# MINIMUM SPANNING TREES

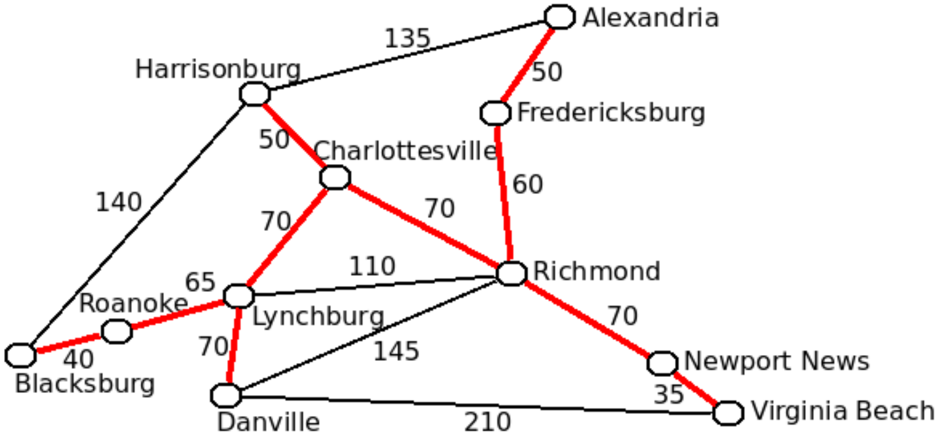
CS340

# Minimum Spanning Tree

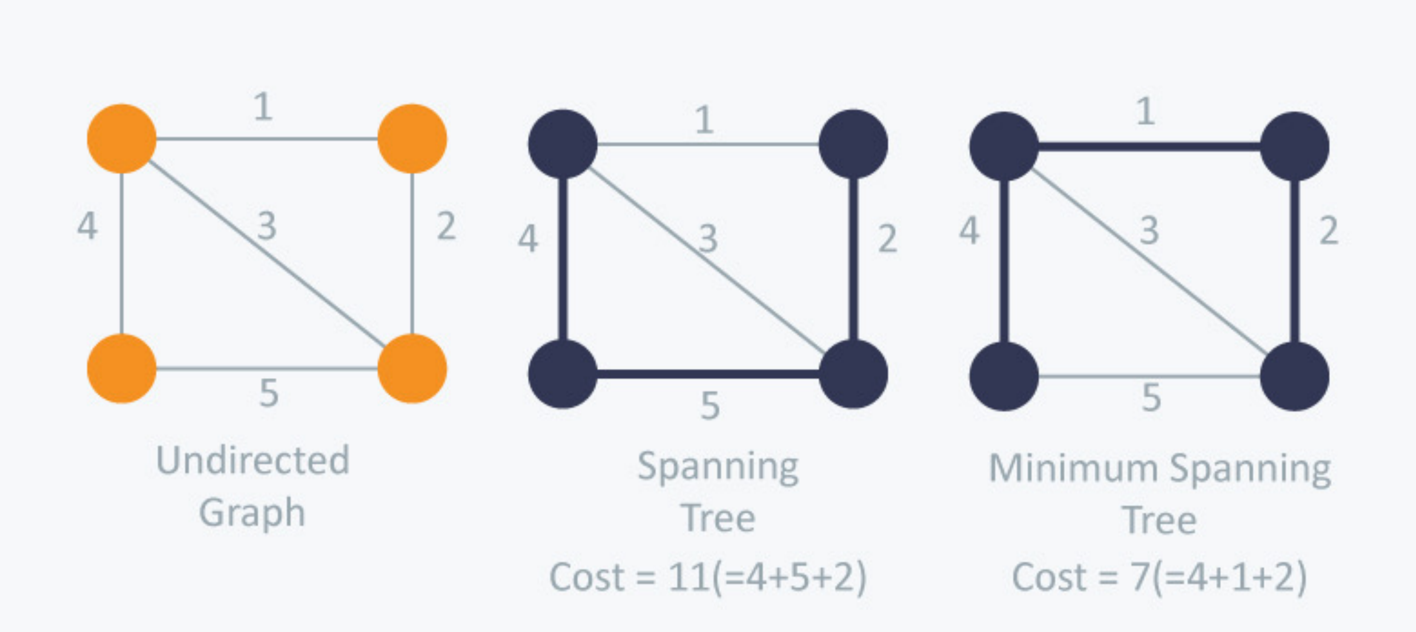
- Given connected graph G with positive edge weights, find a minimum weight set of edges that connects all of the vertices.

# Examples

- Use the least amount of wire to connect a set of pins on a circuit board
- Use the least amount of road to connect every house in a town
- Running electrical wires to electrify many cities



# MST Example


1
4
3
2
5
Undirected Graph
1
4
3
2
5
Spanning Tree
Cost = 11(=4+5+2)
1
4
3
2
5
Minimum Spanning Tree
Cost = 7(=4+1+2)

# Minimum Spanning Trees

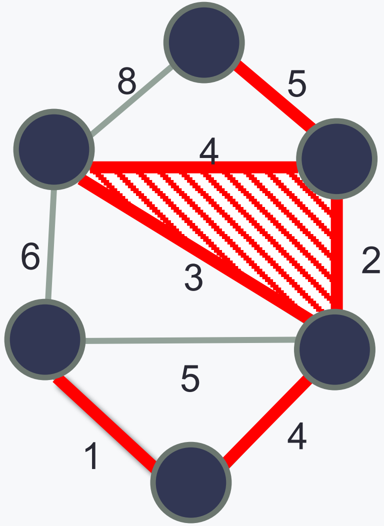
- MSTs have three primary properties
  - It is a tree
    - Connected with N-1 edges
    - No cycles
    - Any 2 vertices are connected by exactly one path
  - It spans
    - All vertices are connected
  - It has minimum weight
    - If T is a minimum spanning tree, $w(T) = \sum_{(u,v)\in T} w(u,v)$ is minimized.

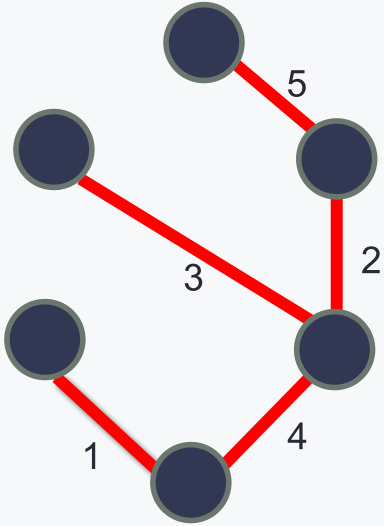
# Kruskal's Algorithm, v1

- **Input**: A weighted connected graph $G = (V, E)$
- **Output:** $E_T$ , the set of edges composing a minimum spanning tree of $G$

sort $E$ in nondecreasing order of edge weights $w(e_1) \leq \ldots \leq w(e_{|E|})$
$E_T = \emptyset$; $ecounter = 0$; $k = 0$;
**while** $ecounter < |V| - 1$ **do**
    $k = k + 1$;
    **if** $E_T \cup \{e_k\}$ is acyclic
        $E_T = E_T \cup \{e_k\}$; $ecounter{+}{+}$;
**return** $E_T$

# Kruskal's Algorithm



# Time complexity

- We don't know.
- We glossed over "if the graph remains acyclic."
- Sorting Edges = $O(E \lg E)$
- How to determine if a graph is acyclic?
  - Do a DFS and check for back edges?
  - DFS time complexity is $\Theta(V+E)$, we would have to do one each time we attempt to add an edge. In the worst case this is E times.
  - So time complexity is $O(E(V+E)) = O(EV + E^2)$

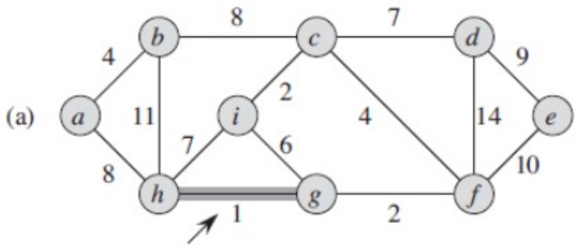
# Another look at Kruskal's Algorithm

- Initially, we have a forest of trivial (one-node) trees.
- On each iteration, consider the next edge *(u, v)* from the sorted list of the graph's edges.
  - Find the trees containing the vertices *u* and *v*
  - *I*f these trees are not the same, unite them in a larger tree by adding the edge *(u, v)*.
- The final forest consists of a single tree, which is a minimum spanning tree of the graph.
- Qualifies as a greedy algorithm because at each step it adds to the forest an edge of least possible weight.

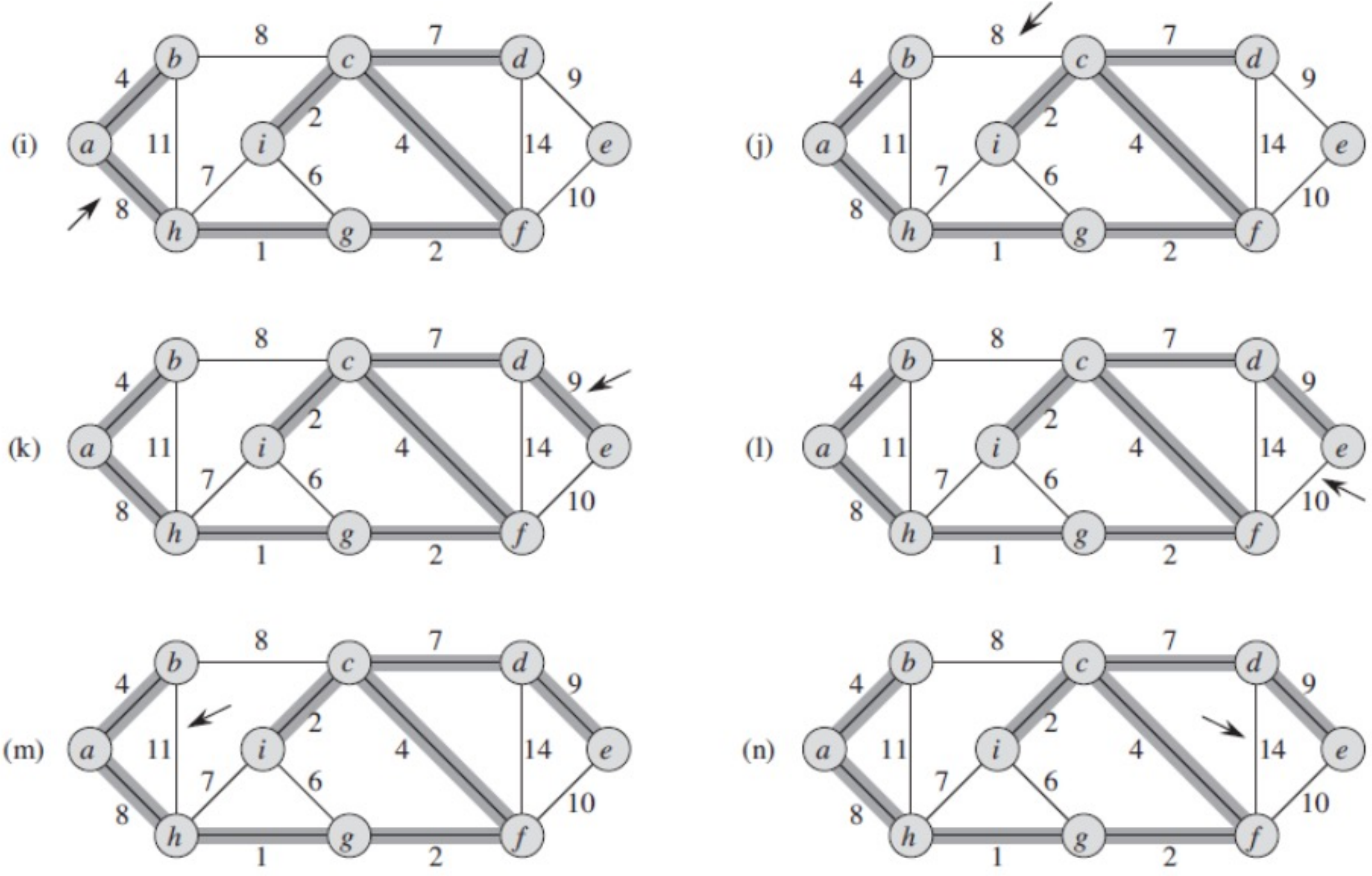
# Kruskal's Algorithm

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

# Kruskal




- Sorted Edges:
- (g,h)
- (c,i)
- (f,g)
- (a,b)
- (c,f)
- (g,i)
- (c,d)
- (h,i)
- (a,h)
- (b,c)
- (d,e)
- (e,f)
- (b,h)
- (d,f)


(i)
(j)
(k)
(l)
(m)
(n)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

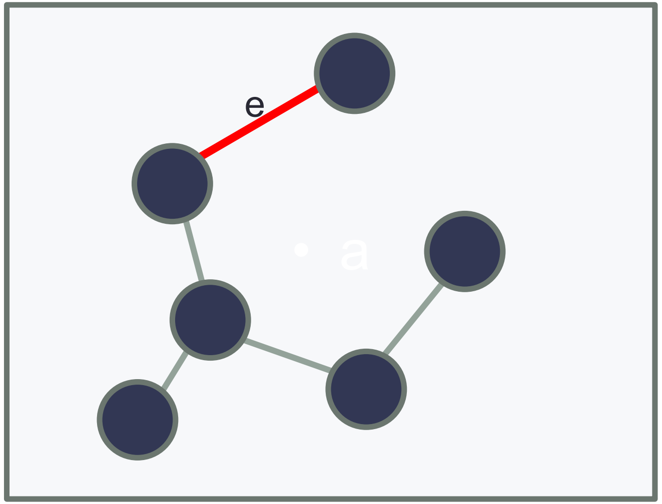
# Kruskal Complexity

- 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 efficient implementation of disjoint-set data structure that uses union by rank and path compression:
- $O((V+E)\ \alpha(V)) + O(E \lg E)$
- Since G is connected, $|E| \geq |V| - 1 \Rightarrow O((E)\ \alpha(V)) + O(E \lg E)$
- $\alpha\ |V| = O(\lg V) = O(\lg E)$
- Therefore, total time is $O(E \lg E)$
- $|E| \leq |V|^2 \Rightarrow \lg |E| = O(2 \lg V) = O(\lg V)$
- **$O(E \lg V)$**

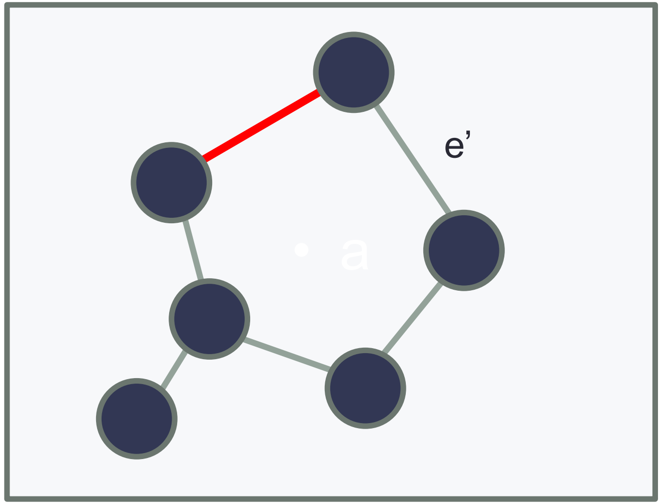
T:
MST Returned by Kruskal’s Algorithm

S:
An MST with lower weight than T
W(T) > W(S)




**S’:**
**W(e) ≤ W(e’)**

**W(S`) ≤ W(S)**




**When S’ = T:**
**W(T) ≤ W(S)**A contradiction

# Proof

1. Let T be the MST returned by Kruskal's algorithm, and S be an MST of the same graph, but with lower weight: i.e. W(T) > W(S).
2. Find e, the smallest edge that is in T that is not in S.
3. S ∪ {e} creates a cycle C in S.
4. Cycle C contains an edge e' that is not in T.
5. Replacing e' in S with e results in spanning tree S' = (S\{e'} ∪{e})
6. W(e) <= W(e'); therefore W(S') <= W(S).
7. S' has one more edge in common with T than S did.
8. We can repeat this process until S'=T; at that point W(T) <= W(S)