ICSI 403: DESIGN AND ANALYSIS OF ALGORITHMS

Spanning Trees – Chapter 23

J Marques de Carvalho

Background – Greedy Algorithms

- “Greedy” algorithms tend to be very simple.
- They make ***locally-optimized*** decisions
  - i.e., they make the most advantageous choice at any given point, without regard for any possible past or future knowledge about the problem.
- As such, the most appealing choice at the moment ***may*** turn out, in the long run, to be a poor choice
- Greedy algorithms never back up and “change their mind” based on what they later learn

Greedy Algorithms - Example

- The Huffman Tree-building algorithm is a greedy algorithm
  - It always picks the two remaining items with the lowest weights to select next for merging into a new subtree
  - In the case of Huffman, the greedy approach does happen to build the optimal encoding tree (as Huffman proved)

Greedy Algorithms – Another Example

- Counting change using standard US coins, with the constraint of using as few coins as possible
  - Coins: half-dollar, quarter, dime, nickel, penny
  - Being greedy, and wanting to use as few coins as possible, we'll always try to use the largest coin we can.

Greedy Algorithms - Example

- Example: Count out 36 cents in change

  - Step 1: The largest coin is half-dollar; too big
  - Step 2: Next-largest coin is quarter; USE ONE
  - Step 3: We still have 11 cents to count out. The half-dollar and quarter are too big; USE A DIME
  - Step 4: The half-dollar, quarter, dime, and nickel are too big; USE A PENNY.
  - Total change = 36 cents; we're done with 3 coins

  - This happens to be the optimal solution.

Greedy Algorithms – Another Example

- Example: Count out 16 cents worth of change, but assume there's a 12-cent coin in addition to the standard denominations:
  - Step 1: The half-dollar and quarter are too big
  - Step 2: Next-largest coin is the 12-cent; USE ONE
  - Step 3: We still have 4 cents to count out. We wind up using FOUR pennies
  - Total change = 16 cents; we're done with 5 coins

  - This is NOT the optimal solution (dime, nickel, penny takes two fewer coins).
    - The decision to use the 12-cent coin looked good at the time but backed us into a non-optimal solution.

Yet Another Greedy Algorithm

- The Discrete (0-1) Knapsack problem:
  - A thief with a knapsack is breaking into a jewelry store
  - If the weight in the knapsack exceeds some threshold, the sack will break, and the thief gets nothing
  - The thief wants to maximize the value of his loot
  - If the jewelry store contains n items from which the thief can choose, there are 2^n combinations to consider: the thief may grow old while planning the heist!
  - The thief can't take part of an item; it's all-or-nothing

The Discrete Knapsack Problem (2)

- One obvious strategy: Take the item with the largest value first
  - The problem: The item with the largest value might also have a correspondingly high weight
  - Suppose we have three items, valued at $10, $9, and $9, with corresponding weights of 25, 10, and 10 pounds. Further, suppose the knapsack’s weight limit is 30 pounds
  - If we start with the most valuable item ($10 and 25 pounds), we can’t take another item, so we wind up with $10 in loot.
  - The optimal solution would be to take the two $9 (10-pound) items, for a total of $18 (and 20 lbs).

The Discrete Knapsack Problem (3)

- Another obvious strategy: Take the item with the smallest weight first
  - The problem: If the items that are light in weight have small values compared with their weights

The Discrete Knapsack Problem (4)

- A more sophisticated strategy: Take the item with the highest value/weight ratio first.
  - Suppose we have three items:
    - Item 1: $50 / 5 lbs = $10/lb
    - Item 2: $60 / 10 lbs = $6/lb
    - Item 3: $140 / 20 lbs = $7/lb
  - If we take the items in order of value per weight, we would take items 1 & 3 ($50 & $140, with weights of 5 & 20 lbs), for a total of $190 / 25 lbs
  - But the optimal solution (which maximizes the total value) is to take items 2 & 3 ($60 & $140, with weights of 10 & 20 lbs), for a total of $200 / 30 lbs.

The Discrete Knapsack Problem (5)

- The greedy approach does not work (i.e., give us an optimal solution) on the discrete knapsack problem!

The Fractional Knapsack Problem

- A variant of the discrete knapsack problem is the ***Fractional Knapsack Problem***:
  - The thief can take ***part*** of an item.
  - Think of the Discrete problem as gold/silver ***bars***
  - Think of the Fractional problem as gold/silver ***dust***
  - By being able to take ***part*** of an item, there is never any wasted space in the knapsack

The Fractional Knapsack Problem

- Consider the original three items:
  - Item 1: $50 / 5 lbs = $10/lb
  - Item 2: $60 / 10 lbs = $6/lb
  - Item 3: $140 / 20 lbs = $7/lb
- The thief would take all of item 1 ($50/5 lb), all of item 3 ($140/20 lb), and have 5 lbs of knapsack space left, with which he could take 5 lbs of item #2 ($30)
- Total haul: $50 + 140 + 30 = $220
- Total haul: 5 + 20 + 5 = 30 lbs

The Fractional Knapsack Problem

- Although the greedy approach does not necessarily provide an optimal solution to the ***discrete*** knapsack problem, it does provide an optimal solution to the ***fractional*** knapsack problem.
- Now, on to spanning trees…

Motivation

- Given: a number of towns in a remote area, where roads are ***very*** expensive (per mile) to build
- Wanted: a way of getting to/from all towns in the region (not necessarily directly)
  - Going from town A to town B by way of some other town X is perfectly acceptable – there just has to be SOME path from A to B.
- Constraint: Given the cost of road building, minimize the total length of all roads required

Motivation – Another Example

- Given: an electric circuit, in which a number of pins need to be connected together (a common point on the schematic)
- Wanted: a way of getting all such pins connected (not necessarily directly to each other), so that they're electrically equivalent
- Constraint: Minimize the total length of all connecting wire required

Spanning Trees

- We can model these problems using *spanning trees*.
- Given a connected, undirected graph $G = (V, E)$, and a weight $w(u, v)$ for each edge $(u, v) \in E$, …
- We want to find a subset of E, which we'll call T (i.e., $T \subseteq E$), such that

$$w(t) = \sum_{(u,v)\in T} w(u,v)$$

is minimized

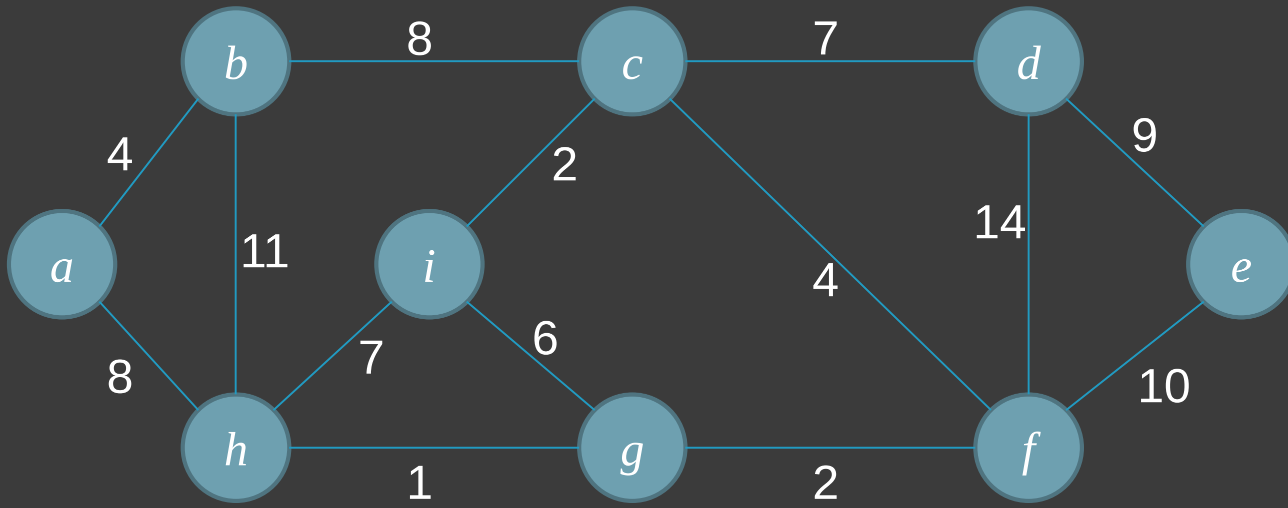
Spanning Trees

- Since T is acyclic, and it connects all of the vertices, it must form a (single) tree, which "spans" the graph, making it a *spanning tree*.
  - It contains all nodes but has no cycles.
- The problem of finding the tree T that minimizes the weights is called the ***minimum spanning tree*** problem.
  - Note that what we're minimizing is the total ***weight*** of all of the edges in the tree; not the size of the tree itself (the *number* of vertices or edges it contains). It's more accurately called the "minimum-weight spanning tree" problem.

Minimum Spanning Trees (MST)

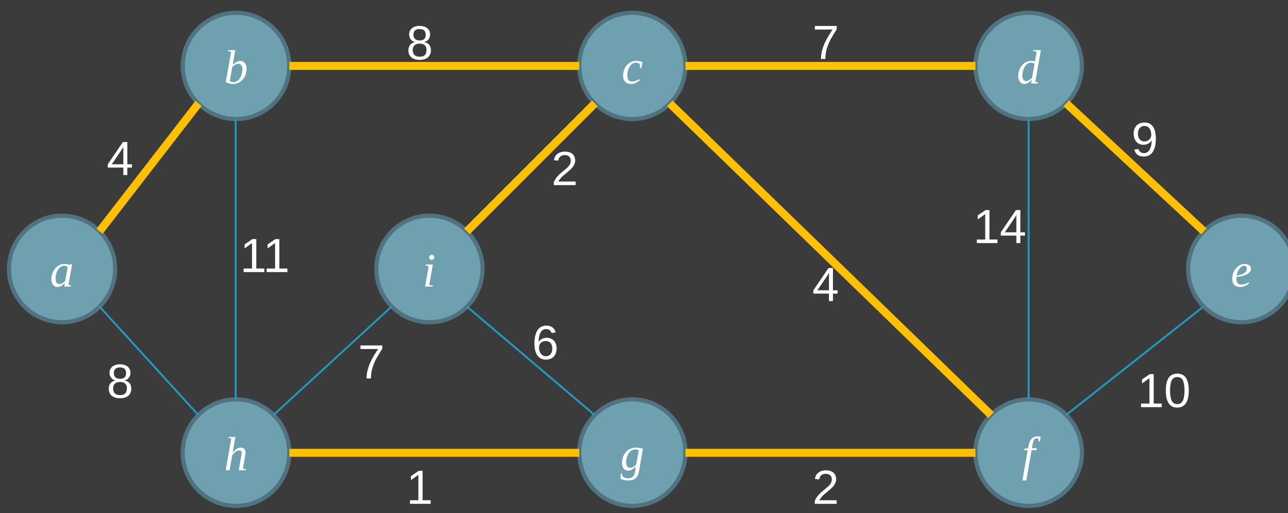
- The two algorithms we consider today, Prim's and Kruskal's, both find minimum spanning trees by using a greedy algorithm.
- We've already seen that greedy algorithms are not always guaranteed to find the optimal solutions.
- For the minimum spanning tree problem, however, just like the fractional knapsack problem, they DO always find an optimal solution!

MST Example



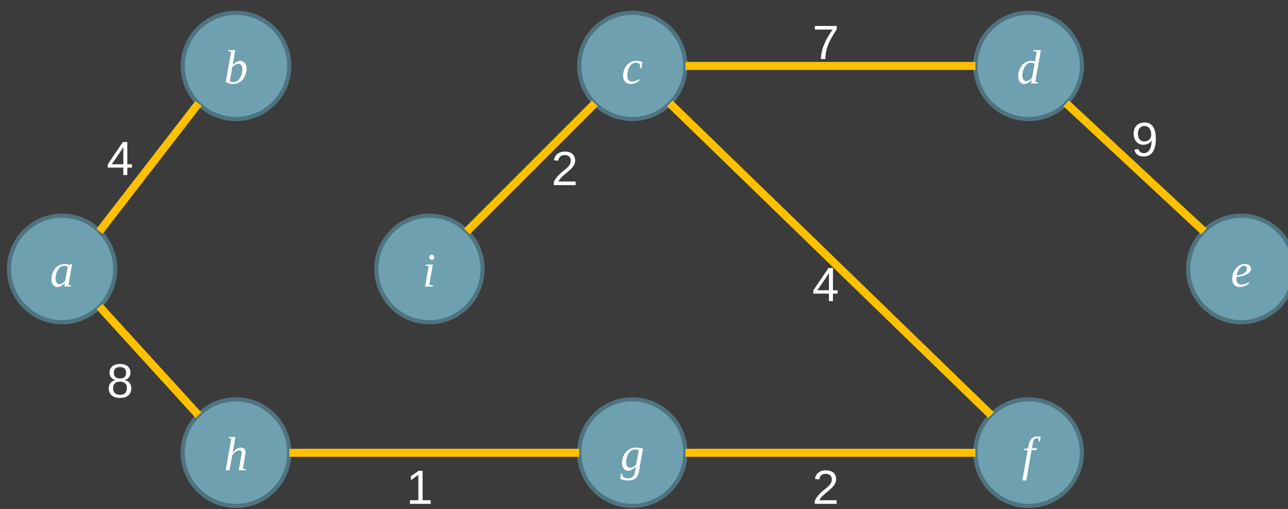
- Given this graph, with vertices $a, b, \ldots, i$, and edges as shown, with labeled weights, find the (a) minimum spanning tree.

MST Example



- You can get from any vertex to any other vertex using this tree. The sum of the weights is 37.

MST Example



- If we drop (b, c), which had a weight of 8, and replace it with (a, h), which also had a weight of 8, we still have a spanning tree with a weight of 37. "The" MST may not be unique!

MST Properties

- Some properties of an MST:
  - It has $|V| - 1$ edges.
  - It has no cycles.
  - It might not be unique

How To Grow an MST?

- Building up the solution:
  - We will build a set A of edges.
  - Initially, A has no edges.
  - As we add edges to A, maintain a loop invariant:
    - A is a subset of some MST.
  - Add only edges that maintain the invariant. If A is a subset of some MST, an edge (u, v) is safe for A if and only if $A \cup \{(u, v)\}$ is also a subset of some MST. So we will add only safe edges.

The Generic MST Algorithm

GENERIC-MST(G, w)

$A = \emptyset$

while A does not form a spanning tree

find and edge (u, v) that is safe for A

$A = A \cup \{(u, v)\}$

return A

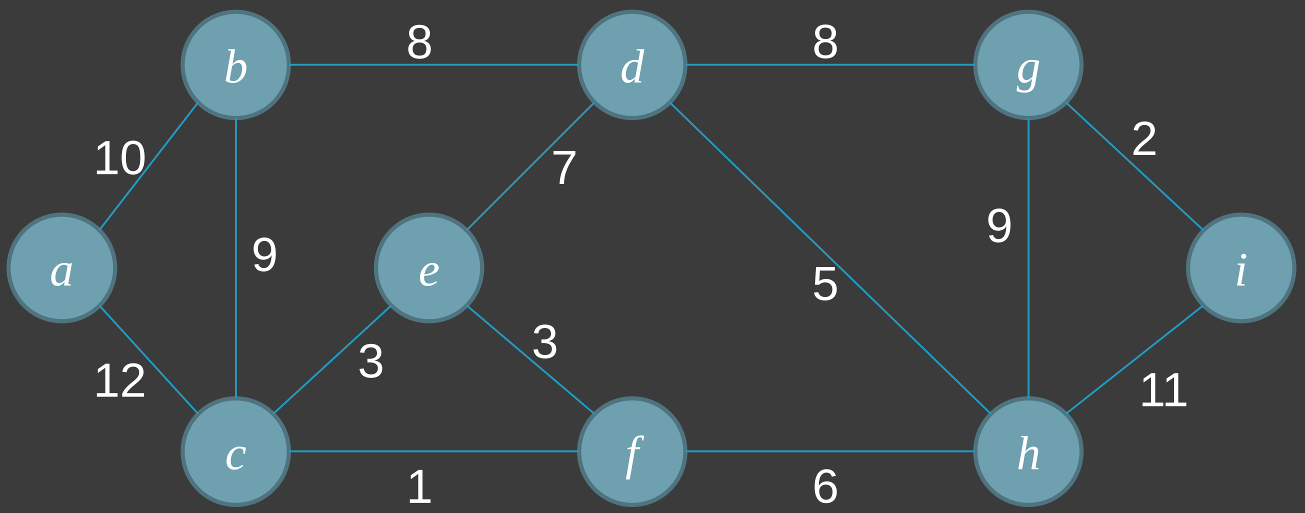
Loop Invariant and Safe Edges

- We can use the loop invariant (A is a subset of some MST) to show that this generic algorithm works.
- *Initialization:* The empty set trivially satisfies the loop invariant.
- *Maintenance:* Since we add only safe edges, A remains a subset of some MST.
- *Termination:* All edges added to A are in an MST, so when we stop, A is a spanning tree that is also an MST.

Loop Invariant and Safe Edges

- Obviously, the key to making this work lies in the details of “find an edge…that is safe for A”
- HOW?
- An example:

Loop Invariant and Safe Edges

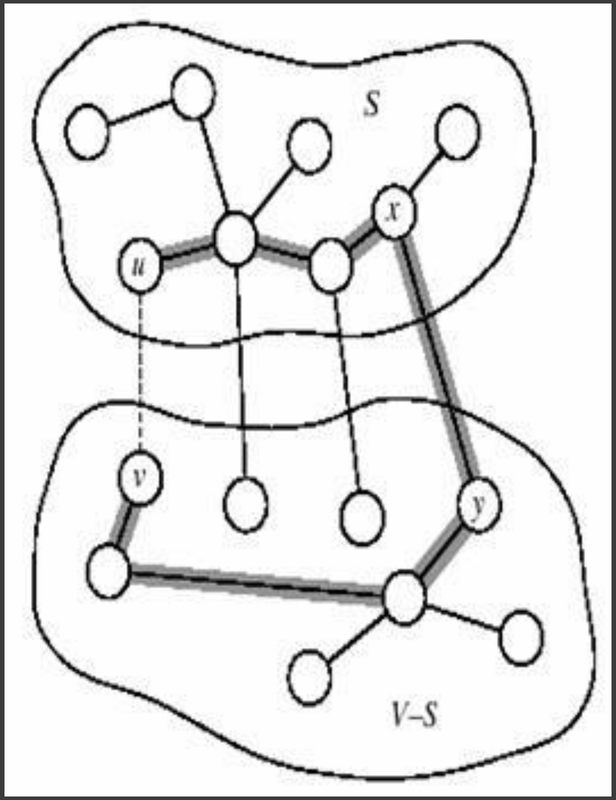


- Edge (c, f) has the lowest weight of any edge in the graph. Is this safe for $A = \emptyset$?

Safe Edges

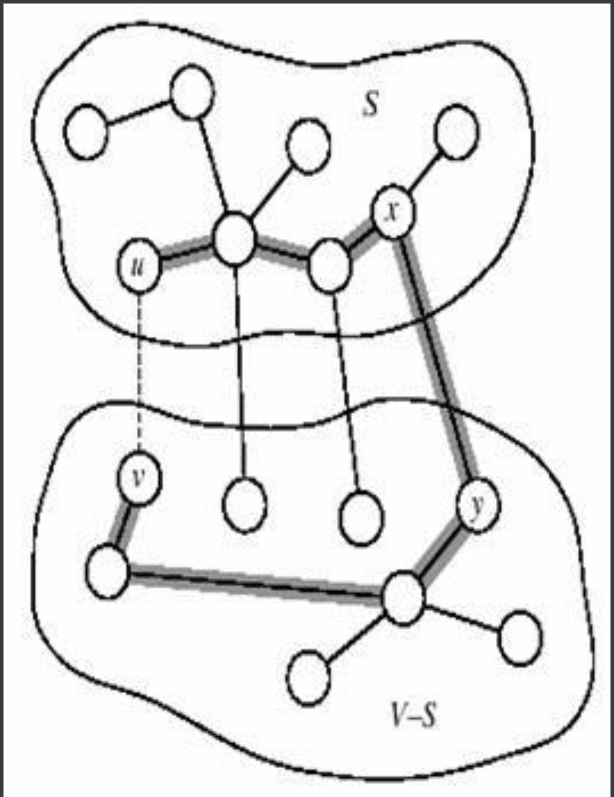
- Intuitively: Let $S \subseteq V$ be any set of vertices that includes c but not f (so that f is in $V - S$).
- In any MST, there has to be at least one edge that connects S with $V - S$.
- Why not choose the edge with minimum weight (which would be (c, f) in this case)?
- Some definitions:

Safe Edges – Definitions (1)



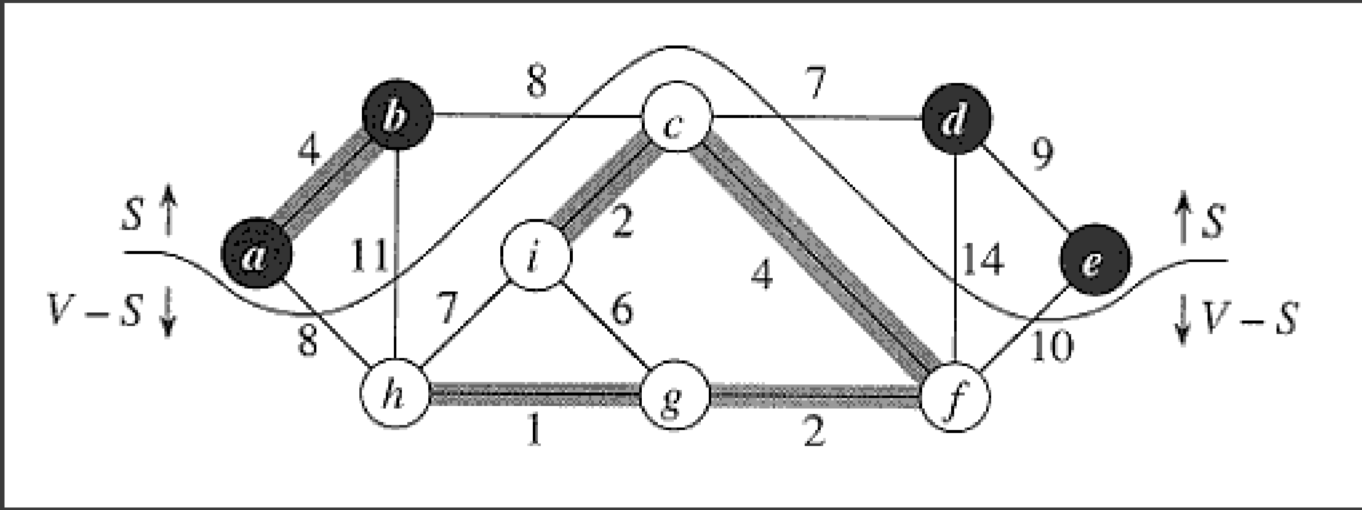
- Let $S \subseteq V$ and $A \subseteq E$
- A <u>*cut*</u> $(S, V-S)$ is a partition of vertices into disjoint sets V and $S-V$.
- Edge $(u, v) \in E$ <u>*crosses*</u> the cut $(S, V-S)$ if one endpoint is in S and the other is in $V-S$.
- A cut <u>*respects*</u> A if and only if no edge in A crosses the cut.

Safe Edges – Definitions (2)



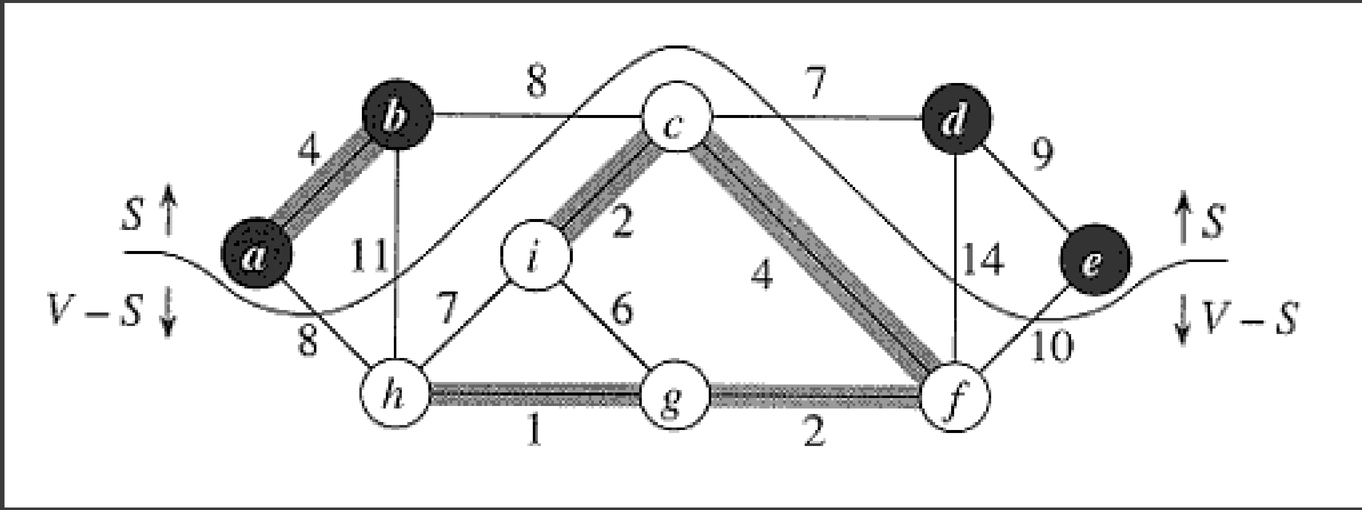
- An edge is a ***light edge*** crossing a cut if and only if its weight is minimum over all edges crossing the cut. For a given cut, there can be > 1 light edge crossing it.
- If A is shown as shaded edges, the cut does *not* respect A; it cuts the edge (x, y)

Safe Edges – Definitions (3)



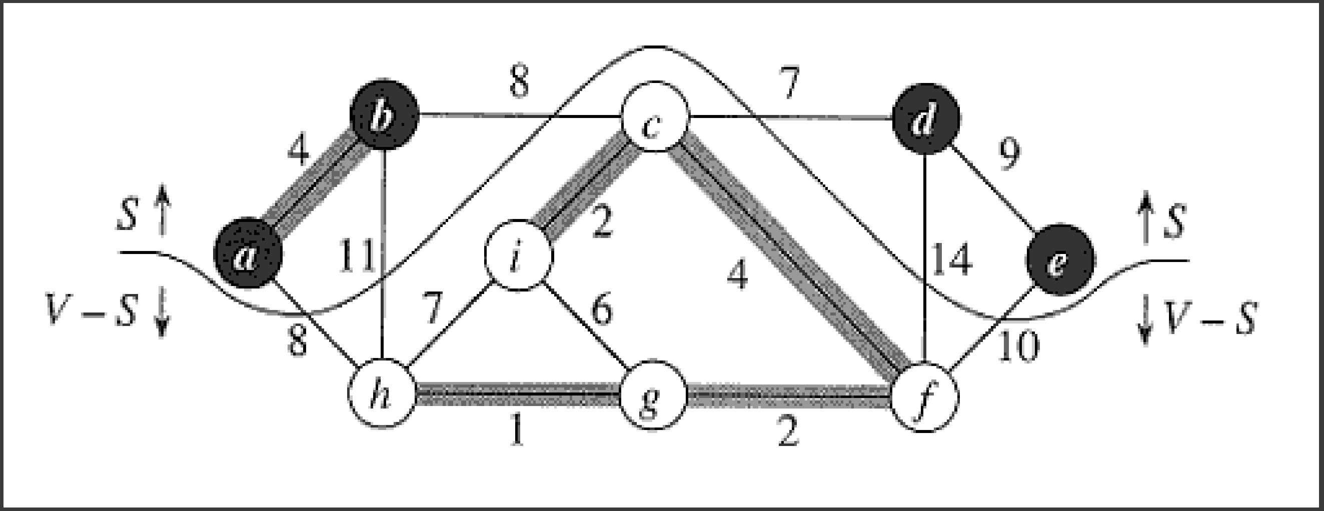
- Vertices in S are shown in black, vertices in $V - S$ are shown in white.
- Edges crossing the cut connect a black vertex to a white one.
- (d, c) is the (unique) light edge crossing the cut

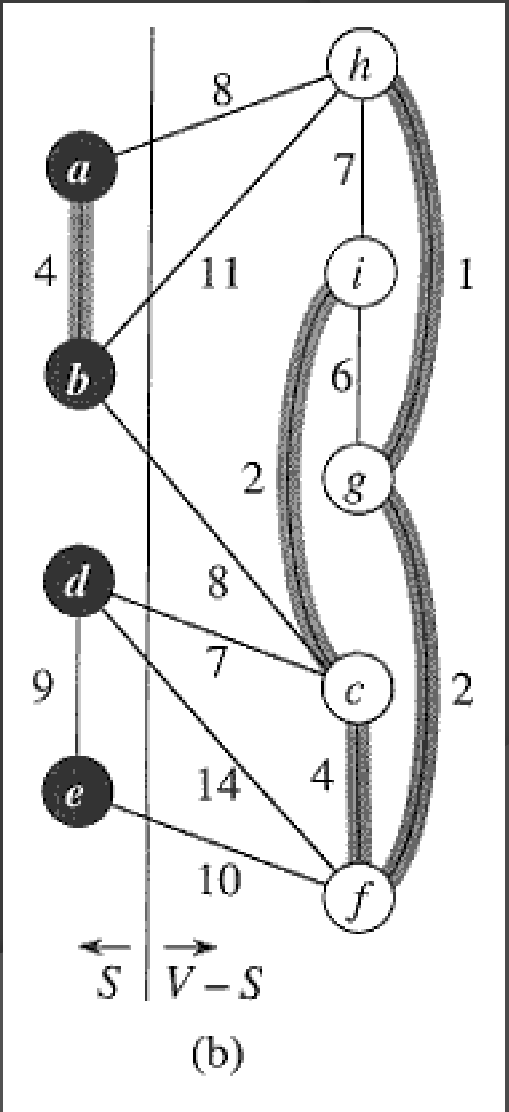
Safe Edges – Definitions (4)



- A subset A of the edges is shaded.
- Note that the cut $(S, V - S)$ respects A, since no edge of A crosses the cut

Safe Edges – Definitions (5)





- Another way of viewing this cut.
- An edge crosses the cut if it connects a vertex on the left (in S) with a vertex on the right (in $V - S$)

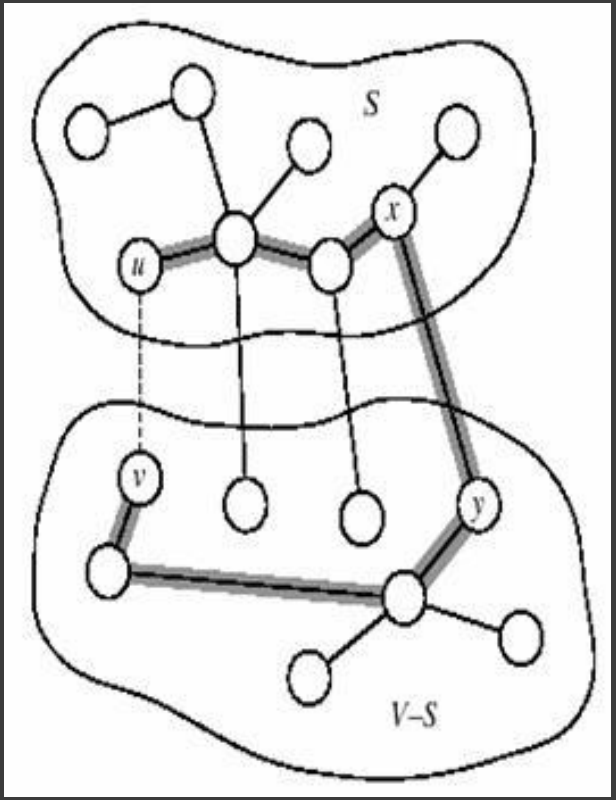
Theorem 23.1

- Let A be a subset of some MST, $(S, V - S)$ be a cut that respects A, and (u, v) be a light edge crossing $(S, V - S)$. Then (u, v) is safe for A.
- ***Proof***
  - Let T be an MST that includes A.
  - If T contains (u, v), done.
  - Assume T does *not* contain (u, v). We'll construct a different MST T that includes $A \cup \{(u, v)\}$.

Theorem 23.1 (2)

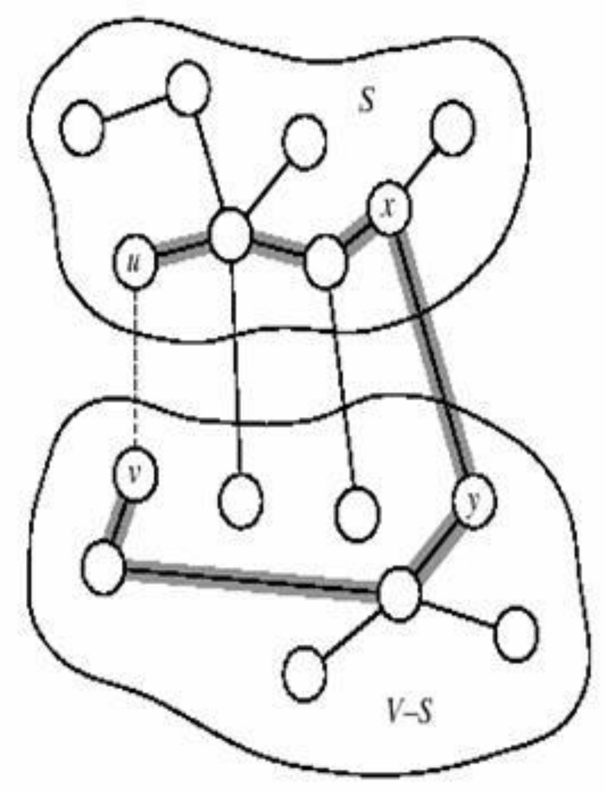
- Recall: a tree has unique path between each pair of vertices. Since T is an MST, it contains a unique path p between u and v. Path p must cross the cut $(S, V - S)$ at least once. Let (x, y) be an edge of p that crosses the cut. From how we chose (u, v), must have $w(u, v) \leq w(x, y)$.
- Since the cut respects A, edge (x, y) is not in A in the graph.

Theorem 23.1 (3)



- To form T' from T :
- Remove (x, y), which breaks T into two components.
- Add (u, v), reconnecting the trees.
- So $T' = T - \{(x, y)\} \cup \{(u, v)\}$, and T' is a spanning tree.

Theorem 23.1 (4)



- $w(T') = w(T) - w(x, y) + w(u, v) \le w(T)$, since $w(u, v) \le w(x, y)$.
- Since T' is a spanning tree, $w(T') \le w(T)$, and T is an MST, then T' must be an MST.
- Need to show that (u, v) is safe for A:
- $A \subseteq T$ and (x, y) not in A; thus, $A \subseteq T$, and $A \cup \{(u, v)\} \subseteq T'$
- Since T is an MST, (u, v) is safe for A

Kruskal's & Prim's Algorithms

- Both are elaborations of GENERIC-MST.
- They differ in how they select a safe edge in line 3 of GENERIC-MST.
- In Kruskal's algorithm, the set A is a forest (multiple trees). The safe edge we add always connects two distinct trees
- In Prim's algorithm, the set A is a single tree. The safe edge we add is always a least-weight edge connecting the tree to a vertex not yet in the tree.

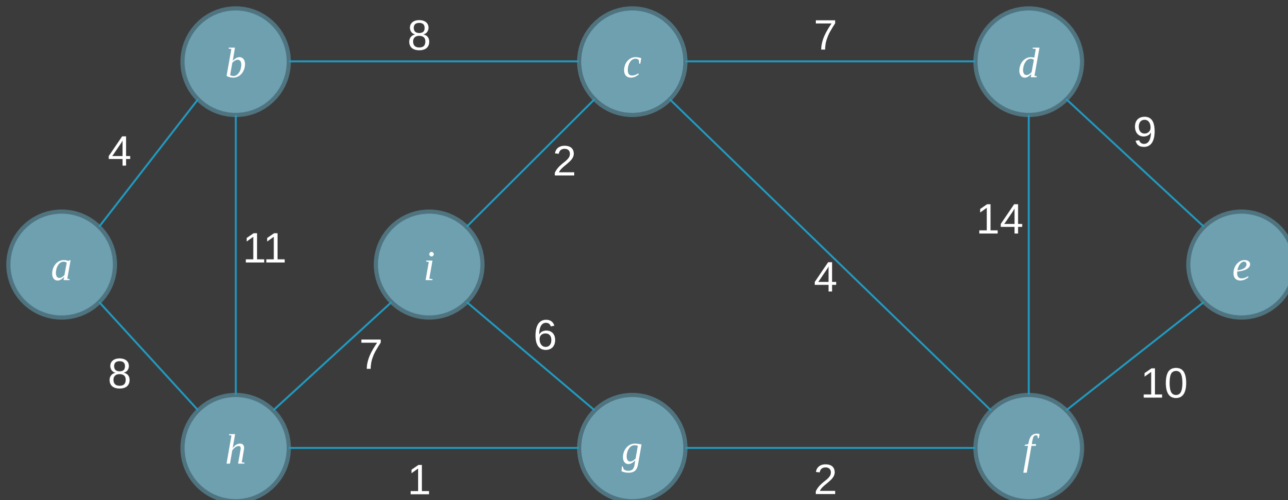
Kruskal's Algorithm

- Kruskal's algorithm depends on the existence of few set-based functions:
- MAKE-SET(x) creates a set containing the single item x.
- FIND-SET(x) looks through the sets it is maintaining, and determines which set x belongs to
- UNION(u, v) merges two sets (one containing u and the other containing v) into one set (u)

MST-Kruskal Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of E by increasing w (weight) value
for each edge (u, v) ∈ E (taken in weight order)
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

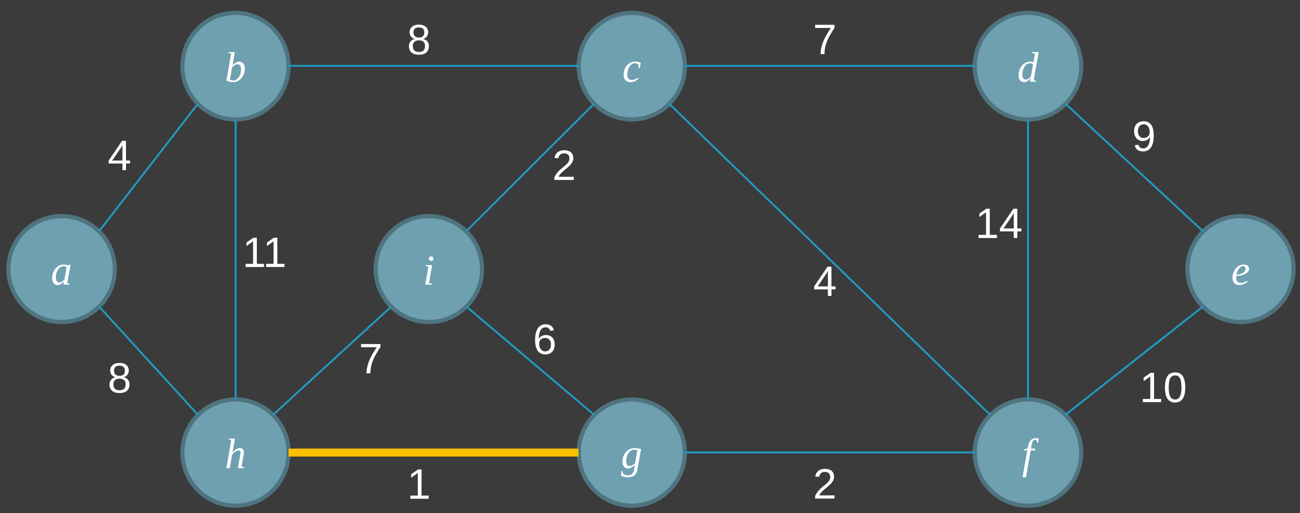
Kruskal's Algorithm



Sets:
$\{a\}$
$\{b\}$
$\{c\}$
$\{d\}$
$\{e\}$
$\{f\}$
$\{g\}$
$\{h\}$
$\{i\}$

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

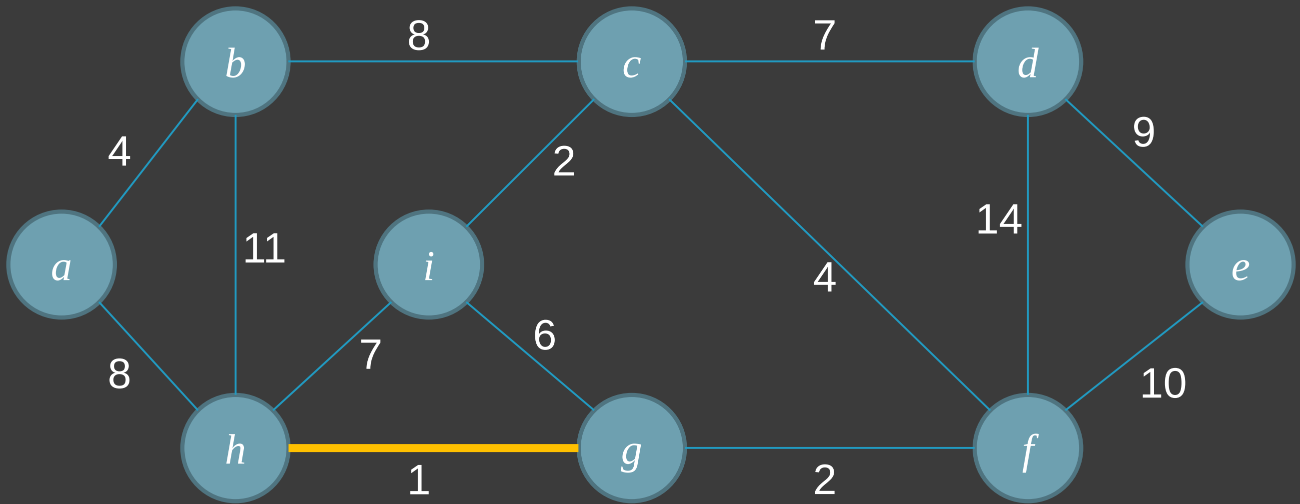


Sets:
$\{a\}$
$\{b\}$
$\{c\}$
$\{d\}$
$\{e\}$
$\{f\}$
$\{g\}$
$\{h\}$
$\{i\}$

The lowest-weight edge is (h, g), and h and g are in different sets, so include the edge and merge the two sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

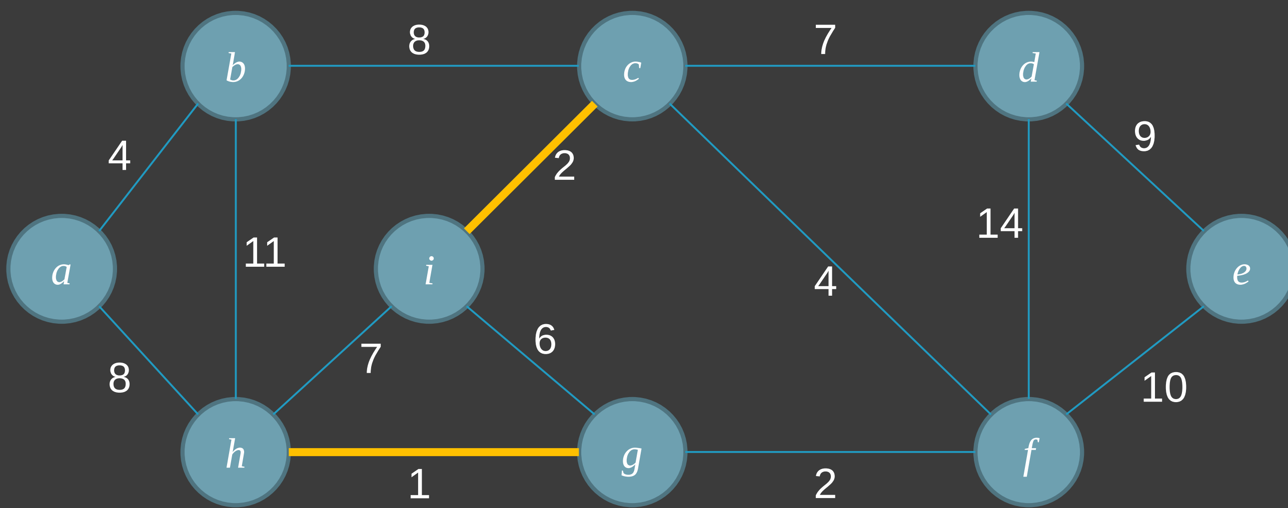


Sets:
$\{a\}$
$\{b\}$
$\{c\}$
$\{d\}$
$\{e\}$
$\{f\}$
$\{g, h\}$
$\{i\}$

The lowest-weight edges are (c, i) and (g, f). Vertices c and i are in different sets, so include the edge and merge the sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal’s Algorithm

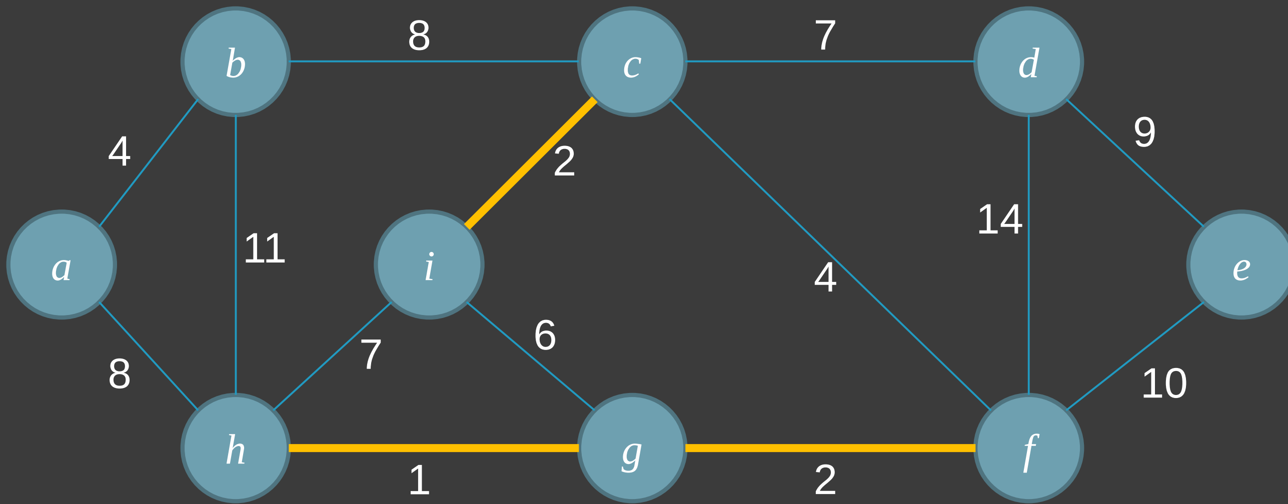


Sets:
{a}
{b}
{c, i}
{d}
{e}
{f}
{g, h}

The lowest-weight remaining edge is (g, f). Vertices g and f are in different sets, so include the edge and merge the sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the the two sets

Kruskal's Algorithm

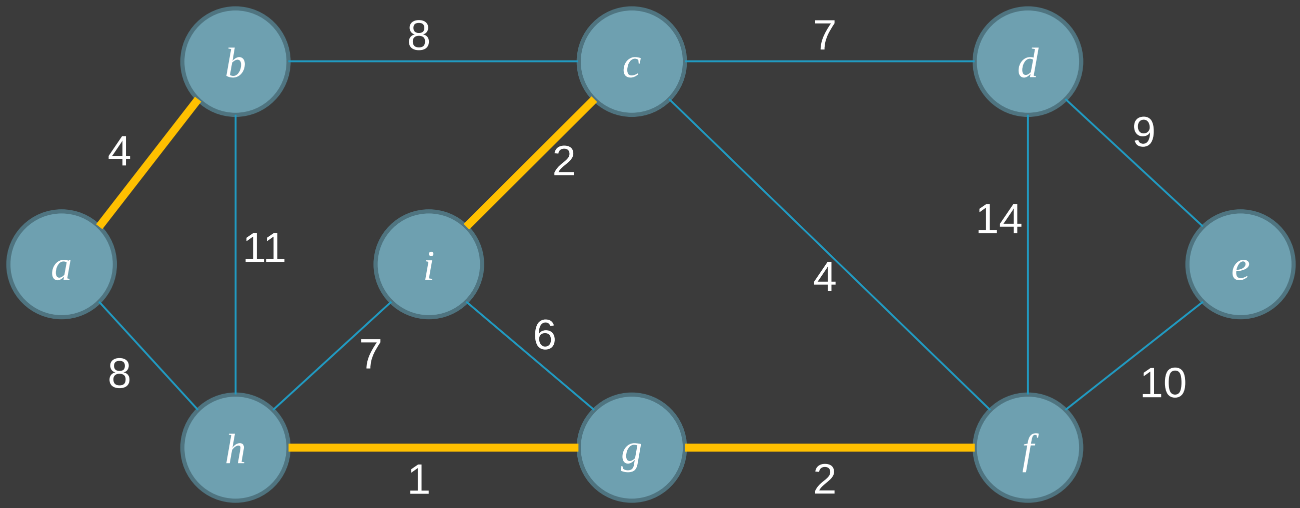


Sets:
$\{a\}$
$\{b\}$
$\{c, i\}$
$\{d\}$
$\{e\}$
$\{f, g, h\}$

The lowest-weight remaining edges are (a, b) and (c, f) Vertices a and b are in different sets, so include the edge and merge the sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

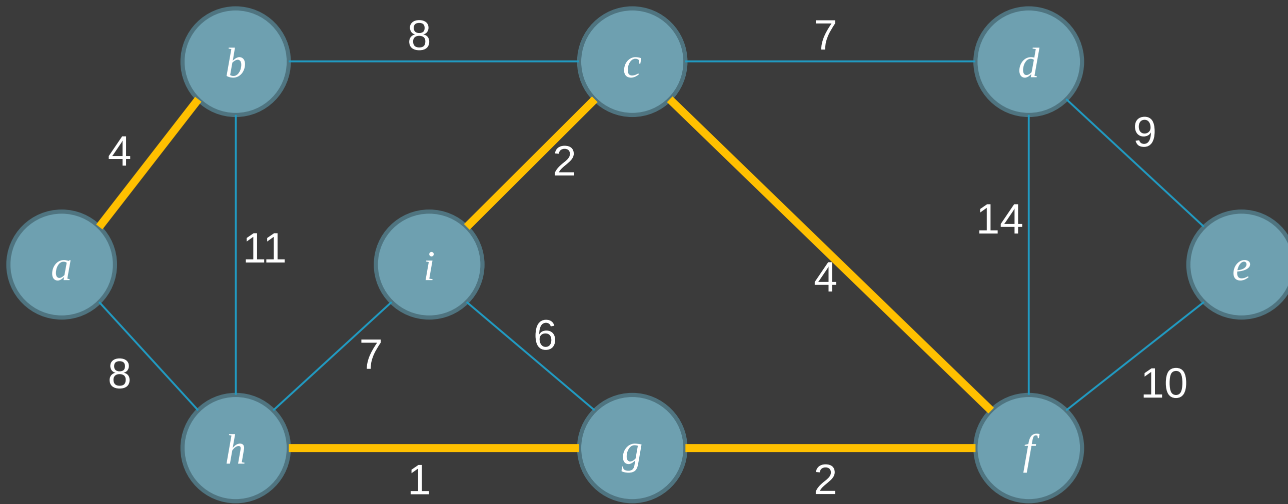


Sets:
{a, b}
{c, i}
{d}
{e}
{f, g, h}

The lowest-weight remaining edge is (c, f) Vertices c and f are in different sets, so include the edge and merge the sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

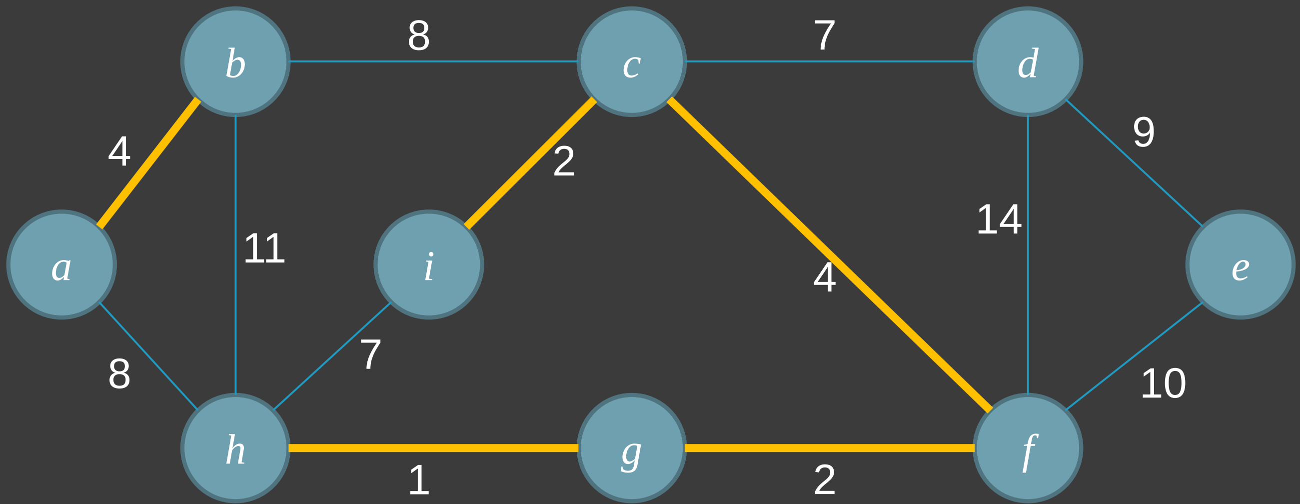


Sets:
$\{a, b\}$
$\{c, f, g, h, i\}$
$\{d\}$
$\{e\}$

The lowest-weight remaining edge is (g, i) Vertices g and i are in the same set, so we drop this edge from consideration, and keep going

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

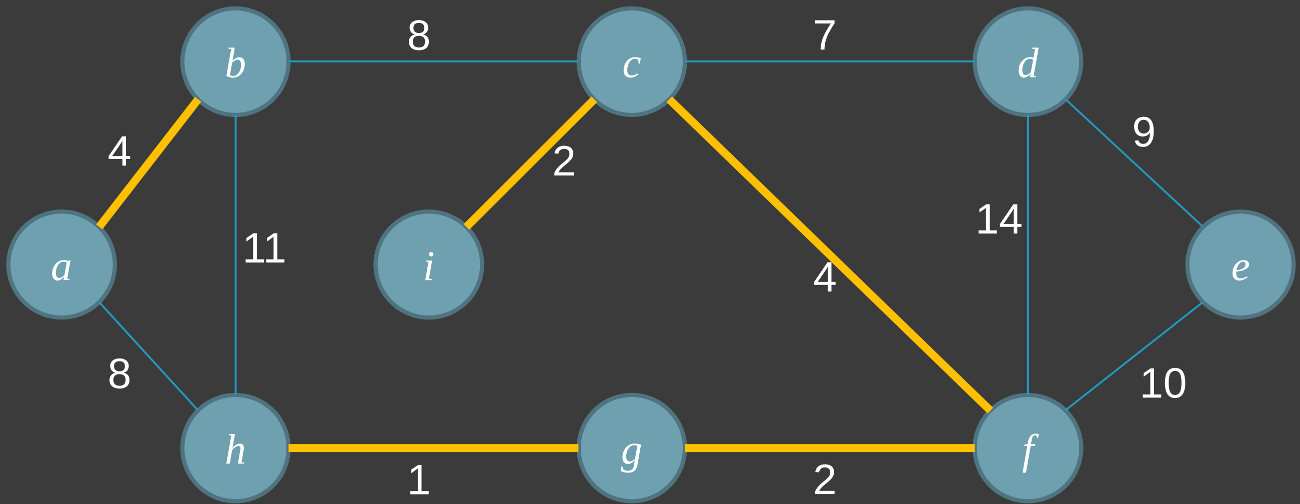


Sets:
$\{a, b\}$
$\{c, f, g, h, i\}$
$\{d\}$
$\{e\}$

The lowest-weight remaining edges are (h, i) and (c, d). Vertices h and i are in the same set, so we drop this edge from consideration, and keep going

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

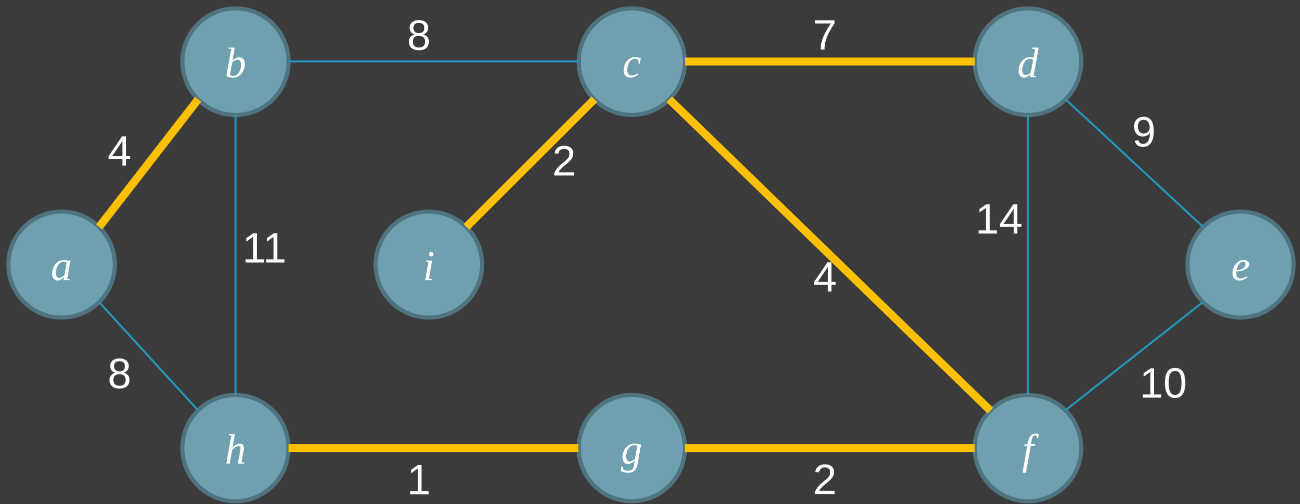


Sets:
$\{a, b\}$
$\{c, f, g, h, i\}$
$\{d\}$
$\{e\}$

The lowest-weight remaining edge is (c, d). Vertices c and d are in different sets, so include this edge and merge the sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal’s Algorithm

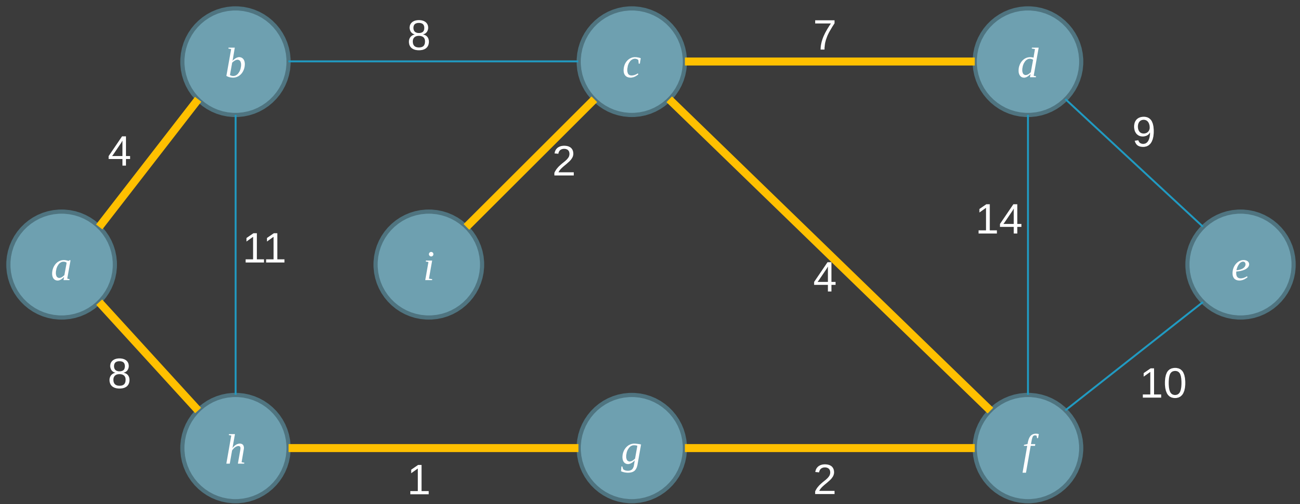


Sets:
{a, b}
{c, d, f, g, h, i}
{e}

The lowest-weight remaining edges are (a, h) and (b, c). Vertices a and h are in different sets, so include this edge and merge the sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

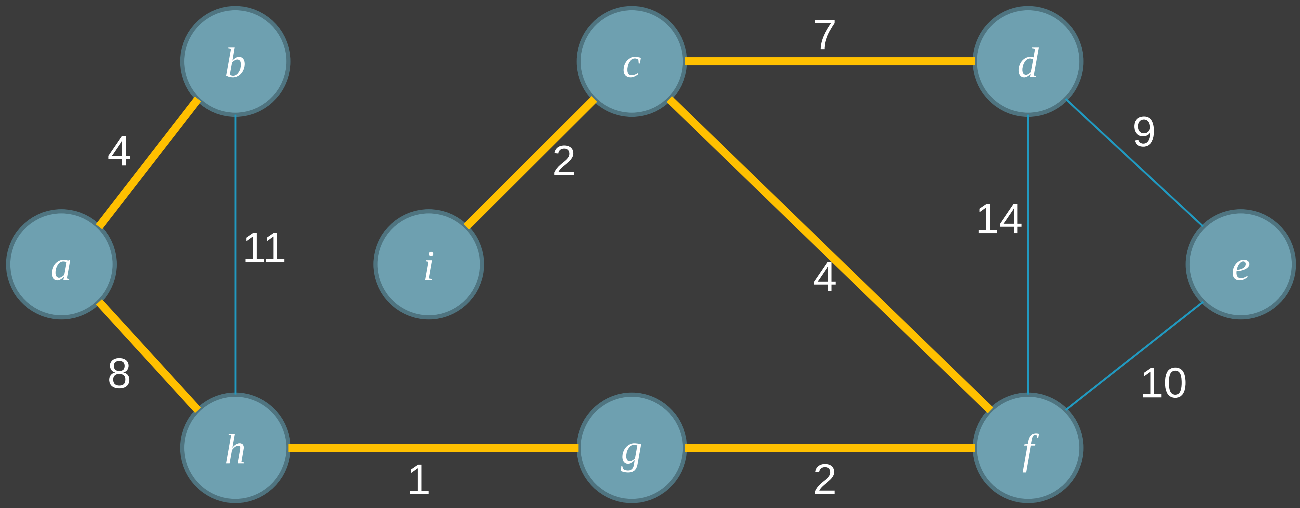


Sets:
$\{a, b, c, d, f, g, h, i\}$
$\{e\}$

The lowest-weight remaining edge is (b, c). Vertices b and c are in the same set, so drop this edge from consideration and keep going

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

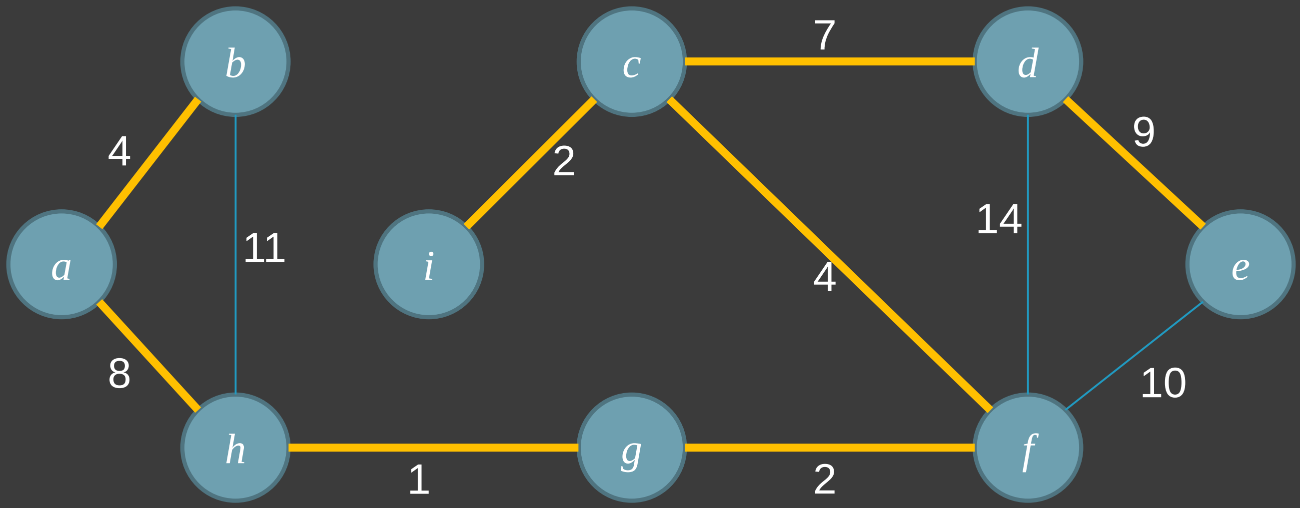


Sets:
$\{a, b, c, d, f, g, h, i\}$
$\{e\}$

The lowest-weight remaining edge is (d, e). Vertices d and e are in different sets, so include this edge and merge the two sets

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm

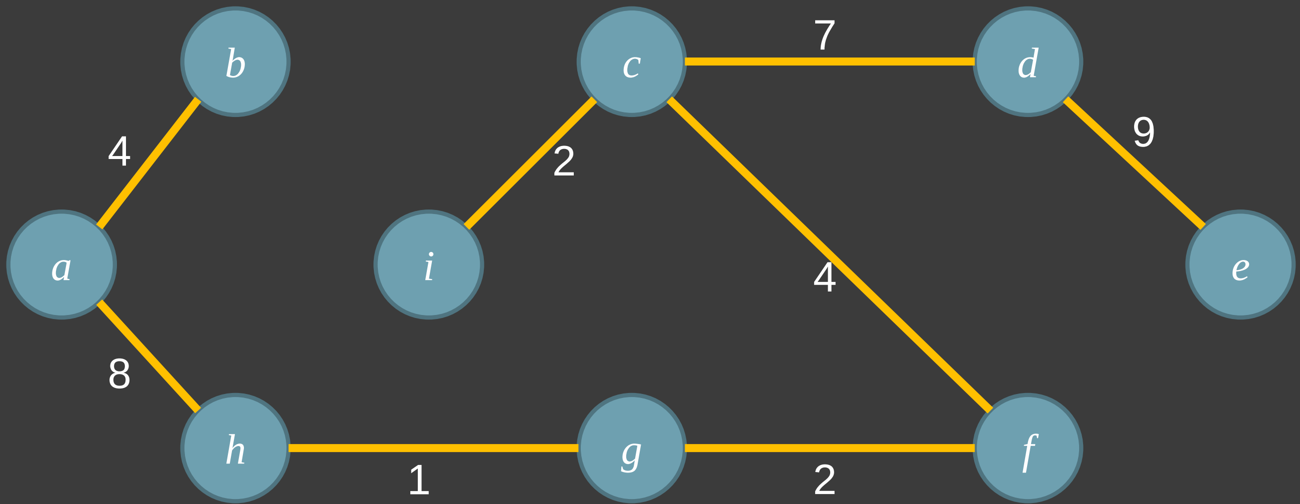


Sets:
$\{a, b, c, d, e, f, g, h, i\}$

We will next consider the edges (f, e), (b, h), and (d, f) (in that order). Because they are all already in the same set, we will drop them all.

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

Kruskal's Algorithm



Sets:
$\{a, b, c, d, e, f, g, h, i\}$

All nodes are connected. Total MST weight is 4 + 8 + 1 + 2 + 4 + 2 + 7 + 9 = 37

- Start with every vertex in its own set
- Consider edges in increasing order ONLY if the edge connects vertices in two different sets.
- If we include the edge, merge the two sets

MST-KRUSKAL Algorithm - Analysis

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
      MAKE-SET(v)
sort the edges of E by increasing w (weight) value
for each edge (u, v) ∈ E (taken in weight order)
      if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
return A
```

Total time: $O(E \lg V)$ (see p. 633)

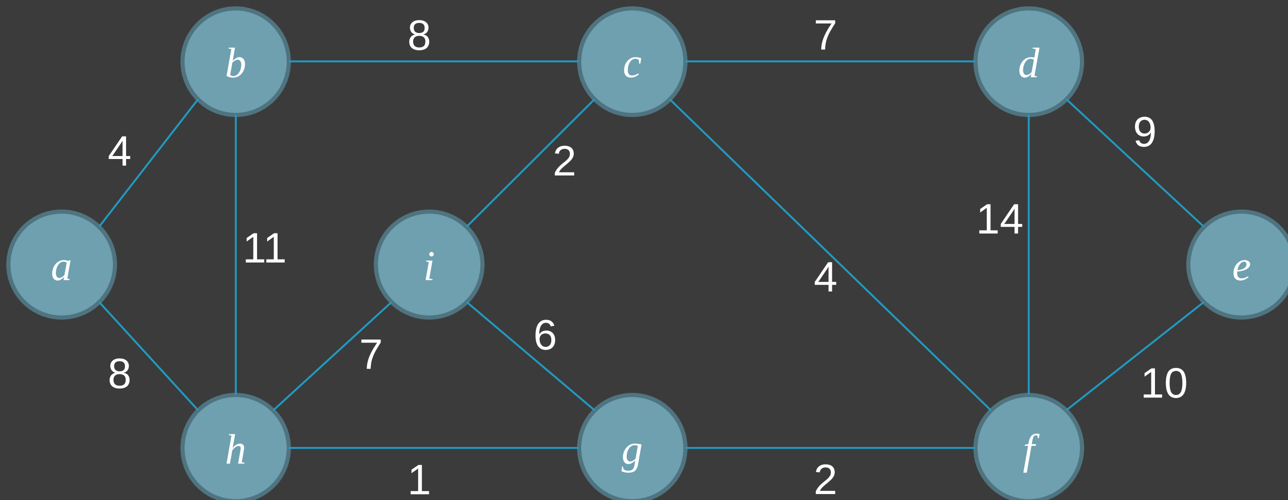
Prim’s Algorithm

- In Prim’s algorithm, the set A is ***one*** tree. The safe edge we add is always a least-weight edge connecting the tree to a vertex not yet in the tree.
- Start with an ***empty*** subset of the edges (F), and a subset of the vertices (Y) containing only a single (arbitrarily chosen) vertex, v_1.
- A vertex nearest to Y (separated by the lowest weight edge) is a vertex in (V–Y) connected to a vertex in Y by an edge of minimum weight.
- The vertex nearest to Y is added to Y, and the edge is added to F. We repeat until $Y = V$

MST-Prim Algorithm

```
MST-PRIM(G, w, r)        // r is an arbitrarily chosen vertex
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V                  // Q is a min-priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

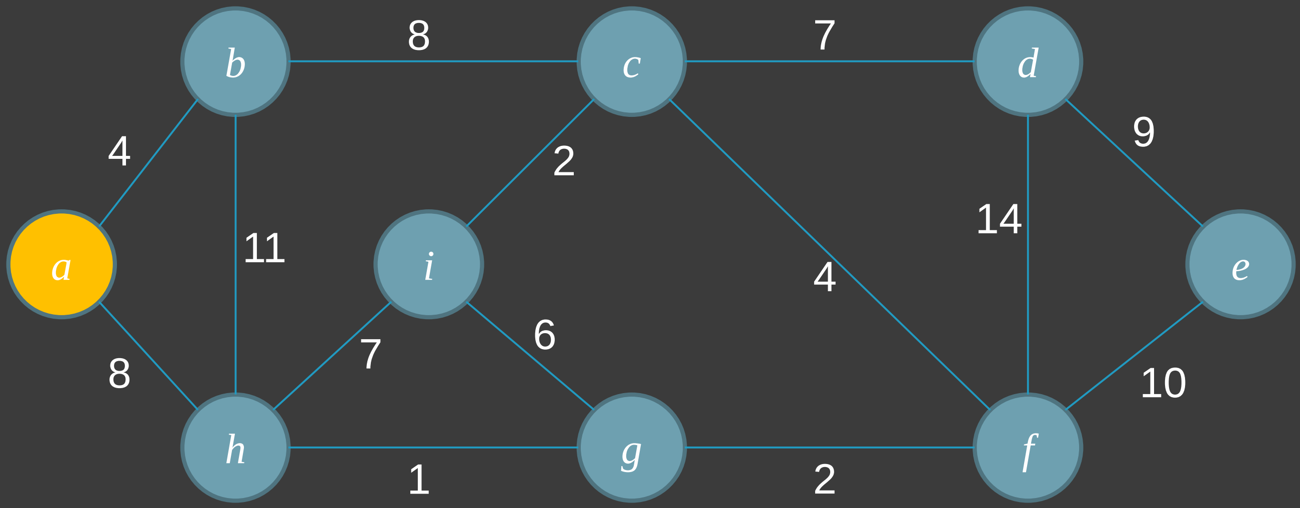
Prim's Algorithm



$Y = \{a\}$
$F = \{\}$

- Let's start (arbitrarily) from a.

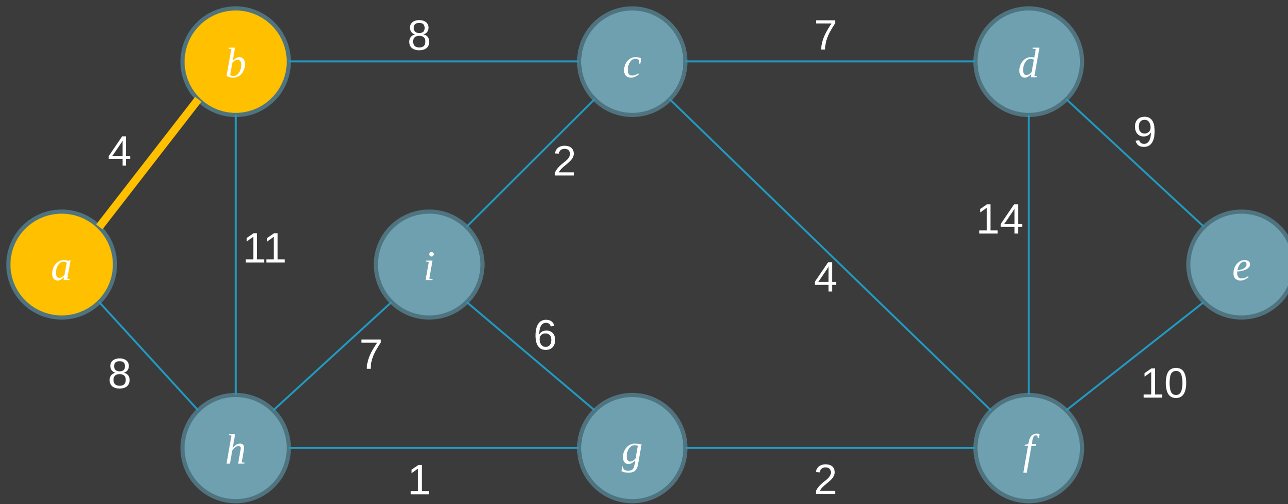
Prim's Algorithm



$Y = \{a\}$
$F = \{\}$

- Let's start (arbitrarily) from a.
- The lowest-weight edge connecting Y to $(V–Y)$ is (a, b)
- Include this edge in F, and include b in Y.

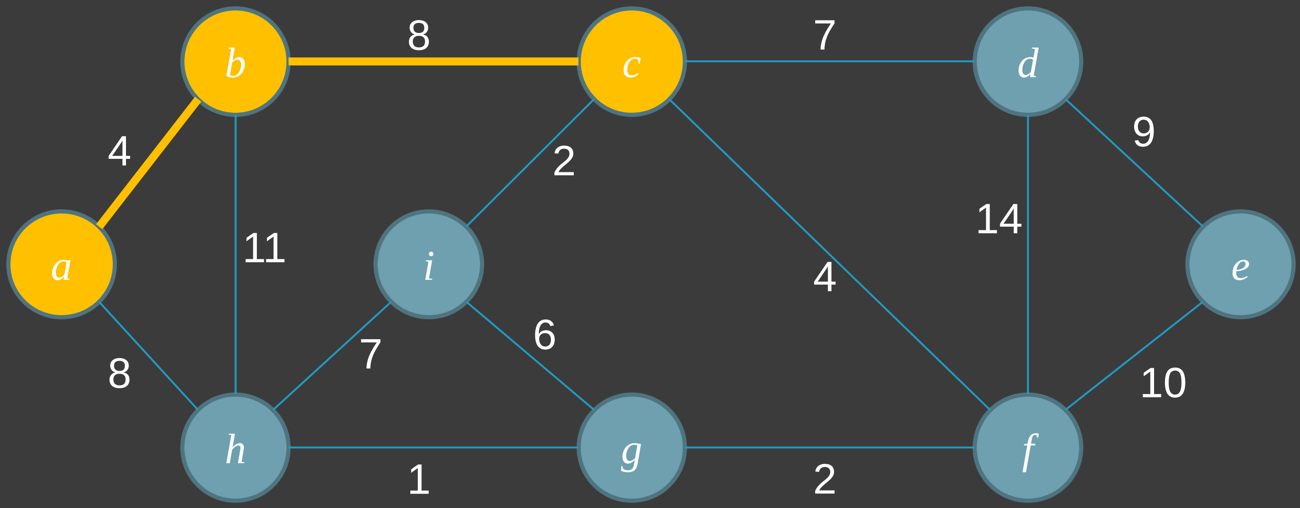
Prim's Algorithm



$Y = \{a, b\}$
$F = \{(a, b)\}$

- The lowest-weight edges connecting Y to $(V–Y)$ are (b, c) and (a, h). We will arbitrarily choose (b, c).
- Include this edge in F, and include c in Y.

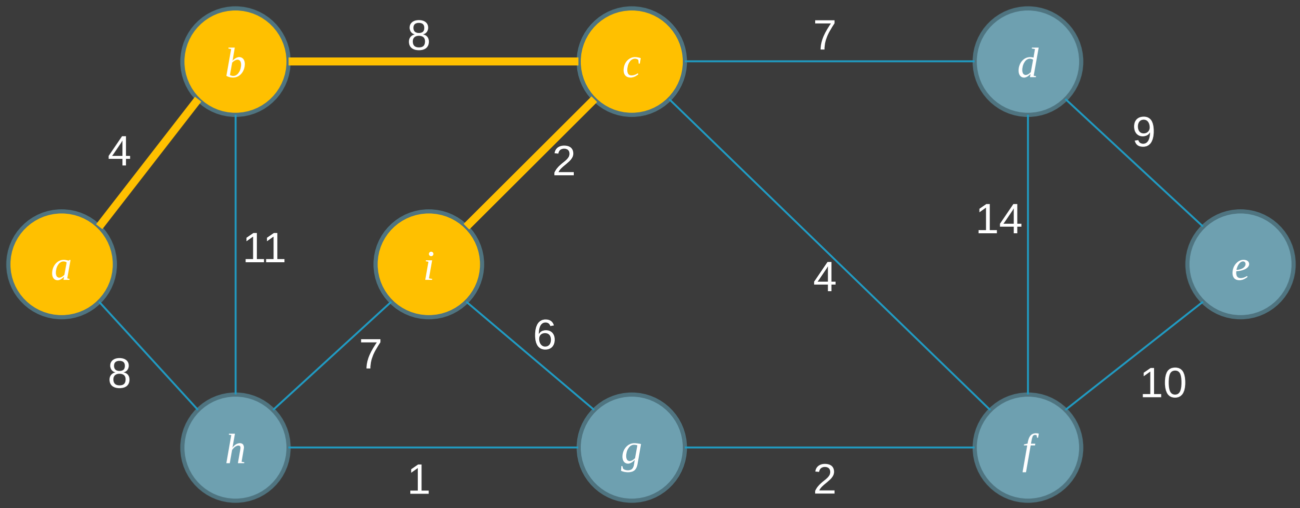
Prim's Algorithm



$Y = \{a, b, c\}$
$F = \{(a, b), (b, c)\}$

- The lowest-weight edge connecting Y to $(V–Y)$ is (c, i)
- Include this edge in F, and include i in Y.

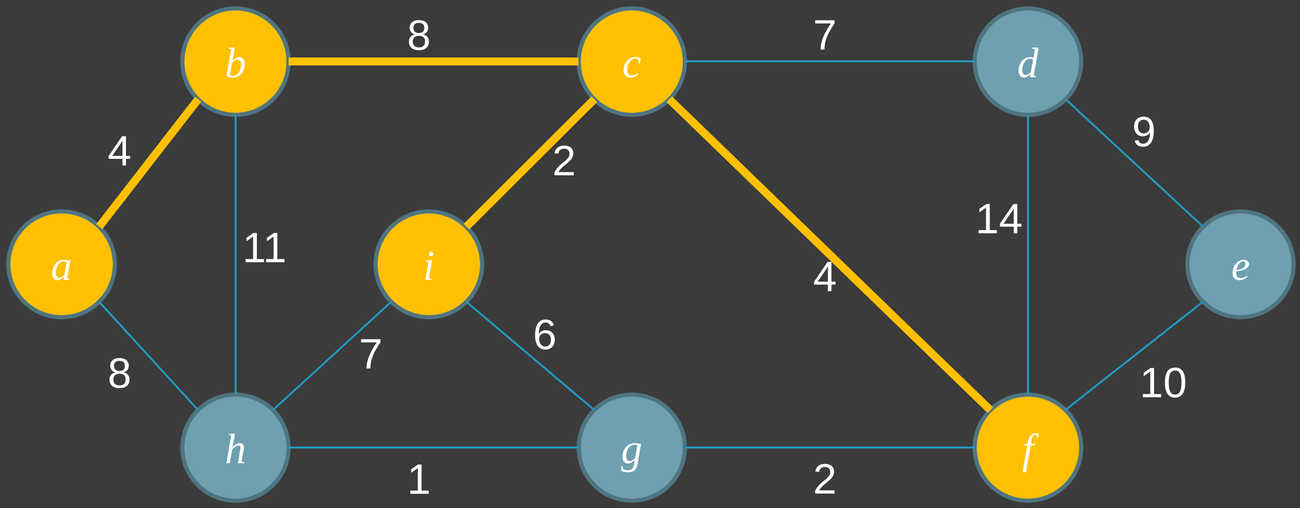
Prim's Algorithm



$Y = \{a, b, c, i\}$
$F = \{(a, b), (b, c), (c, i)\}$

- The lowest-weight edge connecting Y to $(V–Y)$ is (c, f)
- Include this edge in F, and include f in Y.

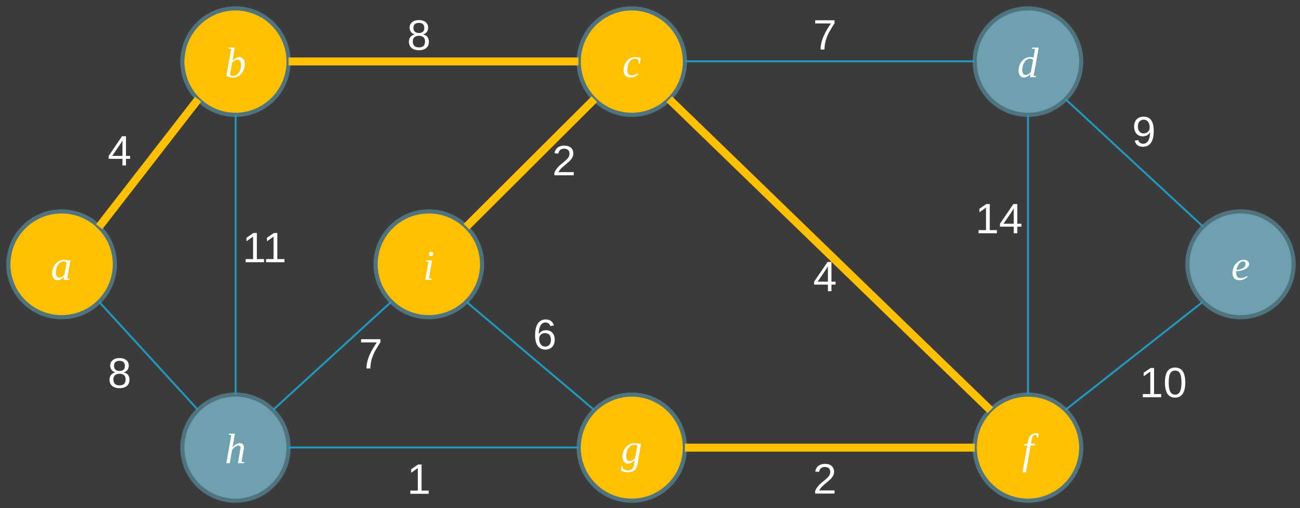
Prim's Algorithm



$Y = \{a, b, c, f, i\}$
$F = \{(a, b), (b, c), (c, i), (c, f)\}$

- The lowest-weight edge connecting Y to $(V–Y)$ is (f, g)
- Include this edge in F, and include g in Y.

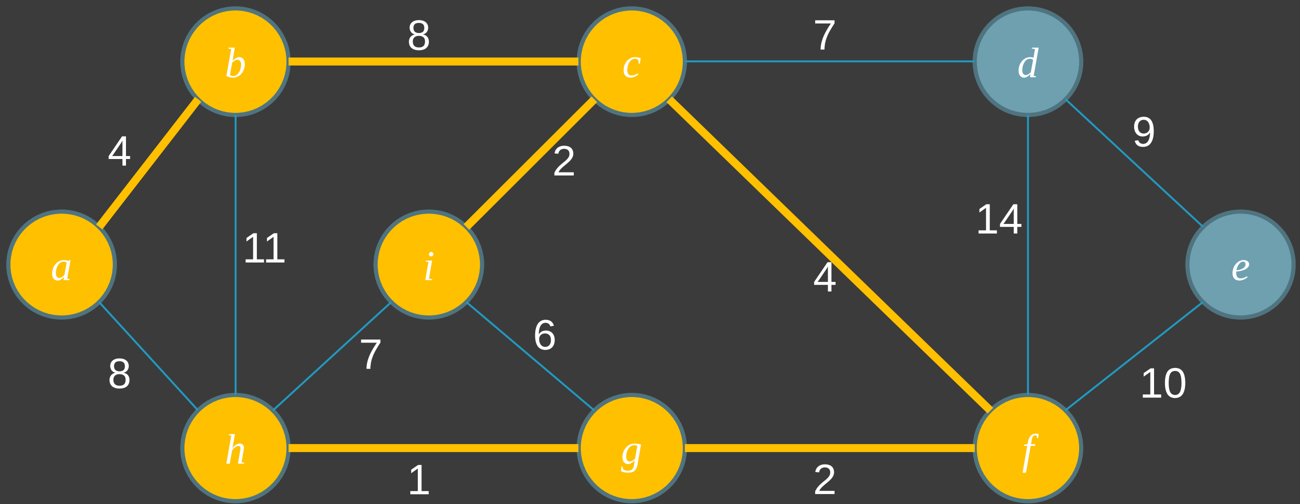
Prim's Algorithm



$Y = \{a, b, c, f, g, i\}$
$F = \{(a, b), (b, c), (c, i),$
$(c, f), (f, g)\}$

- The lowest-weight edge connecting Y to $(V–Y)$ is (g, h)
- Include this edge in F, and include h in Y.

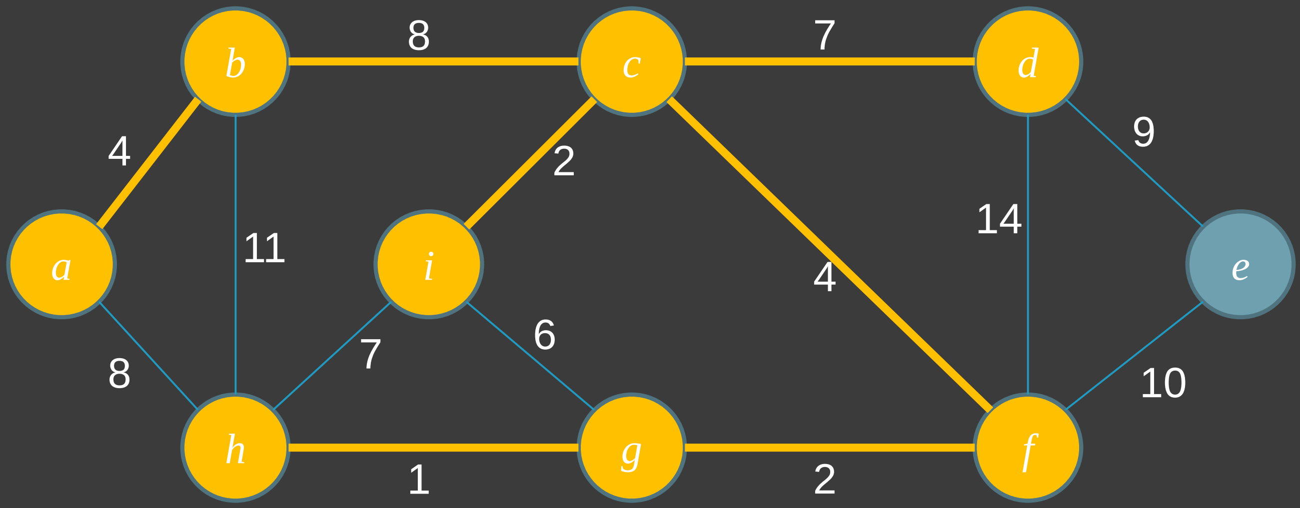
Prim's Algorithm



$Y = \{a, b, c, f, g, h, i\}$
$F = \{(a, b), (b, c), (c, i), (c, f), (f, g), (g, h)\}$

- The lowest-weight edge connecting Y to $(V\text{–}Y)$ is (c, d)
- Include this edge in F, and include d in Y.

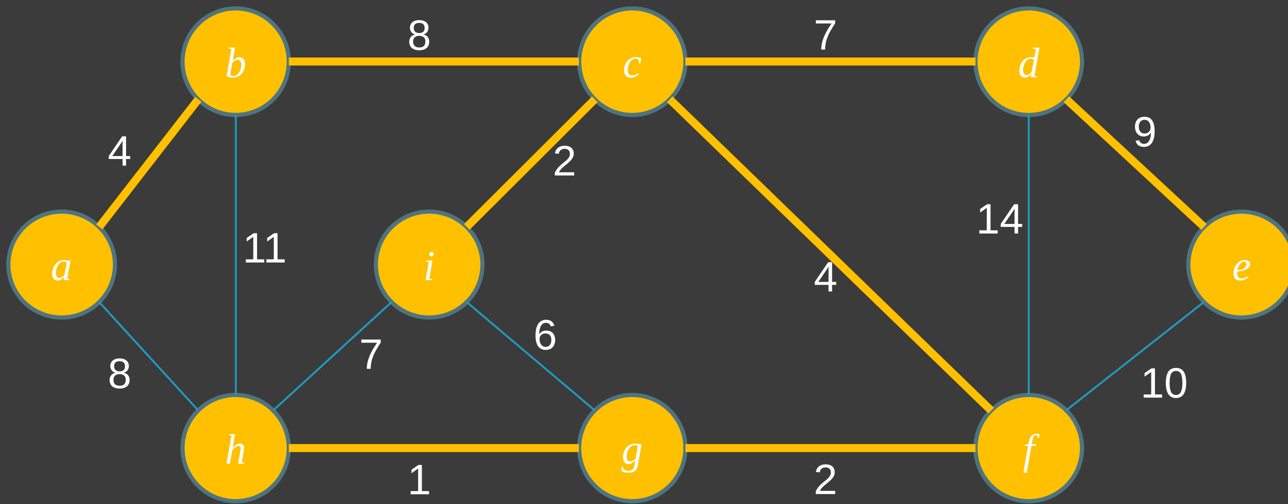
Prim's Algorithm



$Y = \{a, b, c, d, f, g, h, i\}$
$F = \{(a, b), (b, c), (c, i),$
$(c, f), (f, g), (g, h),$
$(c, d)\}$

- The lowest-weight edge connecting Y to $(V–Y)$ is (d, e)
- Include this edge in F, and include e in Y.

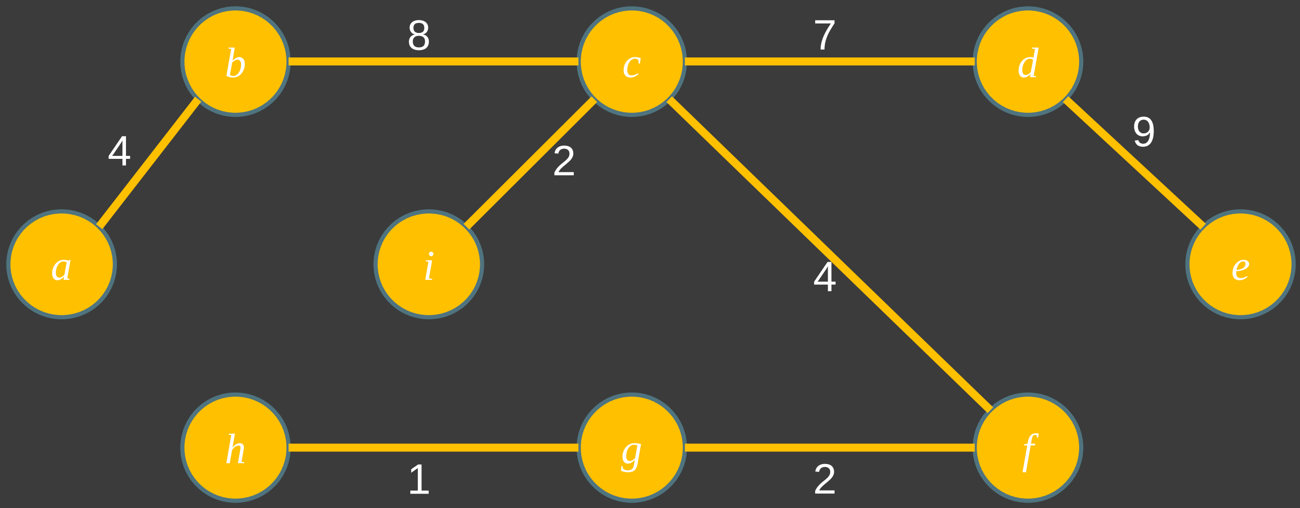
Prim's Algorithm



$Y = \{a, b, c, d, e, f, g, h, i\}$
$F = \{(a, b), (b, c), (c, i), (c, f), (f, g), (g, h), (c, d), (d, e)\}$

- Now $Y = V$ (we have added all of the vertices to Y)
- Remove all remaining edges

Prim's Algorithm



$Y = \{a, b, c, d, e, f, g, h, i\}$
$F = \{(a, b), (b, c), (c, i), (c, f), (f, g), (g, h), (c, d), (d, e)\}$

- Now $Y = V$ (we have added all of the vertices to Y)
- Remove all remaining edges
- MST Weight = 4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = 37

Prim's Algorithm

- Total run time: $O(E \lg V)$ (like Kruskal)
- Depends on how the priority queue is implemented. A MIN-HEAP is a good way, but a FIBONACCI-HEAP (Chapter 19) is even better.

End of Chapter 23

? Questions ?