

COMP 251

Algorithms & Data Structures (Winter 2021)

Graphs – Minimum Spanning Trees

School of Computer Science
McGill University

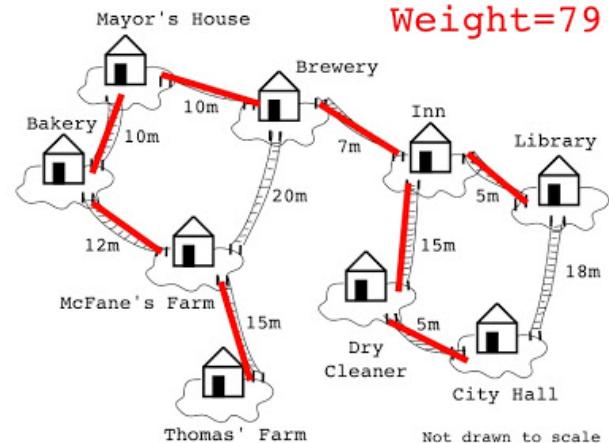
Slides of (Comp321 ,2021), Langer (2014), Kleinberg & Tardos, 2005 & Cormen et al., 2009, Jaehyun Park' slides CS 97SI, Topcoder tutorials, T-414-AFLV Course, Programming Challenges books, slides from D. Plaisted (UNC) and Comp251-Fall McGill.

Minimum Spanning Tree- Problem

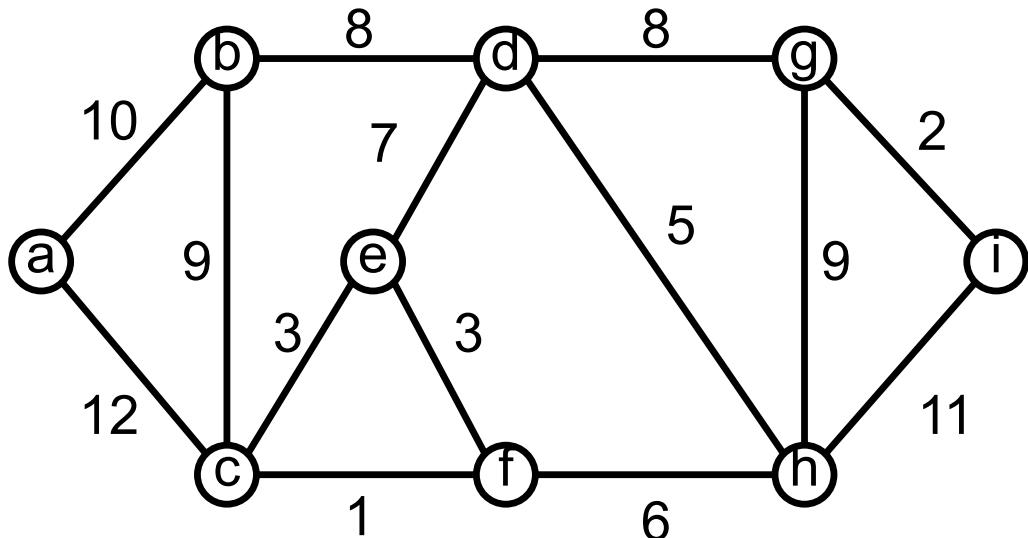
- A town has a set of houses and a set of roads.
- A road connects 2 and only 2 houses.
- A road connecting houses u and v has a repair cost $w(u, v)$.

Goal: Repair enough (and no more) roads such that:

1. everyone stays connected:
can reach every house from
all other houses, and
2. total repair cost is minimum.

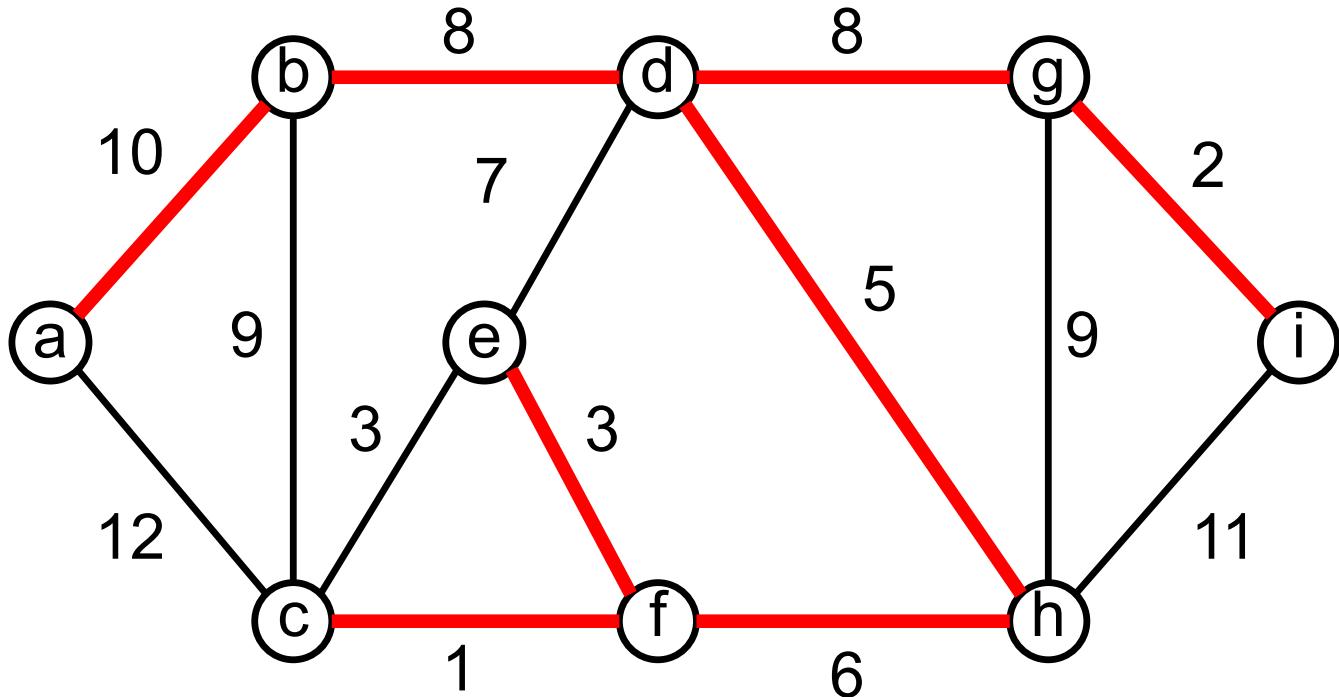


Minimum Spanning Tree – As a graph problem



- Undirected graph $G = (V, E)$.
- **Weight** $w(u, v)$ on each edge $(u, v) \in E$.
- Find $T \subseteq E$ such that:
 1. T connects all vertices (T is a **spanning tree**),
 2. $w(T) = \sum_{(u,v) \in T} w(u,v)$ is minimized.

Minimum Spanning Tree – As a graph problem



- It has $|V| - 1$ edges.
- It has no cycles.
- It might not be unique.

MST– Generic Algorithm

- Initially, A has no edges.
- Add edges to A and maintain the **loop invariant**: “ A is a subset of some MST”.

```
A  $\leftarrow \emptyset$ ;
```

```
while A is not a spanning tree do
```

```
    find a edge  $(u, v)$  that is safe for A;
```

```
    A  $\leftarrow A \cup \{(u, v)\}$ 
```

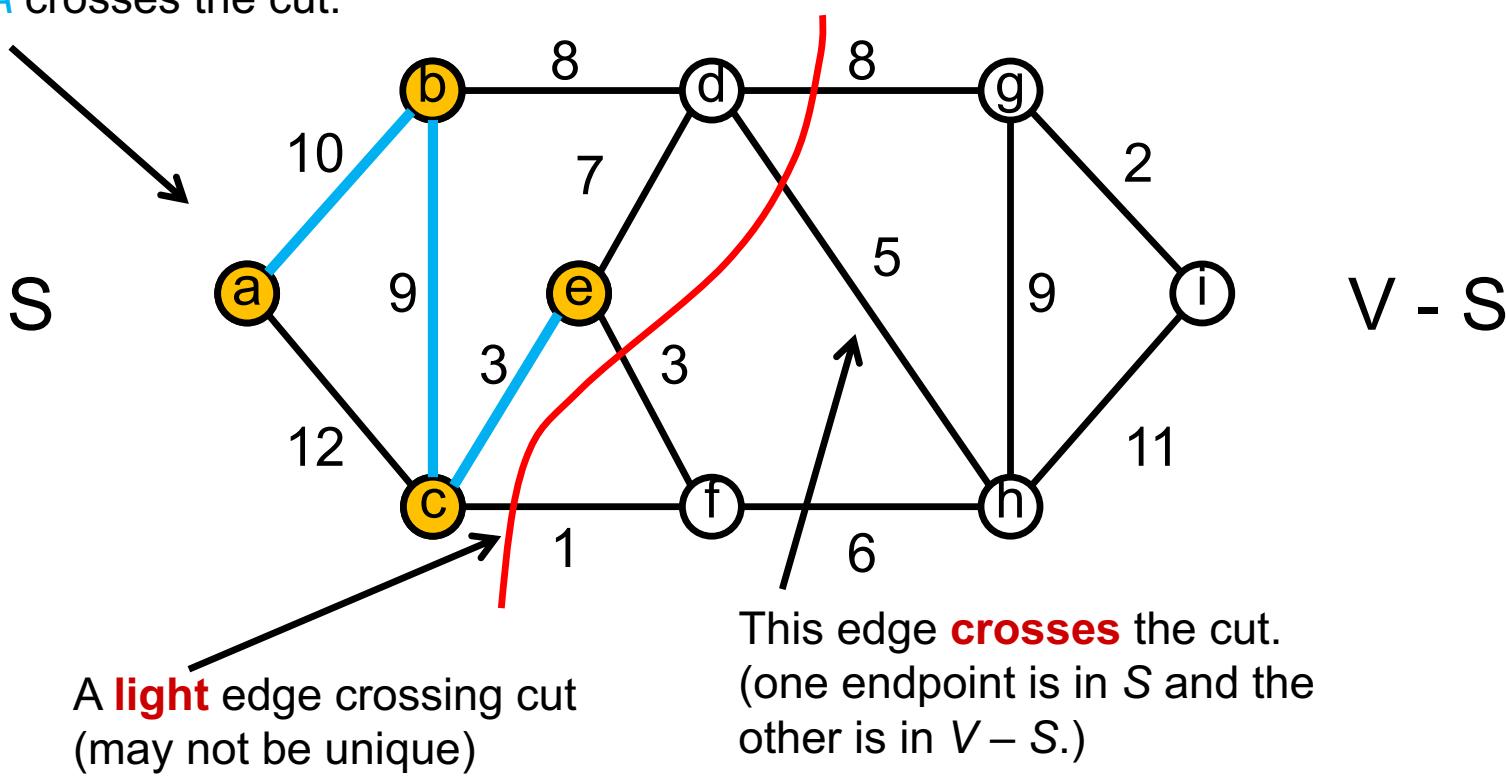
```
return A
```

- **Initialization:** The empty set trivially satisfies the loop invariant.
- **Maintenance:** 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.

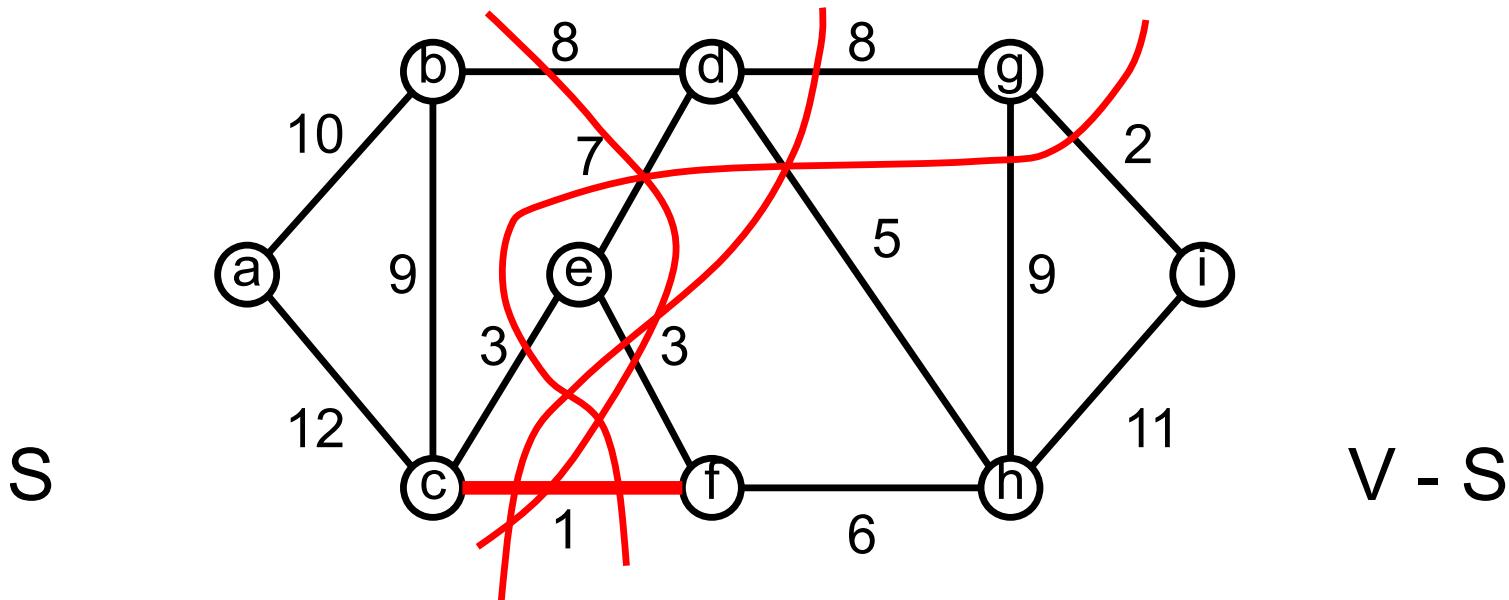
MST– Definitions

A cut **respects A** if and only if no edge in **A** crosses the cut.

cut partitions vertices into disjoint sets, S and $V - S$.



MST – What is a safe edge?



Intuitively: Is (c,f) safe when $A=\emptyset$?

- Let S be any set of vertices including c but not f .
- There has to be one edge (at least) that connects S with $V - S$.
- Why not choosing the one with the minimum weight?

MST– Safe edge

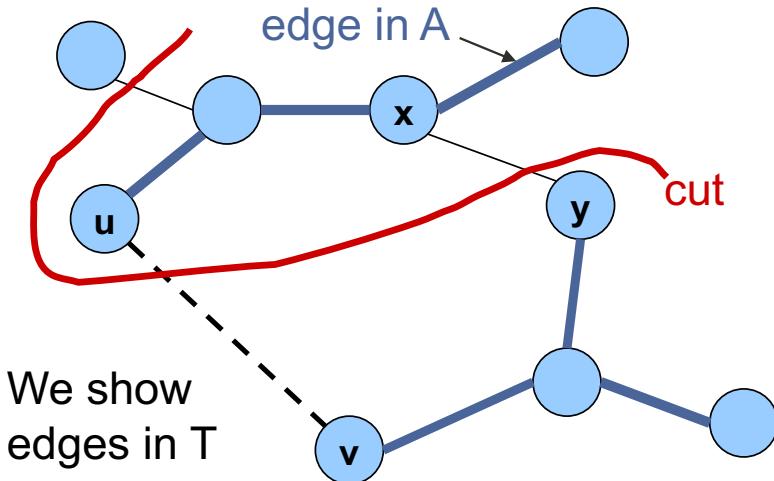
Theorem 1: Let $(S, V-S)$ be any cut that respects A, and let (u, v) be a light edge crossing $(S, V-S)$. Then, (u, v) is safe for A.

Proof:

Let T be a MST that includes A.

Case 1: (u, v) in T . We're done.

Case 2: (u, v) not in T . We have the following:



(x, y) crosses the cut.
Let $T' = T - \{(x, y)\} \cup \{(u, v)\}$.
Because (u, v) is light for cut,
 $w(u, v) \leq w(x, y)$. Thus,
 $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$.
Hence, T' is also a MST.
So, **(u, v) is safe for A**.

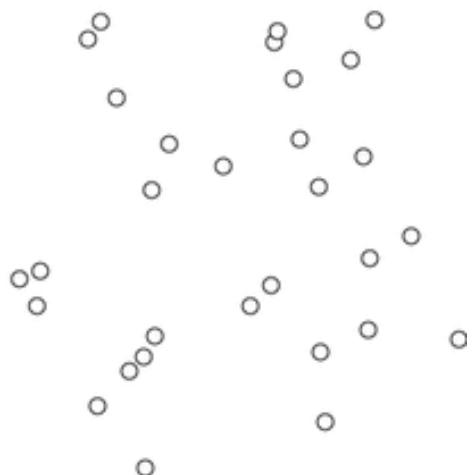
MST– Safe edge

In general, A will consist of several connected components.

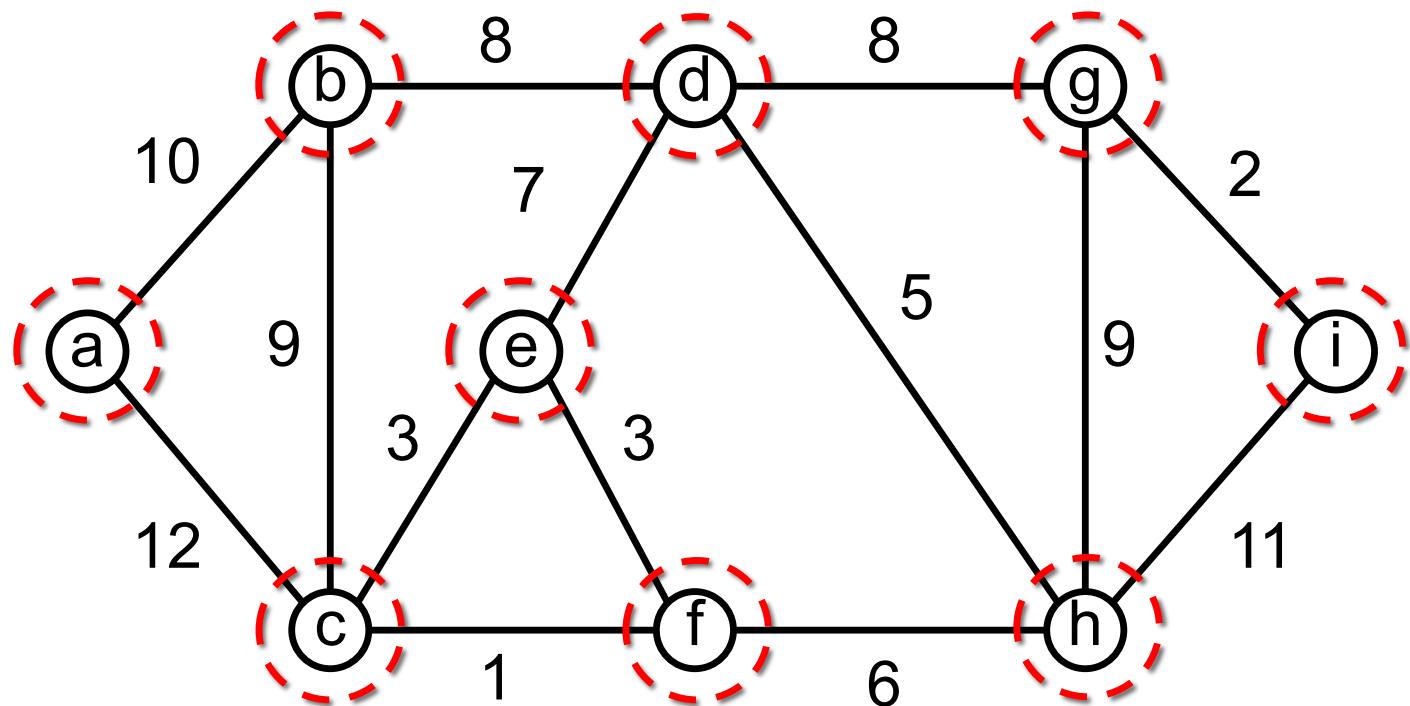
Corollary: If (u, v) is a light edge connecting one CC in (V, A) to another CC in (V, A) , then (u, v) is safe for A.

MST – Kruskal's Algorithm

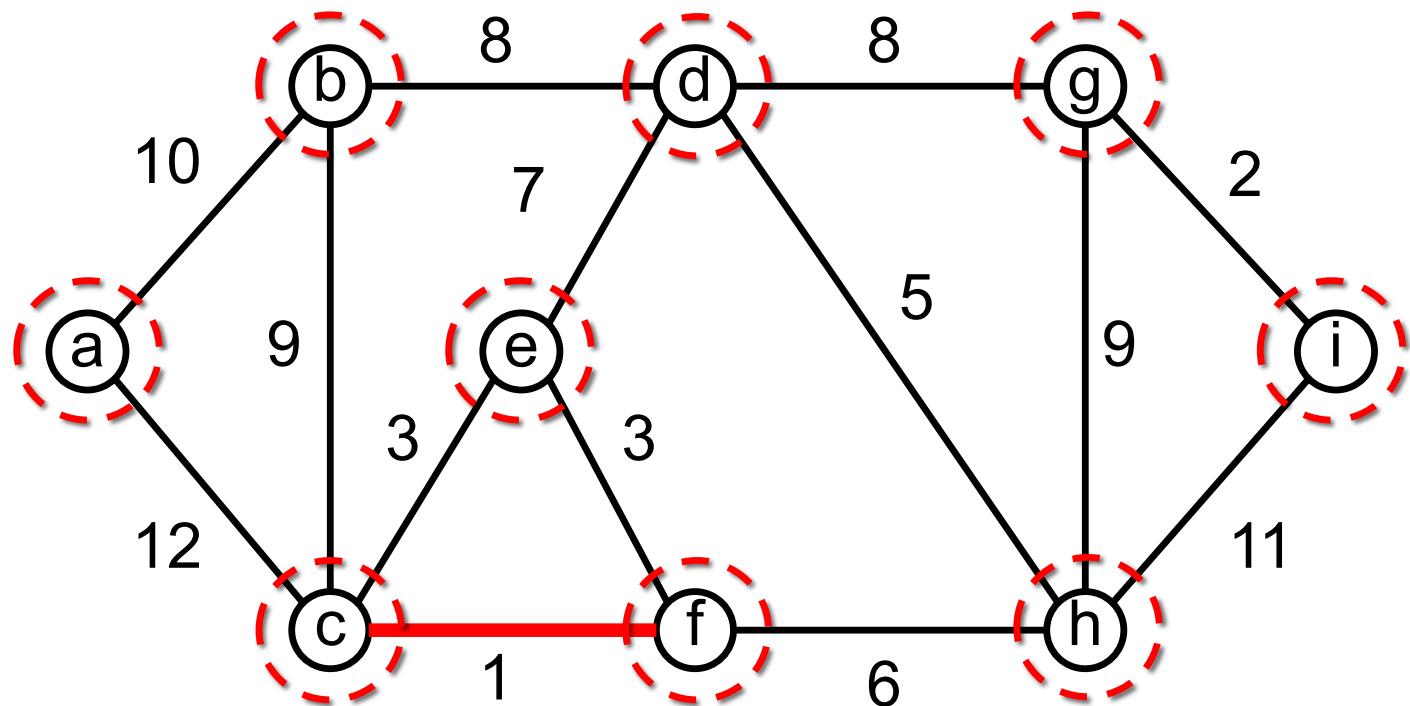
1. Starts with each vertex in its own component.
2. Repeatedly merges two components into one by choosing a light edge that connects them (i.e., a light edge crossing the cut between them).
3. Scans the set of edges in monotonically increasing order by weight.
4. Uses a **disjoint-set data structure** to determine whether an edge connects vertices in different components.



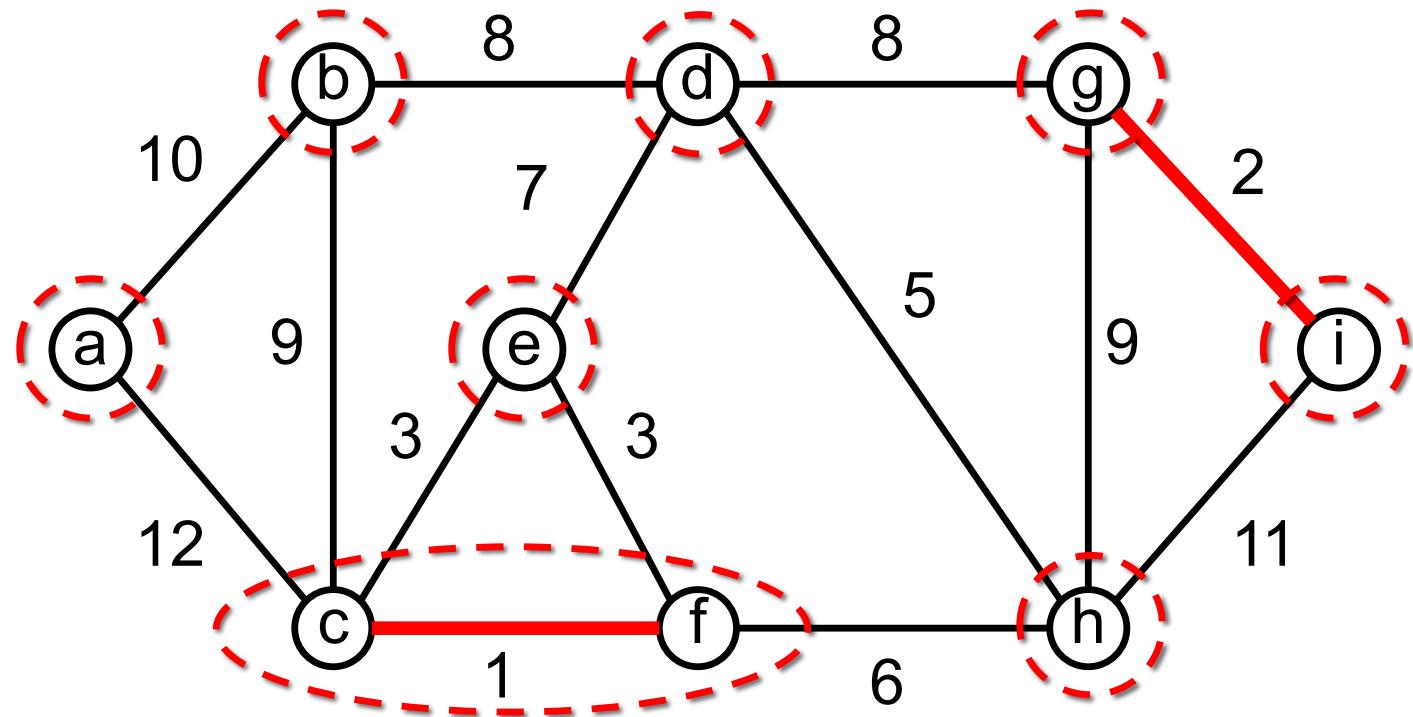
Kruskal's Algorithm - Example



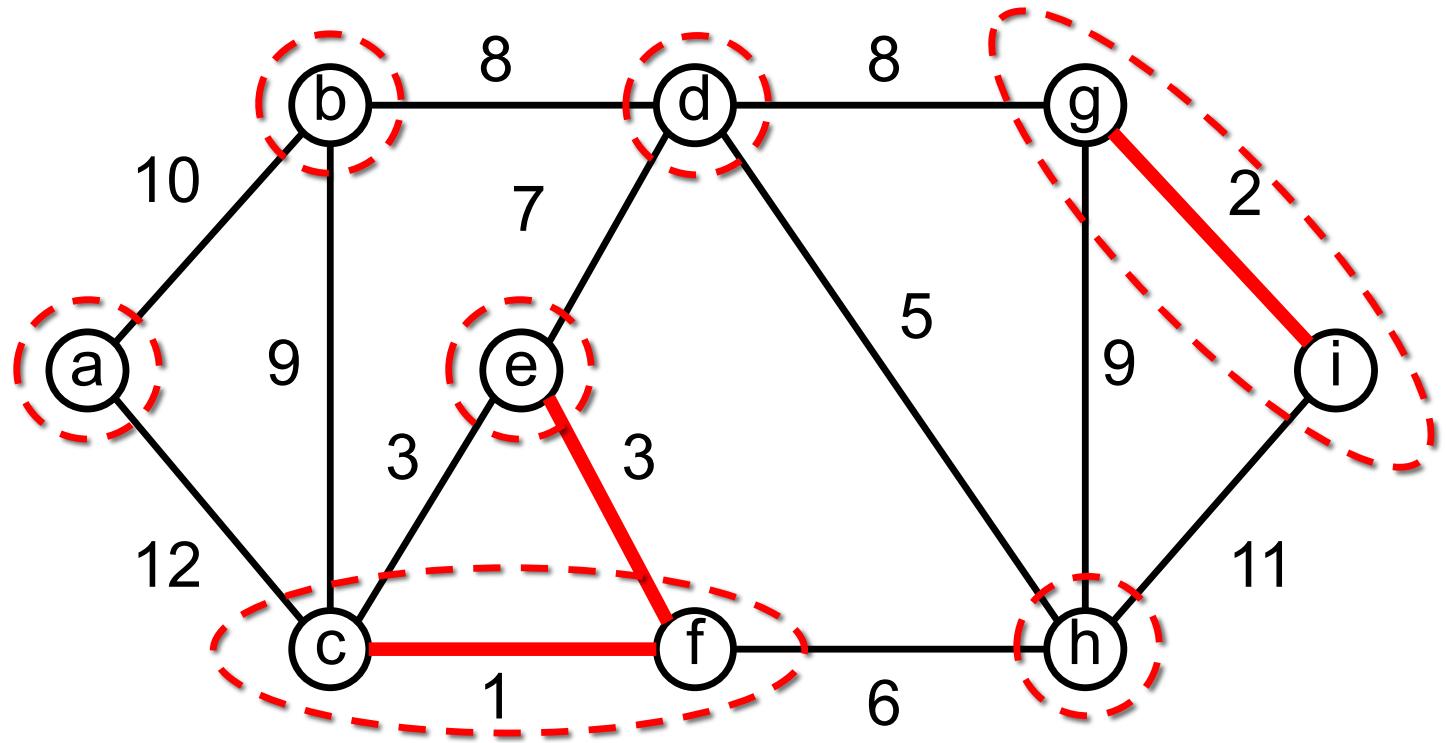
Kruskal's Algorithm - Example



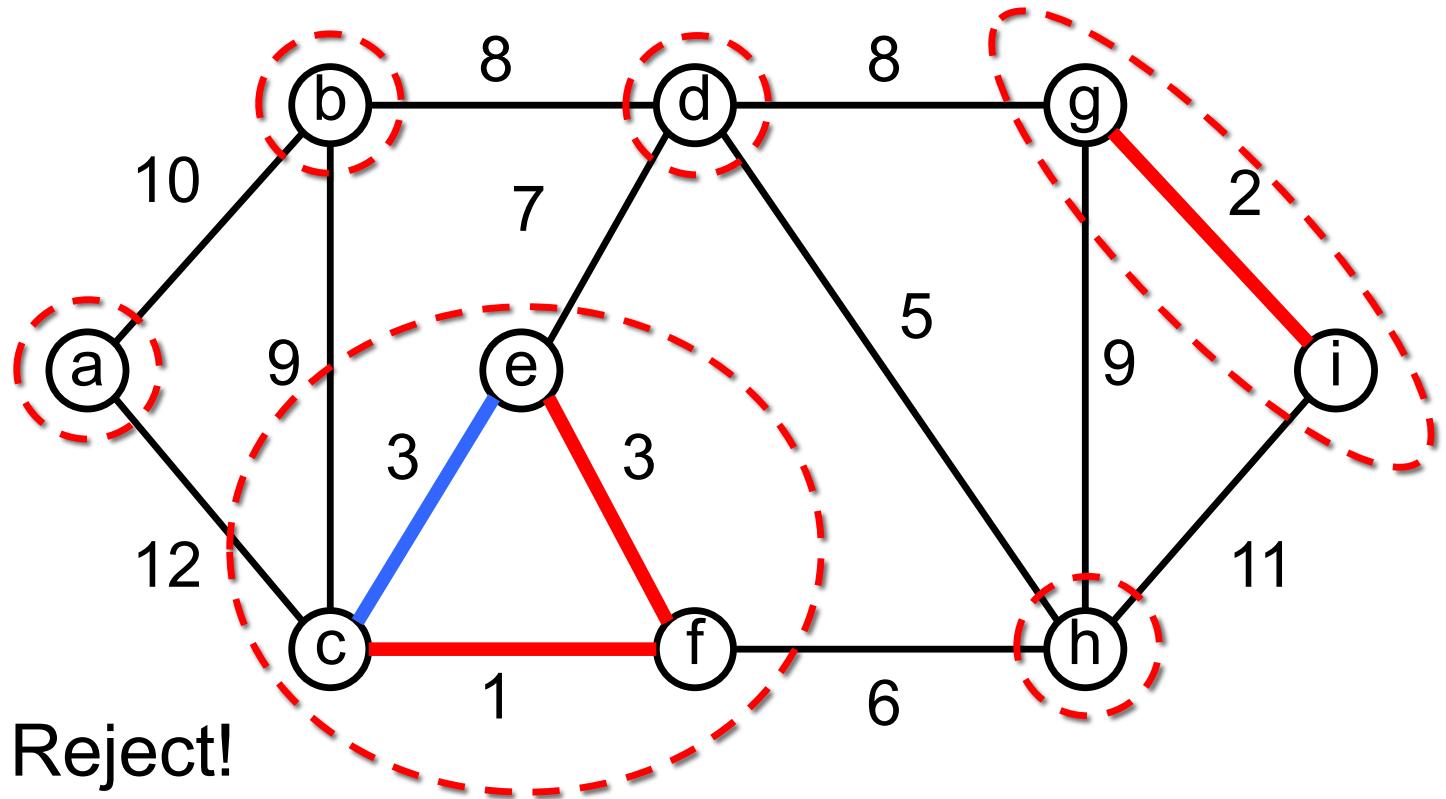
Kruskal's Algorithm - Example



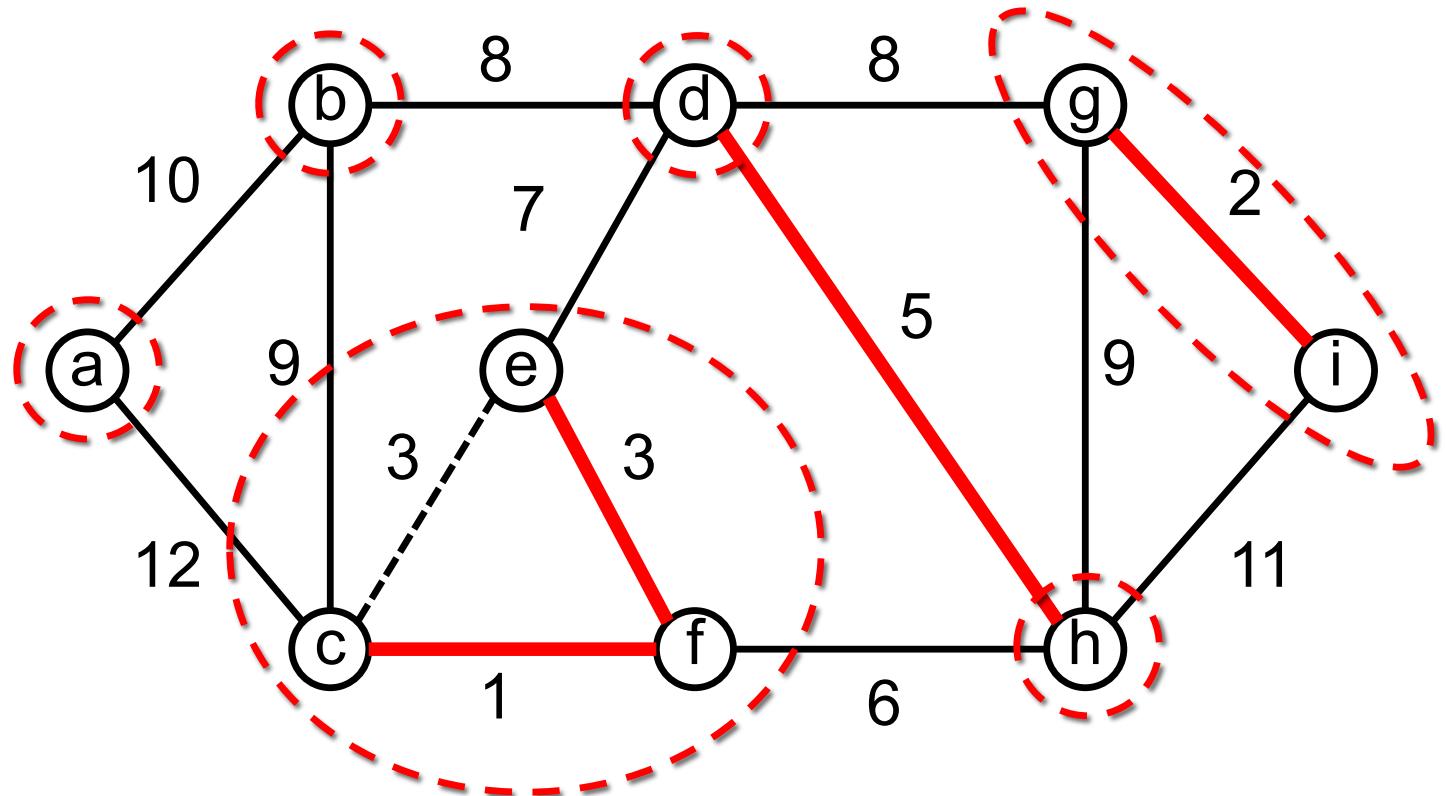
Kruskal's Algorithm - Example



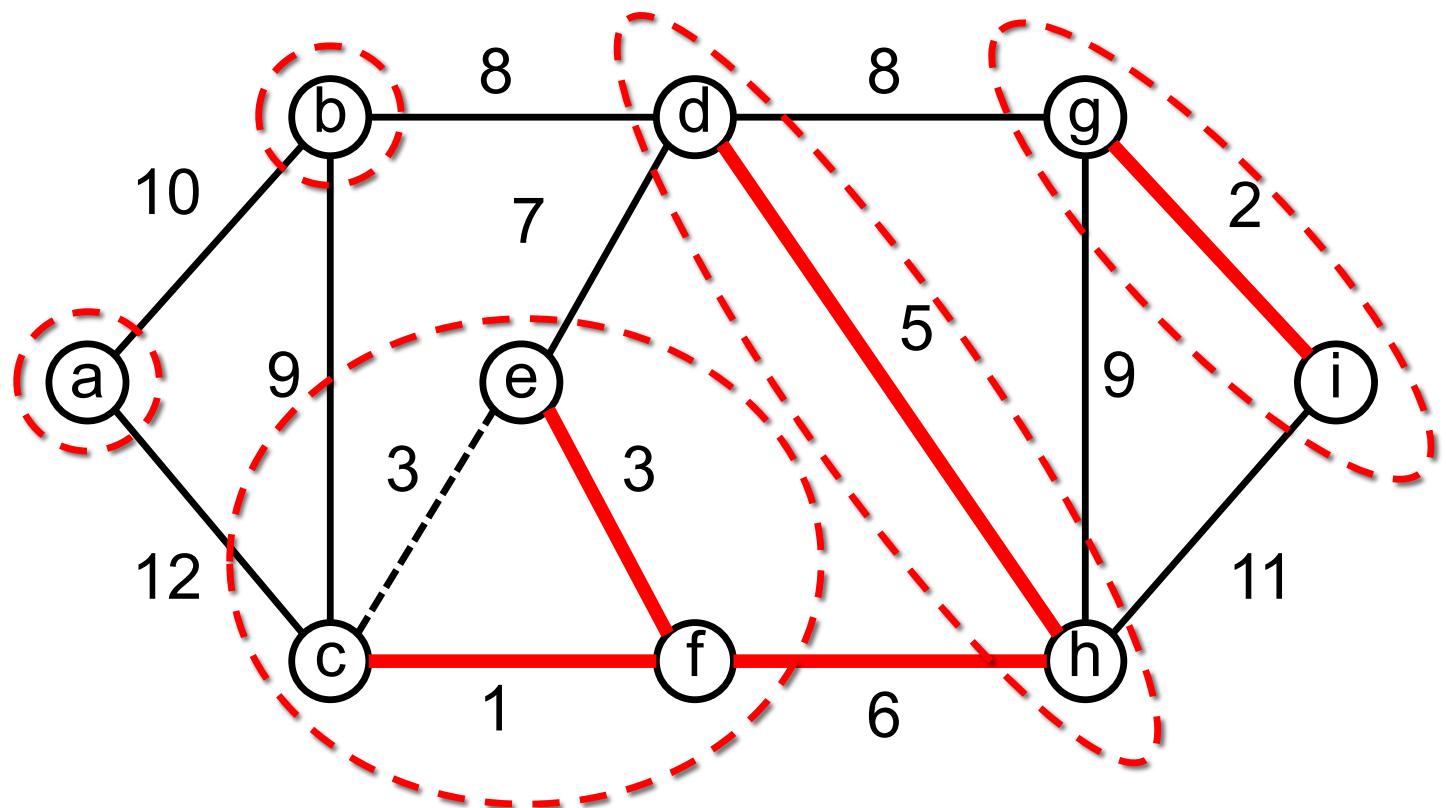
Kruskal's Algorithm - Example



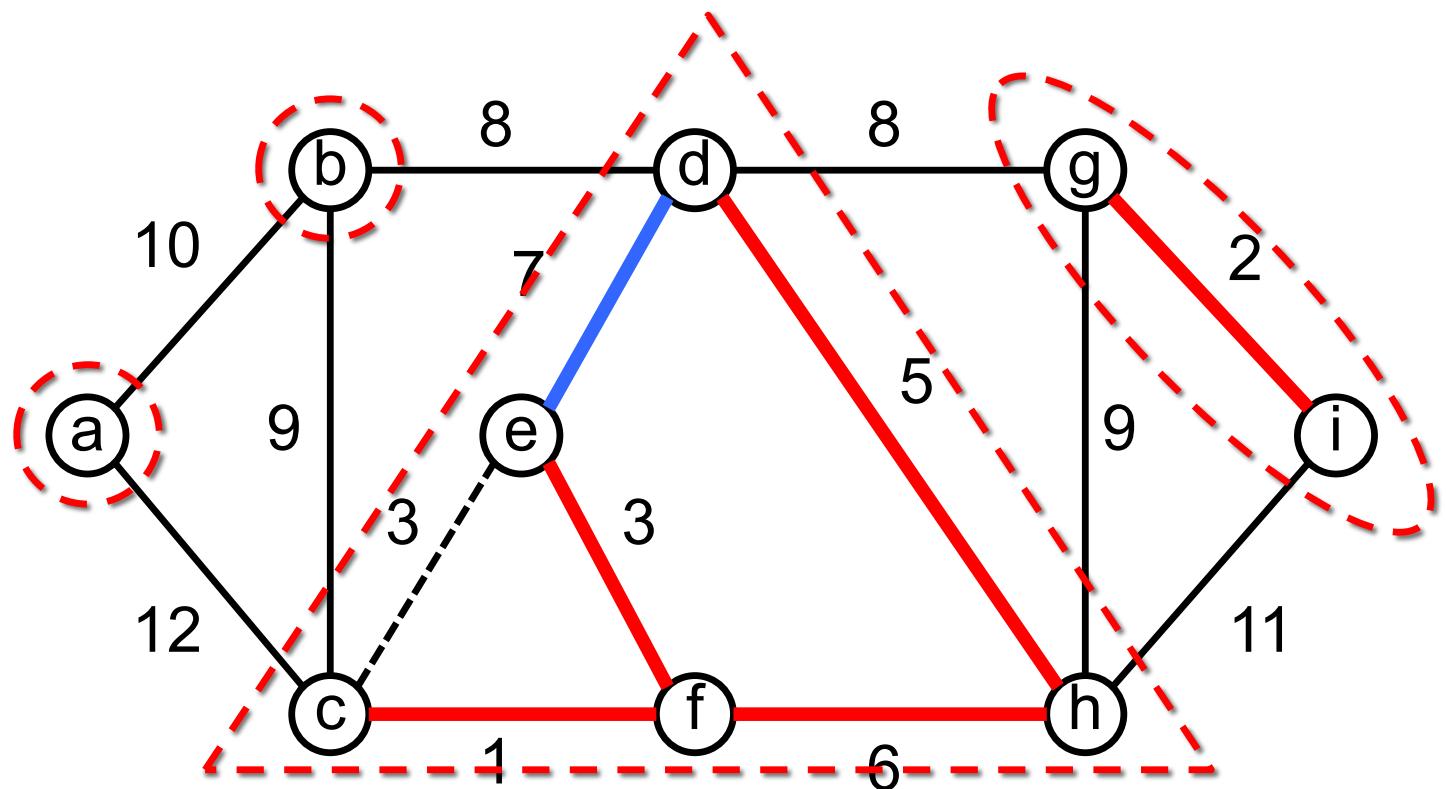
Kruskal's Algorithm - Example



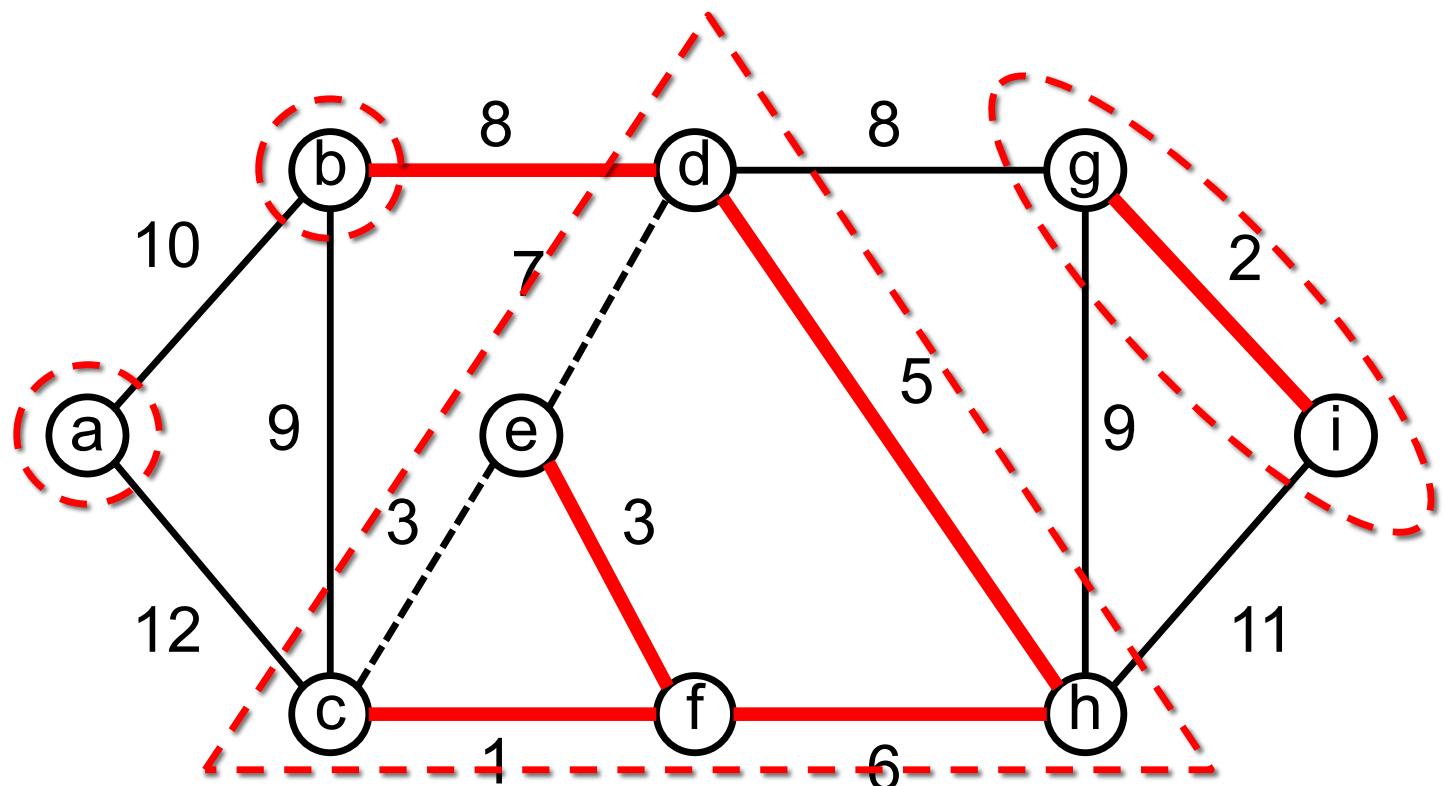
Kruskal's Algorithm - Example



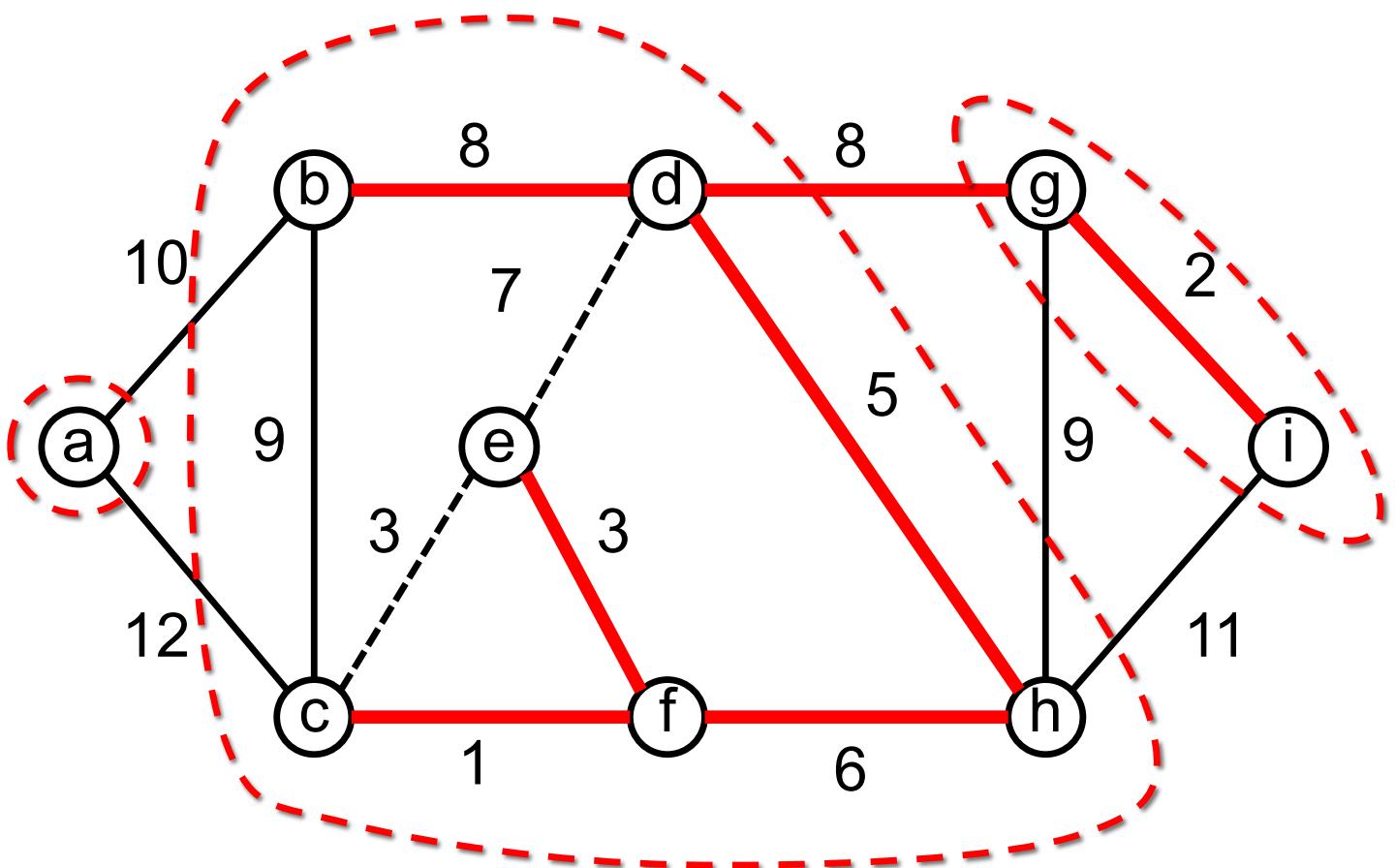
Kruskal's Algorithm - Example



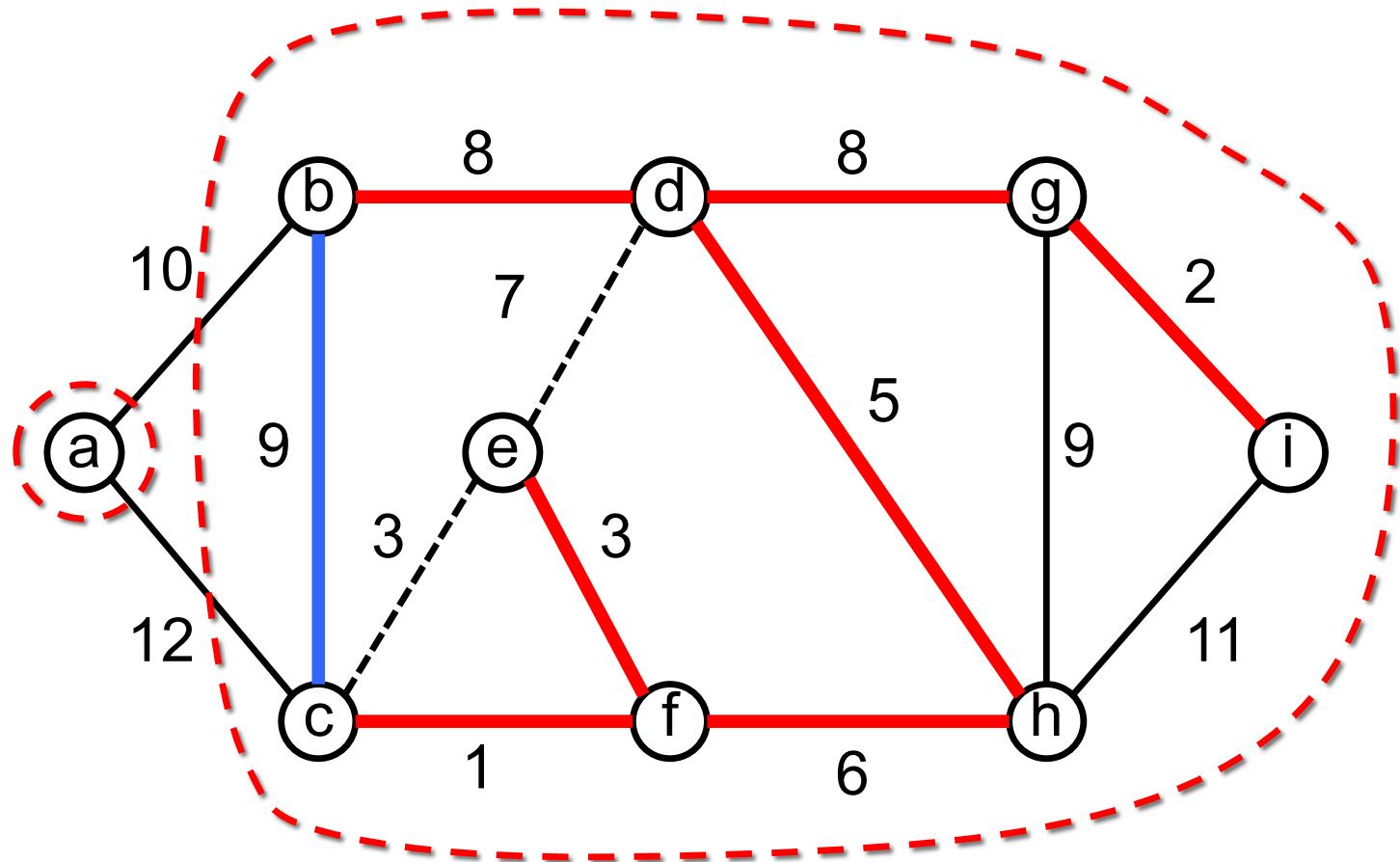
Kruskal's Algorithm - Example



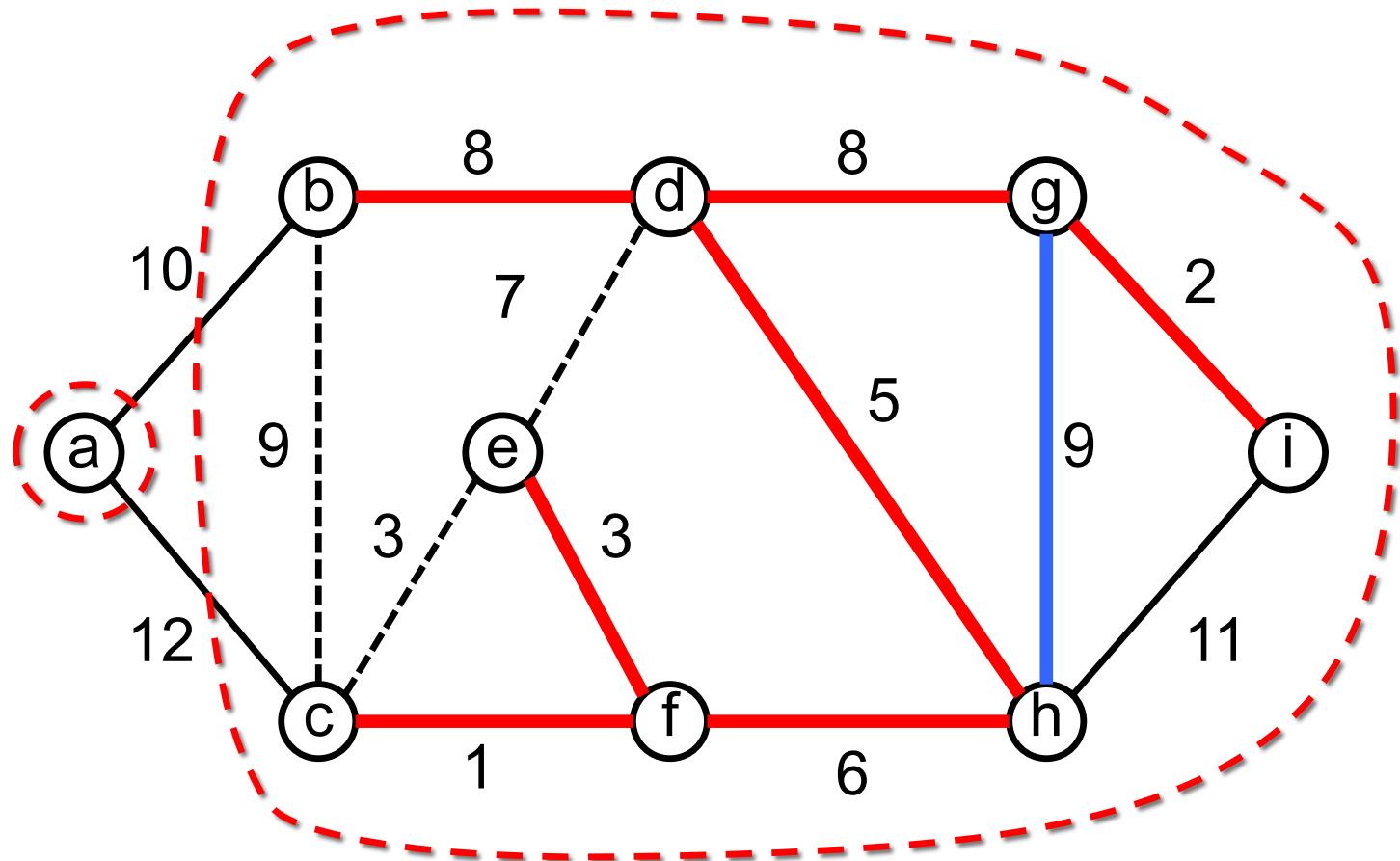
Kruskal's Algorithm - Example



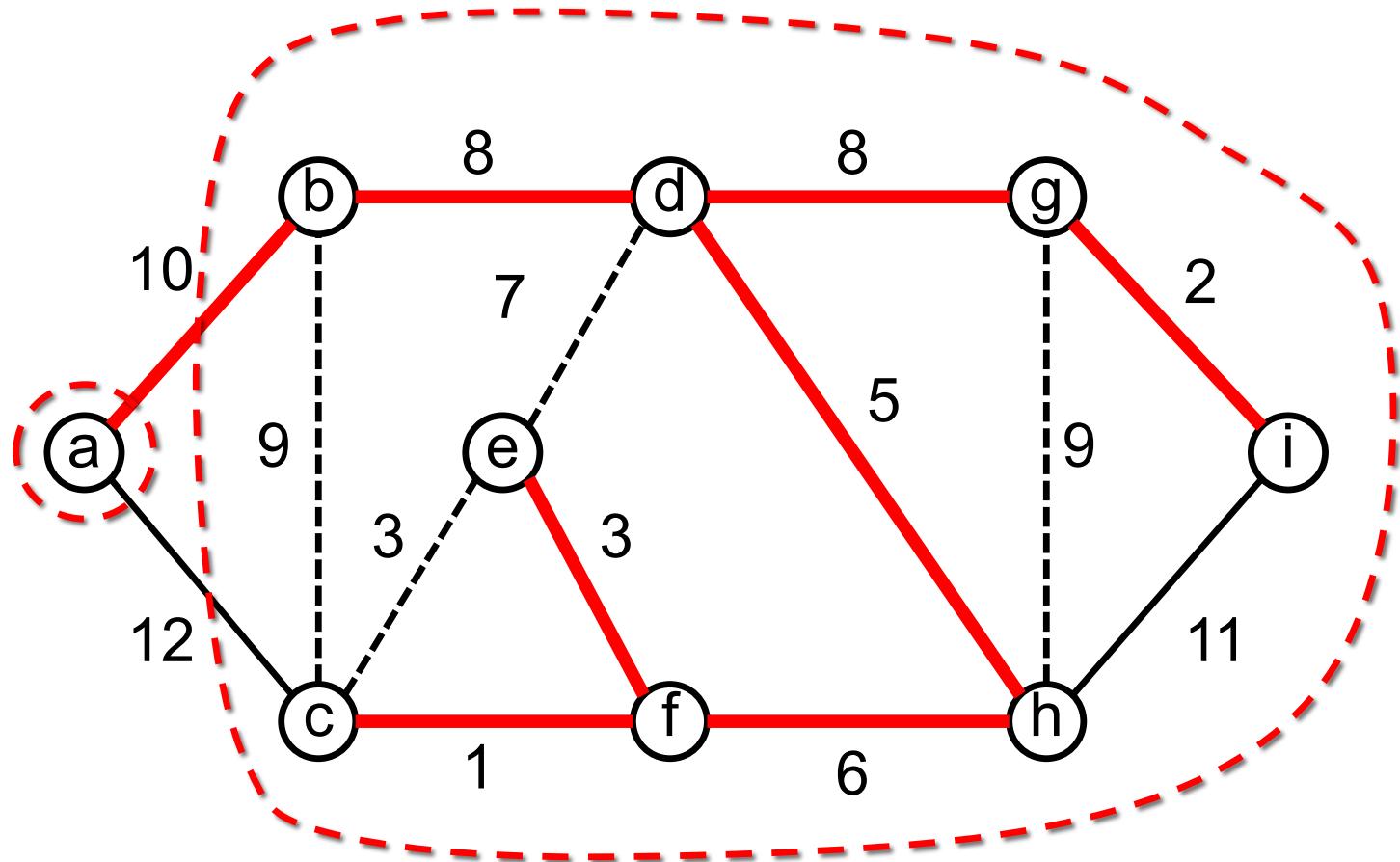
Kruskal's Algorithm - Example



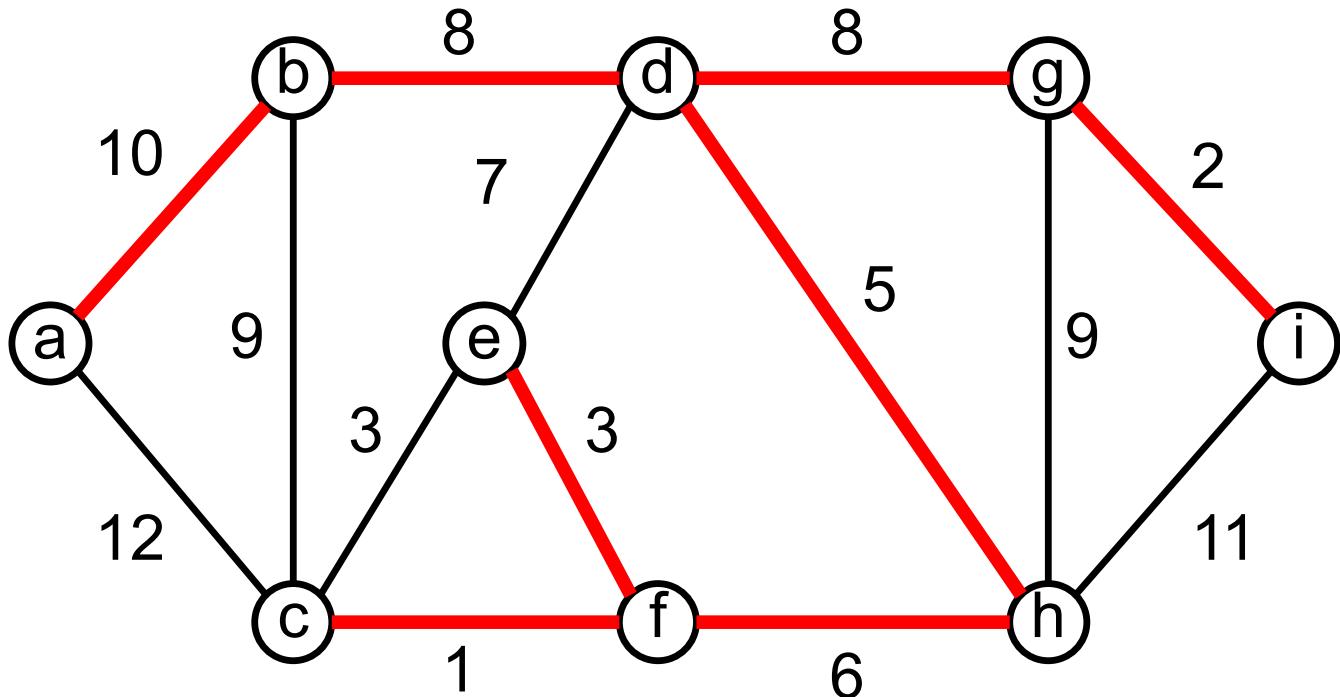
Kruskal's Algorithm - Example



Kruskal's Algorithm - Example



Kruskal's Algorithm - Example



Kruskal's Algorithm - Complexity

MST-KRUSKAL(G, w)

- 1 $A = \emptyset$
 - 2 **for** each vertex $v \in G.V$
 - 3 MAKE-SET(v)
 - 4 sort the edges of $G.E$ into nondecreasing order by weight w
 - 5 **for** each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
 - 6 **if** FIND-SET(u) \neq FIND-SET(v)
 - 7 $A = A \cup \{(u, v)\}$
 - 8 UNION(u, v)
 - 9 **return** A
- | | |
|-----|-------------------------------|
| 1 | O(1) |
| 2-3 | O(v) MAKE-SETS |
| 4 | O($E \lg E$) |
| 5-8 | O(E) FIND-SETS and UNIONs |

Kruskal's Algorithm - Complexity

- Initialize A : $O(1)$
- First **for** loop: $|V|$ MAKE-SETs
- Sort E : $O(E \lg E)$
- Second **for** loop: $O(E)$ FIND-SETs and UNIONs

Assuming union by rank and path compression, m find/union operations on a set with n objects is $O(m \cdot \alpha(n))$:

$$\Rightarrow O(E \cdot \alpha(V)) + O(E \cdot \log(E))$$

Moreover, $\alpha(V) = O(\log V) = O(\log E)$; ($|E| \geq |V| - 1$)

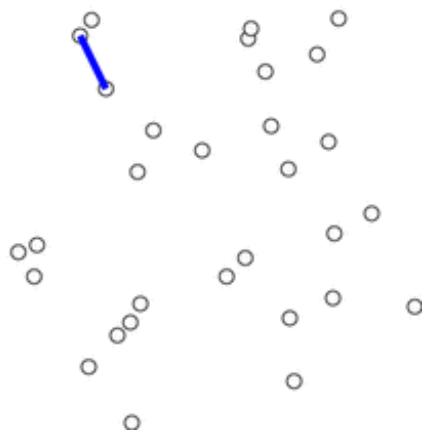
$$\Rightarrow O(E \cdot \log(E))$$

Since, $|E| \leq |V|^2 \Rightarrow \log|E| = O(2 \log V) = O(\log V)$

$$\Rightarrow O(E \cdot \log V)$$

MST– Prim's Algorithm

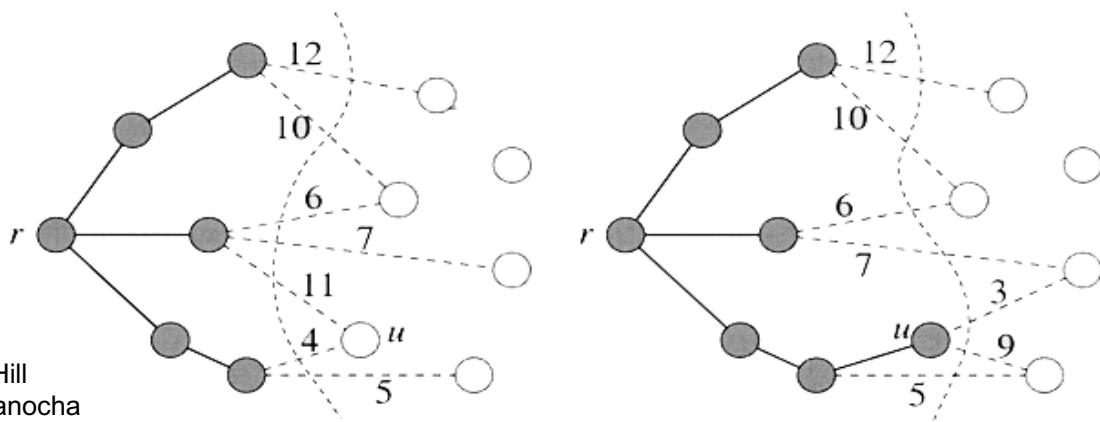
1. Builds **one tree**, so A is always a tree.
2. Starts from an arbitrary “root” r .
3. At each step, **adds a light edge** crossing cut $(V_A, V - V_A)$ to A .
 - Where V_A = vertices that A is incident on.



Differs from Kruskal's in that we grow a single supernode (tree) A instead of growing multiple ones (forest) at the same time

MST– Prim's Algorithm - Intuition

- Consider the set of vertices S currently part of the tree, and its complement $(V-S)$. We have a cut of the graph and the current set of tree edges A is respected by this cut.
- Which edge should we add next? *Light edge!*



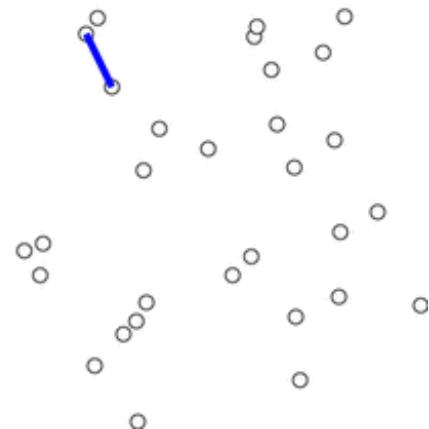
UNC Chapel Hill
Lin/Foskey/Manocha

Prim's Algorithm – Finding a light edge

1. Uses a **priority queue Q** to find a light edge quickly.
2. Each object in Q is a vertex in $V - V_A$.
3. Key of v has minimum weight of any edge (u, v) , where $u \in V_A$.
4. Then the vertex returned by Extract-Min is v such that there exists $u \in V_A$ and (u, v) is light edge crossing $(V_A, V - V_A)$.
5. Key of v is ∞ if v is not adjacent to any vertex in V_A .

Prim's Algorithm – Basics

- It works by adding leaves on at a time to the current tree.
 - Start with the root vertex r (it can be any vertex). At any time, the subset of edges A forms a single tree. $S = \text{vertices of } A$.
 - At each step, a light edge connecting a vertex in S to a vertex in $V - S$ is added to the tree.
 - The tree grows until it spans all the vertices in V .
- Implementation Issues:
 - How to update the cut efficiently?
 - How to determine the light edge quickly?



Prim's Algorithm – Implementation – Priority Queue

- Priority queue implemented using heap can support the following operations in $O(\lg n)$ time:
 - Insert (Q, u, key): Insert u with the key value key in Q
 - $u = \text{Extract_Min}(Q)$: Extract the item with minimum key value in Q
 - $\text{Decrease_Key}(Q, u, new_key)$: Decrease the value of u 's key value to new_key
- All the vertices that are *not* in S reside in a priority queue Q based on a key field. When the algorithm terminates, Q is empty.

Prim's Algorithm

```
Q := V[G];  
for each  $u \in Q$  do  
    key[u] :=  $\infty$   
     $\pi[u]$  := Nil;  
    Insert(Q,u)  
Decrease-Key(Q,r,0);  
while  $Q \neq \emptyset$  do  
     $u := \text{Extract-Min}(Q)$ ;  
    for each  $v \in \text{Adj}[u]$  do  
        if  $v \in Q \wedge w(u, v) < \text{key}[v]$  :  
             $\pi[v] := u$ ;  
            Decrease-Key(Q,v,w(u,v));
```

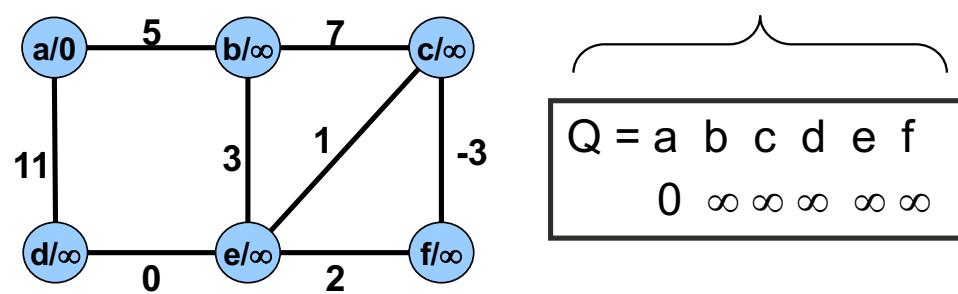
Complexity:

Using binary heaps: $O(E \lg V)$.
Building initial queue: $O(V)$.
Initialization: $O(V)$.
 V Extract-Min: $O(V \lg V)$.
 E Decrease-Key: $O(E \lg V)$.

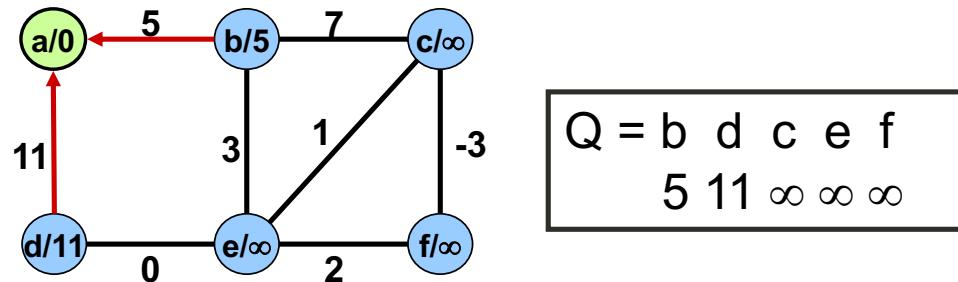
Using Fibonacci heaps:
 $O(E + V \lg V)$.

Notes: (i) $A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$. (ii) r is the root.

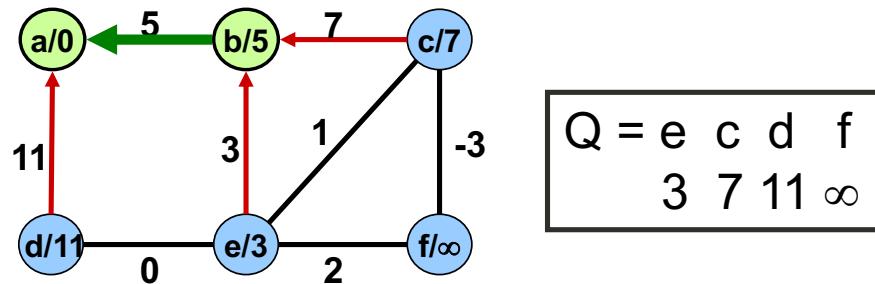
Prim's Algorithm - Example



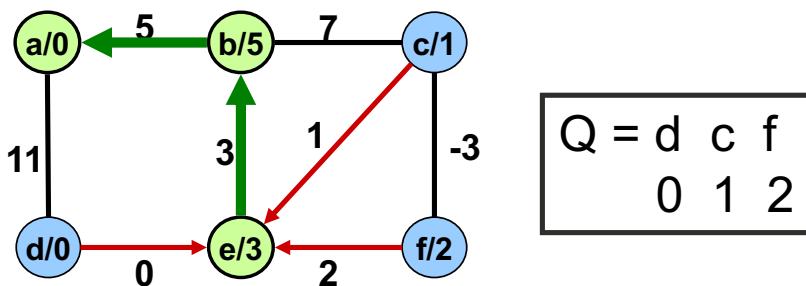
Prim's Algorithm - Example



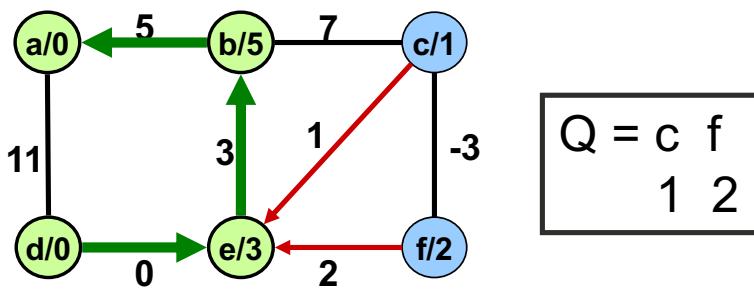
Prim's Algorithm - Example



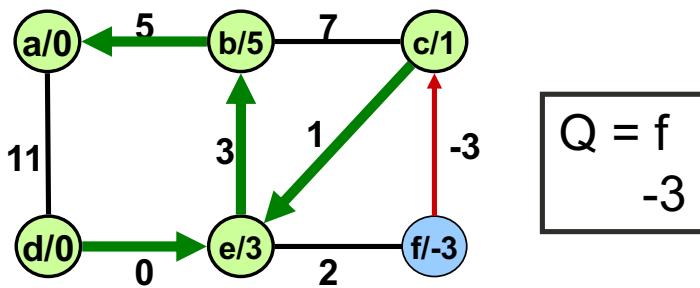
Prim's Algorithm - Example



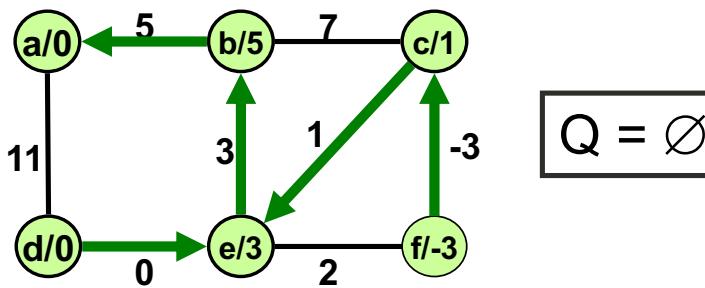
Prim's Algorithm - Example



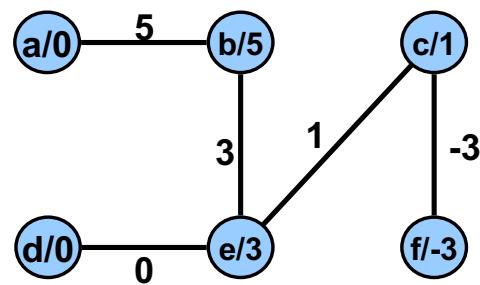
Prim's Algorithm - Example



Prim's Algorithm - Example



Prim's Algorithm - Example



Prim's Algorithm - Correctness

- Again, show that every edge added is a safe edge for A
- Assume (u, v) is next edge to be added to A .
- Consider the cut $(A, V-A)$.
 - This cut respects A
 - and (u, v) is the light edge across the cut
- Thus, by the Theorem 1, (u, v) is safe.

Prim's Algorithm – Time complexity

- Initialization: $O(V)$
 - Extract the light edge from the queue: $O(V \log V)$
 - Relax the neighbour edges: $O(E \log V)$
- $\Rightarrow O(V \log V + E \log V) = O(E \log V)$ // same as Kruskall
- Note: Using Fibonacci heaps, we can obtain $O(E + V \log V)$.

Outline

- Graphs.
 - Introduction.
 - Topological Sort. / Strong Connected Components
 - Network Flow 1.
 - Introduction
 - Ford-Fulkerson
 - Network Flow 2.
 - Min-cuts
 - Shortest Path.
 - Minimum Spanning Trees.
 - Bipartite Graphs.

