

CSE 373: Minimum Spanning Trees: Prim and Kruskal

Michael Lee

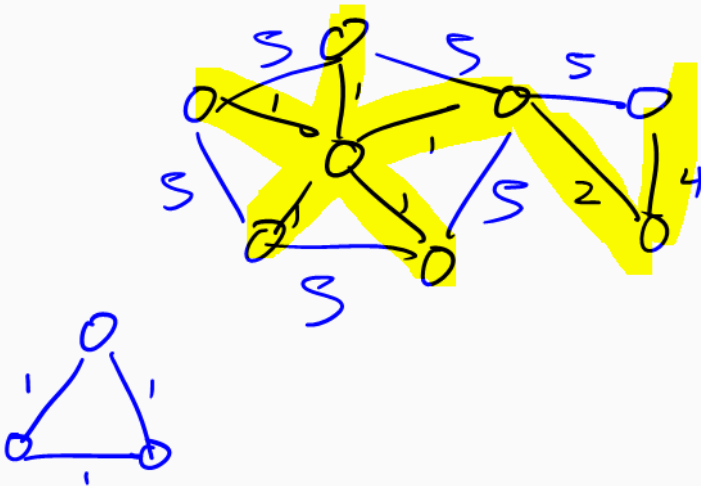
Monday, Feb 26, 2018

Warm-up:

- Remind your neighbor:
What is a MST?
- Can an MST contain
a cycle?

Minimum spanning trees

Punchline: a MST of a graph connects all the vertices together while minimizing the number of edges used (and their weights).



Minimum spanning trees

Punchline: a MST of a graph connects all the vertices together while minimizing the number of edges used (and their weights).

Minimum spanning trees

Given a connected, undirected graph $G = (V, E)$, a **minimum spanning tree** is a *subgraph* $G' = (V', E')$ such that...

Minimum spanning trees

Punchline: a MST of a graph connects all the vertices together while minimizing the number of edges used (and their weights).

Minimum spanning trees

Given a connected, undirected graph $G = (V, E)$, a **minimum spanning tree** is a *subgraph* $G' = (V', E')$ such that...

- ▶ $V = V'$ (G' is *spanning*)

Minimum spanning trees

Punchline: a MST of a graph connects all the vertices together while minimizing the number of edges used (and their weights).

Minimum spanning trees

Given a connected, undirected graph $G = (V, E)$, a **minimum spanning tree** is a *subgraph* $G' = (V', E')$ such that...

- ▶ $V = V'$ (G' is *spanning*)
- ▶ There exists a path from any vertex to any other one

Minimum spanning trees

Punchline: a MST of a graph connects all the vertices together while minimizing the number of edges used (and their weights).

Minimum spanning trees

Given a connected, undirected graph $G = (V, E)$, a **minimum spanning tree** is a *subgraph* $G' = (V', E')$ such that...

- ▶ $V = V'$ (G' is *spanning*)
- ▶ There exists a path from any vertex to any other one
- ▶ The sum of the edge weights in E' is *minimized*.

Minimum spanning trees

Punchline: a MST of a graph connects all the vertices together while minimizing the number of edges used (and their weights).

Minimum spanning trees

Given a connected, undirected graph $G = (V, E)$, a **minimum spanning tree** is a *subgraph* $G' = (V', E')$ such that...

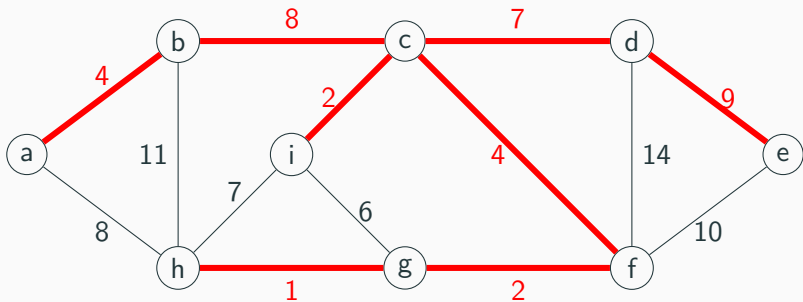
- ▶ $V = V'$ (G' is *spanning*)
- ▶ There exists a path from any vertex to any other one
- ▶ The sum of the edge weights in E' is *minimized*.

In order for a graph to have a MST, the graph must...

- ▶ ...be connected – there is a path from a vertex to any other vertex. (Note: this means $|V| \leq |E|$).
- ▶ ...be undirected.

Minimum spanning trees: example

An example of an minimum spanning tree (MST):



Minimum spanning trees: Applications

Example questions:

- ▶ We want to connect phone lines to houses, but laying down cable is expensive. How can we minimize the amount of wire we must install?

Minimum spanning trees: Applications

Example questions:

- ▶ We want to connect phone lines to houses, but laying down cable is expensive. How can we minimize the amount of wire we must install?
- ▶ We have items on a circuit we want to be “electrically equivalent”. How can we connect them together using a minimum amount of wire?

Minimum spanning trees: Applications

Example questions:

- ▶ We want to connect phone lines to houses, but laying down cable is expensive. How can we minimize the amount of wire we must install?
- ▶ We have items on a circuit we want to be “electrically equivalent”. How can we connect them together using a minimum amount of wire?

Other applications:

Minimum spanning trees: Applications

Example questions:

- ▶ We want to connect phone lines to houses, but laying down cable is expensive. How can we minimize the amount of wire we must install?
- ▶ We have items on a circuit we want to be “electrically equivalent”. How can we connect them together using a minimum amount of wire?

Other applications:

- ▶ Implement efficient multiple constant multiplication
- ▶ Minimizing number of packets transmitted across a network
- ▶ Machine learning (e.g. real-time face verification)
- ▶ Graphics (e.g. image segmentation)

Minimum spanning trees: properties

Important properties:

- ▶ A valid MST cannot contain a cycle

Minimum spanning trees: properties

Important properties:

- ▶ A valid MST cannot contain a cycle
- ▶ If we add or remove an edge from an MST, it's no longer a valid MST for that graph.

Adding an edge introduces a cycle; removing an edge means vertices are no longer connected.

Minimum spanning trees: properties

Important properties:

- ▶ A valid MST cannot contain a cycle
- ▶ If we add or remove an edge from an MST, it's no longer a valid MST for that graph.
Adding an edge introduces a cycle; removing an edge means vertices are no longer connected.
- ▶ If there are $|V|$ vertices, the MST contains exactly $|V| - 1$ edges.

Minimum spanning trees: properties

Important properties:

- ▶ A valid MST cannot contain a cycle
- ▶ If we add or remove an edge from an MST, it's no longer a valid MST for that graph.
Adding an edge introduces a cycle; removing an edge means vertices are no longer connected.
- ▶ If there are $|V|$ vertices, the MST contains exactly $|V| - 1$ edges.
- ▶ An MST is always a tree.

Minimum spanning trees: properties

Important properties:

- ▶ A valid MST cannot contain a cycle
- ▶ If we add or remove an edge from an MST, it's no longer a valid MST for that graph.
Adding an edge introduces a cycle; removing an edge means vertices are no longer connected.
- ▶ If there are $|V|$ vertices, the MST contains exactly $|V| - 1$ edges.
- ▶ An MST is always a tree.
- ▶ If every edge has a unique weight, there exists a unique MST.

Minimum spanning trees: algorithm

Design question: how would you implement an algorithm to find the MST of some graph, assuming the edges *all have the same weight*?

Minimum spanning trees: algorithm

Design question: how would you implement an algorithm to find the MST of some graph, assuming the edges *all have the same weight*?

Hints:

- ▶ Try modifying DFS or BFS.

Minimum spanning trees: algorithm

Design question: how would you implement an algorithm to find the MST of some graph, assuming the edges *all have the same weight*?



Hints:

- ▶ Try modifying DFS or BFS.
- ▶ Try using an *incremental* approach: start with an empty graph, and steadily add nodes and edges.

Minimum spanning trees: approach 1, adding nodes

Intuition: We start with an “empty” MST, and steadily grow it.

Minimum spanning trees: approach 1, adding nodes

Intuition: We start with an “empty” MST, and steadily grow it.

Core algorithm:

1. Start with an arbitrary node.

Minimum spanning trees: approach 1, adding nodes

Intuition: We start with an “empty” MST, and steadily grow it.

Core algorithm:

1. Start with an arbitrary node.
2. Run either DFS or BFS, storing *edges* in our stack or queue.

Minimum spanning trees: approach 1, adding nodes

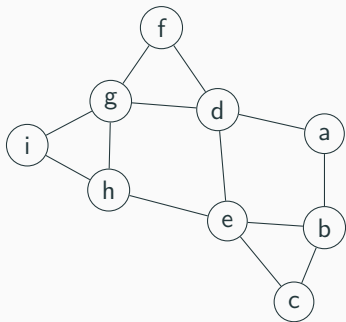
Intuition: We start with an “empty” MST, and steadily grow it.

Core algorithm:

1. Start with an arbitrary node.
2. Run either DFS or BFS, storing *edges* in our stack or queue.
3. As we visit nodes, add each edge we remove to our MST.

Minimum spanning trees: approach 1, adding nodes

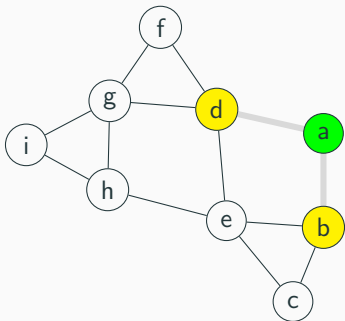
An example using a modified version of DFS:



Stack:

Minimum spanning trees: approach 1, adding nodes

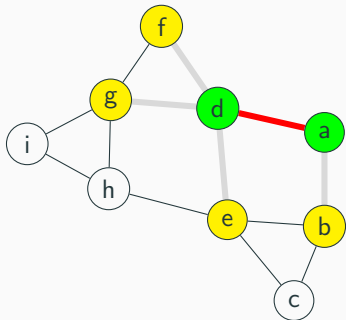
An example using a modified version of DFS:



Stack: (a, b) , (a, d) ,

Minimum spanning trees: approach 1, adding nodes

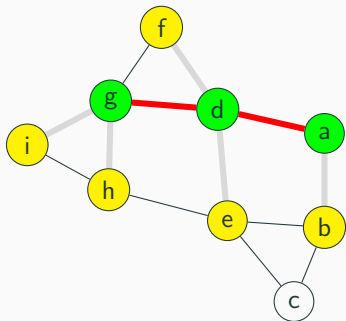
An example using a modified version of DFS:



Stack: (a, b) , (d, e) , (d, f) , (d, g) ,

Minimum spanning trees: approach 1, adding nodes

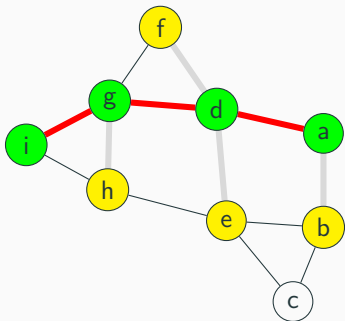
An example using a modified version of DFS:



Stack: (a, b) , (d, e) , (d, f) , (g, h) , (g, i) ,

Minimum spanning trees: approach 1, adding nodes

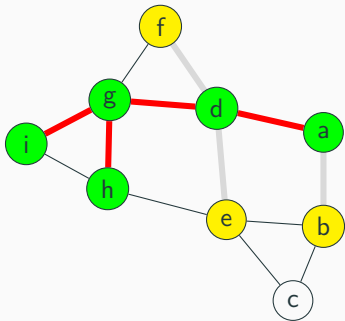
An example using a modified version of DFS:



Stack: (a, b) , (d, e) , (d, f) , (g, h) ,

Minimum spanning trees: approach 1, adding nodes

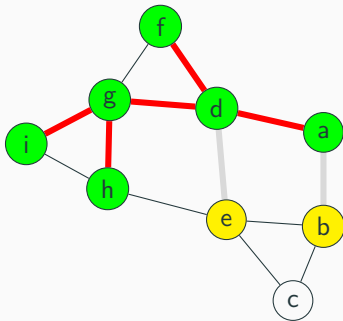
An example using a modified version of DFS:



Stack: (a, b) , (d, e) , (d, f) ,

Minimum spanning trees: approach 1, adding nodes

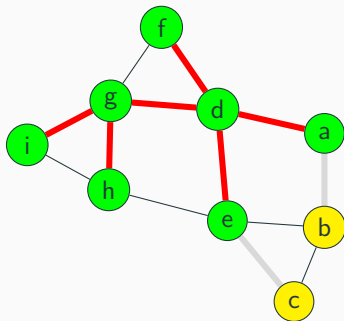
An example using a modified version of DFS:



Stack: (a, b) , (d, e) ,

Minimum spanning trees: approach 1, adding nodes

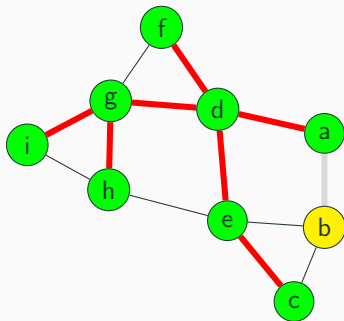
An example using a modified version of DFS:



Stack: (a, b) , (e, c) ,

Minimum spanning trees: approach 1, adding nodes

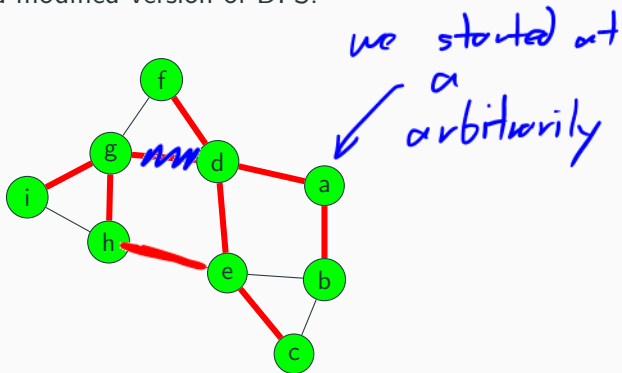
An example using a modified version of DFS:



Stack: (a, b) ,

Minimum spanning trees: approach 1, adding nodes

An example using a modified version of DFS:



Stack:

Minimum spanning trees: approach 1, adding nodes

What if the edges have different weights?

Minimum spanning trees: approach 1, adding nodes

What if the edges have different weights?

Observation:

We solved a similar problem earlier this quarter, when studying shortest path algorithms!

Interlude: finding the shortest path

Review: How do we find the shortest path between two vertices?

- ▶ If the graph is unweighted: run BFS

Interlude: finding the shortest path

Review: How do we find the shortest path between two vertices?

- ▶ If the graph is unweighted: run BFS
- ▶ If the graph is weighted: run Dijkstra's

Interlude: finding the shortest path

Review: How do we find the shortest path between two vertices?

- ▶ If the graph is unweighted: run BFS
- ▶ If the graph is weighted: run Dijkstra's

How does Dijkstra's algorithm work?

1. Give each vertex v a "cost": the cost of the shortest-known path so far between v and the start.
(The cost of a path is the sum of the edge weights in that path)

Interlude: finding the shortest path

Review: How do we find the shortest path between two vertices?

- ▶ If the graph is unweighted: run BFS
- ▶ If the graph is weighted: run Dijkstra's

How does Dijkstra's algorithm work?

1. Give each vertex v a "cost": the cost of the shortest-known path so far between v and the start.
(The cost of a path is the sum of the edge weights in that path)
2. Pick the node with the smallest cost, update adjacent node costs, repeat

Minimum spanning trees: approach 1, adding nodes

Intuition: We can use the same idea to find a MST!

Minimum spanning trees: approach 1, adding nodes

Intuition: We can use the same idea to find a MST!

Core idea: Use the exact same algorithm as Dijkstra's algorithm, but **redefine the cost:**

► **Previously, for Dijkstra's:**

The cost of vertex v is the cost of the **shortest-known path** so far between **v and the start**

Minimum spanning trees: approach 1, adding nodes

Intuition: We can use the same idea to find a MST!

Core idea: Use the exact same algorithm as Dijkstra's algorithm, but redefine the cost:

► **Previously, for Dijkstra's:**

The cost of vertex v is the cost of the shortest-known path so far between v and the start

► **Now:**

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*

Minimum spanning trees: approach 1, adding nodes

Intuition: We can use the same idea to find a MST!

Core idea: Use the exact same algorithm as Dijkstra's algorithm, but redefine the cost:

▶ **Previously, for Dijkstra's:**

The cost of vertex v is the cost of the shortest-known path so far between v and the start

▶ **Now:**

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*

This algorithm is known as **Prim's algorithm**.

Compare and contrast: Dijkstra vs Prim

Pseudocode for Dijkstra's algorithm:

```
def dijkstra(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(cost(current) + edge.cost, cost(edge.dst))  
            update cost(edge.dst) to newCost  
            backpointers.put(edge.dst, edge.src)  
  
    return backpointers
```

Compare and contrast: Dijkstra vs Prim

Pseudocode for Prim's algorithm:

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

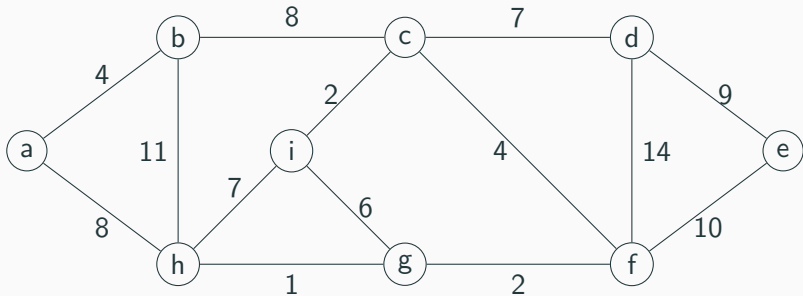
    for (v : vertices):
        set cost(v) to infinity
    set cost(start) to 0

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

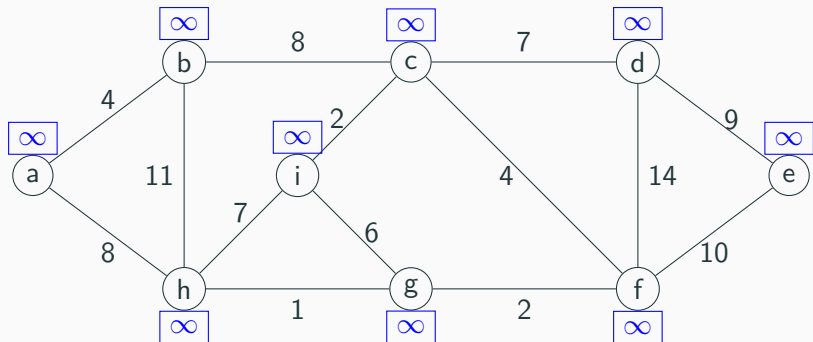
        for (edge : current.getOutEdges()):
            newCost = min(edge.cost, cost(edge.dst))
            update cost(edge.dst) to newCost
            backpointers.put(edge.dst, edge.src)

    return backpointers
```

Prim's algorithm: an example

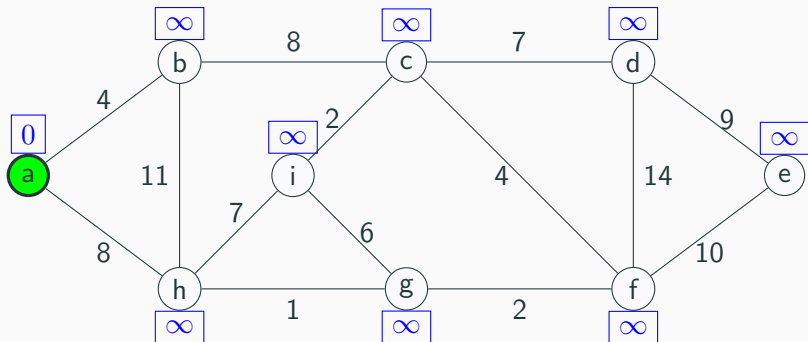


Prim's algorithm: an example



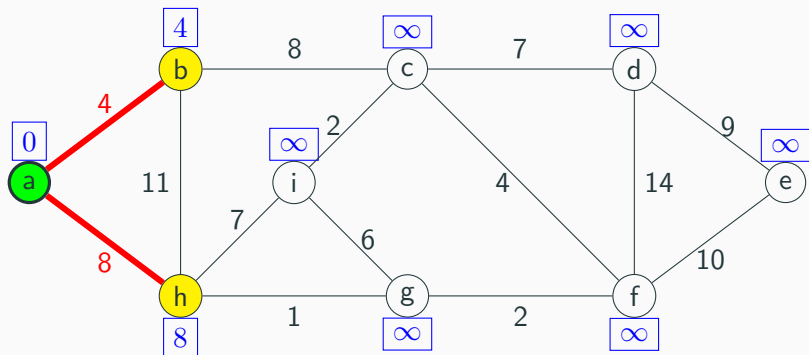
We initially set all costs to ∞ , just like with Dijkstra.

Prim's algorithm: an example



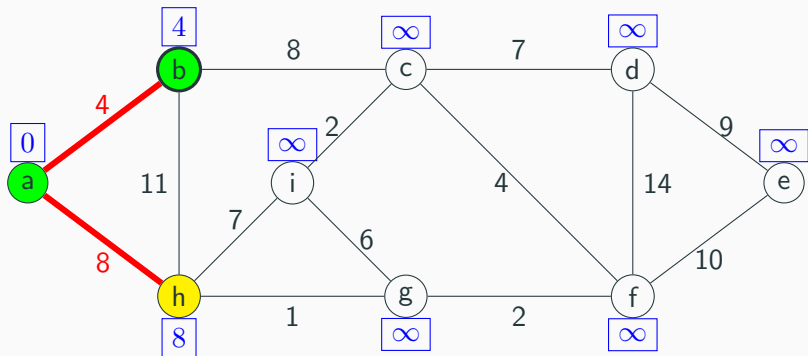
We pick an arbitrary node to start.

Prim's algorithm: an example



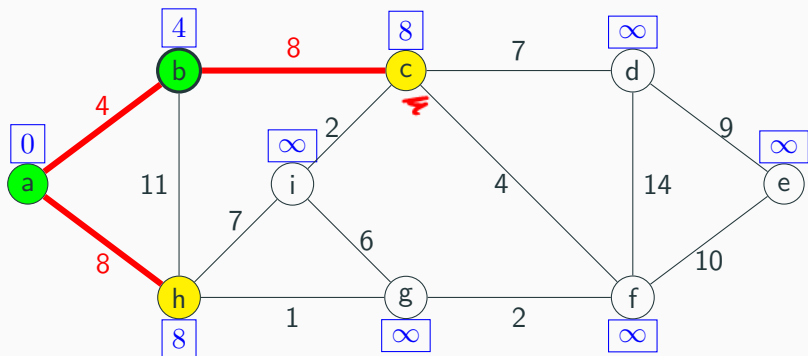
We update the adjacent nodes.

Prim's algorithm: an example



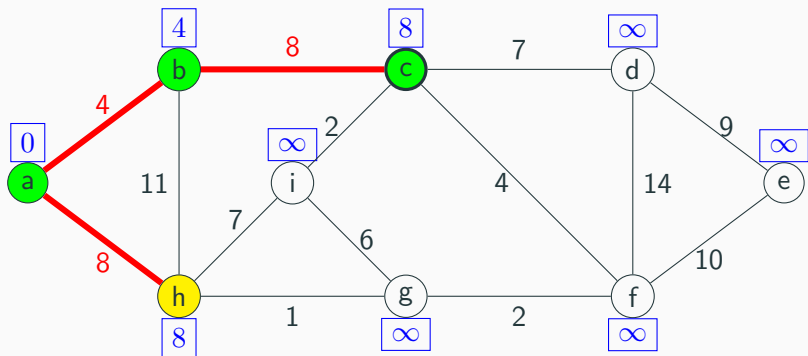
We select the one with the smallest cost.

Prim's algorithm: an example



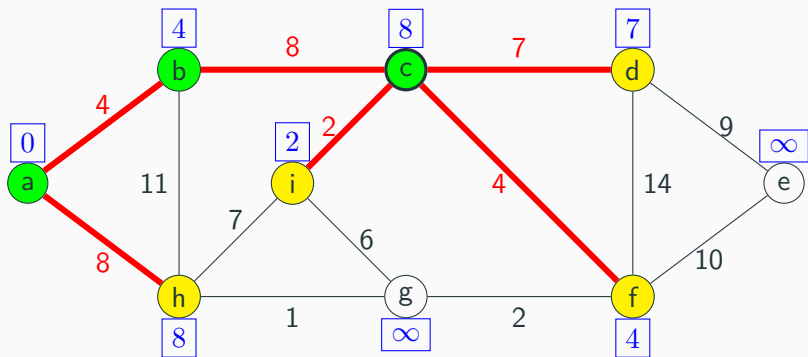
We potentially need to update h and c , but only c changes.

Prim's algorithm: an example



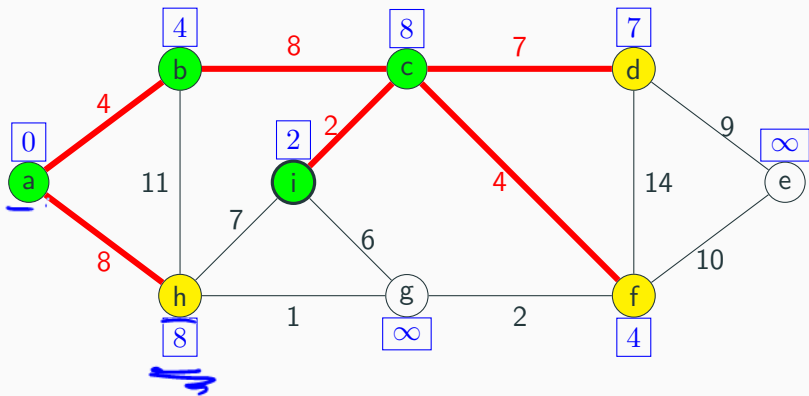
We (arbitrarily) pick c.

Prim's algorithm: an example



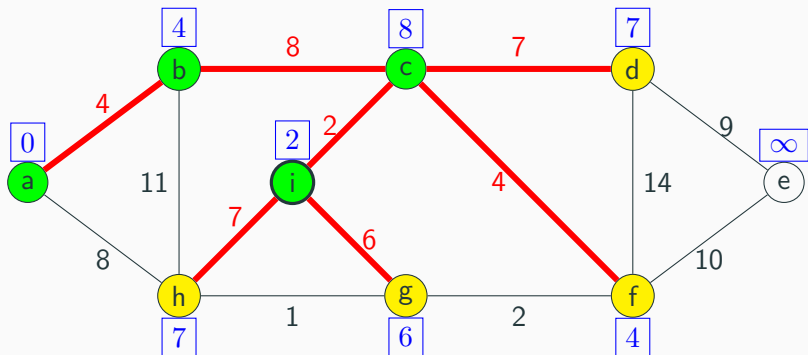
...and update the adjacent nodes. Note that we don't add the cumulative cost: the cost represents the shortest path to *any* green node, not to the start.

Prim's algorithm: an example



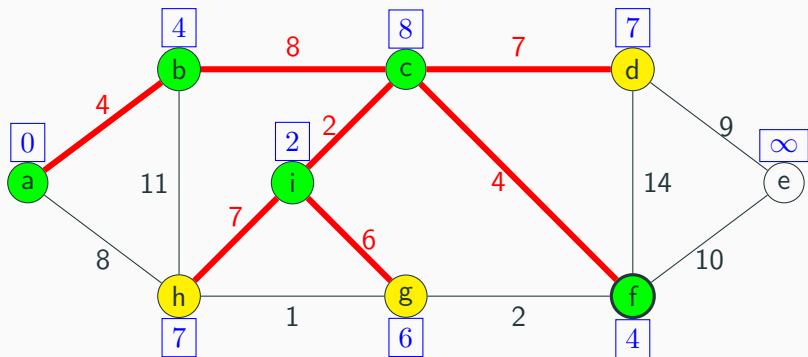
i has the smallest cost.

Prim's algorithm: an example



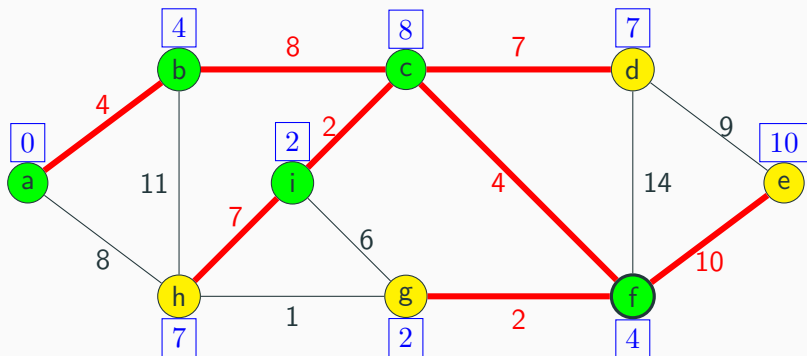
We update both unvisited nodes, and modify the edge to h since we now have a better option.

Prim's algorithm: an example



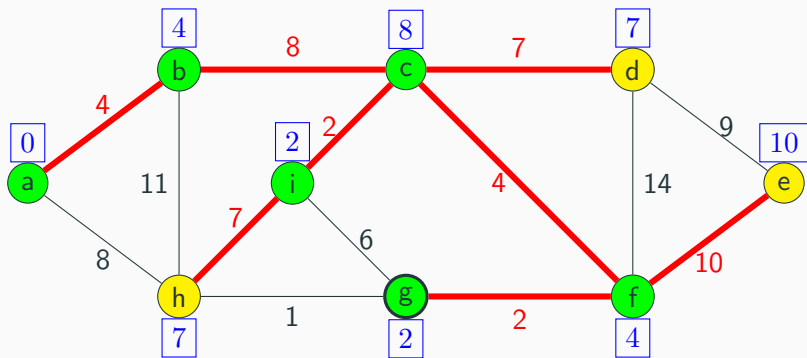
f has the smallest cost.

Prim's algorithm: an example



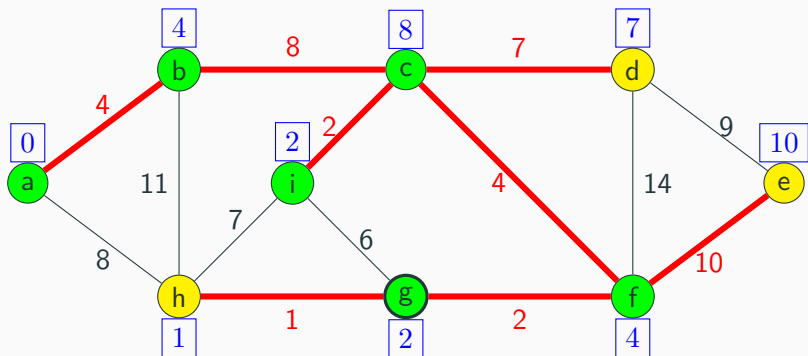
Again, we update the adjacent unvisited nodes.

Prim's algorithm: an example



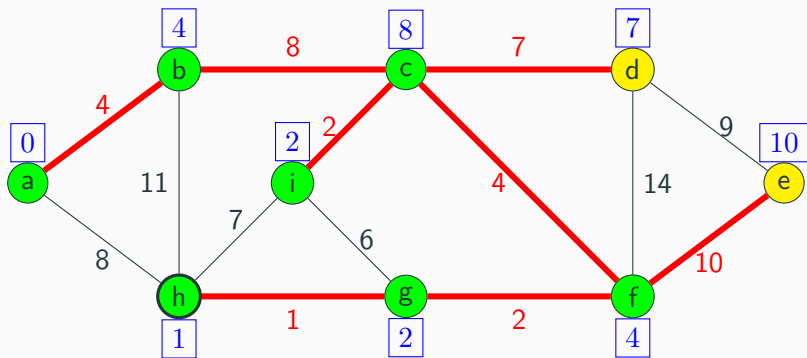
g has the smallest cost.

Prim's algorithm: an example



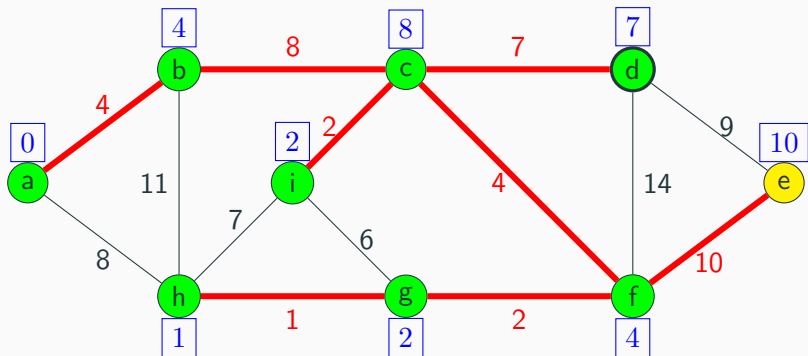
We update h again.

Prim's algorithm: an example



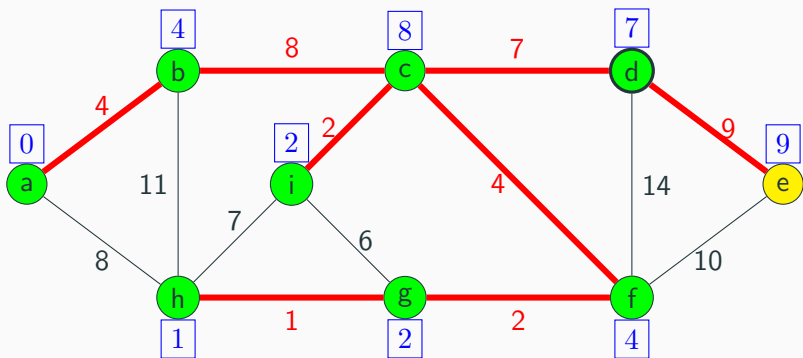
h has the smallest cost. Note that there nothing to update here.

Prim's algorithm: an example



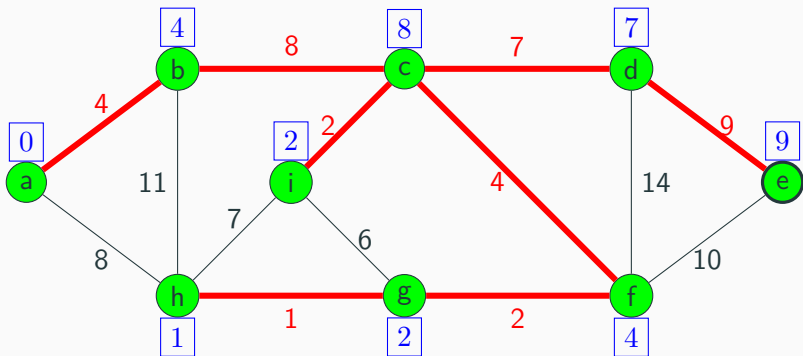
d has the smallest cost.

Prim's algorithm: an example



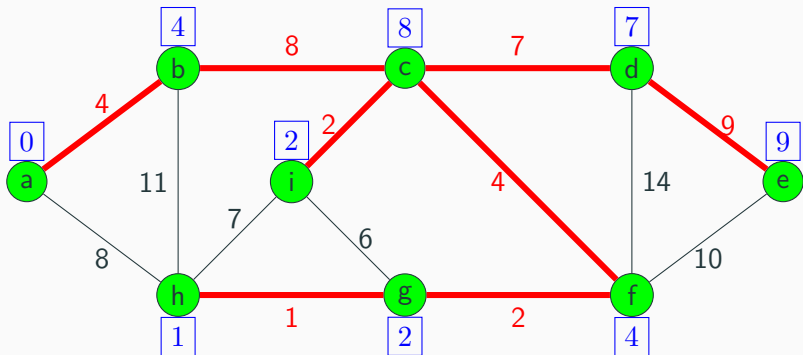
We can update e.

Prim's algorithm: an example



e has the smallest cost.

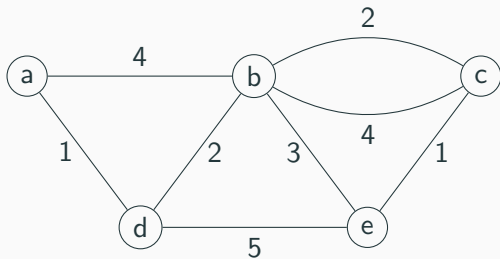
Prim's algorithm: an example



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

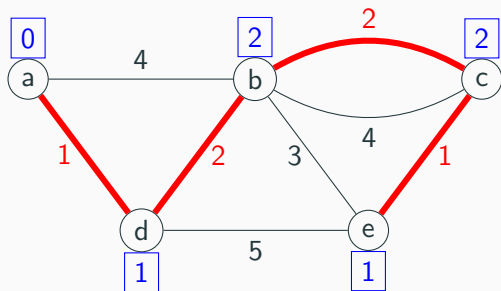
Prim's algorithm: another example

Now you try. Start on node *a*.



Prim's algorithm: another example

Now you try. Start on node *a*.



Analyzing Prim's algorithm

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

Analyzing Prim's algorithm

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

Answer: The same as Dijkstra's: $\mathcal{O}(|V|t_s + |E|t_u)$ where...

- ▶ t_s = time needed to get next smallest node
- ▶ t_u = time needed to update vertex costs

Analyzing Prim's algorithm

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

Answer: The same as Dijkstra's: $\mathcal{O}(|V|t_s + |E|t_u)$ where...

- ▶ t_s = time needed to get next smallest node
- ▶ t_u = time needed to update vertex costs

So, $\mathcal{O}(|V| \log(|V|) + |E| \log(|V|))$ if we stick to data structures we know how to implement; $\mathcal{O}(|V| \log(|V|) + |E|)$ if we use Fibonacci heaps.

Minimum spanning trees, approach 2

Recap: Prim's algorithm works similarly to Dijkstra's – we start with a single node, and “grow” our MST.

Minimum spanning trees, approach 2

Recap: Prim's algorithm works similarly to Dijkstra's – we start with a single node, and “grow” our MST.

A second approach: instead of “growing” our MST, we...

- ▶ Initially place each node into its own MST of size 1 – so, we start with $|V|$ MSTs in total.

Minimum spanning trees, approach 2

Recap: Prim's algorithm works similarly to Dijkstra's – we start with a single node, and “grow” our MST.

A second approach: instead of “growing” our MST, we...

- ▶ Initially place each node into its own MST of size 1 – so, we start with $|V|$ MSTs in total.
- ▶ Steadily combine together different MSTs until we have just one left

Minimum spanning trees, approach 2

Recap: Prim's algorithm works similarly to Dijkstra's – we start with a single node, and “grow” our MST.

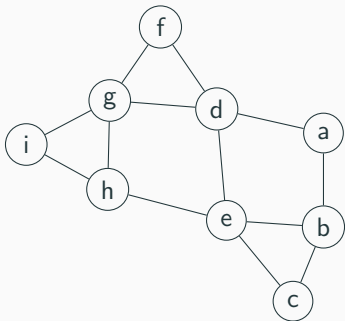
A second approach: instead of “growing” our MST, we...

- ▶ Initially place each node into its own MST of size 1 – so, we start with $|V|$ MSTs in total.
- ▶ Steadily combine together different MSTs until we have just one left
- ▶ How? Loop through every single edge, see if we can use it to join two different MSTs together.

This algorithm is called **Kruskal's algorithm**

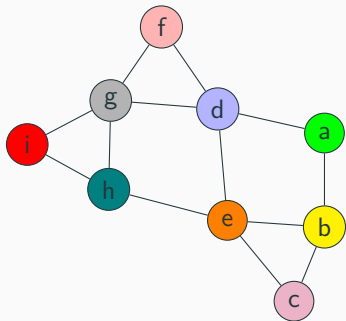
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



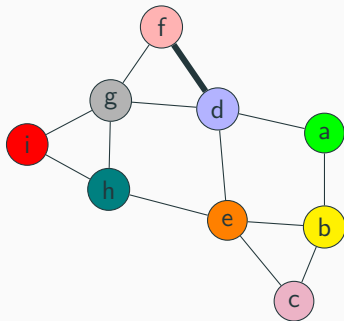
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



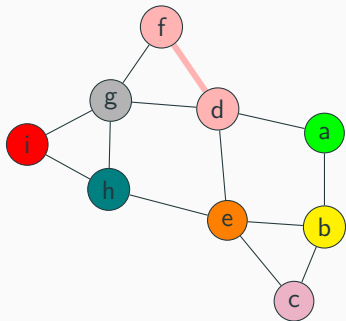
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



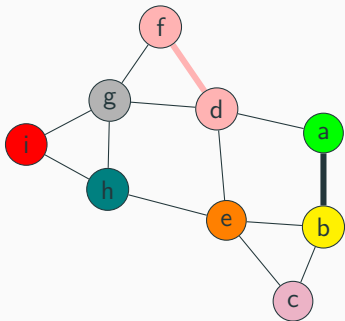
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



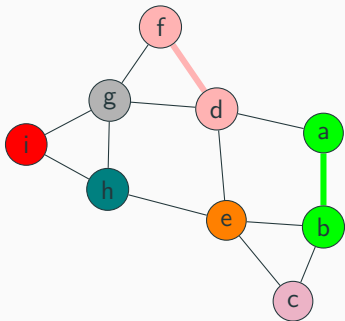
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



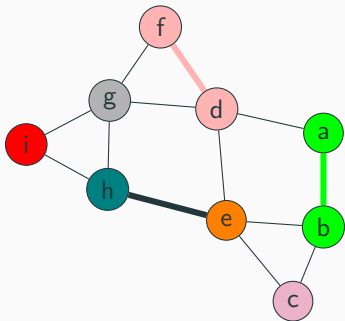
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



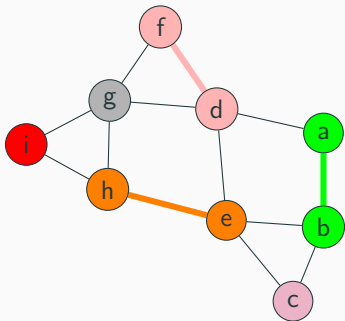
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



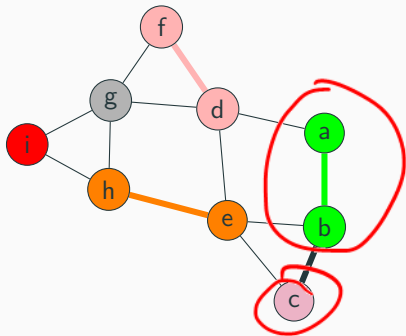
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



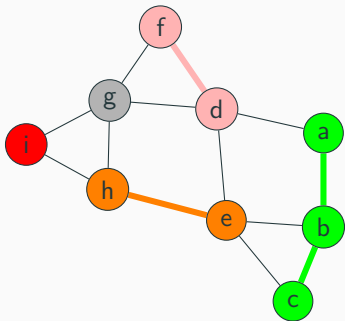
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



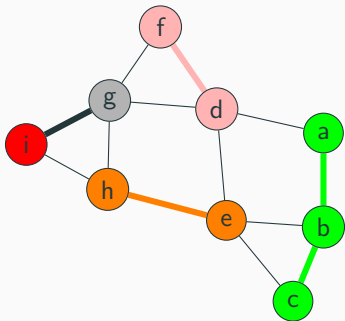
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



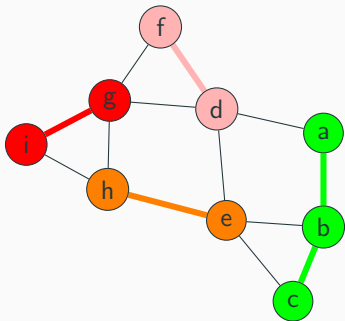
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



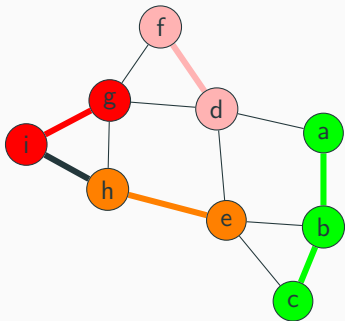
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



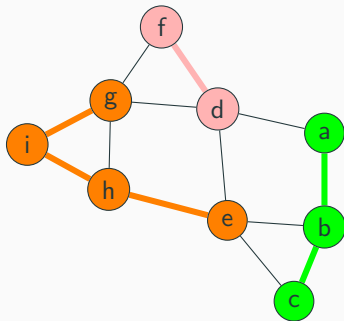
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



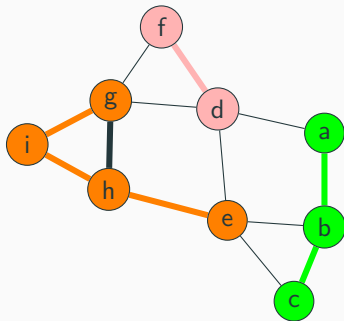
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



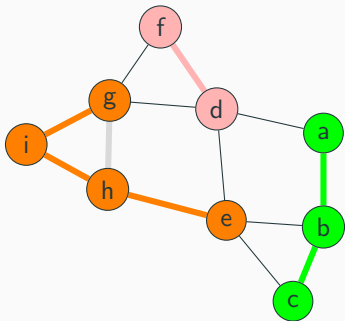
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



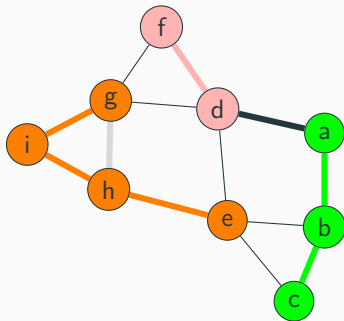
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



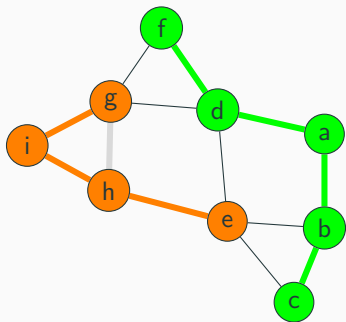
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



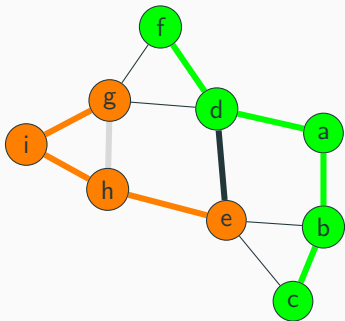
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



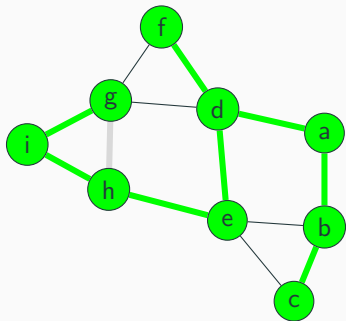
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



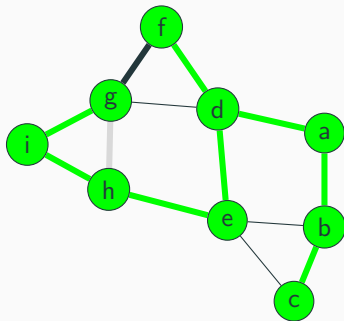
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



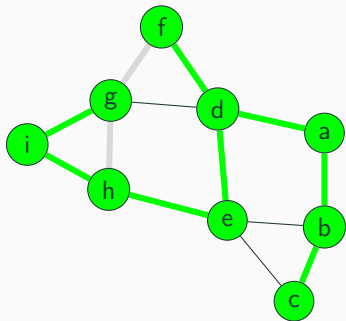
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



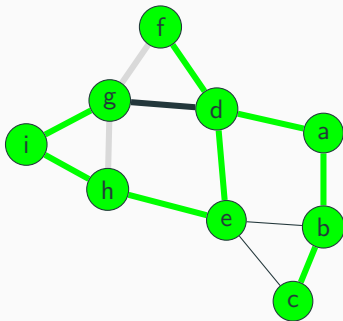
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



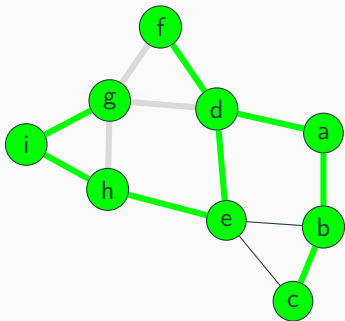
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



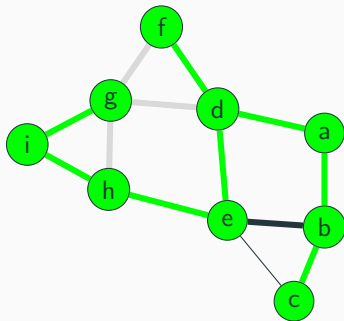
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



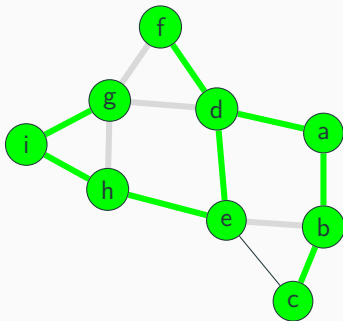
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



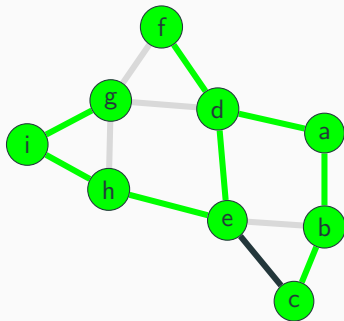
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



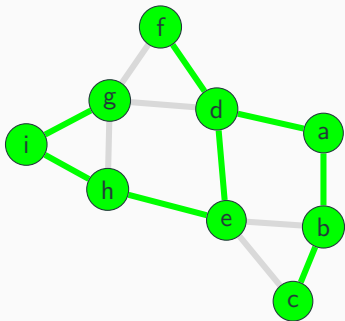
Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



Kruskal's algorithm

An example, for unweighted graphs. Note: each MST has a different color.



Kruskal's algorithm: weighted graphs

Question: How do we handle edge weights?

Kruskal's algorithm: weighted graphs

Question: How do we handle edge weights?

Answer: Consider edges sorted in ascending order by weight.

So, we look at the edge with the smallest weight first, the edge with the second smallest weight next, etc.

Kruskal's algorithm: pseudocode

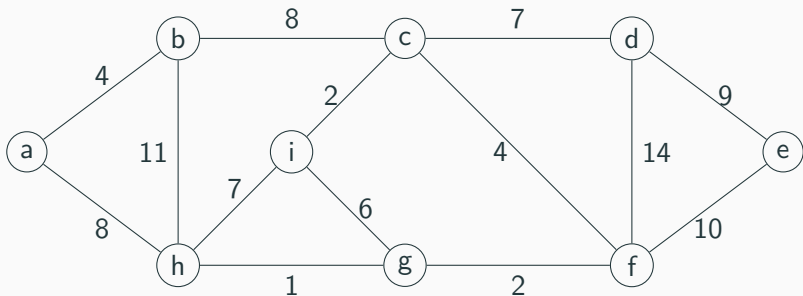
Pseudocode for Kruskal's algorithm:

```
def kruskal():  
    mst = new SomeSet<Edge>()  
    for (v : vertices):  
        makeMST(v)  
    sort edges in ascending order by their weight  
    for (edge : edges):  
        if findMST(edge.src) != findMST(edge.dst):  
            union union(edge.src, edge.dst)  
            mst.add(edge)  
    return mst
```

- ▶ makeMST(v): stores v as a MST containing just one node
- ▶ findMST(v): finds the MST that vertex is a part of
- ▶ union(u, v): combines the two MSTs of the two given vertices, using the edge (u, v)

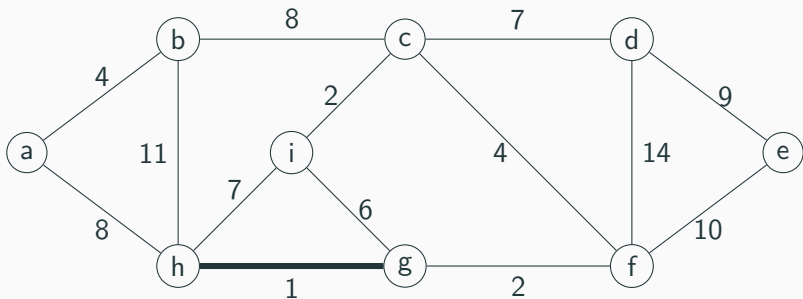
Kruskal's algorithm: example with a weighted graph

Now you try:



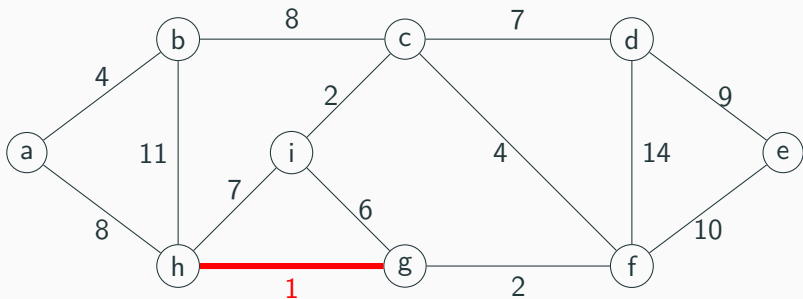
Kruskal's algorithm: example with a weighted graph

Now you try:



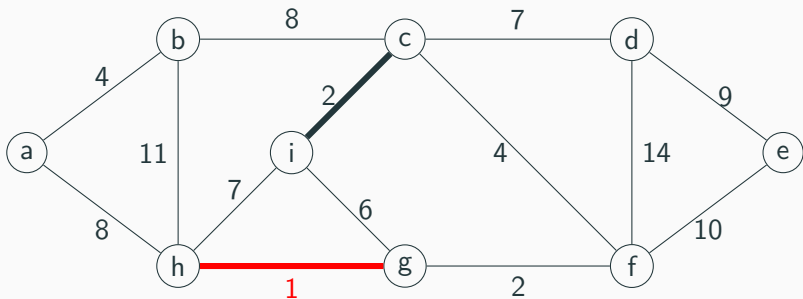
Kruskal's algorithm: example with a weighted graph

Now you try:



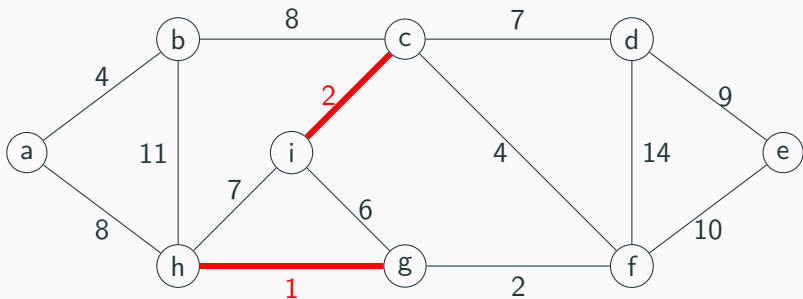
Kruskal's algorithm: example with a weighted graph

Now you try:



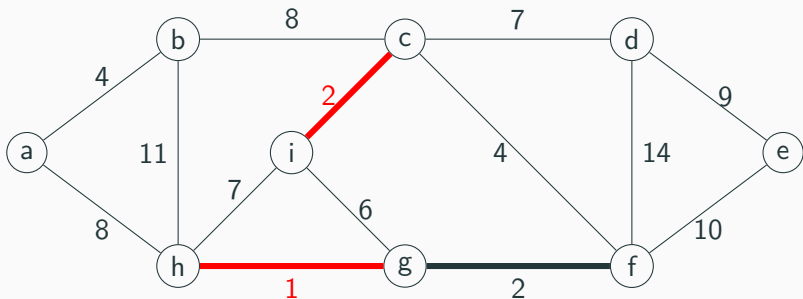
Kruskal's algorithm: example with a weighted graph

Now you try:



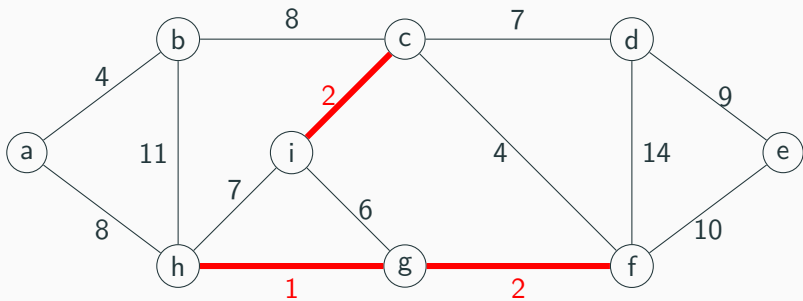
Kruskal's algorithm: example with a weighted graph

Now you try:



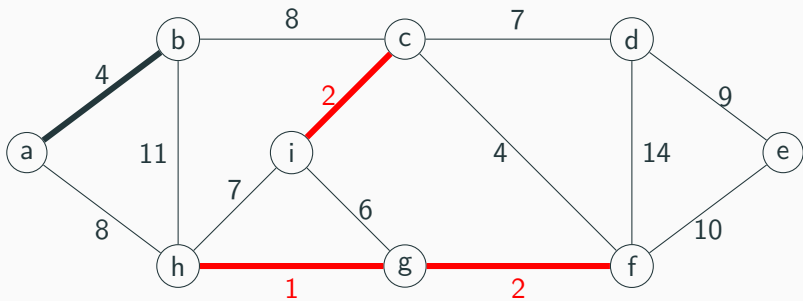
Kruskal's algorithm: example with a weighted graph

Now you try:



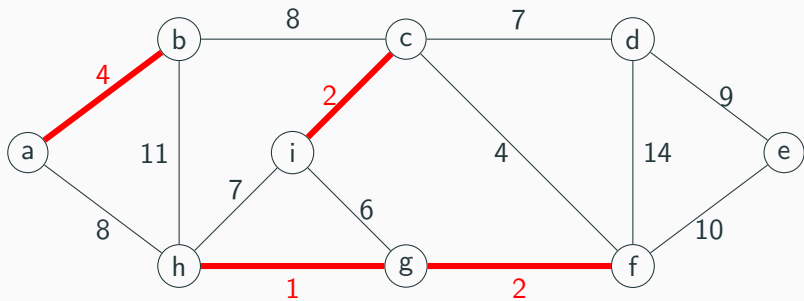
Kruskal's algorithm: example with a weighted graph

Now you try:



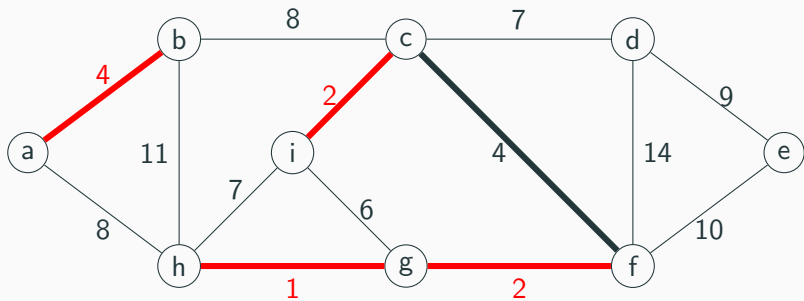
Kruskal's algorithm: example with a weighted graph

Now you try:



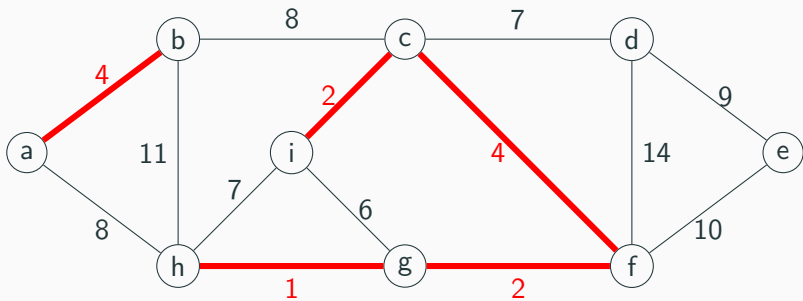
Kruskal's algorithm: example with a weighted graph

Now you try:



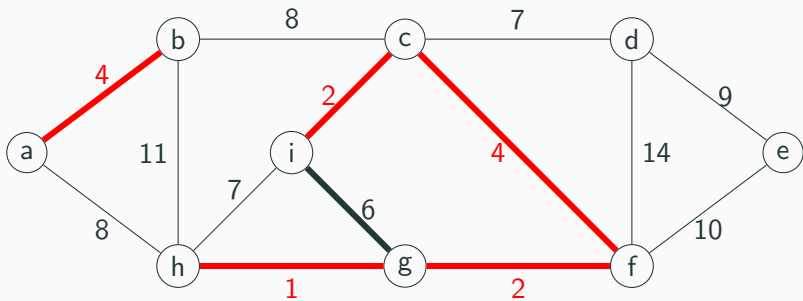
Kruskal's algorithm: example with a weighted graph

Now you try:



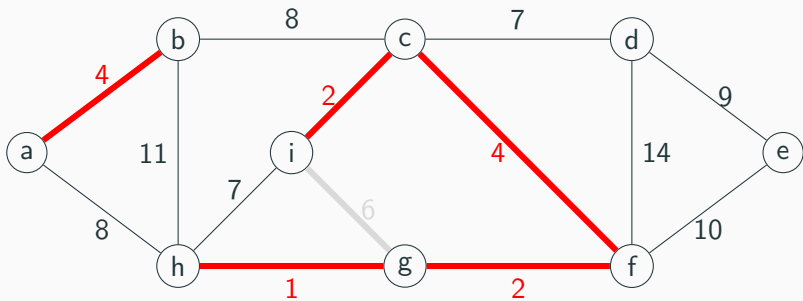
Kruskal's algorithm: example with a weighted graph

Now you try:



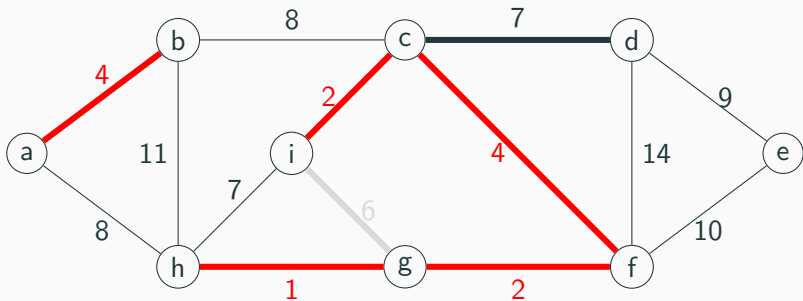
Kruskal's algorithm: example with a weighted graph

Now you try:



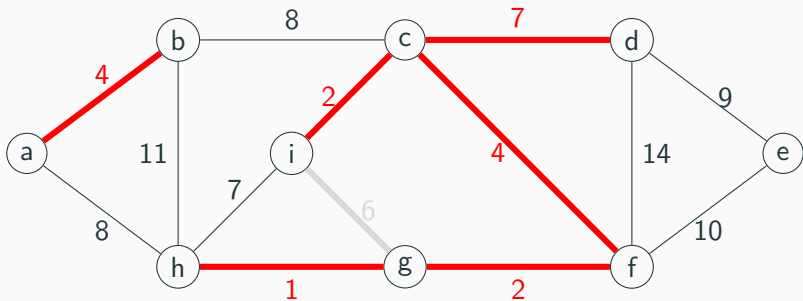
Kruskal's algorithm: example with a weighted graph

Now you try:



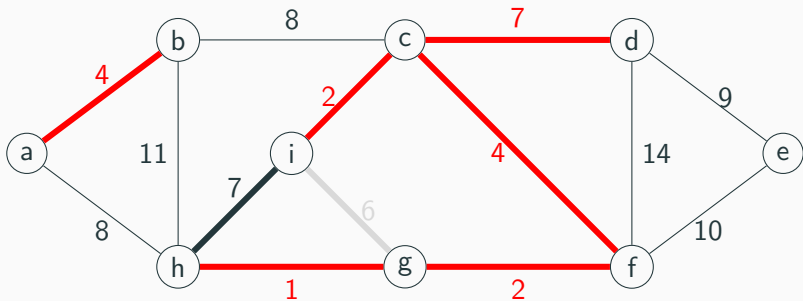
Kruskal's algorithm: example with a weighted graph

Now you try:



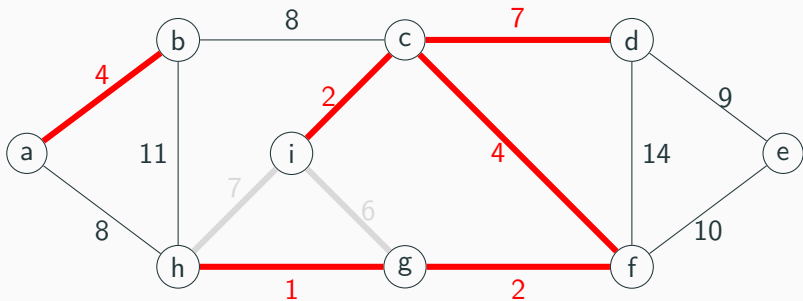
Kruskal's algorithm: example with a weighted graph

Now you try:



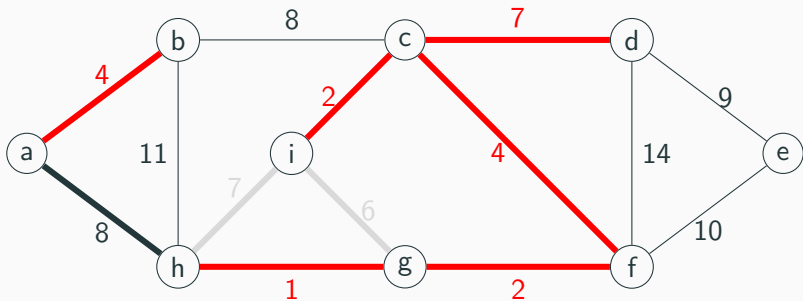
Kruskal's algorithm: example with a weighted graph

Now you try:



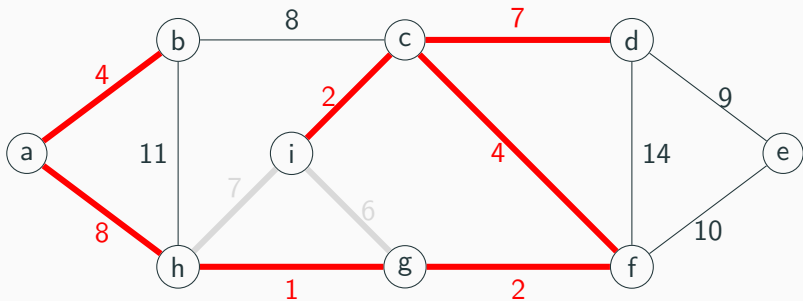
Kruskal's algorithm: example with a weighted graph

Now you try:



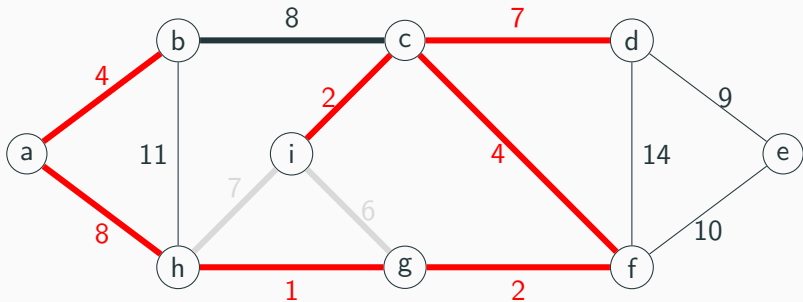
Kruskal's algorithm: example with a weighted graph

Now you try:



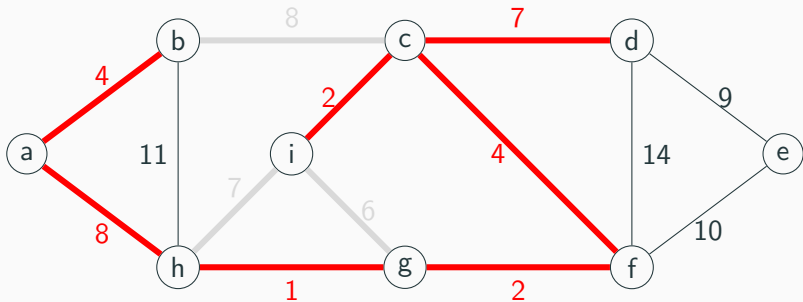
Kruskal's algorithm: example with a weighted graph

Now you try:



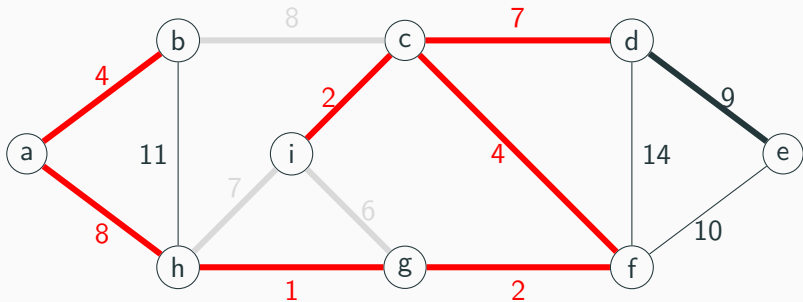
Kruskal's algorithm: example with a weighted graph

Now you try:



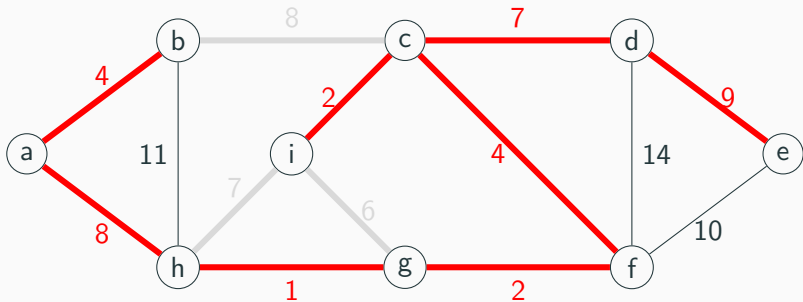
Kruskal's algorithm: example with a weighted graph

Now you try:



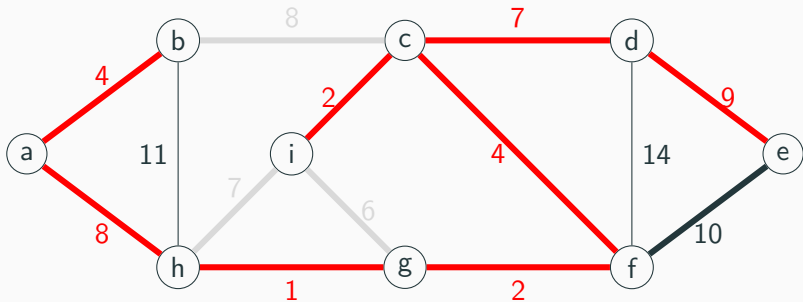
Kruskal's algorithm: example with a weighted graph

Now you try:



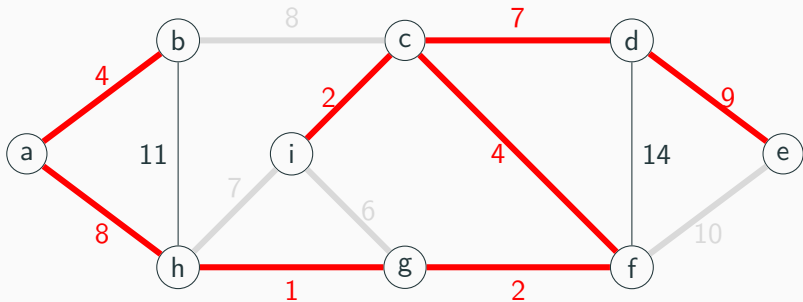
Kruskal's algorithm: example with a weighted graph

Now you try:



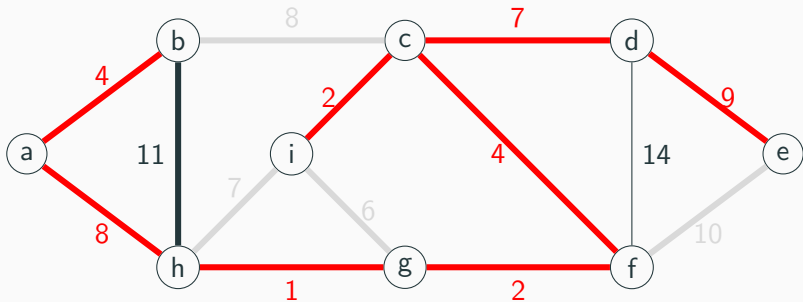
Kruskal's algorithm: example with a weighted graph

Now you try:



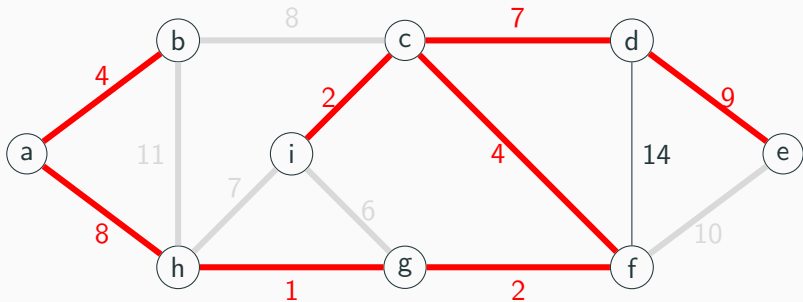
Kruskal's algorithm: example with a weighted graph

Now you try:



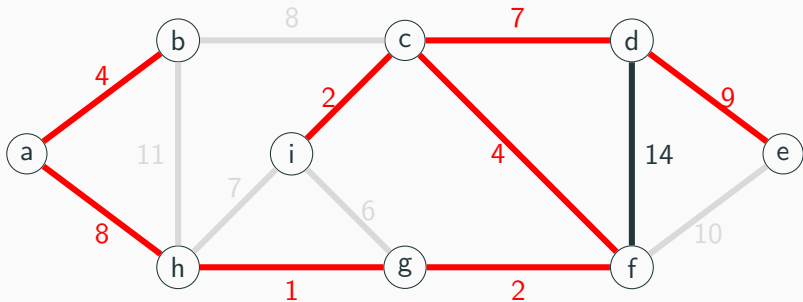
Kruskal's algorithm: example with a weighted graph

Now you try:



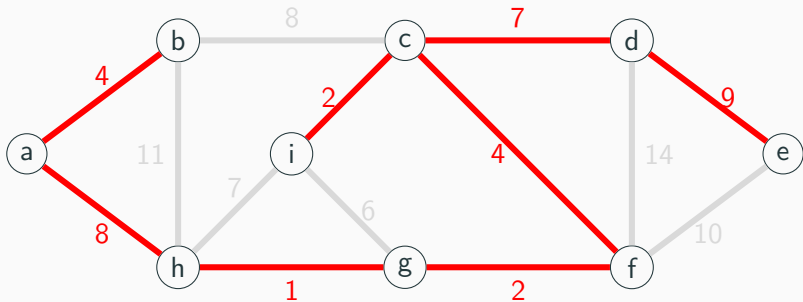
Kruskal's algorithm: example with a weighted graph

Now you try:



Kruskal's algorithm: example with a weighted graph

Now you try:



Kruskal's algorithm: analysis

What is the worst-case runtime?

$n \log(n)$

```
def kruskal():  
    mst = new SomeSet<Edge>()
```

```
    for (v : vertices):  
        makeMST(v)
```

$\mathcal{O}(|V| \cdot t_m)$

```
    sort edges in ascending order by their weight
```

$\mathcal{O}(|E| \cdot \log |E|)$

```
    for (edge : edges):
```

```
        if findMST(edge.src) != findMST(edge.dst):
```

```
            union(edge.src, edge.dst)  
            mst.add(edge)
```

```
    return mst
```



$\mathcal{O}(|V| - 1)$

$\mathcal{O}((1 + t_u) |E|)$

$t_f + t_u$

Note: assume that...

- ▶ makeMST(v) takes $\mathcal{O}(t_m)$ time
- ▶ findMST(v): takes $\mathcal{O}(t_f)$ time
- ▶ union(u, v): takes $\mathcal{O}(t_u)$ time

$\mathcal{O}(|V| \cdot t_v + |E| \cdot t_f)$

Kruskal's algorithm: analysis

- ▶ Making the $|V|$ MSTs takes $\mathcal{O}(|V| \cdot t_m)$ time
- ▶ Sorting the edges takes $\mathcal{O}(|E| \cdot \log(|E|))$ time, assuming we use a general-purpose comparison sort
- ▶ The final loop takes $\mathcal{O}(|E| \cdot t_f + |V| \cdot t_u)$ time

Kruskal's algorithm: analysis

- ▶ Making the $|V|$ MSTs takes $\mathcal{O}(|V| \cdot t_m)$ time
- ▶ Sorting the edges takes $\mathcal{O}(|E| \cdot \log(|E|))$ time, assuming we use a general-purpose comparison sort
- ▶ The final loop takes $\mathcal{O}(|E| \cdot t_f + |V| \cdot t_u)$ time

Putting it all together:

$$\mathcal{O}(|V| \cdot t_m + |E| \cdot \log(|E|) + |E| \cdot t_f + |V| \cdot t_u)$$

But wait, what exactly is t_m , t_f , and t_u ? How exactly do we implement `makeMST(v)`, `findMST(v)`, and `union(u, v)`?

The DisjointSet ADT

But wait, what exactly is t_m , t_f , and t_u ? How exactly do we implement `makeMST(v)`, `findMST(v)`, and `union(u, v)`?

We can do so using a new ADT called the DisjointSet ADT!