CSCI 4470 Algorithms

Part VI Graph Algorithms Notes

- 20 Elementary Graph Algorithms
- **21 Minimum Spanning Trees**
- 22 Single-Source Shortest Paths
- 23 All-Pairs Shortest Paths
- 24 Maximum Flow
- 25 Matchings in Bipartite Graphs

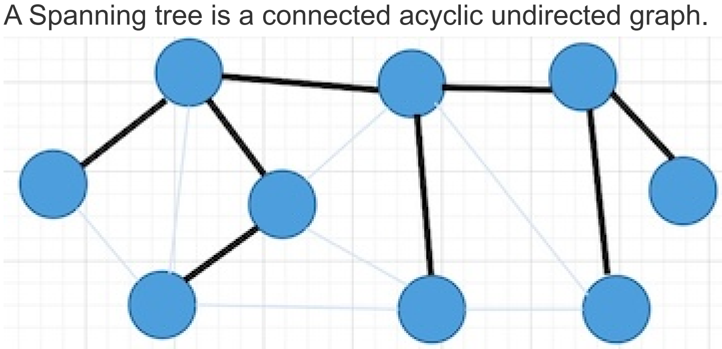
Chapter 21: Minimum Spanning Trees

```
21 Minimum Spanning Trees
  21.1 Growing a minimum spanning tree
  21.2 The algorithms of Kruskal and Prim
```

SPANNING TREES

SPANNING TREES ARE GRAPHS:

- A spanning tree is a connected, undirected, acyclic graph that includes all the vertices of the original graph.
- A set of spanning trees is called a forest.
- A graph G is a spanning tree if and only if there is exactly one path between every pair of vertices in G.
- A **DAG** is a directed acyclic graph.
- A Spanning tree is a connected acyclic undirected graph.



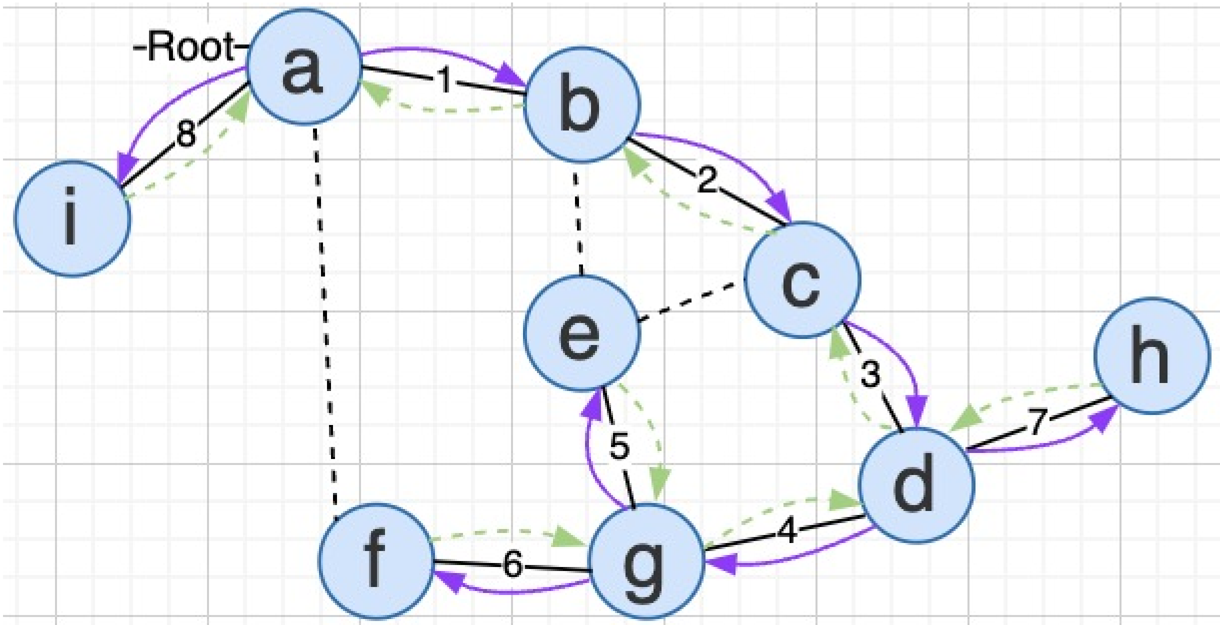
Tree: an undirected connected graph without cycles.

Observations about undirected graphs

1. A connected undirected graph with n vertices must have at least n - 1 edges.
2. A connected undirected graph with n vertices and exactly n - 1 edges cannot contain a cycle.

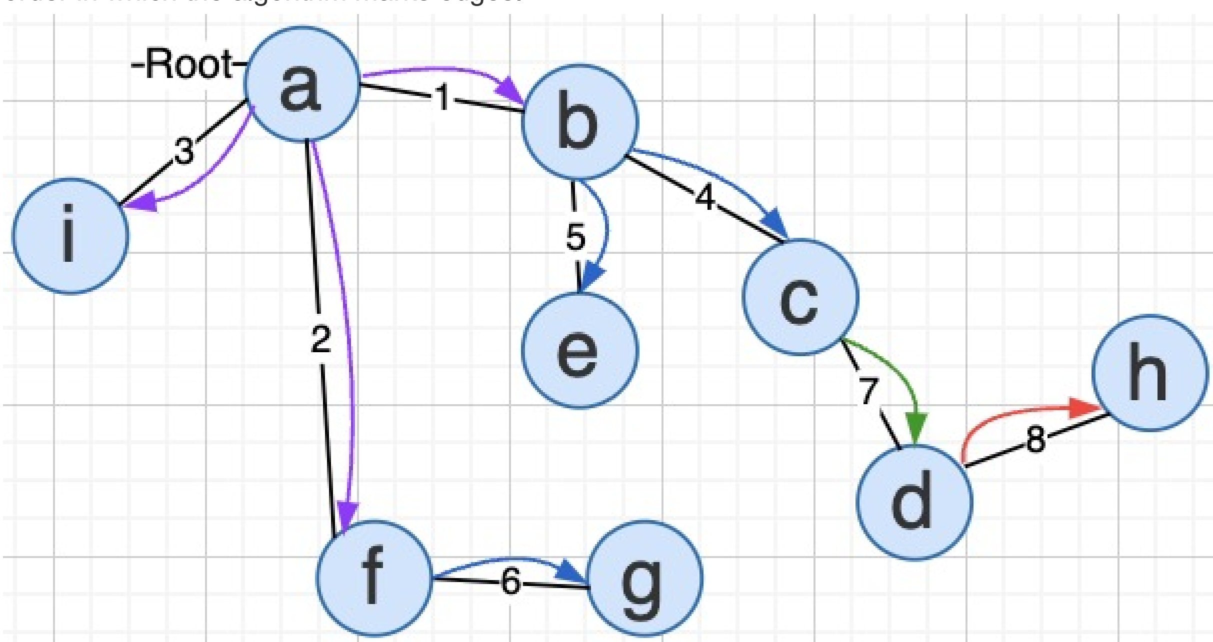
Spanning Tree by DFS Example The DFS Spanning Tree rooted at vertex `a` for the graph

- The DFS spanning tree algorithm visits vertices in this order: `a, b, c, d, g, e, f, h, i`. Numbers indicate the order in which the algorithm marks edges.

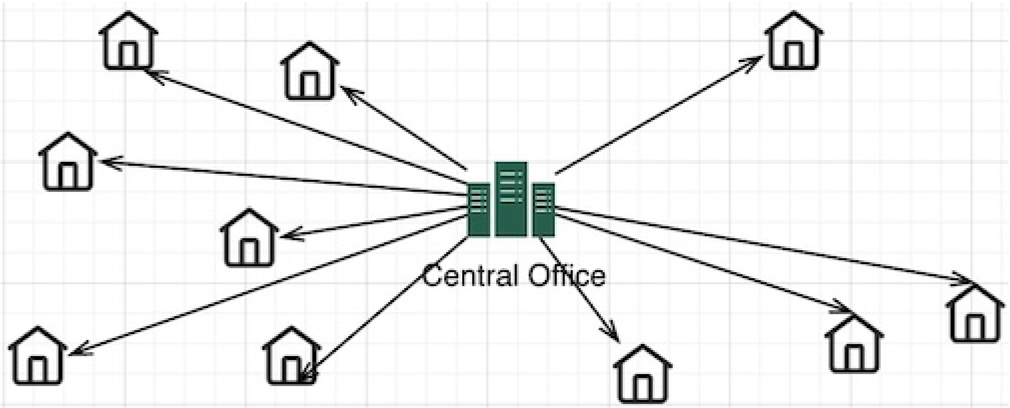


Spanning Tree by BFS Example The BFS Spanning Tree rooted at vertex `a` for the graph

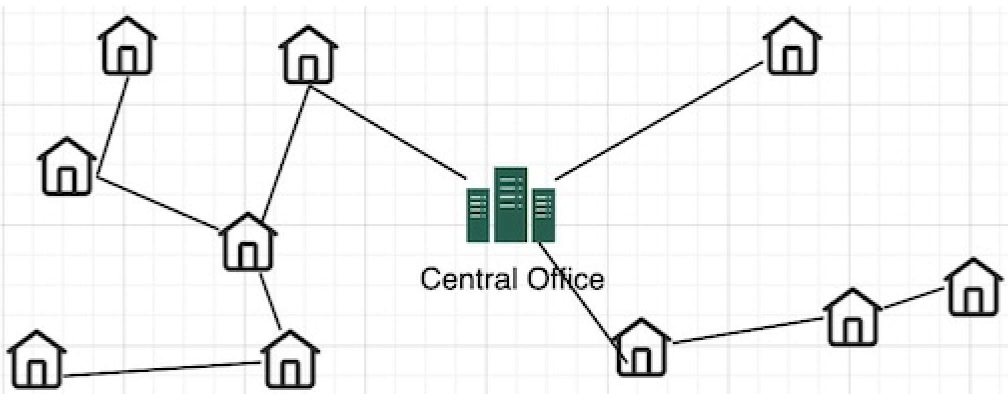
- The BFS spanning tree algorithm visits vertices in this order: `a, b, f, i, c, e, g, d, h`. Numbers indicate the order in which the algorithm marks edges.



Problem: Laying Telephone Wire (Expensive)



Wiring: Better Approach (Minimum Spanning Tree)



Minimize the total length of wire connecting the customers (MST)

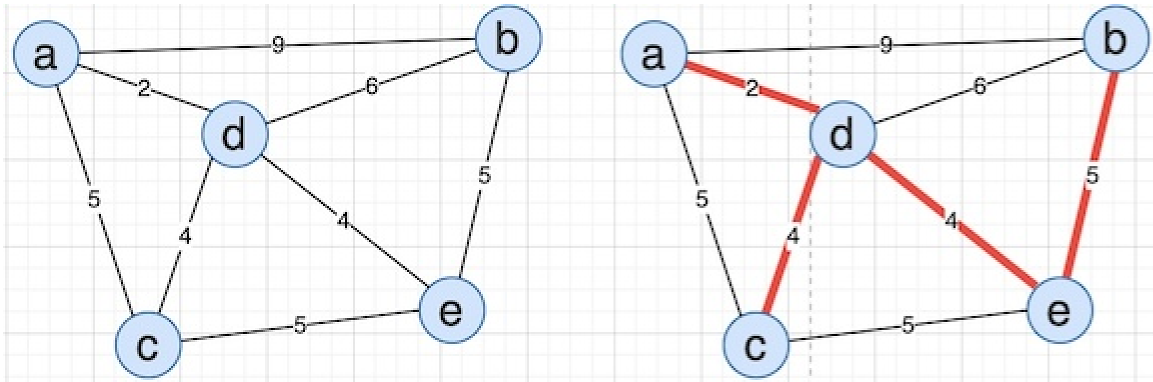
Minimum Spanning Tree (MST)

A **minimum spanning tree (MST)** is a special type of spanning tree that minimizes the total edge weight in a **Weighted, Connected, Undirected Graph** G.

- It is a tree, meaning it is acyclic and connected.
- It includes all vertices V of the graph, and has $|V| - 1$ edges.
- The total weight (cost) of its edges is minimized across all possible spanning trees of the graph.
- It may not be unique; there can be multiple MSTs for a given graph.

How to generate a MST

- Two Common Algorithms: Kruskal's algorithm, and Prim's Algorithm



- The total weight (cost) of the example is $2+4+4+6=16$

MINIMUM SPANNING TREES

Given a connected, undirected graph $G=(V,E)$ with edge **weights** $w(u,v)$, find acyclic subset $T \subseteq E$ that connects all vertices and minimizes total weight

- T is an MST, T may not be unique
- T will have $|V|-1$ edges, Having more than $|V|-1$ edges would mean there is a cycle in T, and T is acyclic by definition

T is called a **Minimum Spanning Tree**

- Any subset of E containing $|V|-1$ edges that connects all vertices is called a spanning tree
- T is minimum because it has minimum weight

Having fewer than $|V|-1$ edges would mean (V,T) is not connected, and T is connected by definition

MST and TWO ALGORITHMS

Kruskal’s and Prim’s, are Greedy and have same general format, For finding a MST

- Add safe edges to A one by one until $|A|=|V|-1$
- Given a subset A of E, where A is a subset of some minimum spanning tree, we say (u,v) is safe for A if $A \bigcup \{(u,v)\}$ is also a subset of a minimum spanning tree

Generic Properties of MSTs and then discuss Kruskal’s and Prim’s algorithms

```
GENERIC-MST(G,w)
A = ∅
while A does not form a spanning tree
     find an edge (u.v) that is safe for A
     A = A ⋃ {(u.v)}
return A
```

- Challenge: determining which edges are safe for A

- The `GENERIC-MST` function outlines a generic approach for finding a Minimum Spanning Tree (MST) in a weighted graph G with a weight function w. Here's a concise and comprehensive explanation:

Generic-MST(G, w) Explanation

1. Initialization:

- `A = ∅` : Start with an empty set A, which will eventually contain the edges of the MST.

2. Building the MST:

- `while A does not form a spanning tree` : Continue the process as long as A does not yet form a spanning tree.
- A spanning tree connects all the vertices in G with the minimum number of edges.

3. Selecting Safe Edges:

- `find an edge (u, v) that is safe for A` : Choose an edge (u, v) that can be added to A without creating a cycle and ensures the minimality of the spanning tree.
- A safe edge connects two distinct components of the spanning forest in A and is of minimum weight among all such edges.

4. Updating the MST:

- `A = A ⋃ {(u, v)}` : Add the selected safe edge to A.

5. Completion:

- `return A` : Once A forms a spanning tree, return it as the MST of G.

Complexity and Properties

- The complexity depends on how the safe edge is found in step 3.
- This generic algorithm underlies specific MST algorithms like Kruskal's and Prim's, which provide efficient ways to find safe edges.
- The algorithm ensures that the final set A is a Minimum Spanning Tree of G, connecting all vertices with the minimum possible total edge weight.

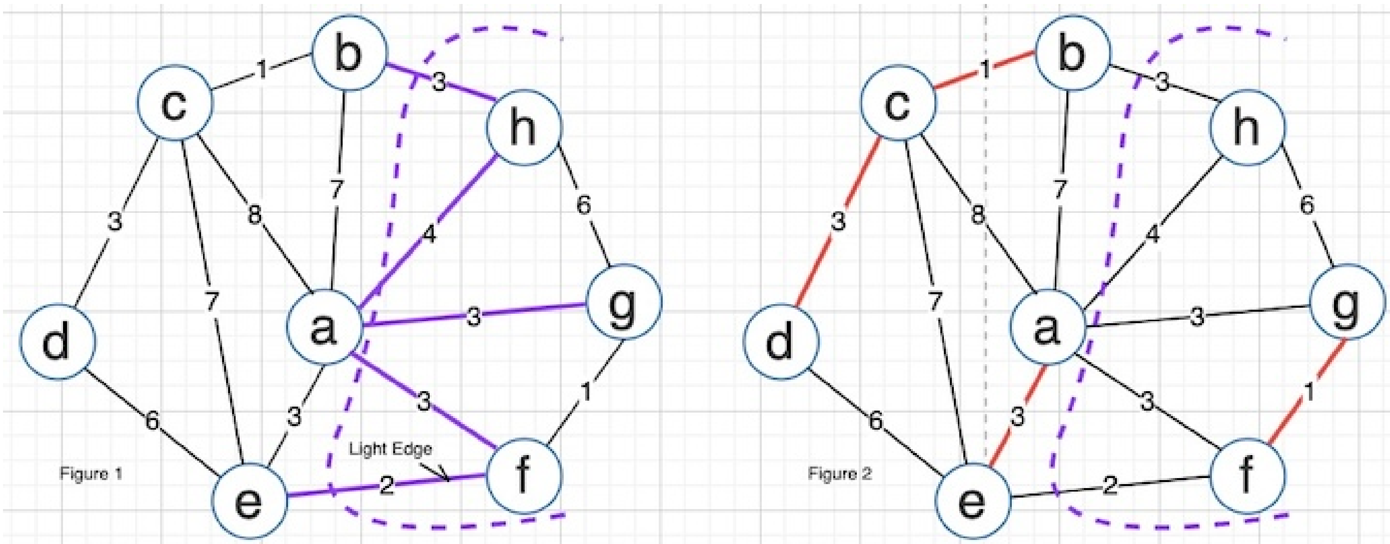
Before Finding Safe Edges

Cut, A **cut** $(S, V - S)$ in a graph G is a partition of the vertex set V into two disjoint subsets S and $V - S$.

- Example: $S = \{a, b, c, d, e\}, V - S = \{g, h, f\}$.

Crossing Edge, An Edge (u, v) is said to **cross the cut** if one endpoint is in S and the other is in $V - S$.

- Example of crossing edges: `{b, h}` , `{a, h}` , `{a, g}` , `{a, f}` , `{e, f}` .



Cut Respecting A, A cut $(S, V - S)$ **respects a set of edges** A if no edge in A crosses the cut.

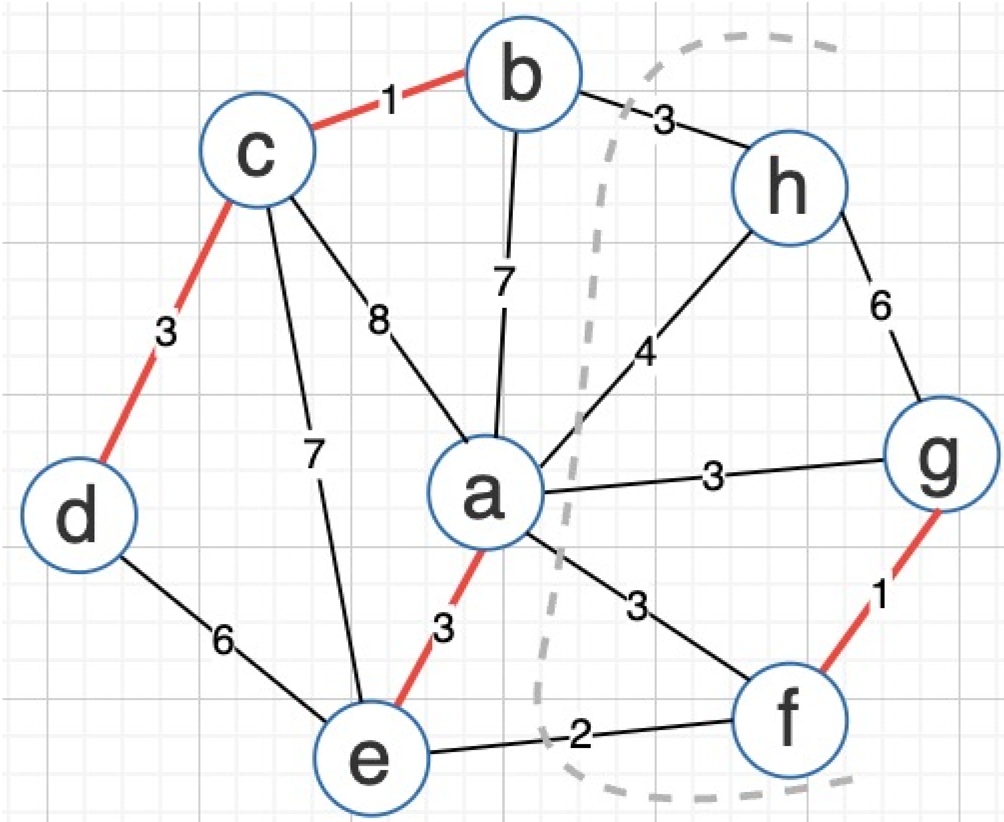
- A is the set of red edges, $A \subseteq E$, and $A \subseteq \text{MST}$.
- Both endpoints of each edge in A are either in S or in $V - S$.

Light Edge, A **light edge** crossing a cut is an edge with the minimum weight among all edges crossing the cut.

- Light edges may not be unique. For example, edge `{e, f}` has weight 2, which is lighter than the other crossing edges.
- A light edge is safe to be added to the MST.

FINDING SAFE EDGES

Theorem: If a cut $(S, V - S)$ respects a set A, and edge (u, v) is the lightest edge crossing the cut, then (u, v) is safe to add to A.



Proof by contradiction:

- Assume MST T exists where $A \subseteq T$ and $(u, v) \notin T$. Let P be the unique path from u to v in T, and identify an edge (x, y) in P that crosses the cut $(S, V - S)$.
- Construct MST T' by replacing (x, y) with (u, v), which does not increase the total weight since (u, v) is lighter.

Completing the Proof:

REPLACE(x, y) **with** (u, v), Claim: T' is a spanning tree and $w(T') \leq w(T)$.

- Adding (u, v) to T creates a cycle. Removing (x, y) from T breaks the cycle, leaving a spanning tree T'.
    - Since (u, v) is lighter than (x, y), $w(T')$ does not exceed $w(T)$.
- Let $T' = T - \{(x, y)\} \cup \{(u, v)\}$. For any two vertices q, r, a path in T' exists, ensuring it's a spanning tree.
- If the path from q to r in T used (x, y), then in T', the path will use (u, v) instead, maintaining connectivity.
- Therefore, T' is a spanning tree, and since (u, v) is the lightest edge crossing the cut, $w(T')$ is less than or equal to $w(T)$, making (u, v) a safe edge.

2 THINGS TO SHOW ABOUT T

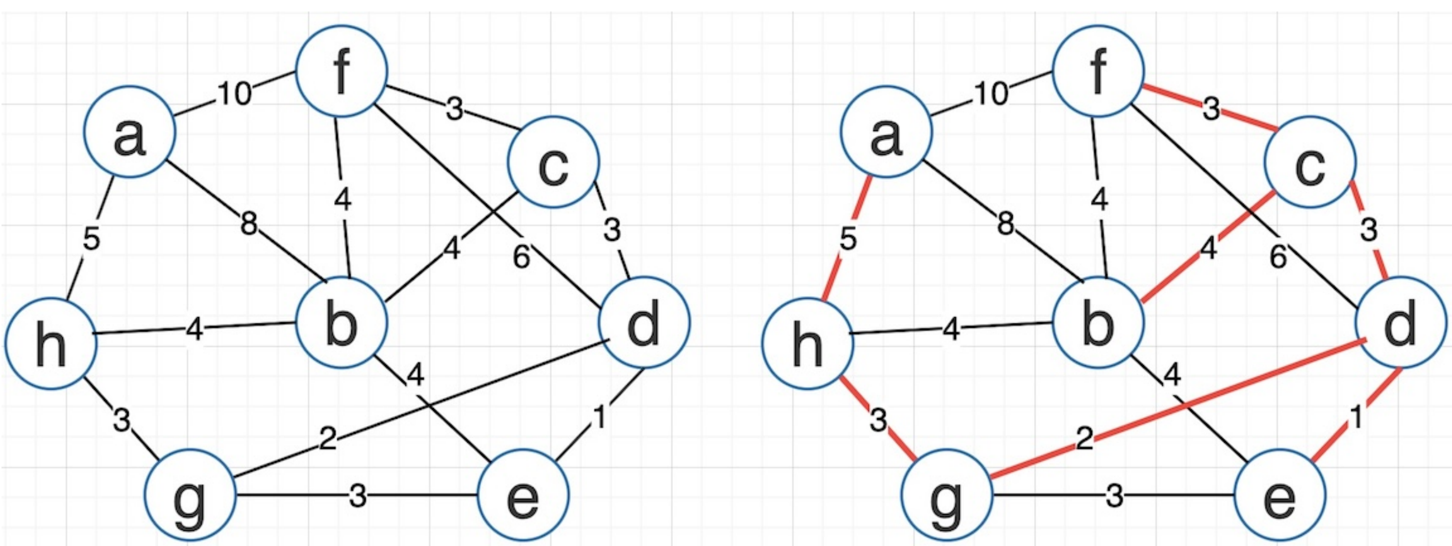
path from q to r in T'
W(T) ≥ w(T)

There exists a path from p to q inT’
If the path from p to q in T did not include (x,y), then the same path still exists in T’
If the path from p to q included (x,y), then the path in T’ will get from x to y by taking the other edges in the cycle. The rest of the path will be the same as the path in T.
W(T) ≥ W(T’)
W(T’) = W(T) – w(x,y) + w(u,v) ≤ W(T)
Because w(x,y) ≥ w(u,v)

MST and KRUSKAL(G,W) Algorithm

```
KRUSKAL(G,W) Algorithm
A = ∅
for each v in V
   MAKE-SET(v) // uses the disjoint forest data structure
sort the edges of E in nondecreasing order by w
for each (u,v) ∈ G.E // consider in sorted order
   if FIND-SET(u) ≠ FIND-SET(v)
       A = A ∪ {(u,v)}
       UNION(u,v)
return A
```

Kruskal’s Example, Given the connected, undirected graph $G(V, E)$, $V = \{a, b, c, d, e, f, g, h\}$, and $E = \{\}$



- Select first |V|-1 edges which do not generate a cycle

Edge	d_v		Edge	d_v	
(d, e)	1	√	(b, e)	4	×
(d, g)	2	√	(b, f)	4	×

Edge	d_v		Edge	d_v	
(e, g)	3	×	(b, h)	4	×
(c, d)	3	√	(a, h)	5	√
(g, h)	3	√	(d, f)	6	
(c, f)	3	√	(a, b)	8	
(b, c)	4	√	(a, f)	10	

Total Weight(Cost) = $1 + 2 + 3 + 3 + 3 + 4 + 5 = 21$

Run Time, There are v `MAKE-SET()` , and E `FIND-SET()` & `UNION()` Operations, $O(V + E)\alpha(V)$

- $\alpha(V) \in O(\log(V))$, The Cost is $O(V + E)\log(V)$, Since $E \geq V - 1$, the Cost can be $O(E\log(V))$

MST and Prim(G,W,R) Algorithm

An implementation of Prim's algorithm, which is used to find the minimum spanning tree of a connected, undirected graph with weighted edges.

```
PRIM(G, W, r)
// r is root of spanning tree
for each u in G.V
    u.key = ∞    // Initialize all vertices' keys to infinity except the root
    u.π = NIL    // Initialize all vertices' parent pointers to NIL
r.key = 0        // Set the key of the root vertex to 0
Q = G.V          // Initialize the priority queue with all vertices
while Q ≠ 0
    u = EXTRACT-MIN(Q)    // Remove and return the vertex with the smallest key from the priority queue
    for each v ∈ G.adj[u]     // For each neighbor v of u
        if v ∈ Q and w(u,v) < v.key      // If v is still in the priority queue and the edge (u,v) has a smaller weight than v's current key
            v.π = u    // Update v's parent to u
            v.key = w(u,v) //decrease v's key    // Update v's key to the weight of edge (u,v)
            Decrease-Key(Q, v, w(u, v))    // Update v's position in the priority queue since its key has decreased
```

1. Initialization: Set all vertices' keys to ∞ and parents to NIL. Root's key is 0.
2. Priority Queue: Maintain vertices in Q, extracting the minimum key vertex in each iteration.
3. Building MST: For each extracted vertex u, update adjacent v if edge (u, v) offers a smaller key.
4. Termination: When Q is empty, MST is defined by the parent pointers.
`DECREASE-KEY(Q, v)` ensures Q maintains order after key updates.

Steps for Solving MST by Prim's Algorithm:

1. **Initialization**: Start with an empty set A to store the edges of the MST. Initialize all vertices' keys to infinity and their parent pointers to NIL. Choose a starting vertex and set its key to 0.
2. **Priority Queue**: Add all vertices to a priority queue Q.
3. **Building MST**: While Q is not empty:
   - Extract the vertex u with the minimum key from Q.
   - Add u's parent edge to A if u is not the starting vertex.
   - For each neighbor v of u:
     - If v is in Q and the weight of edge (u, v) is less than v's key, update v's parent to u and decrease v's key.
4. **Result**: Once Q is empty, A contains the edges of the MST.