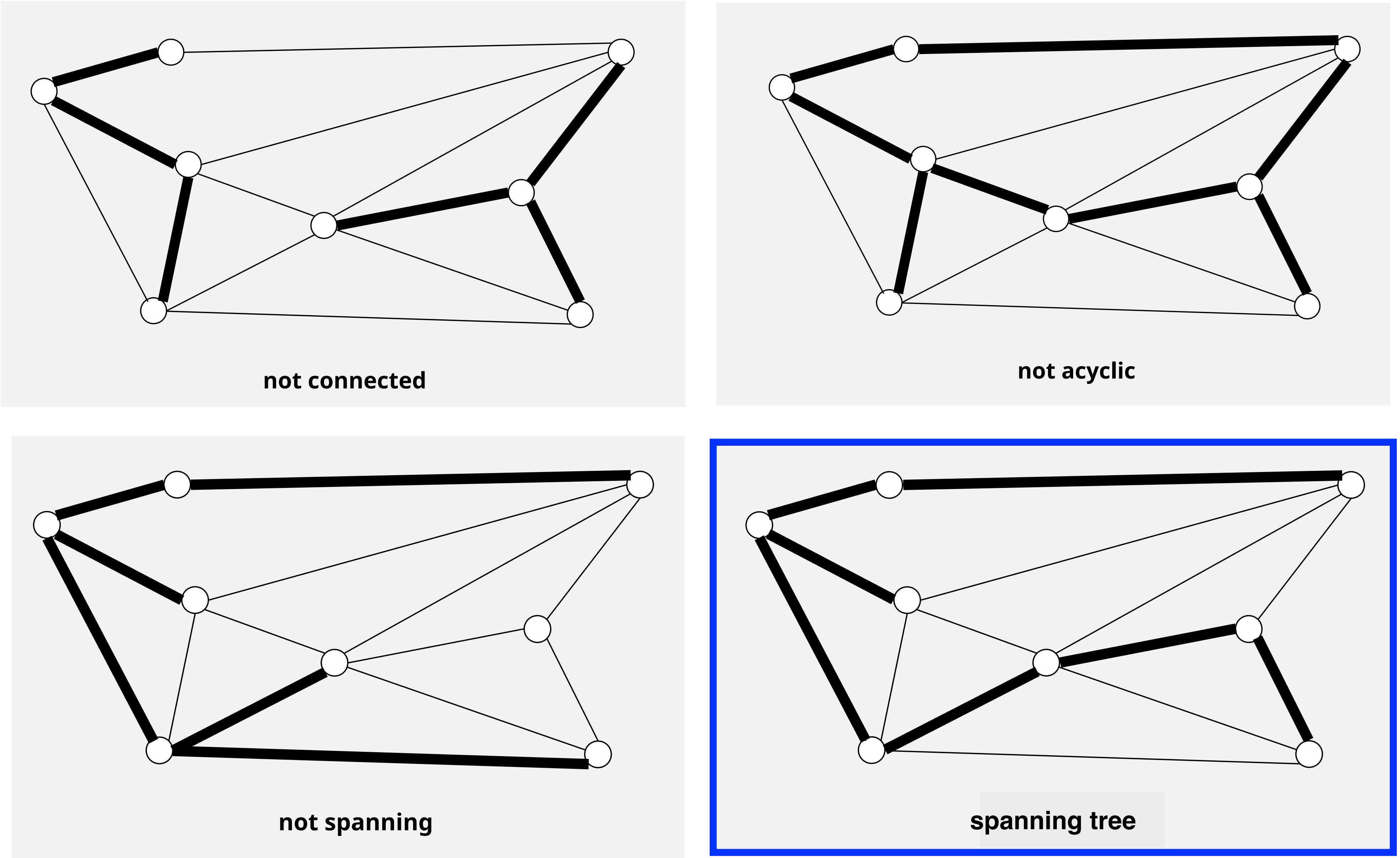
Minimum spanning tree

Def. A spanning tree of G is a subgraph T that is:

- Connected.
- Acyclic.
- Includes all of the vertices.

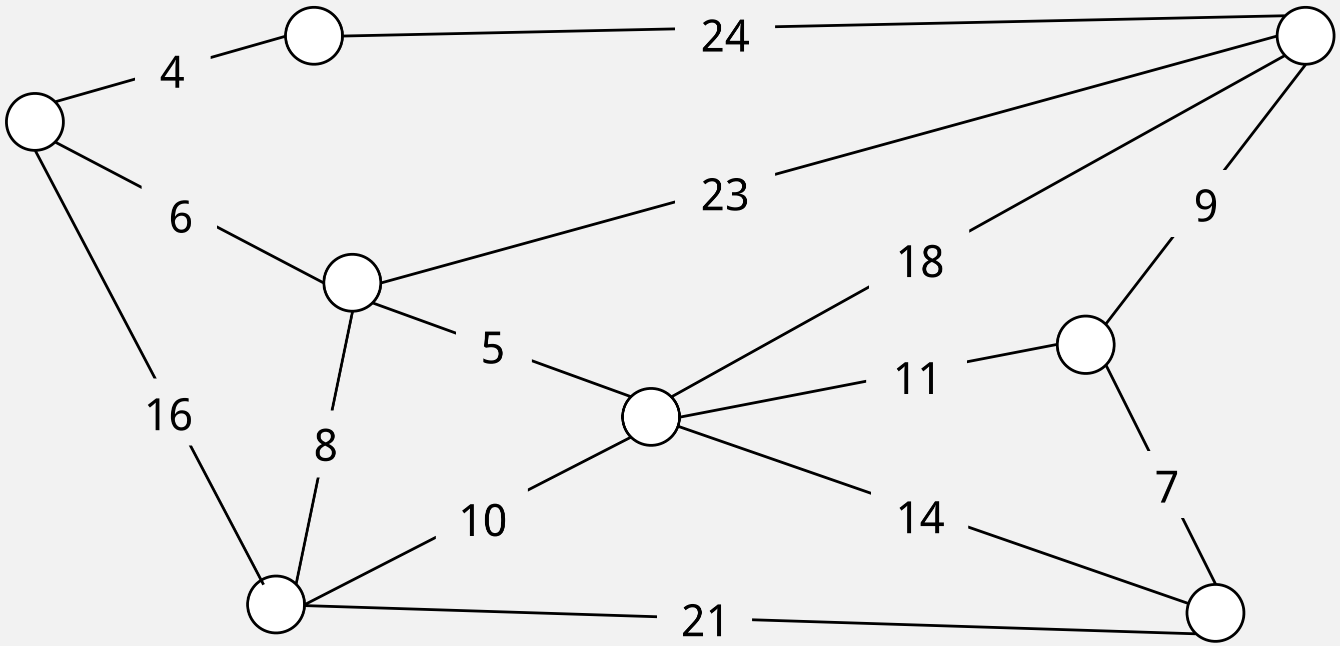


graph G


not connected
not acyclic
not spanning
spanning tree

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

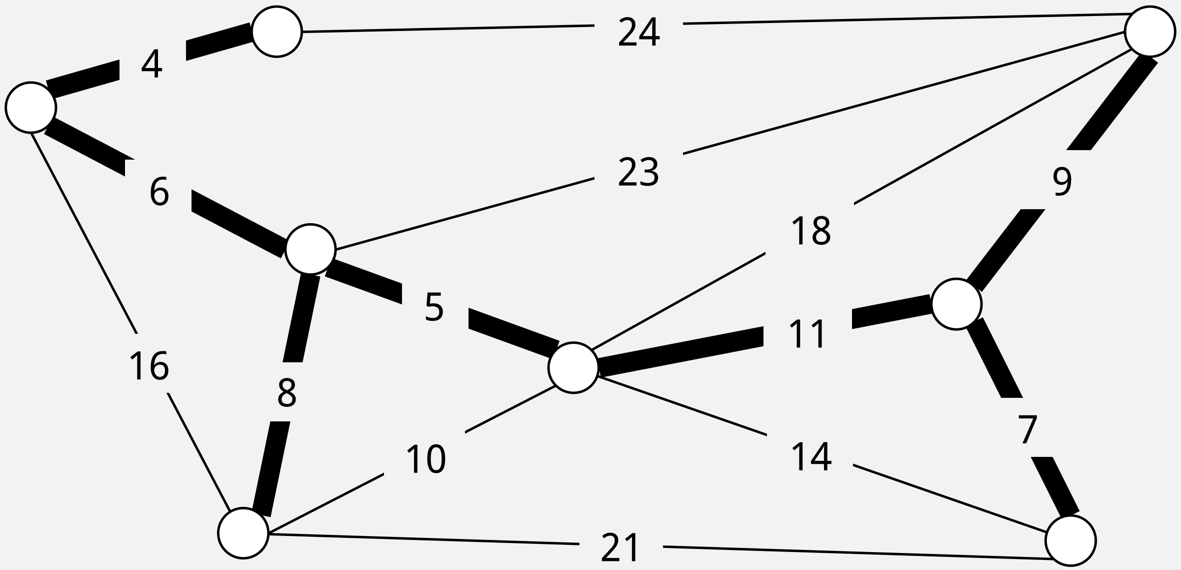


edge-weighted graph G

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

Output. A minimum weight spanning tree.

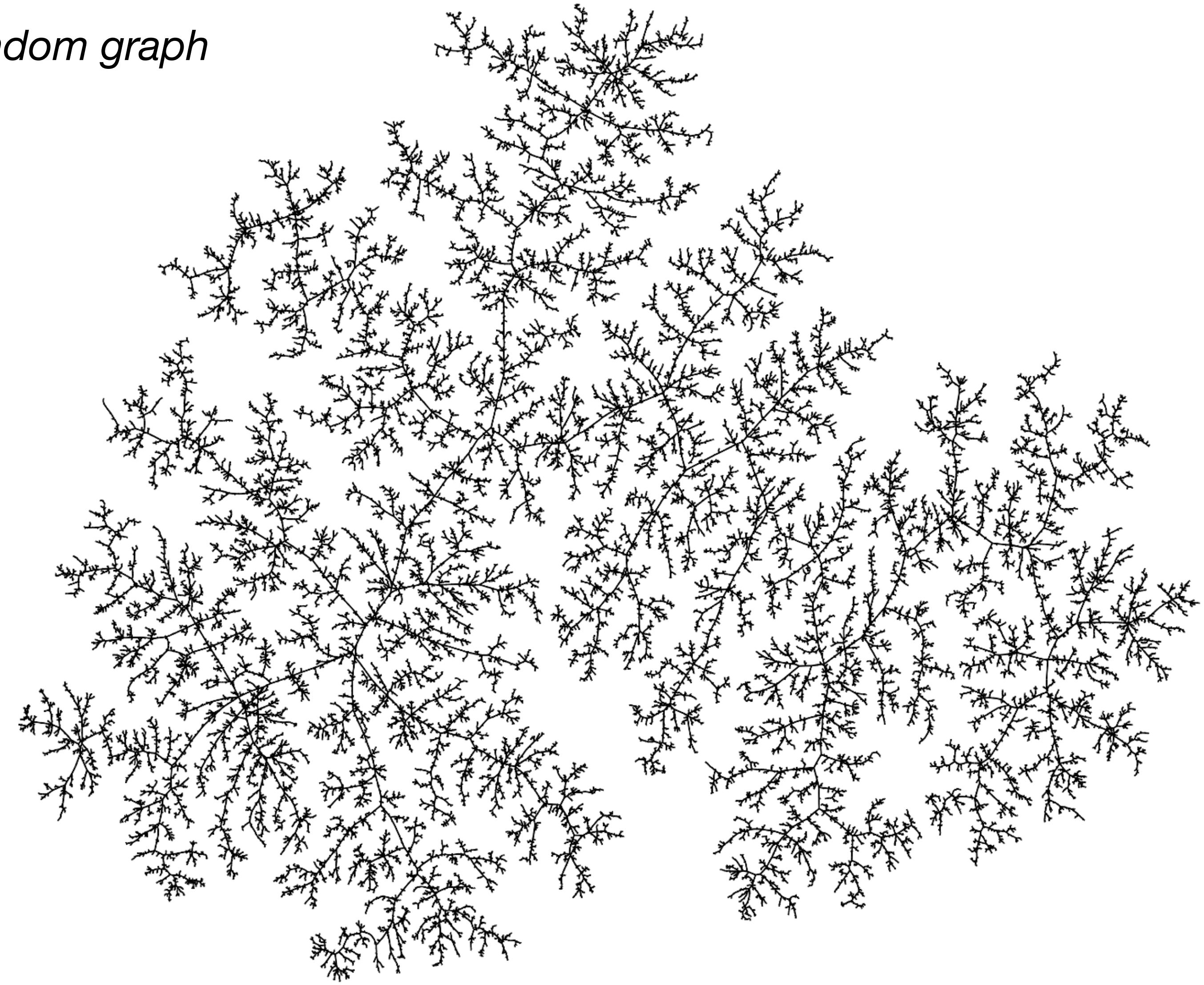


minimum spanning tree T

(weight = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7)

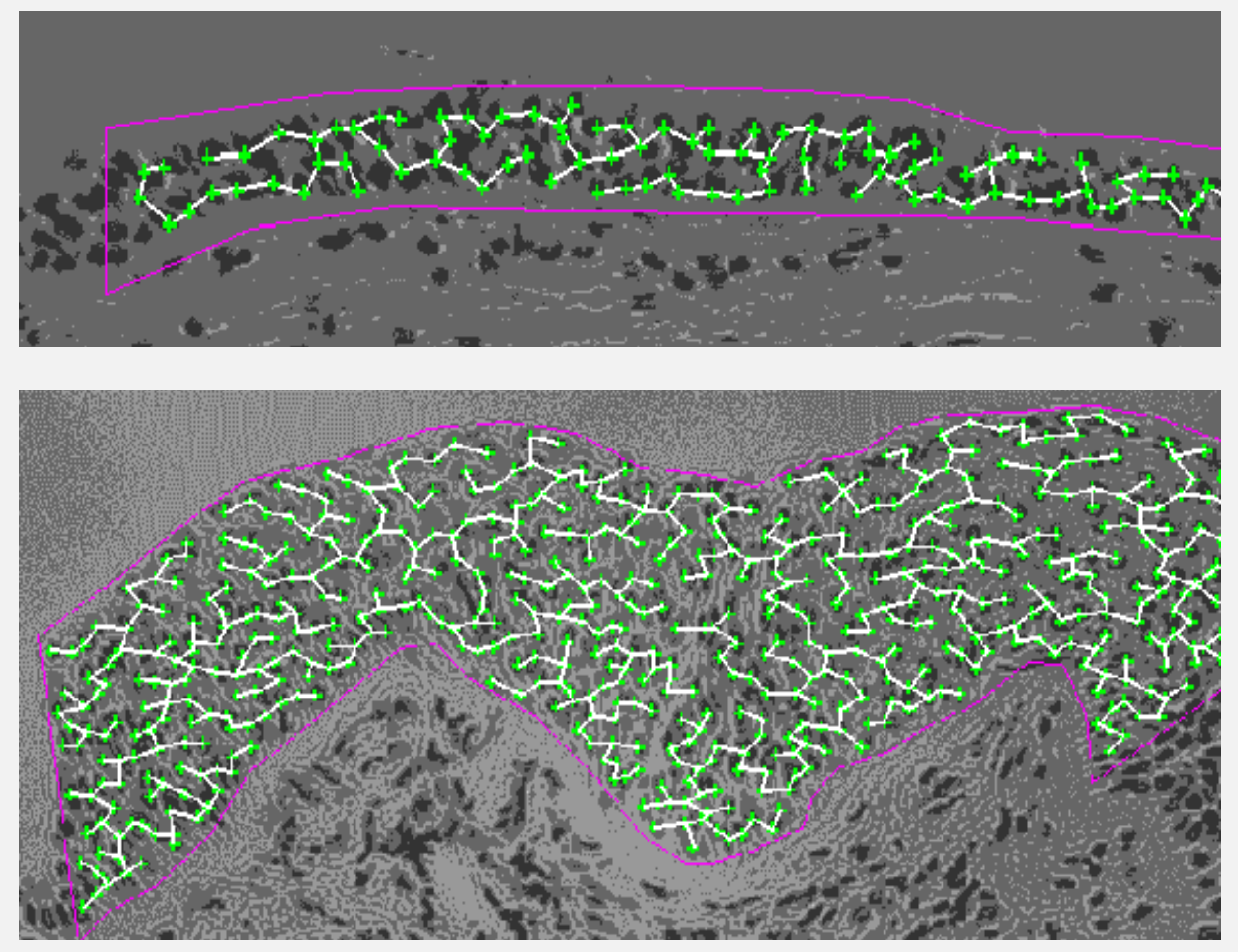
Natural examples

MST of a random graph



Medical image processing

MST describes arrangement of nuclei in the epithelium for cancer research



MST is fundamental problem with diverse **applications**.

1. Dithering.
2. Cluster analysis.
3. Max bottleneck paths.
4. Real-time face verification.
5. LDPC codes for error correction.
6. Image registration with Renyi entropy.
7. Find road networks in satellite and aerial imagery.
8. Reducing data storage in sequencing amino acids in a protein.
9. Model locality of particle interactions in turbulent fluid flows.
10. Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
11. Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
12. Network design (communication, electrical, hydraulic, computer, road).
13. …

From a simple theorem to efficient algorithms

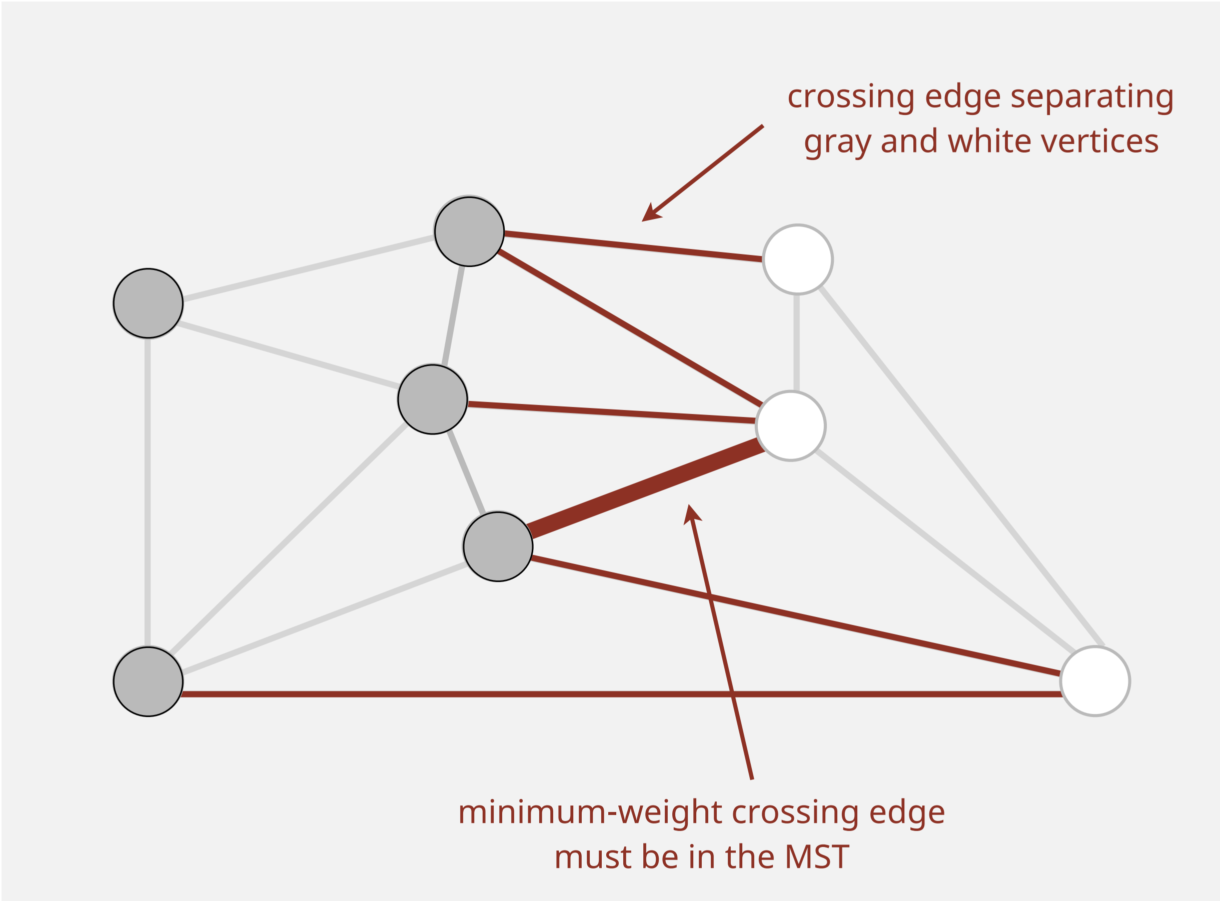
Cut property

Def. A cut in a graph is a partition of its vertices into two (nonempty) sets.

Def. A crossing edge connects a vertex in one set with a vertex in the other.

Theorem (the cut property):

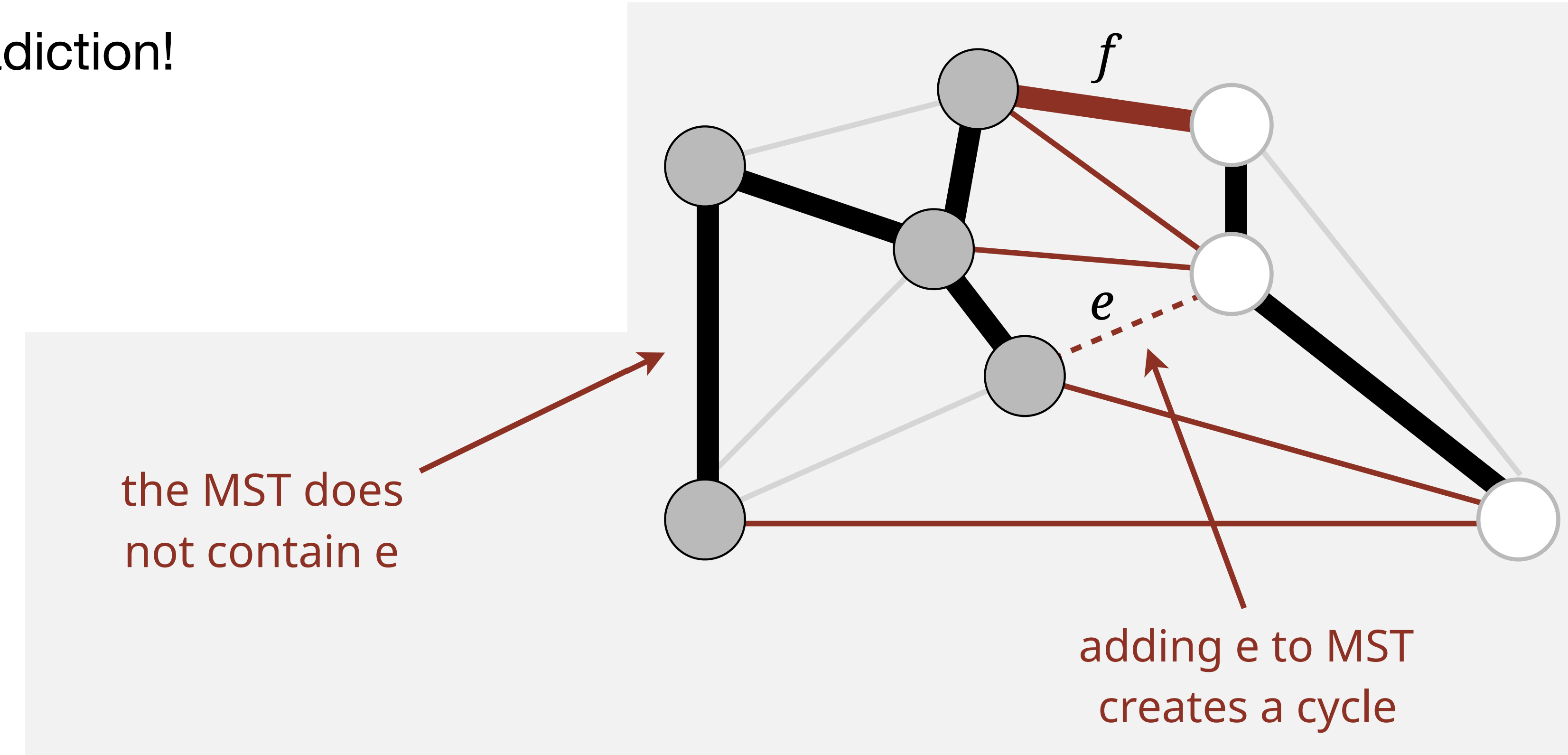
Given any cut, the crossing edge of minimal weight is in the MST.



Proof. (by contradiction):

Suppose min-weight crossing edge *e* is not in the MST.

- Adding *e* to the MST creates a cycle.
- Some other edge *f* in the cycle must be a crossing edge.
- Removing *f* and adding *e* is also a spanning tree.
- Since weight of *e* is less than the weight of *f*, that spanning tree has lower weight.
- Contradiction!

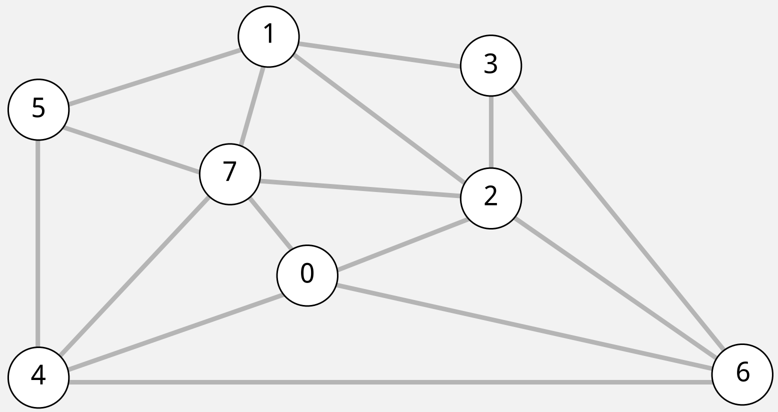


Greedy MST algorithm demo

I. Start with all edges colored gray.

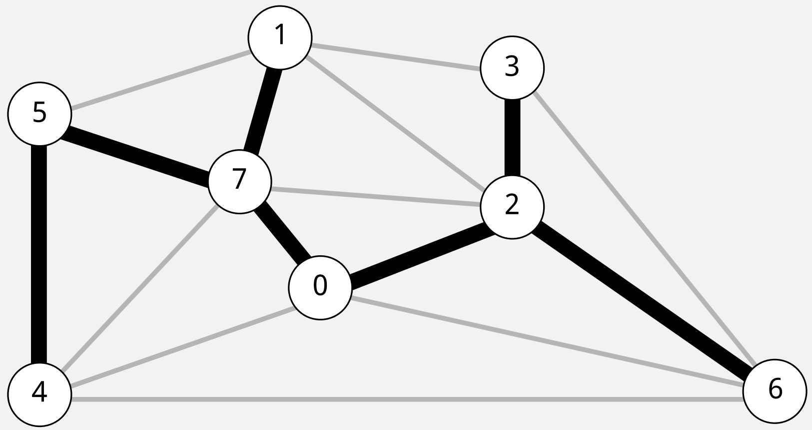
II. Find cut with no black crossing edges; color its min-weight edge black.

III. Repeat until $V - 1$ edges are colored black.



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93



MST edges

0-2 5-7 6-2 0-7 2-3 1-7 4-5

Greedy MST algorithm: efficient design

Proposition. The greedy algorithm computes the MST.

Efficient implementations: key steps

I)How do we choose the cut?

II)How do we find the min-weight edge?

Two important examples:

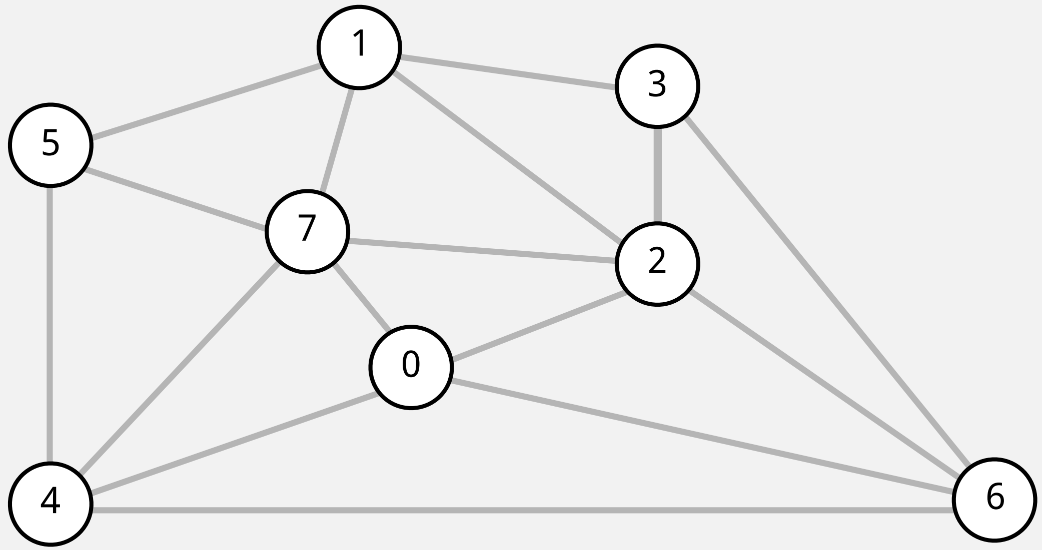
1. Kruskal's algorithm.

2. Prim's algorithm.

Kruskal's algorithm

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle.



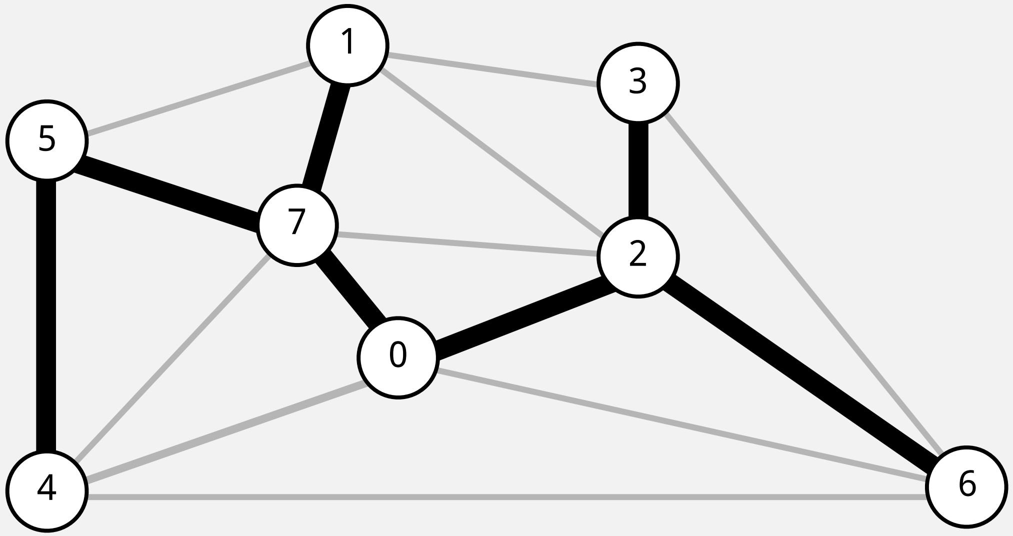
an edge-weighted graph

graph edges sorted by weight

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle.



a minimum spanning tree

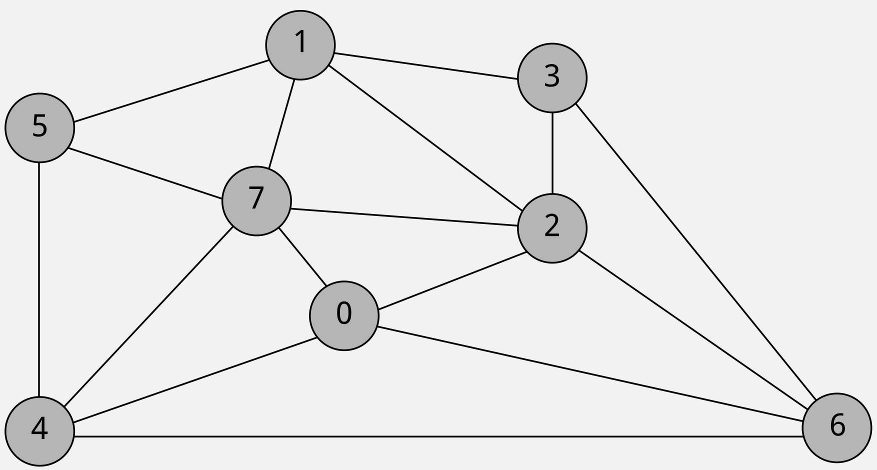
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm

I) Start with vertex 0 and greedily grow tree T.

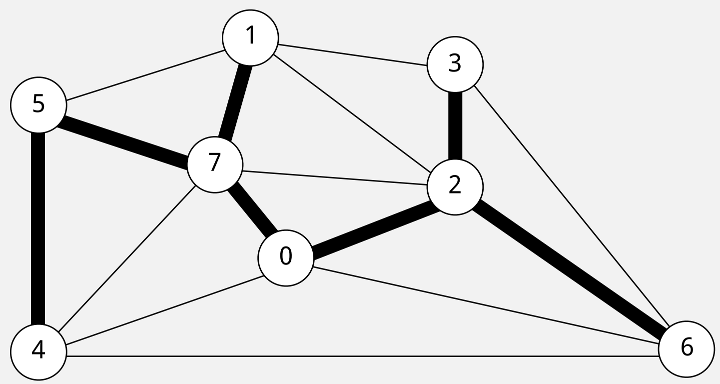
II) Add to T the min weight edge with exactly one endpoint in T.

III) Repeat until $V - 1$ edges.



an edge-weighted graph

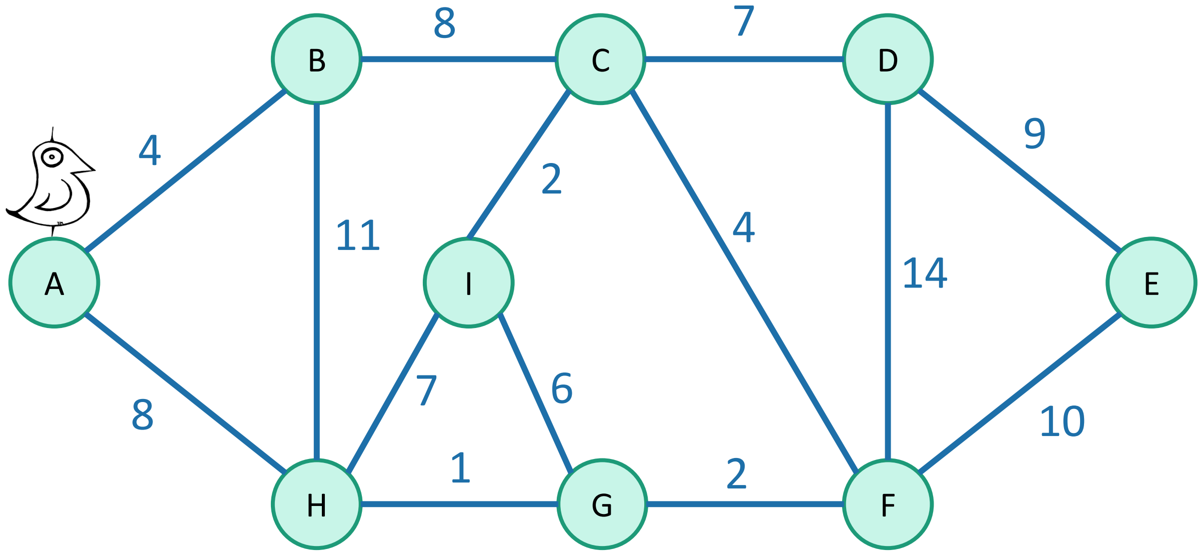
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

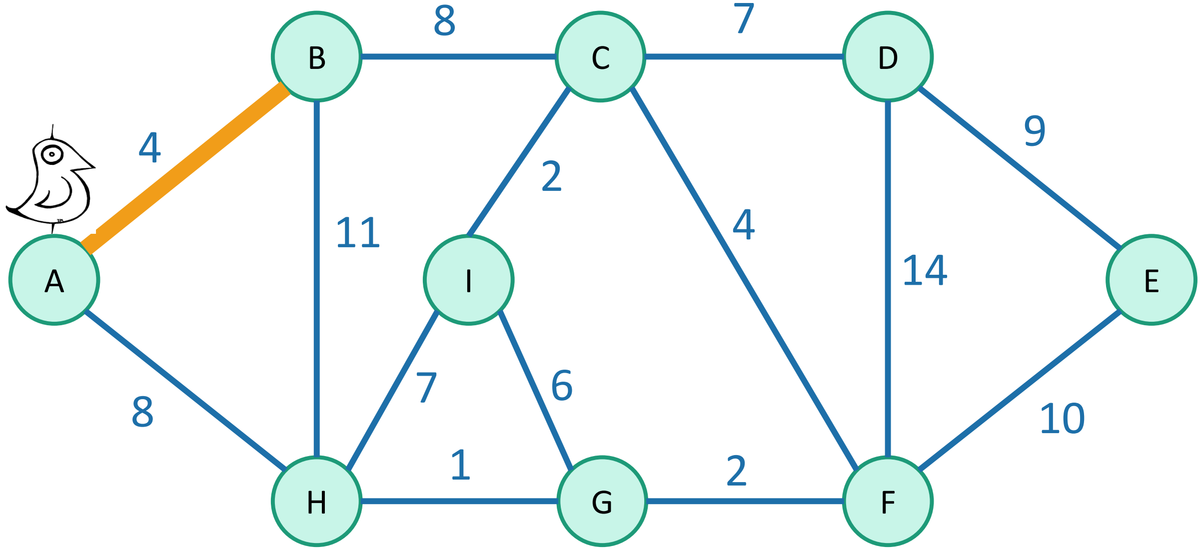


MST edges

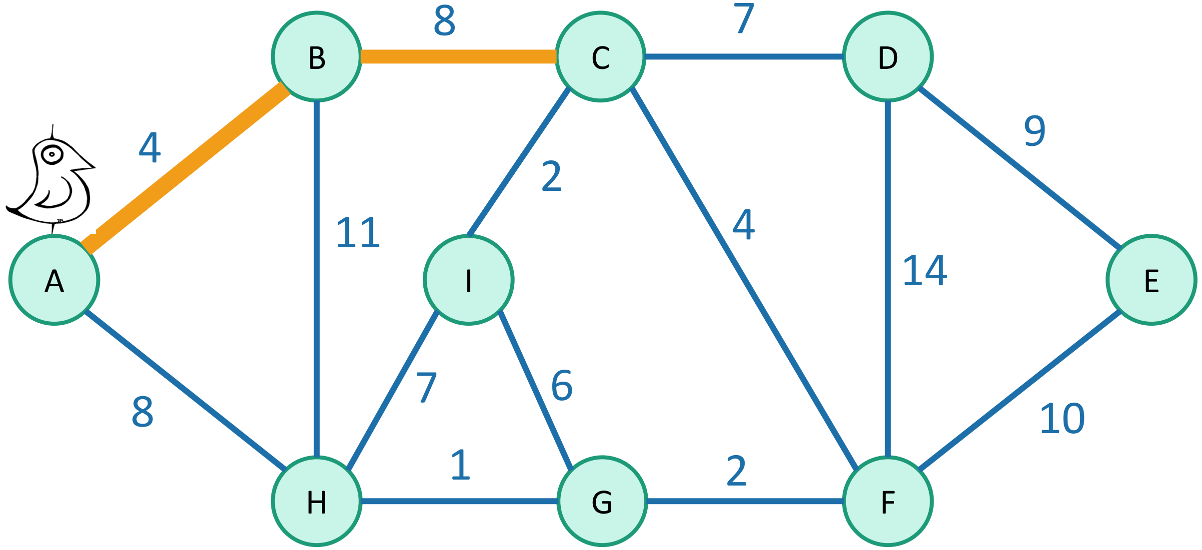
0-7 1-7 0-2 2-3 5-7 4-5 6-2

Start growing a tree, greedily add the shortest edge we can to grow the tree.

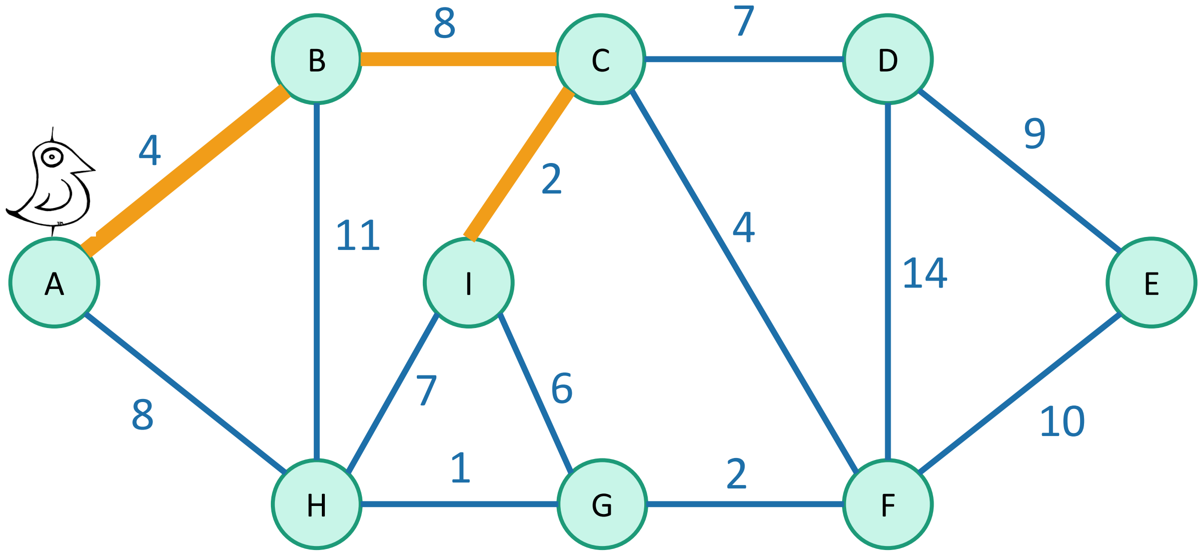




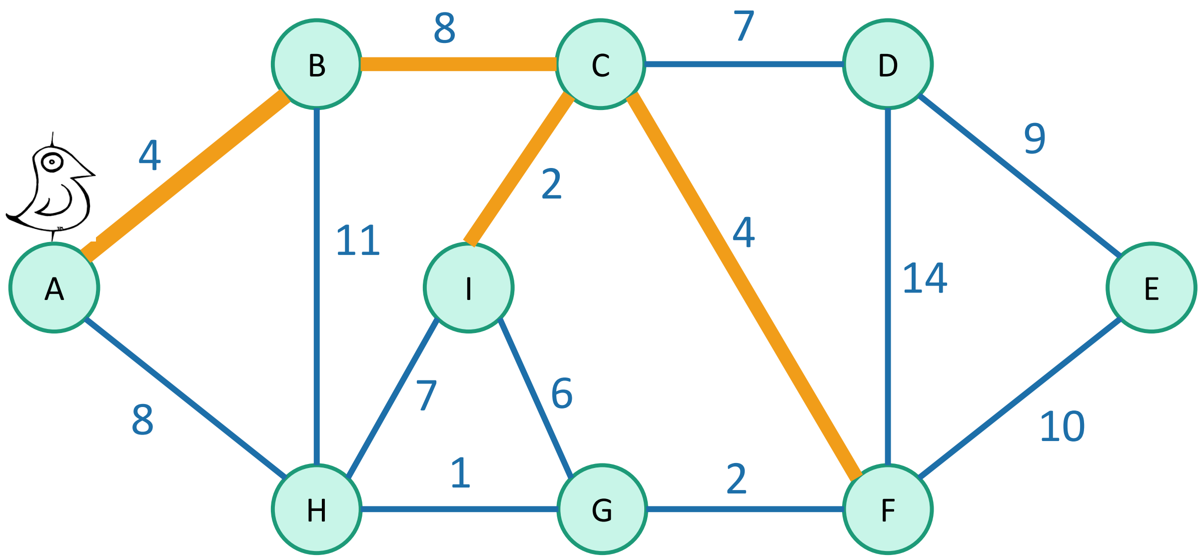
A
B
C
D
E
F
G
H
I
4
8
8
7
9
11
2
4
14
7
6
1
2
10



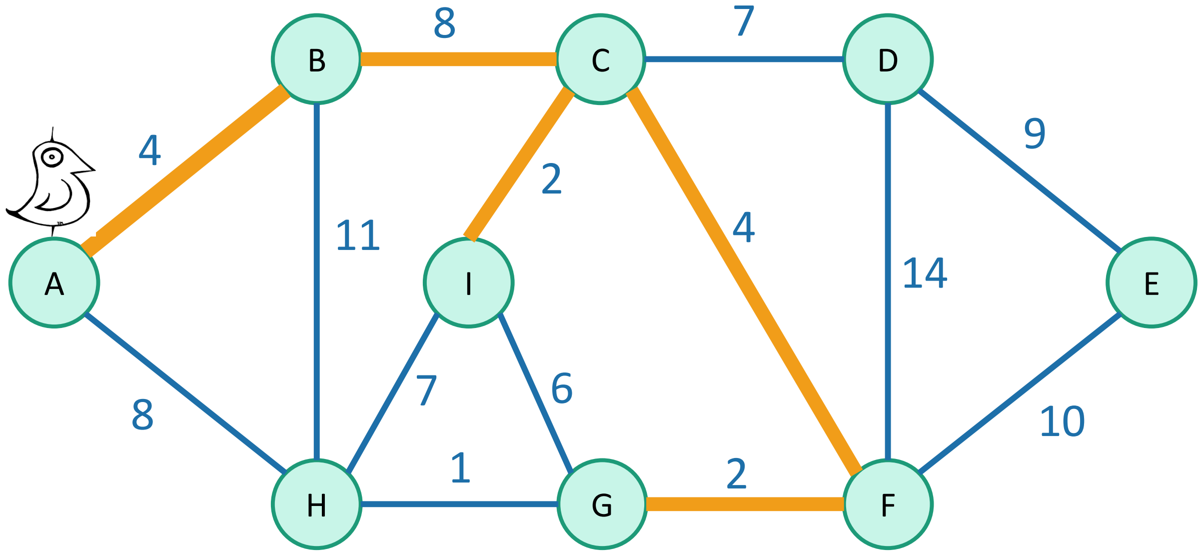
A
B
C
D
E
F
G
H
I
4
8
11
8
7
2
4
14
9
10
7
6
1
2



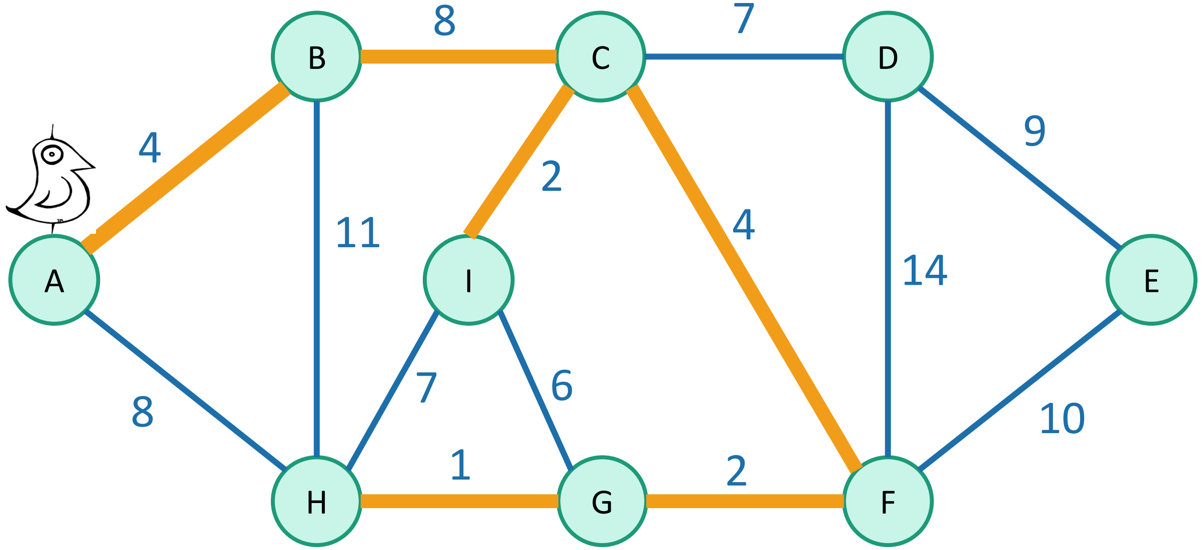
A
B
C
D
E
F
G
H
I
4
8
7
9
11
2
4
14
7
6
8
1
2
10



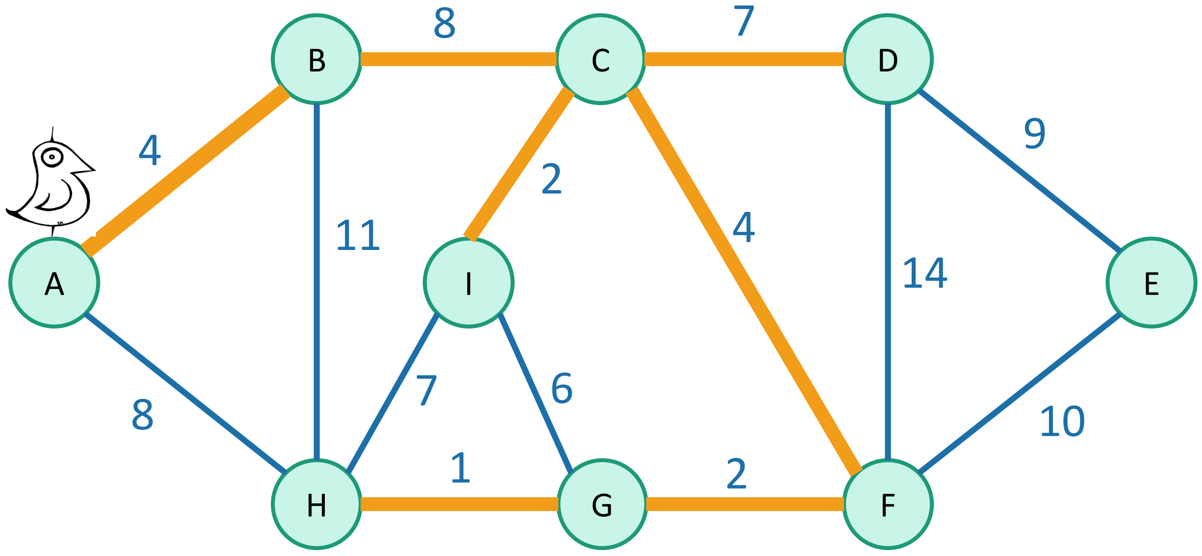
A
B
C
D
E
F
G
H
I
4
8
7
9
11
2
4
14
7
6
1
2
10
8

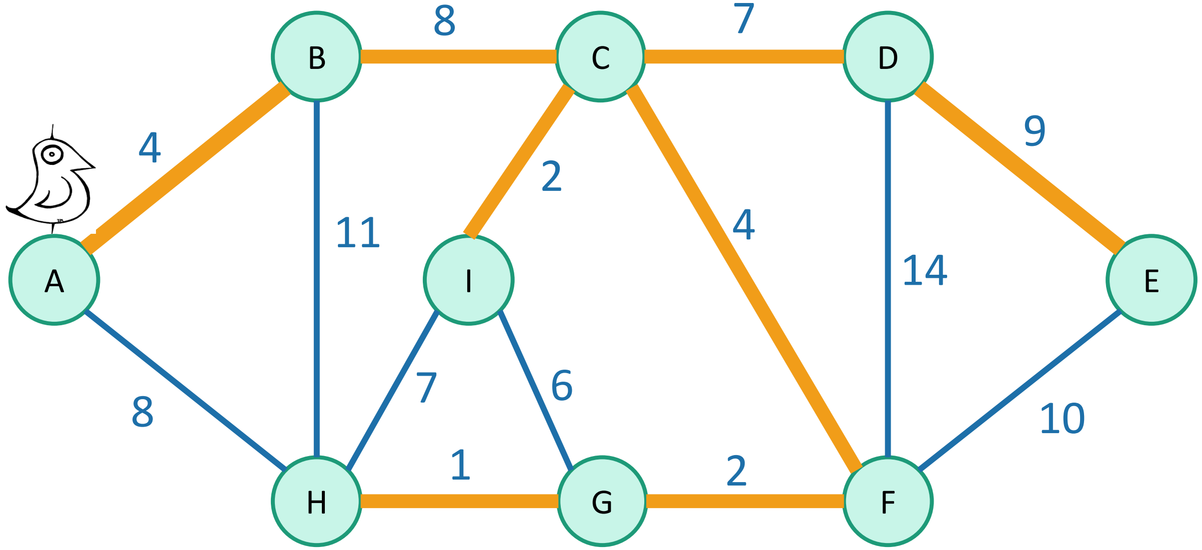


A
B
C
D
E
F
G
H
I
4
8
7
9
11
2
4
14
8
7
6
10
1
2



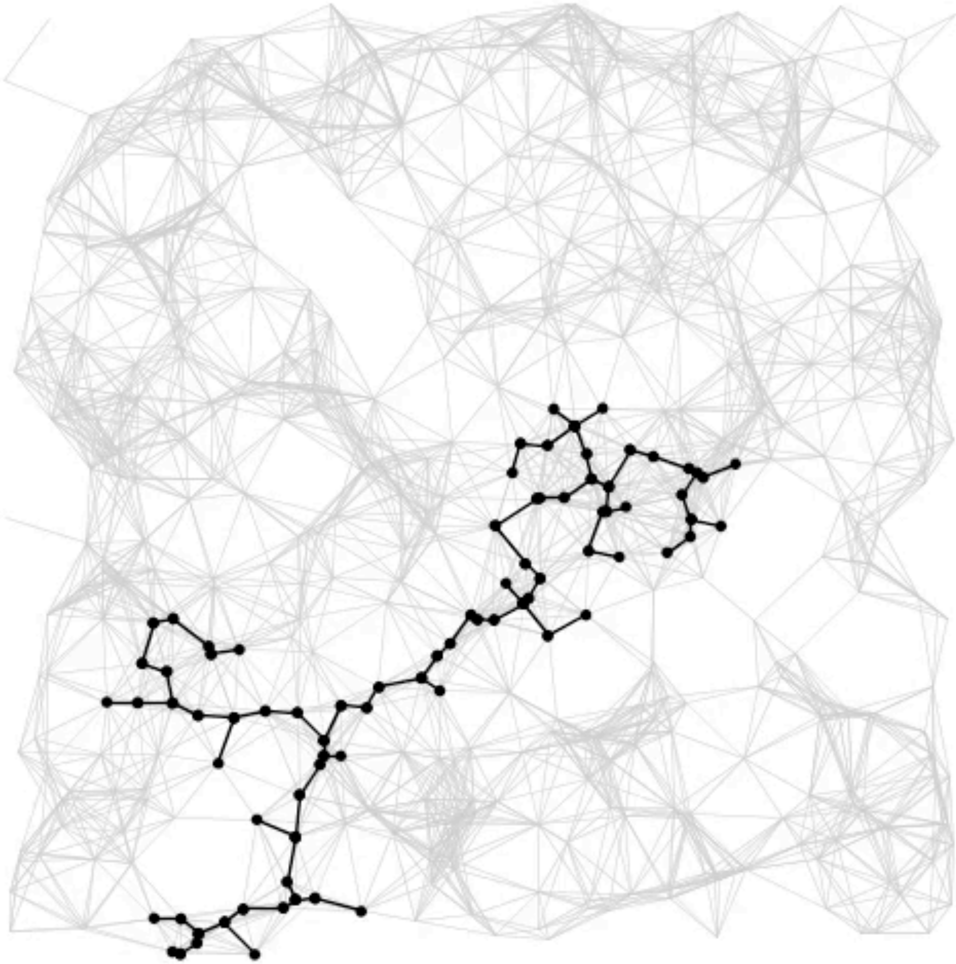
A
B
C
D
E
F
G
H
I
4
8
7
9
11
2
4
14
7
6
1
2
10
8

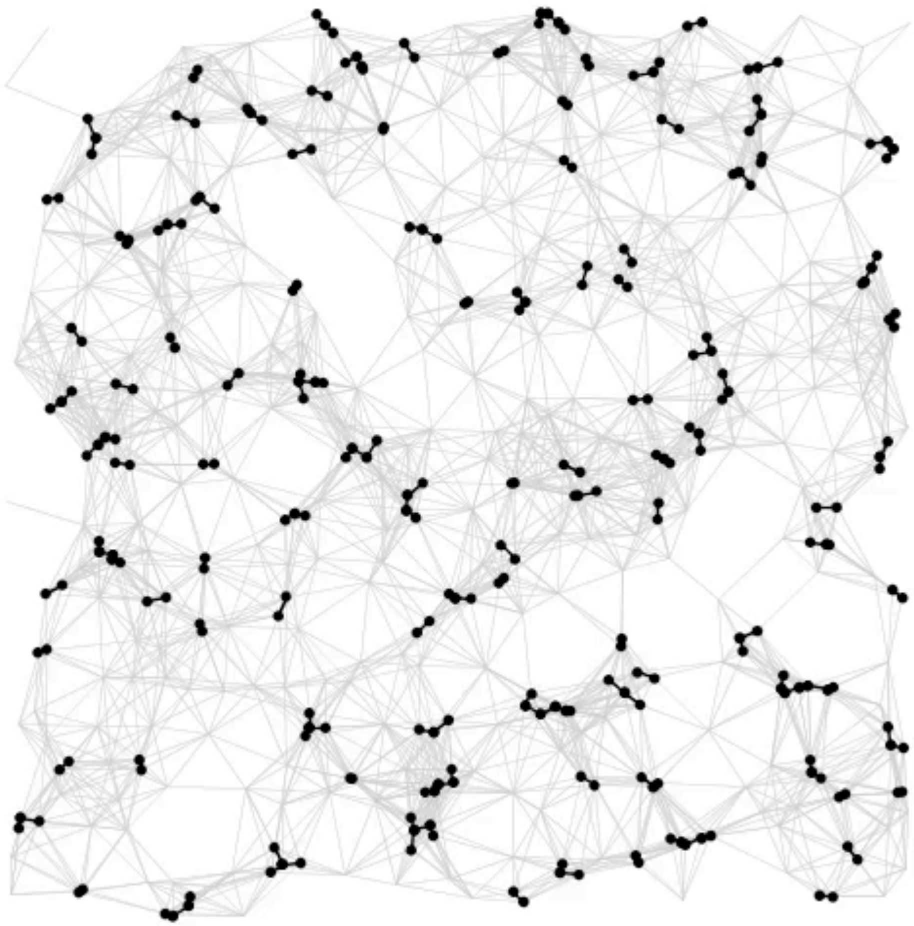

A
B
C
D
E
F
G
H
I
4
8
7
9
11
2
4
14
7
6
8
1
2
10



A
B
C
D
E
F
G
H
I
4
8
7
9
11
2
4
14
7
6
8
1
2
10

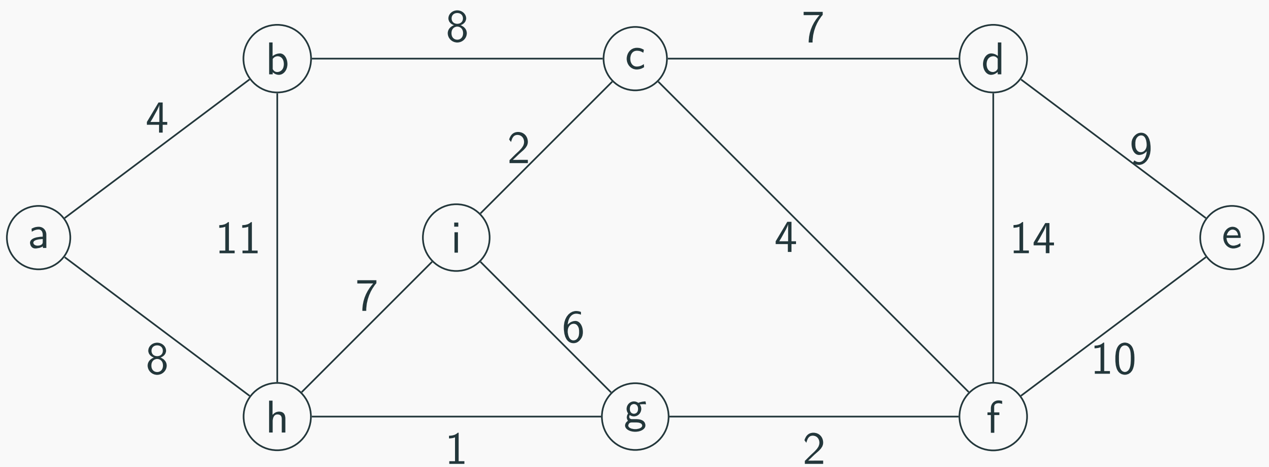
Prim vs Kruskal: visualization

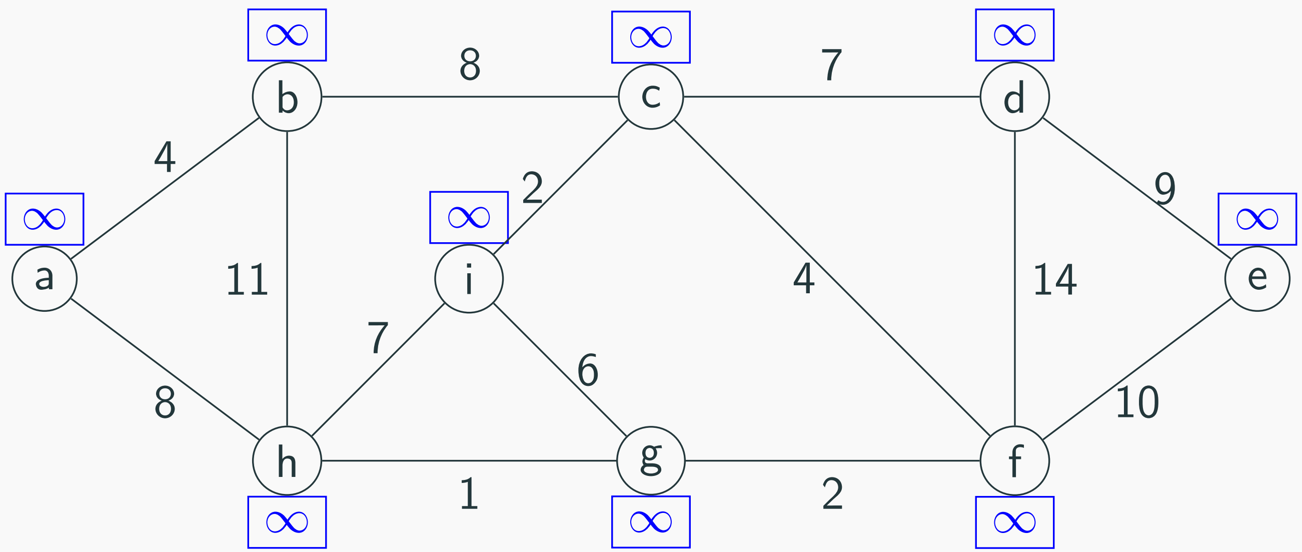




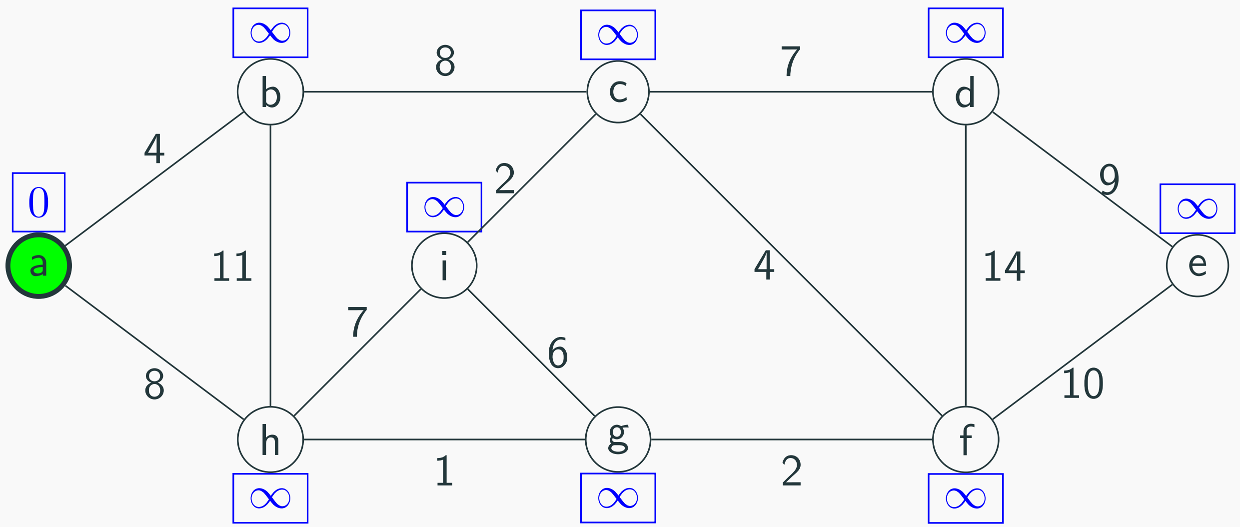
Smart implementation of Prim’s algorithm

Def.
The cost of vertex v is the cost of the shortest-known path so far between v and any node we’ve visited so far

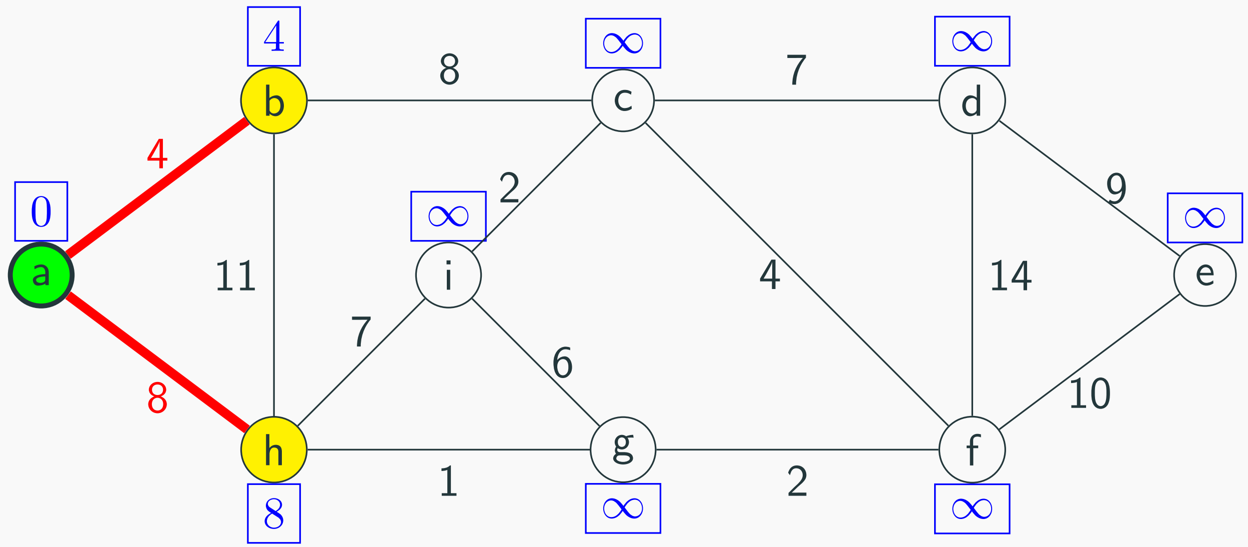

a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
1
7
4
2
14
9
10



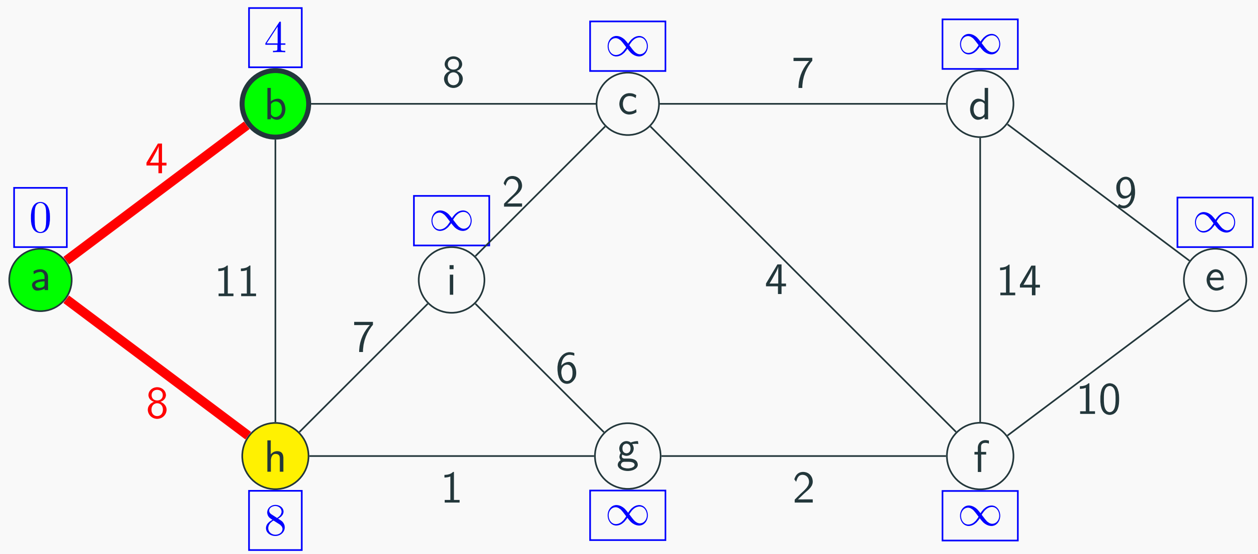
We initially set all costs to ∞



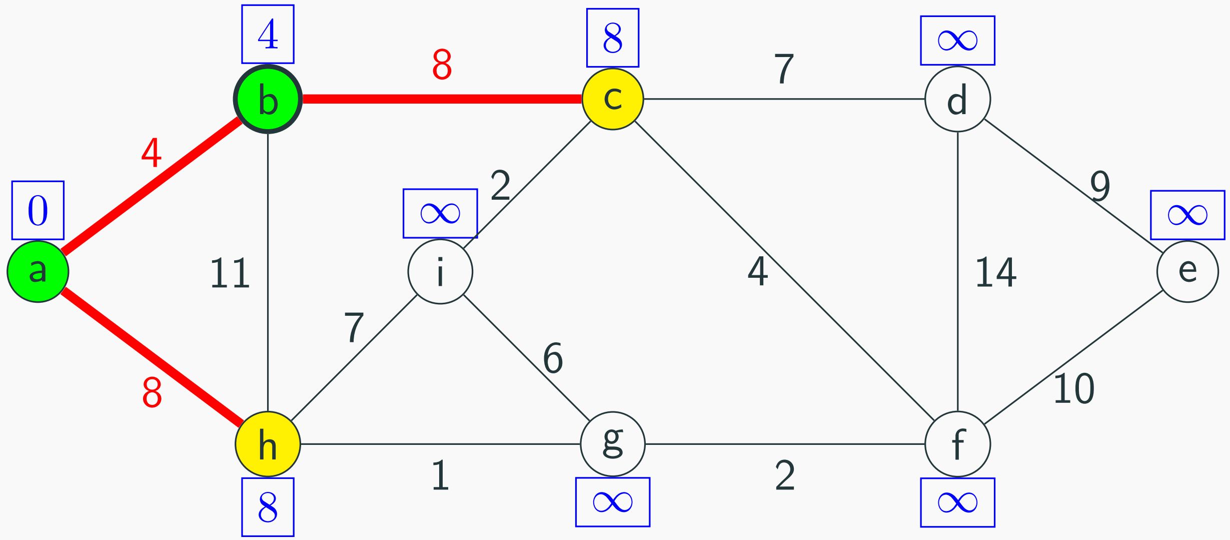
We pick an arbitrary node to start.



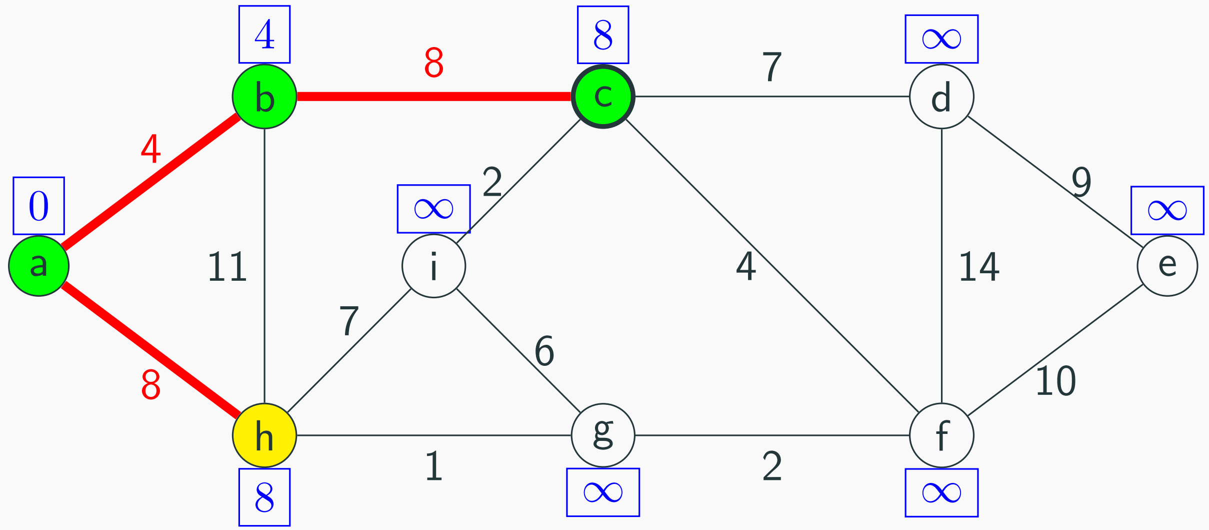
We update the adjacent nodes.



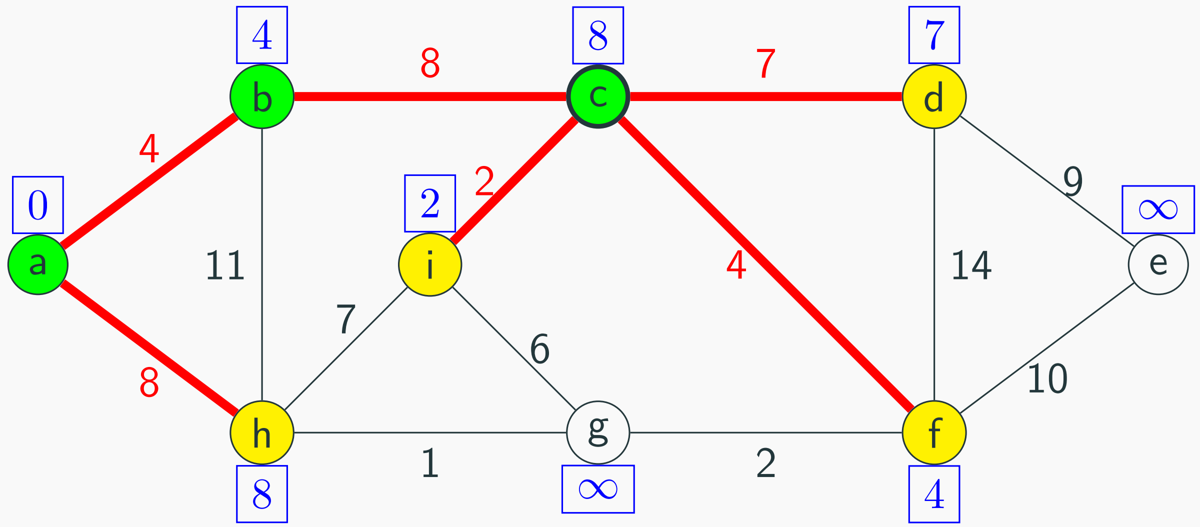
We select the one with the smallest cost.



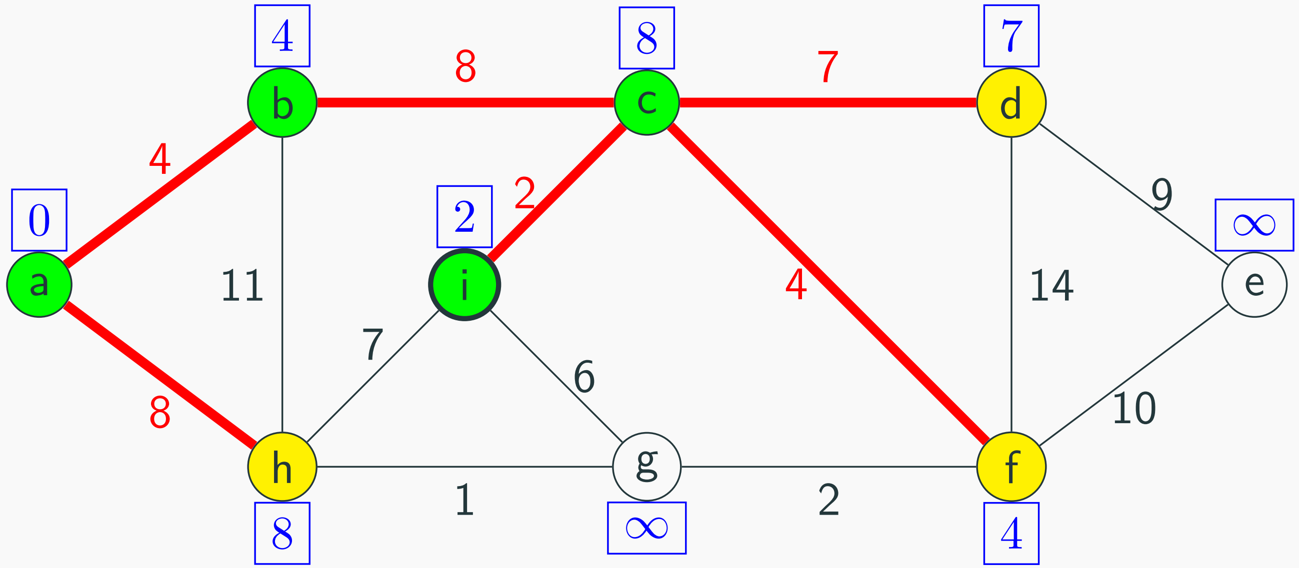
We potentially need to update h and c, but only c changes.



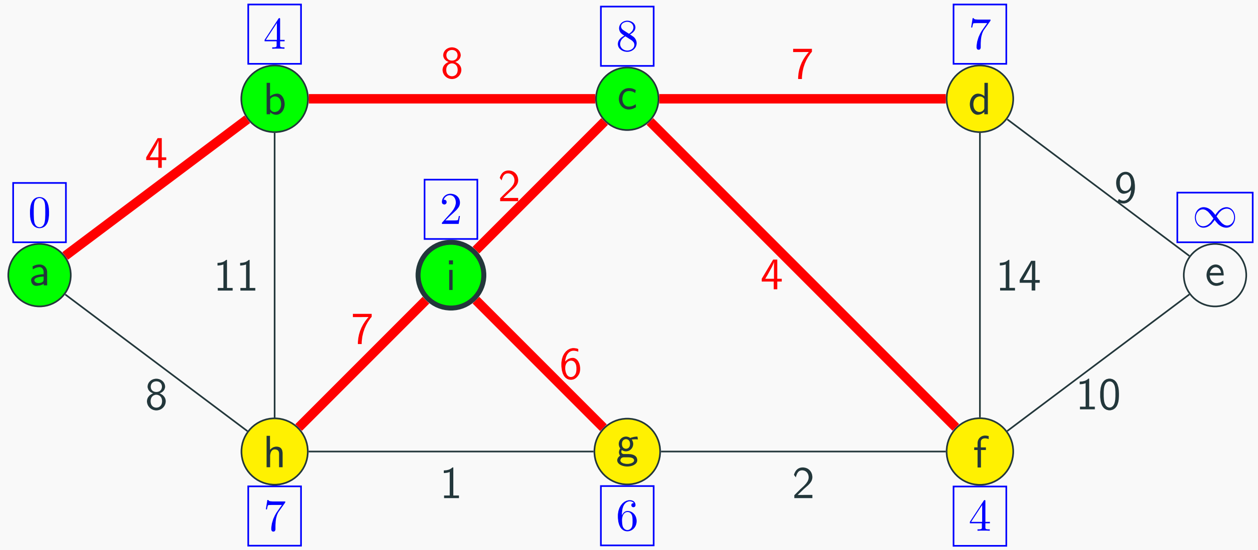
We (arbitrarily) pick c.



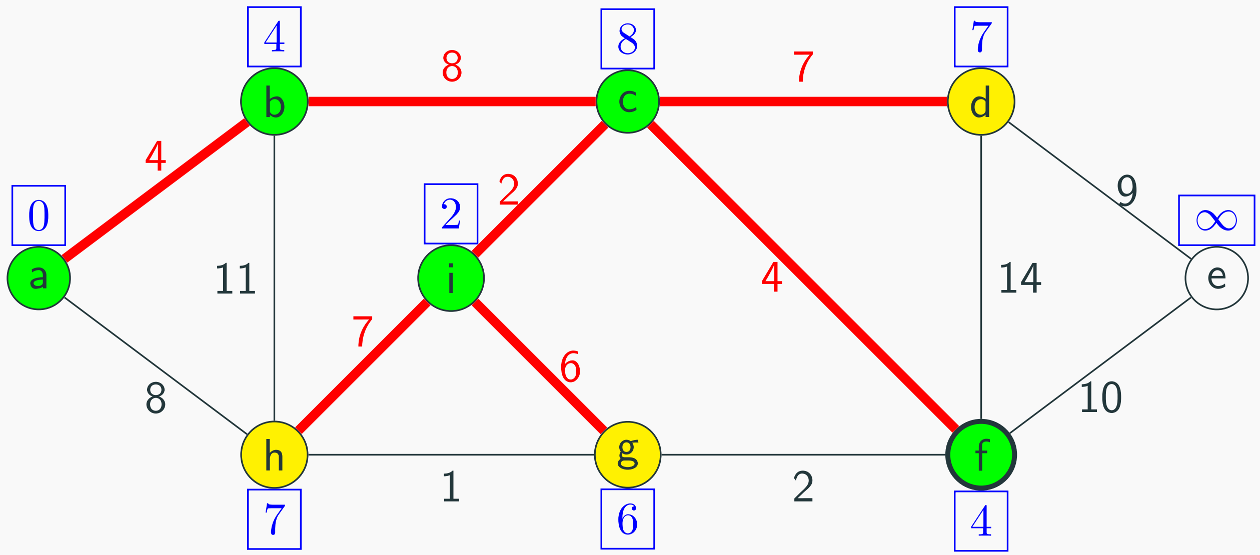
...and update the adjacent nodes. Note that we don't add the cumulative cost: the cost represents the shortest path to *any* green node, not to the start.



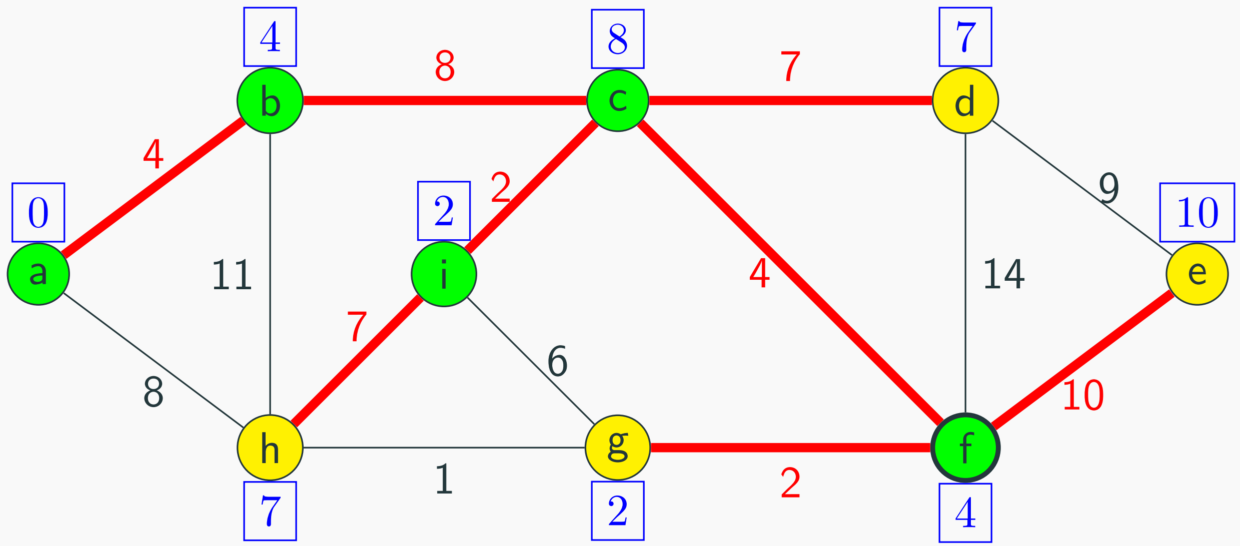
i has the smallest cost.



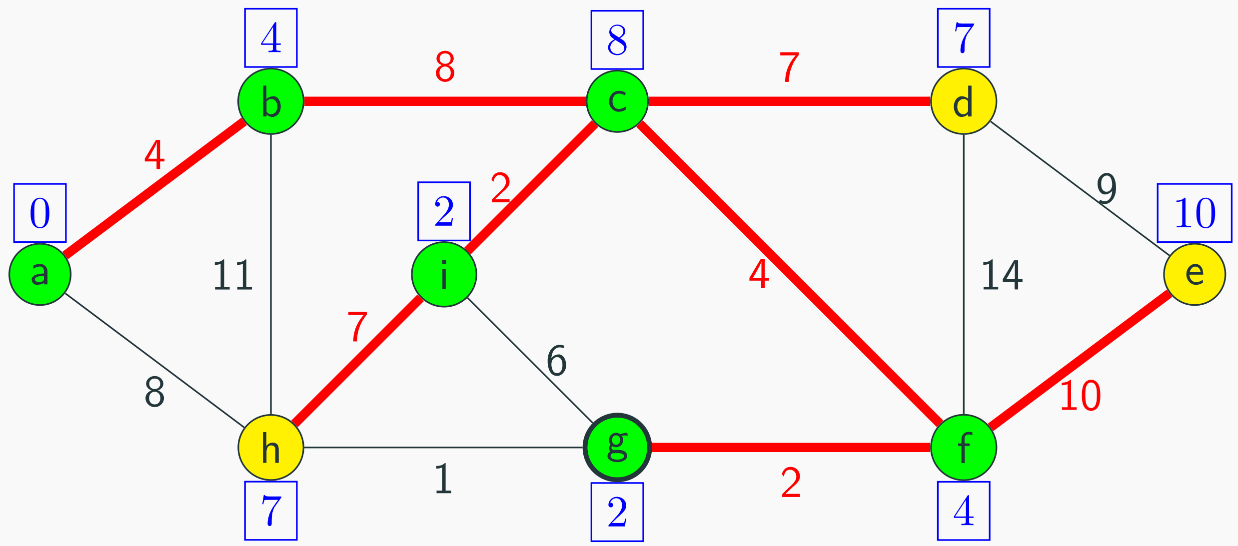
We update both unvisited nodes, and modify the edge to *h* since we now have a better option.



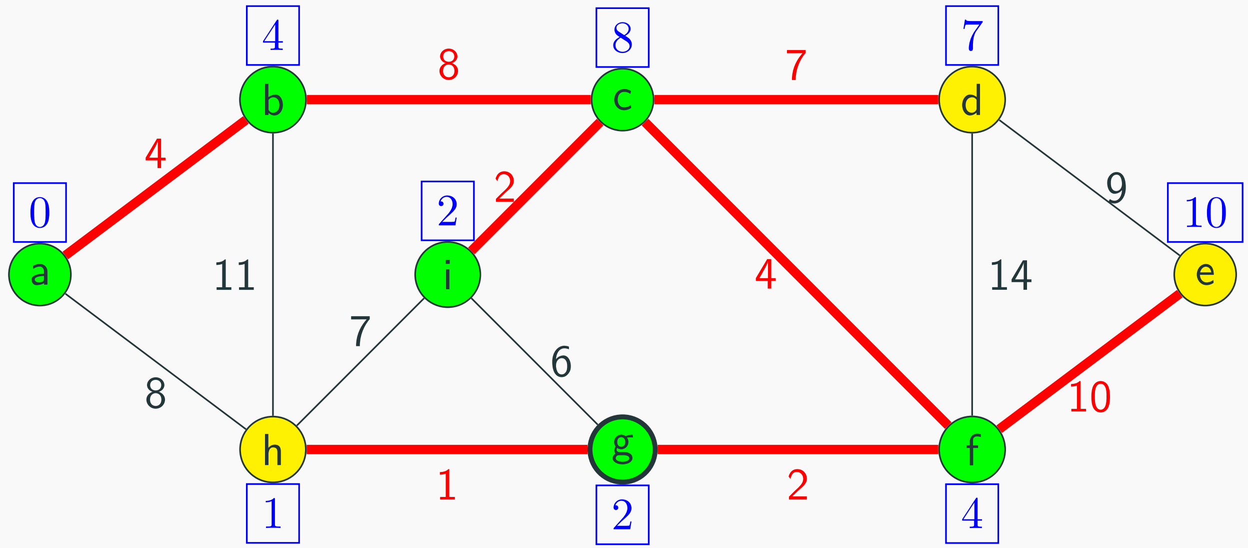
f has the smallest cost.



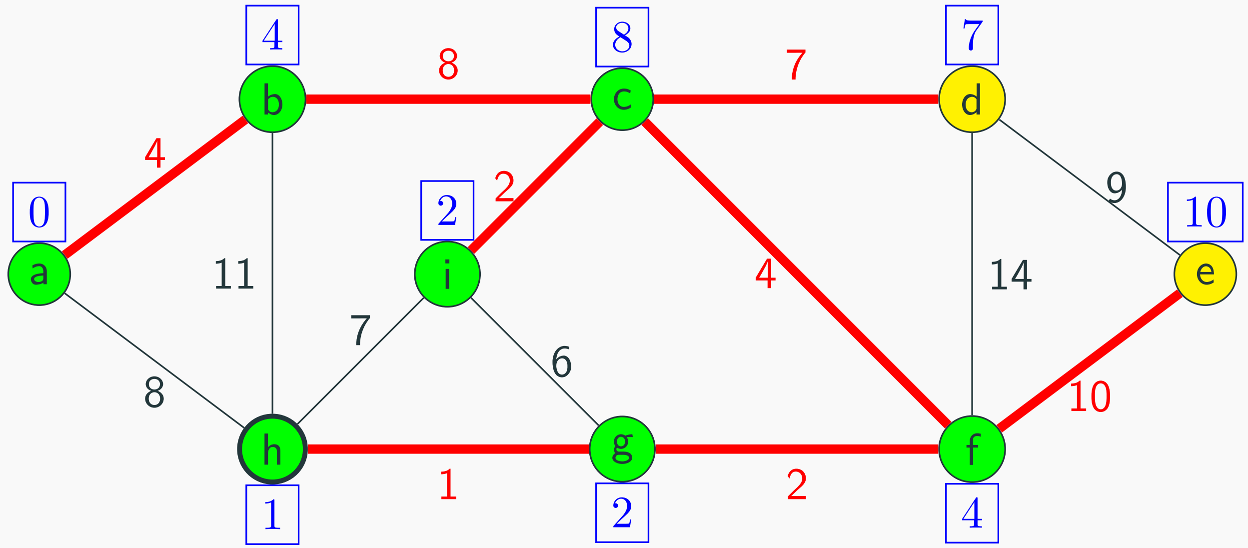
Again, we update the adjacent unvisited nodes.



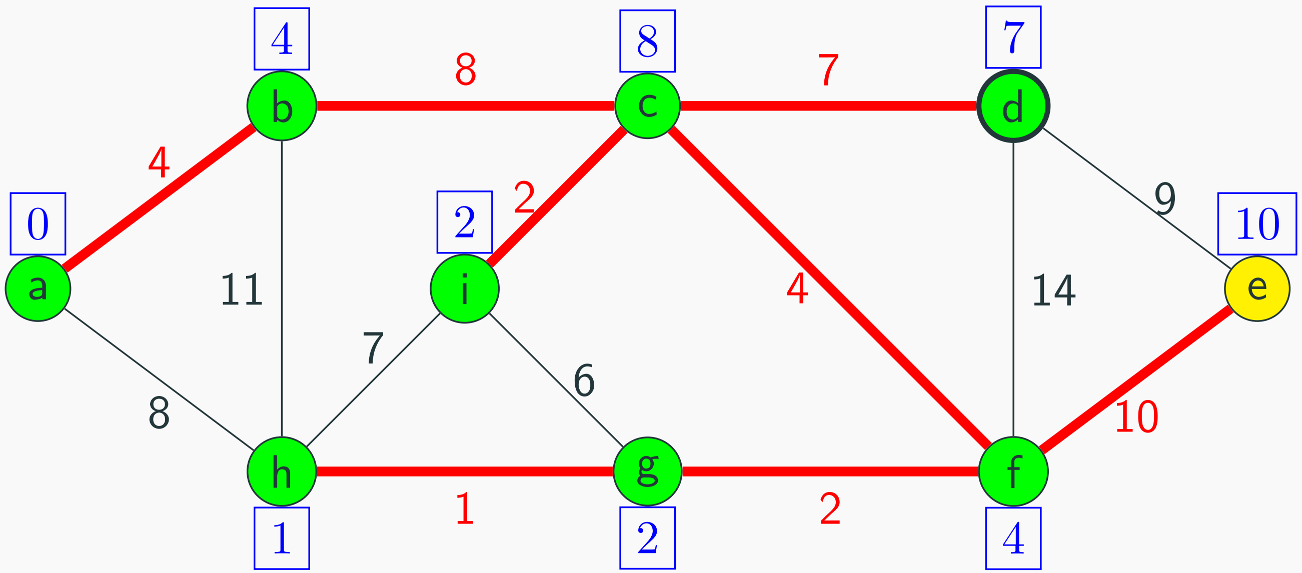
g has the smallest cost.



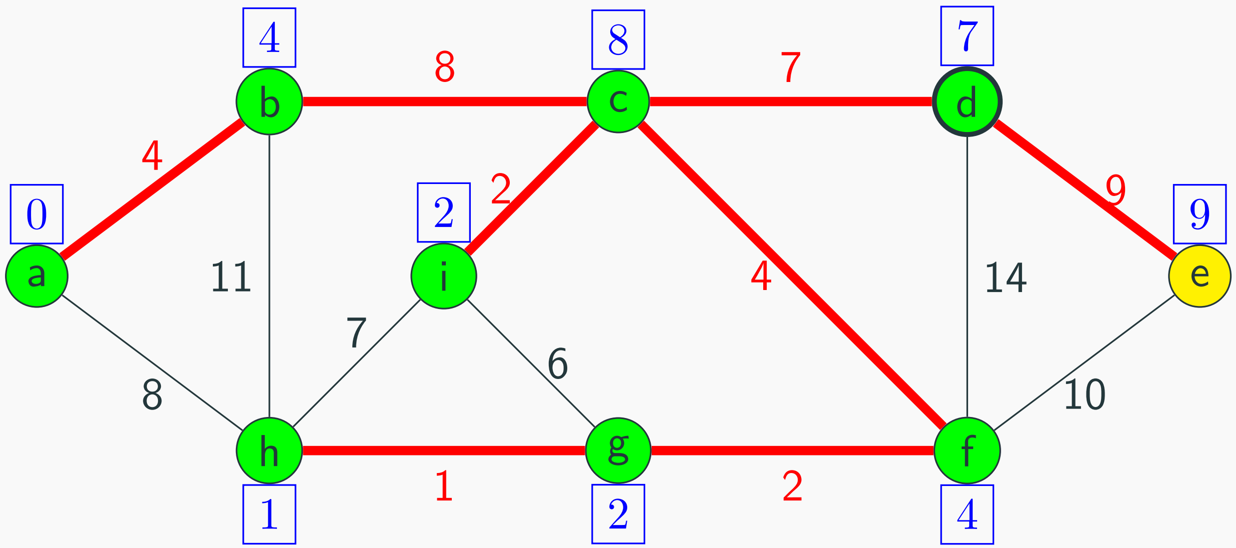
We update *h* again.



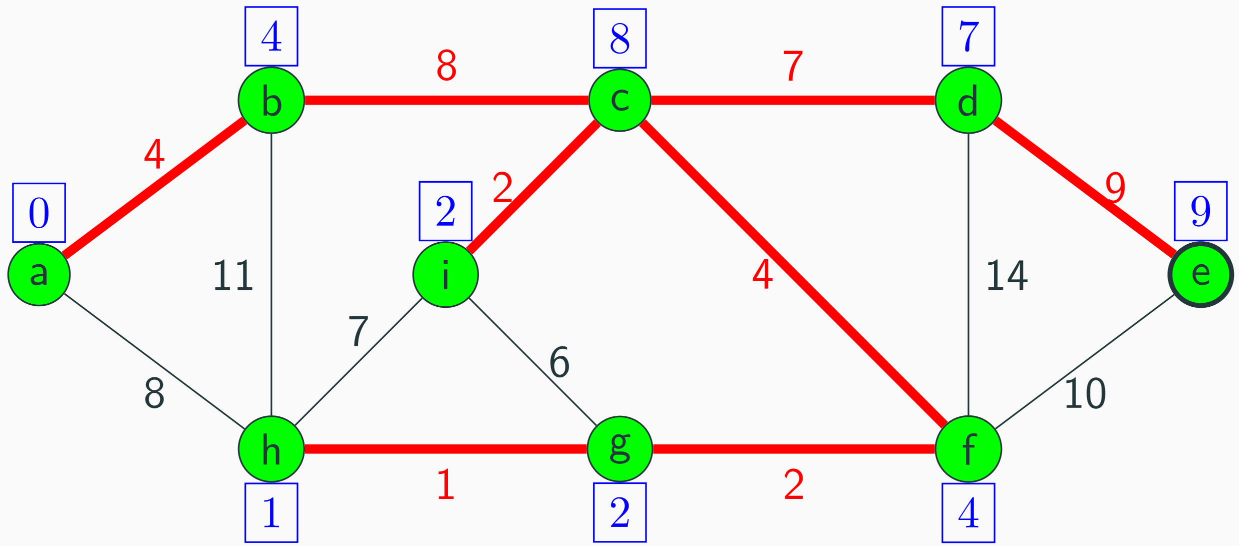
h has the smallest cost. Note that there nothing to update here.



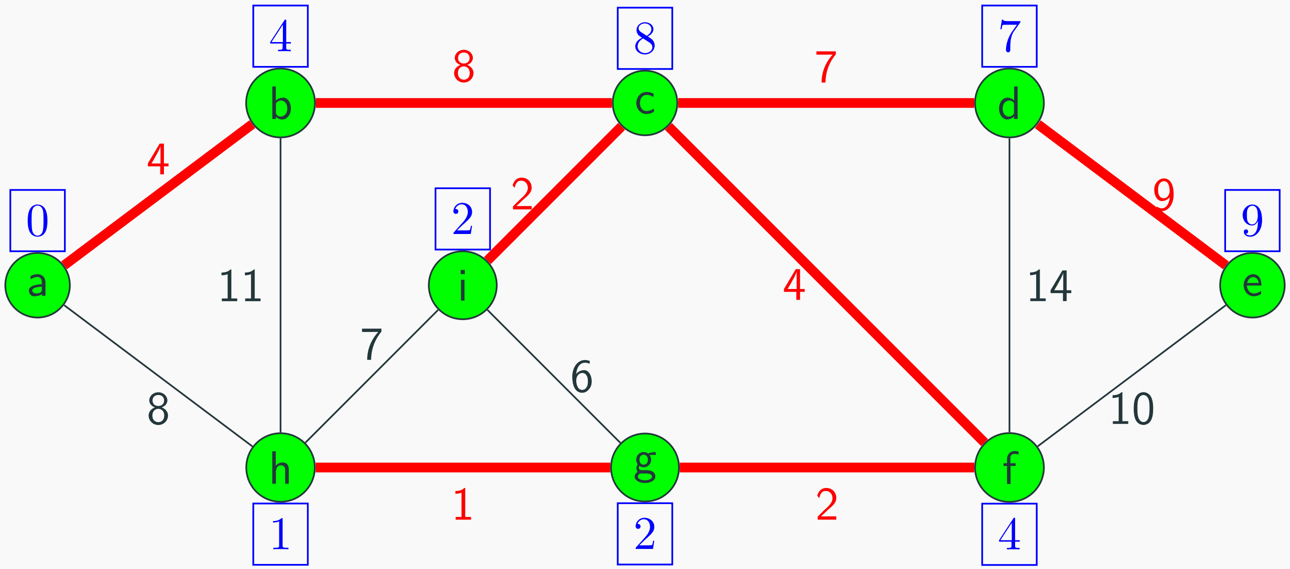
d has the smallest cost.



We can update *e*.



e has the smallest cost.



There are no more nodes left, so we're done.

Pseudocode for Prim's algorithm:

```
def prim(start):
    backpointers = new SomeDictionary<Vertex, Vertex>()

    for (v : vertices):
        set cost(v) to infinity
    set cost(start) to 0

    while (we still have unvisited nodes):
        current = get next smallest node

        for (edge : current.getOutEdges()):
            newCost = min(edge.cost, cost(edge.dst))
            update cost(edge.dst) to newCost
            backpointers.put(edge.dst, edge.src)

    return backpointers
```

Question: What is the worst-case asymptotic runtime of Prim's algorithm?

Answer: O(|V | ts + |E | tu) where...

- t_S = time needed to get next smallest node (log(lVl in the worst case)
- t_U = time needed to update vertex costs (log(lVl in the worst case)

So, O(|V | log(|V |) + |E | log(|V |))