Lecture 15

Minimum Spanning Trees

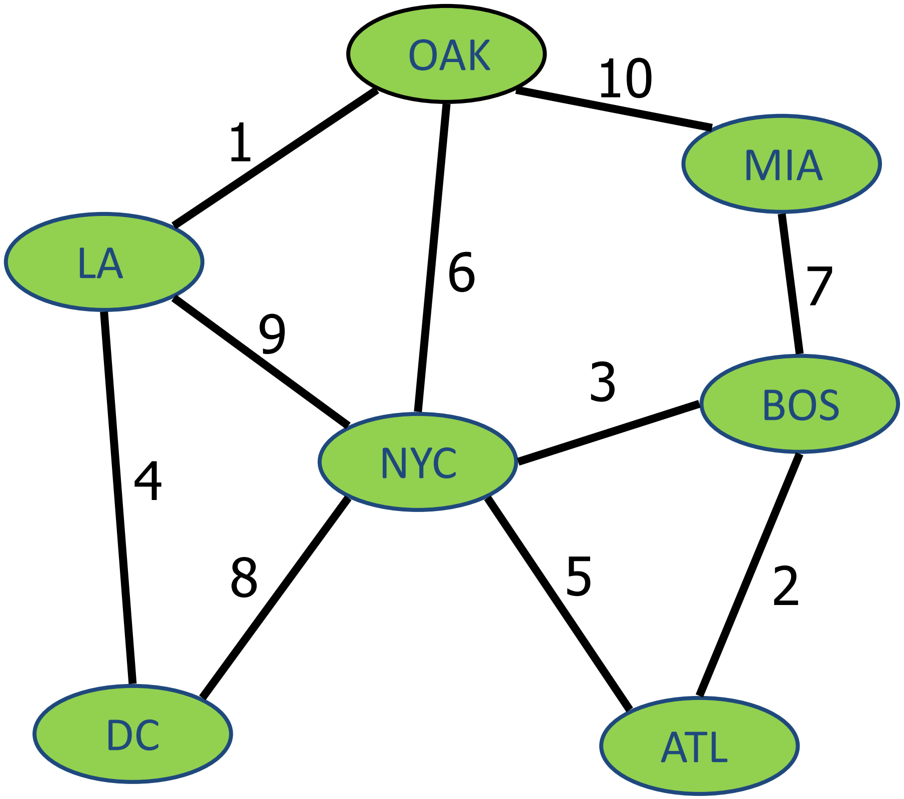
CS 161 Design and Analysis of Algorithms

Ioannis Panageas

Spanning Tree

Definition: We are given an undirected, weighted graph G. A spanning tree of G is a connected acyclic (tree) subgraph of G that includes all the vertices of G (spanning).

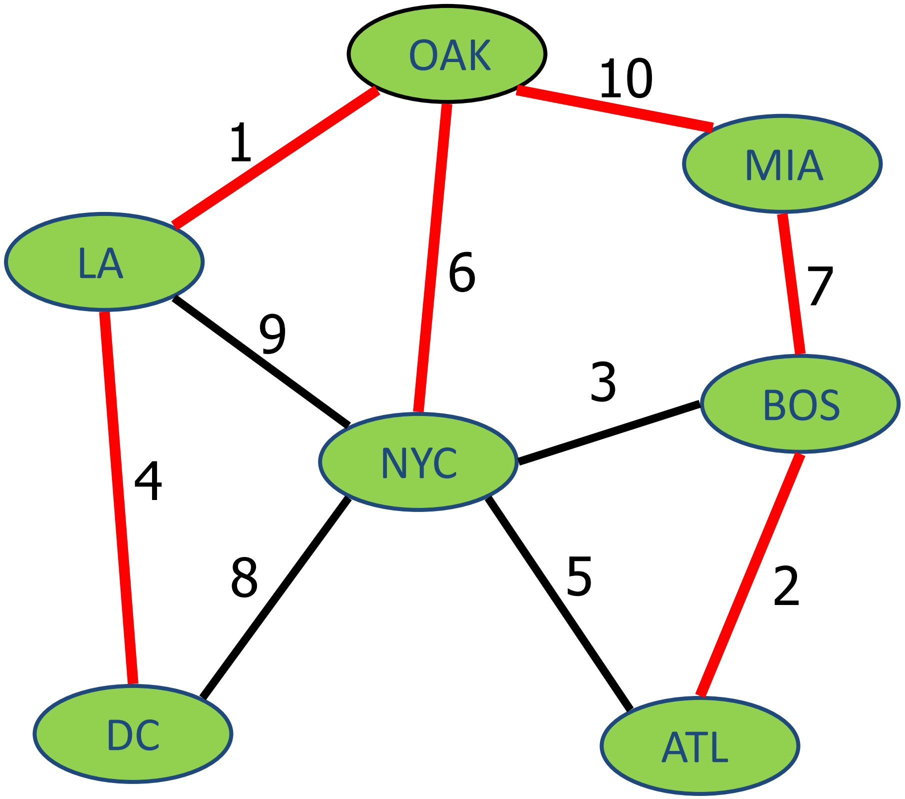
Example:



Spanning Tree

Definition: We are given an undirected, weighted graph G. A spanning tree of G is a connected acyclic (tree) subgraph of G that includes all the vertices of G (spanning).

Example:



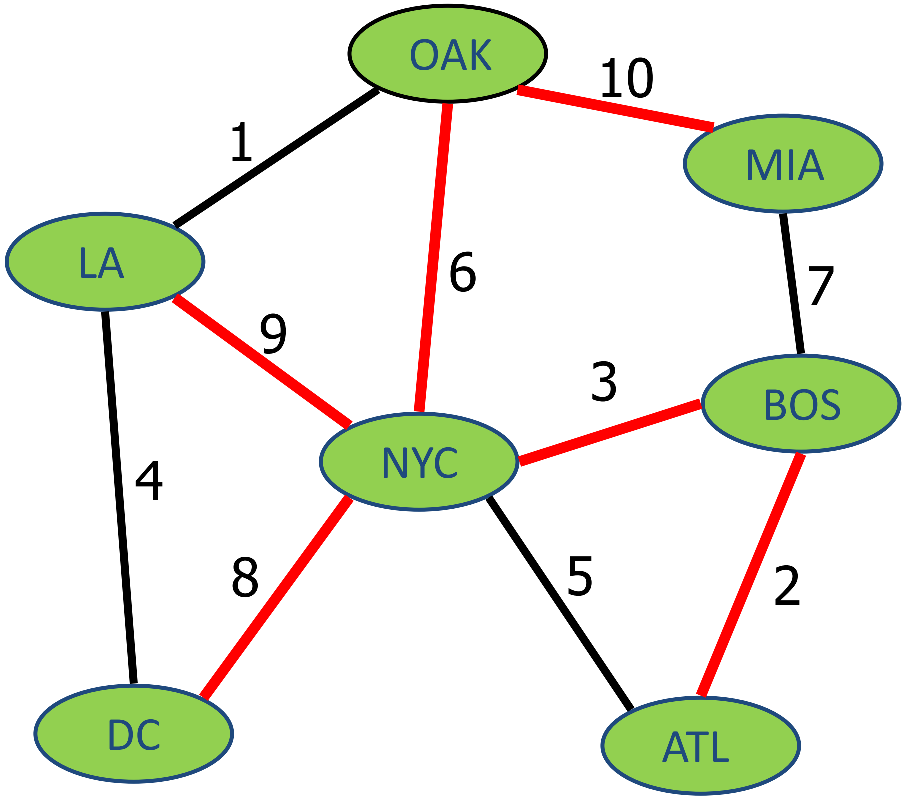
Total cost

$4+1+10+6+7+2 = 30$

Spanning Tree

Definition: We are given an undirected, weighted graph G. A spanning tree of G is a connected acyclic (tree) subgraph of G that includes all the vertices of G (spanning).

Example:



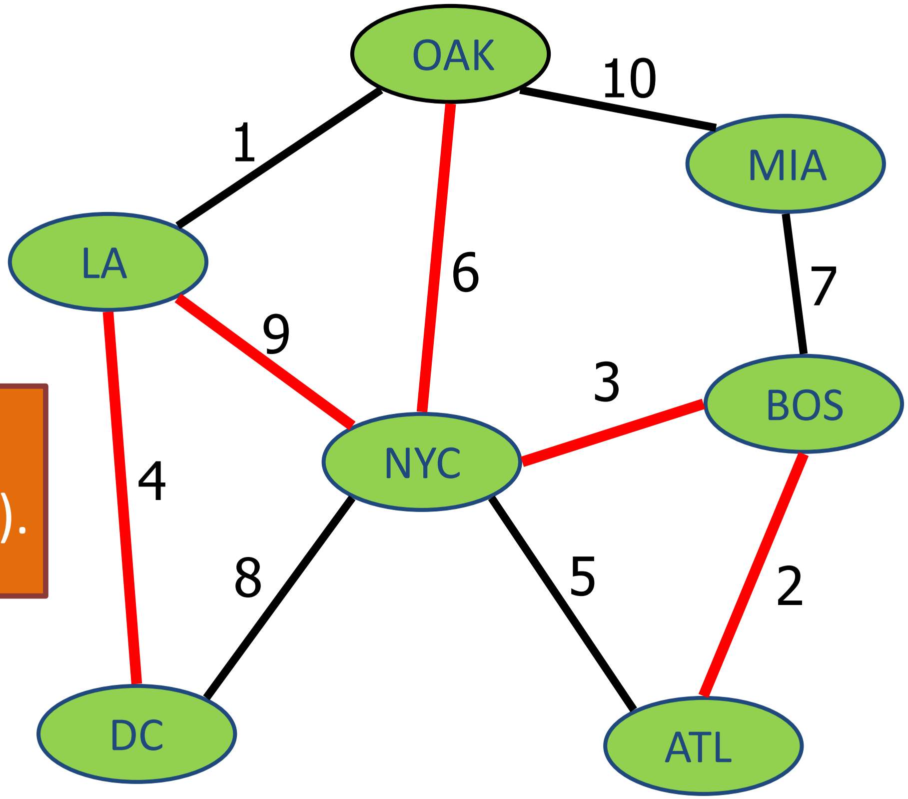
Total cost

$8+9+6+10+3+2 = 38$

Spanning Tree

Definition: We are given an undirected, weighted graph G. A spanning tree of G is a connected acyclic (tree) subgraph of G that includes all the vertices of G (spanning).

Example:

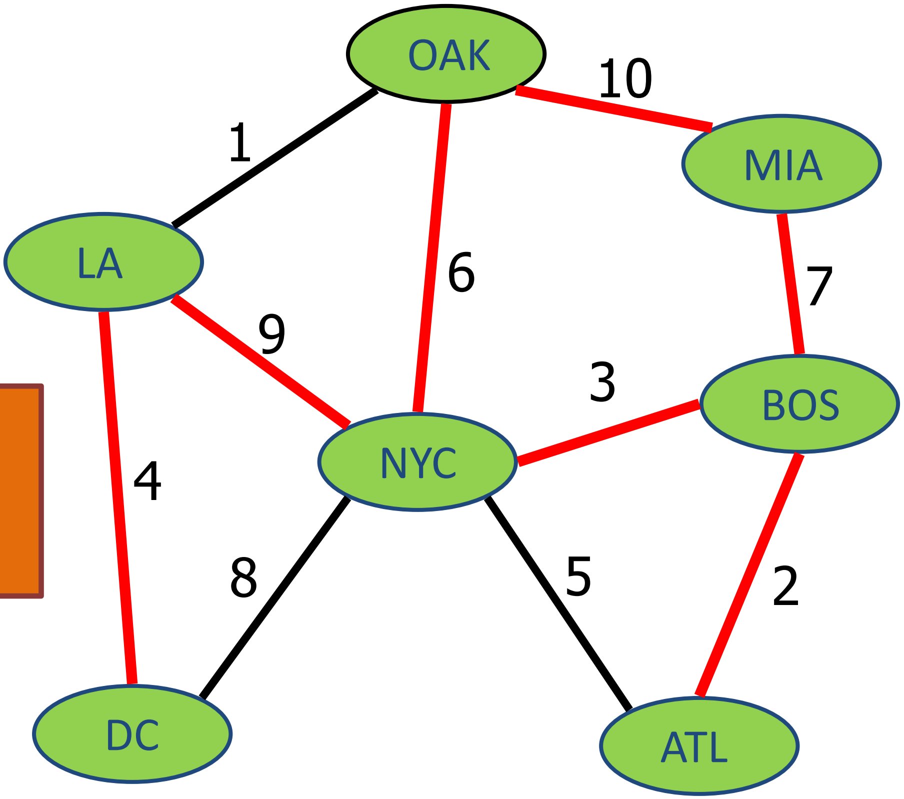


Not a spanning tree.
It is not **spanning** (MIA).

Spanning Tree

Definition: We are given an undirected, weighted graph G. A spanning tree of G is a connected acyclic (tree) subgraph of G that includes all the vertices of G (spanning).

Example:

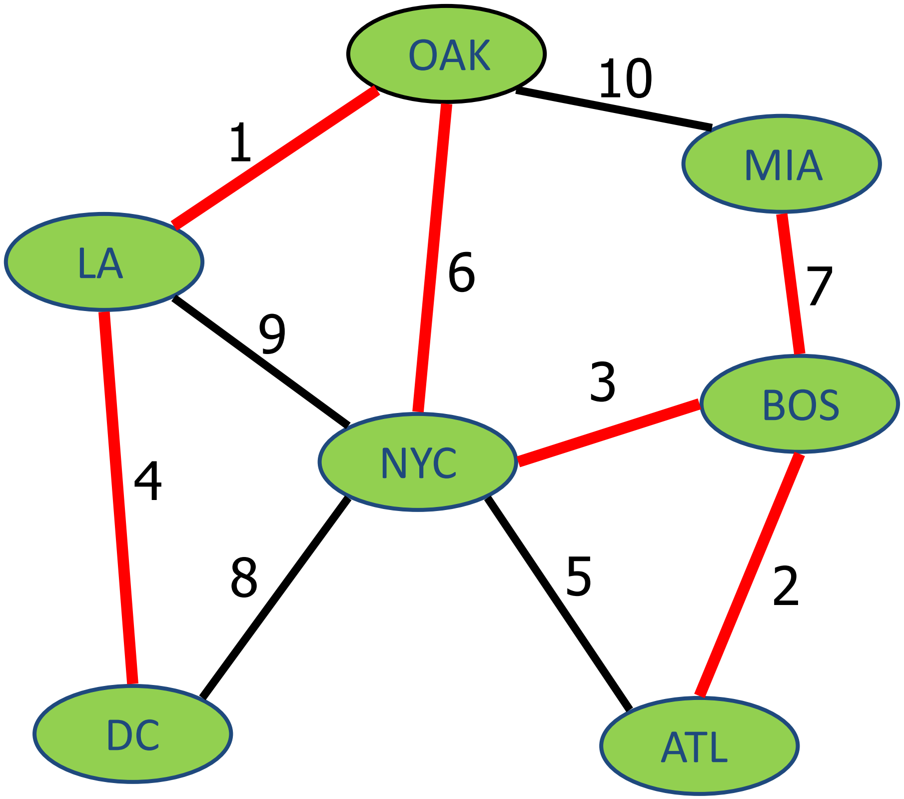


Not a spanning tree.
It is not **a tree** (cycle).

Minimum Spanning Tree

Problem: We are given an undirected, weighted graph G, find the **minimum spanning tree (MST).**

Example:



Total cost

$1{+}4{+}6{+}7{+}3{+}2 = 23$

Minimum Spanning Tree

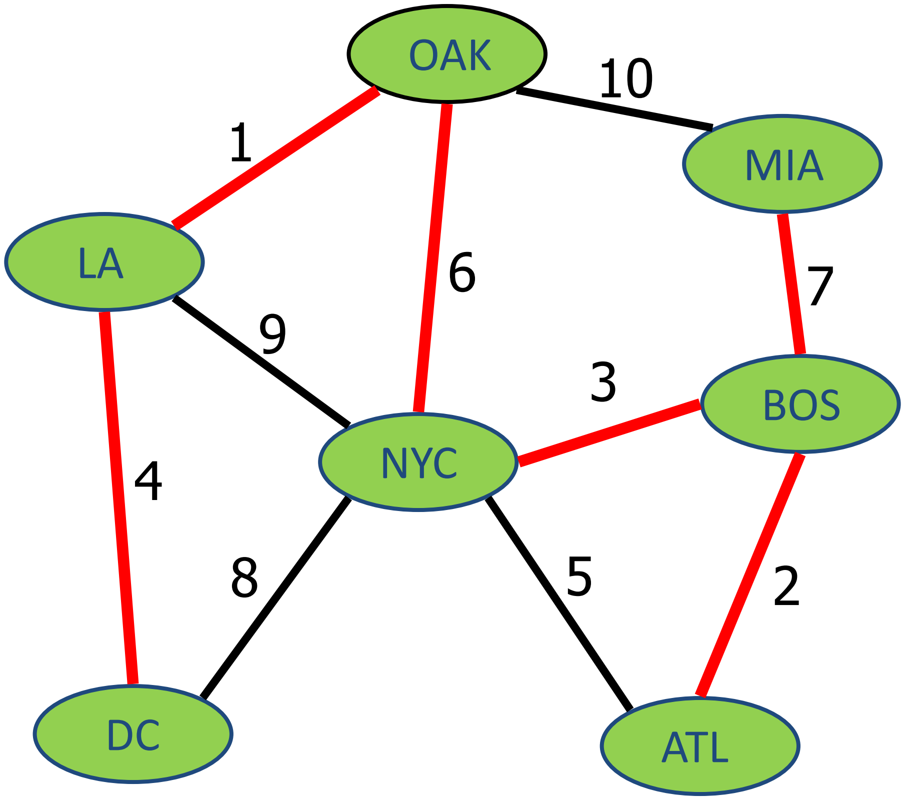
Cycle Property

Let T be a minimum spanning tree of a weighted graph G.

- Let $\boldsymbol{e}$ be an edge of G that is not in T and C let be the cycle formed by $\boldsymbol{e}$ with T.

It holds that:

For every edge $\boldsymbol{f}$ of C, $weight(f) \leq weight(e)$.



Minimum Spanning Tree

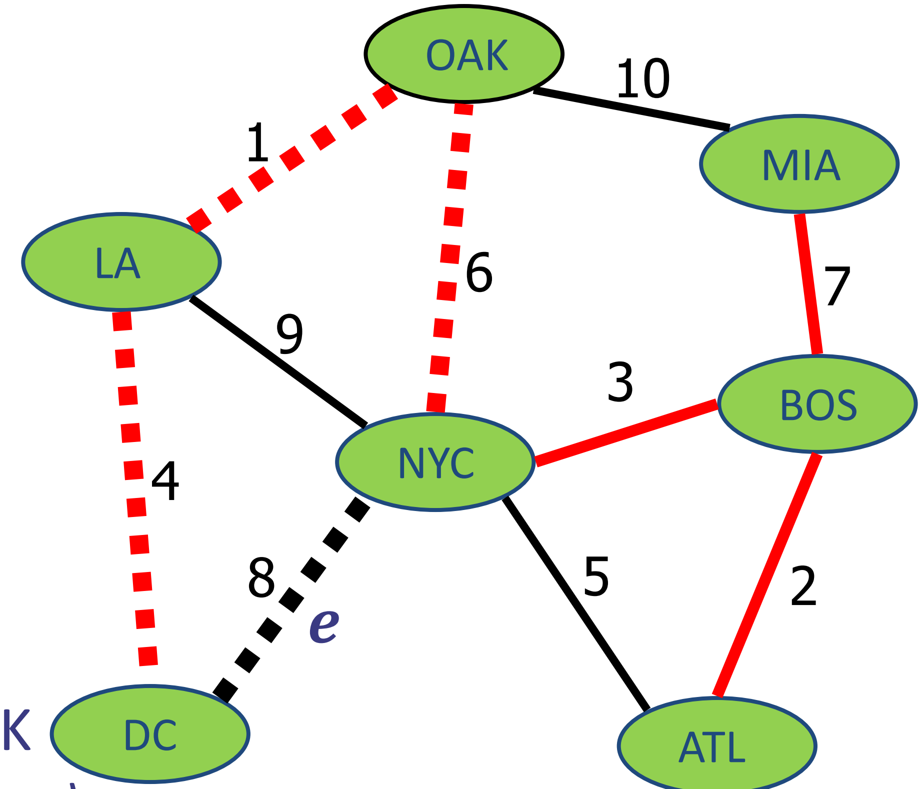
Cycle Property

Let T be a minimum spanning tree of a weighted graph G.

- Let $\boldsymbol{e}$ be an edge of G that is not in T and C let be the cycle formed by $\boldsymbol{e}$ with T.

It holds that:

For every edge $\boldsymbol{f}$ of C, $weight(f) \leq weight(e)$.



Example 1: Cycle LA, DC, NYC, OAK

$w(e) = 8 \geq 1, 6, 4$ *(rest of weights)*

Minimum Spanning Tree

Cycle Property

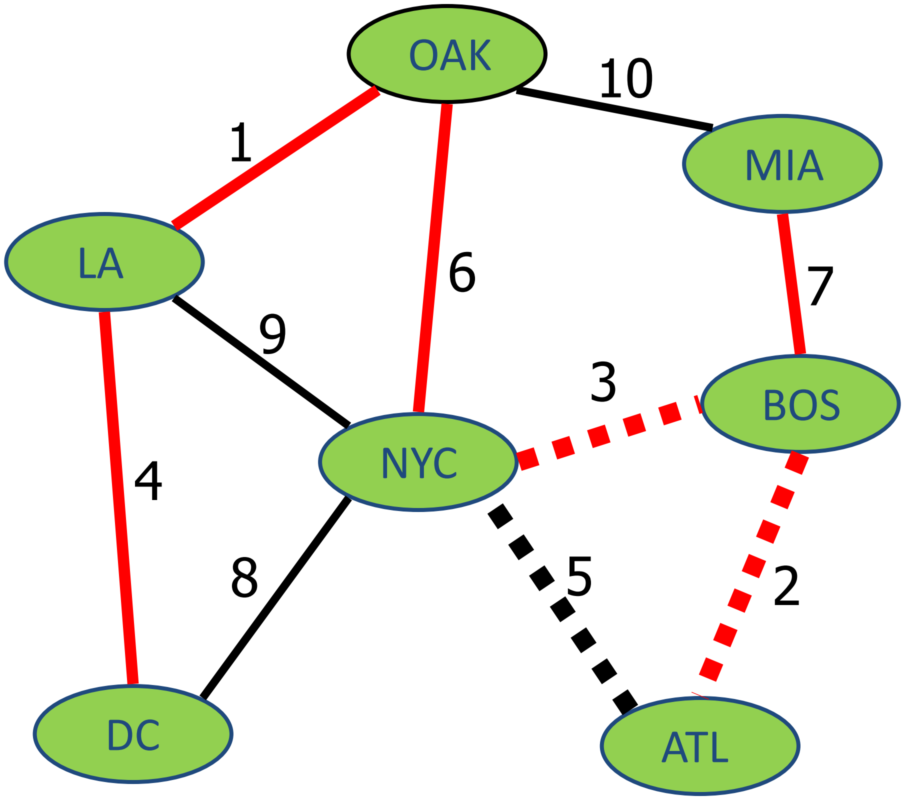
Let T be a minimum spanning tree of a weighted graph G.

- Let $\boldsymbol{e}$ be an edge of G that is not in T and C let be the cycle formed by $\boldsymbol{e}$ with T.

It holds that:

For every edge $\boldsymbol{f}$ of C, $weight(f) \leq weight(e)$.

Example 2: Cycle BOS, ATL, NYC
$w(e) = 5 \geq 2, 3$ *(rest of weights)*



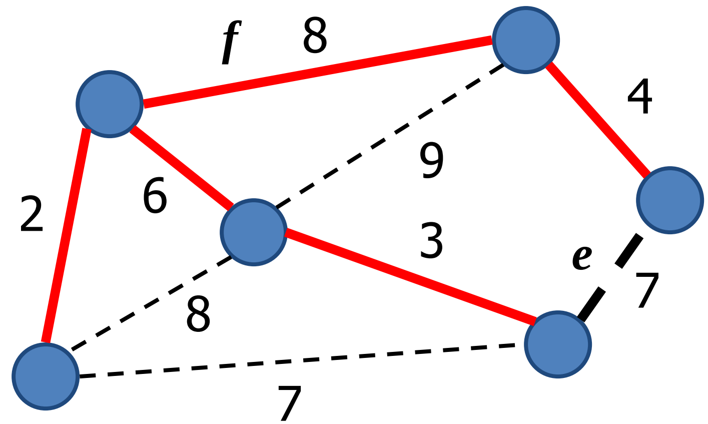
Minimum Spanning Tree

Cycle Property

For the sake of contradiction:

Assume there exist f, e so that

$$weight(f) > weight(e)$$



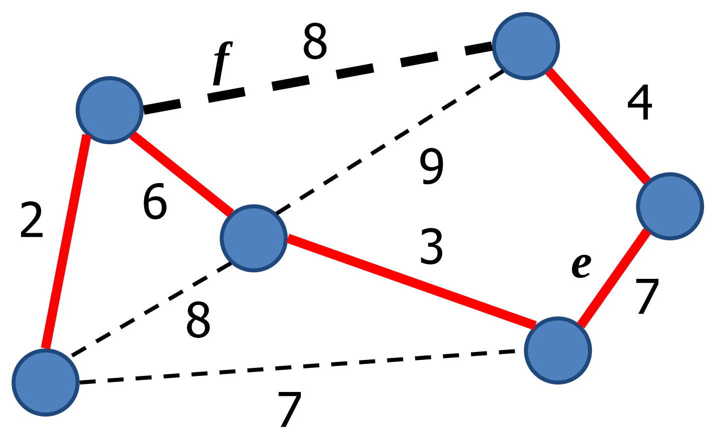
Total cost

$2+3+4+6+8 = 23$

Replacing f with e yields a better spanning tree

Total cost

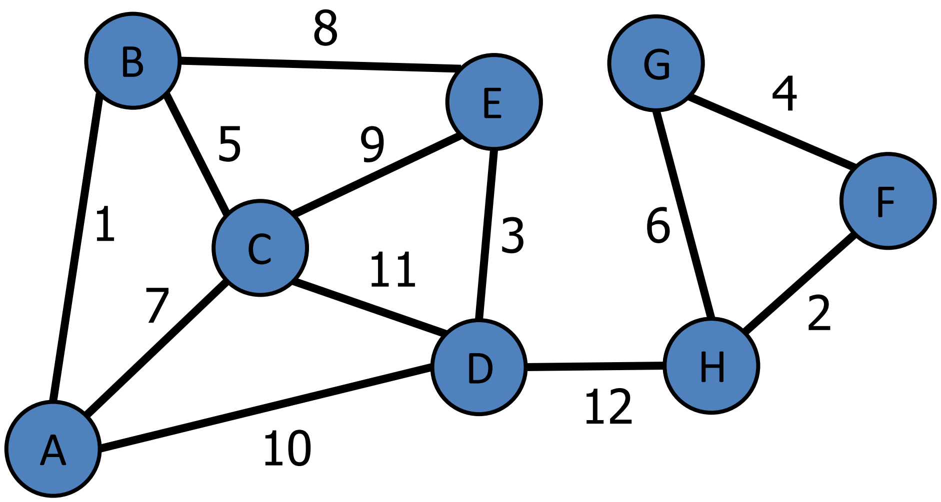
$2+3+4+6+7 = 22$



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

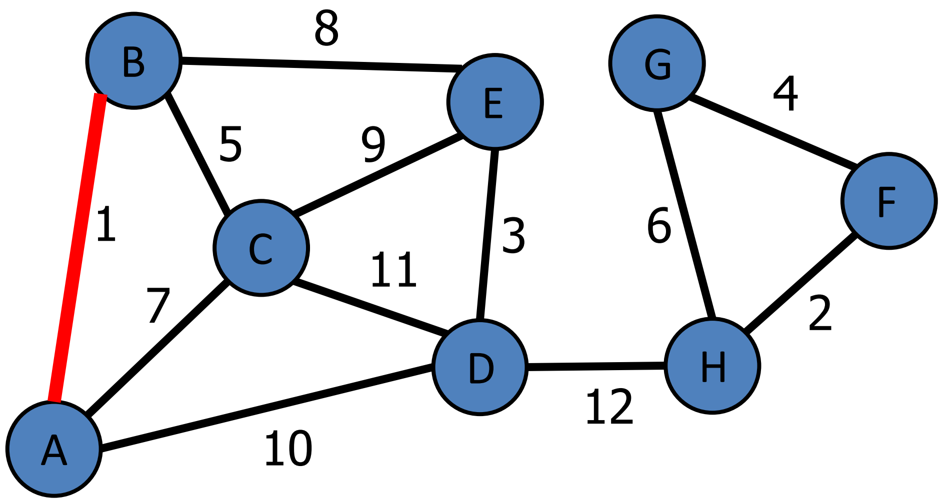
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

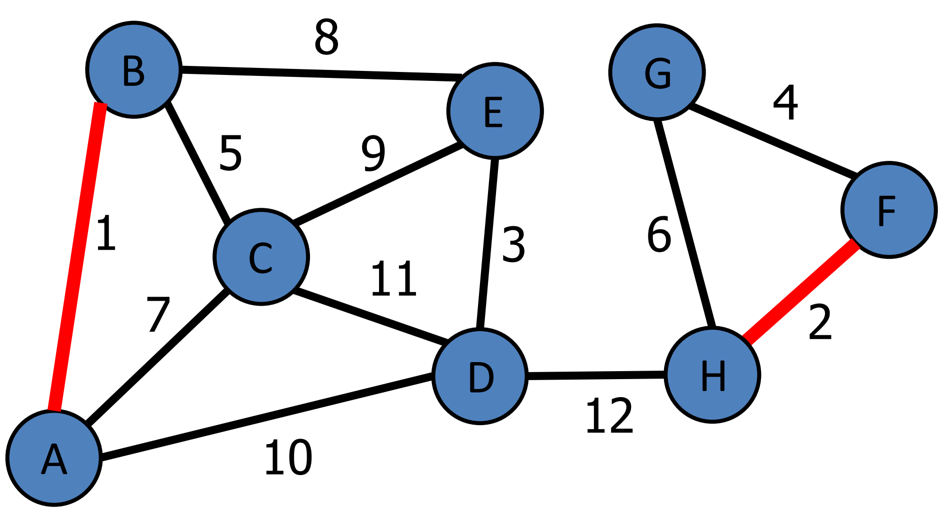
Example:



Kruskal’s Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

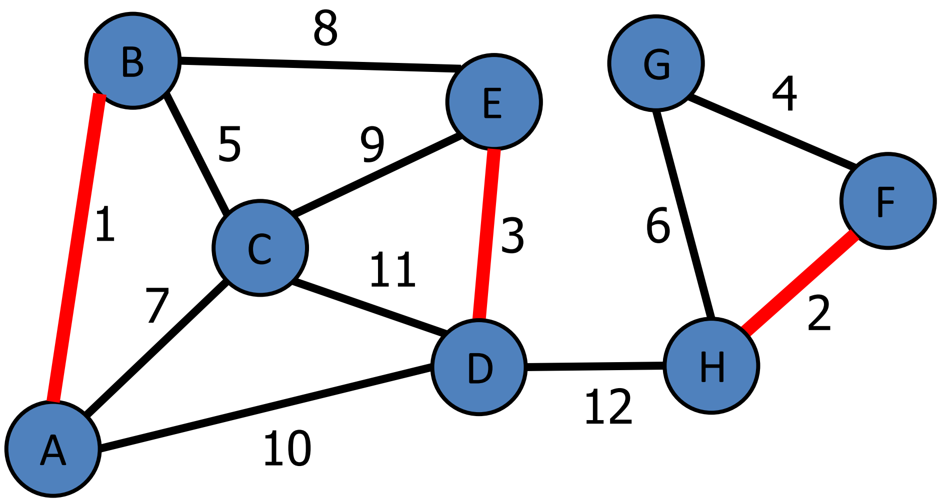
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

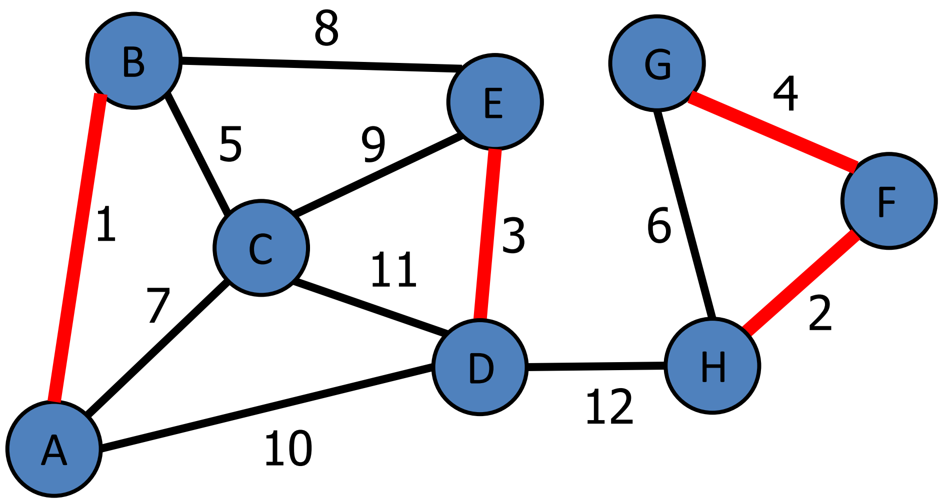
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

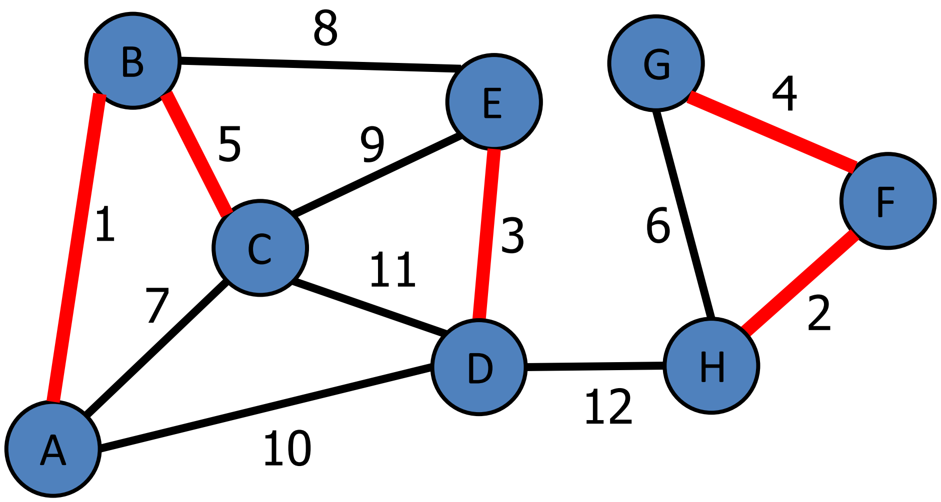
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

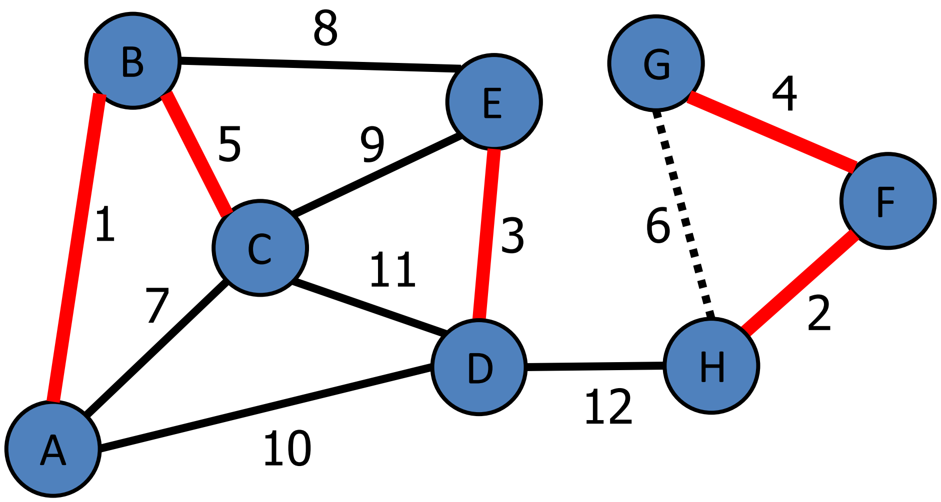
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

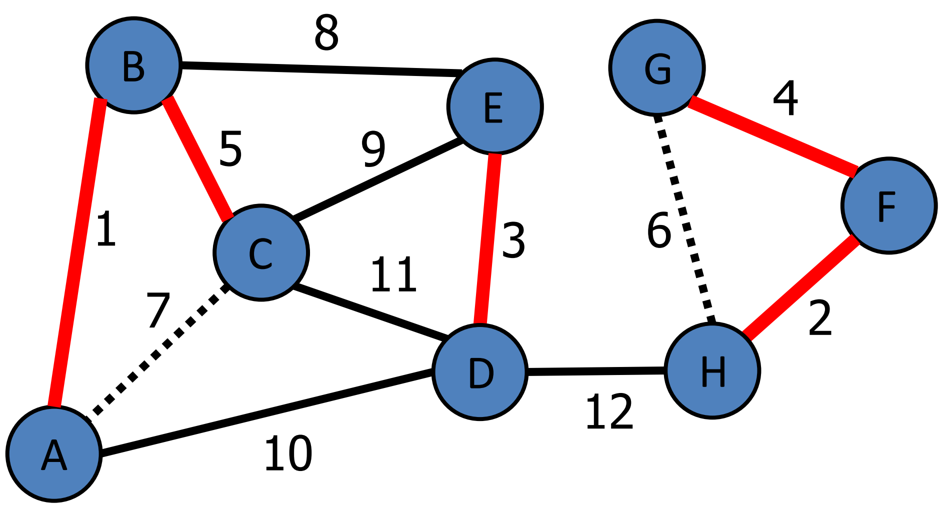
Example:



Kruskal’s Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

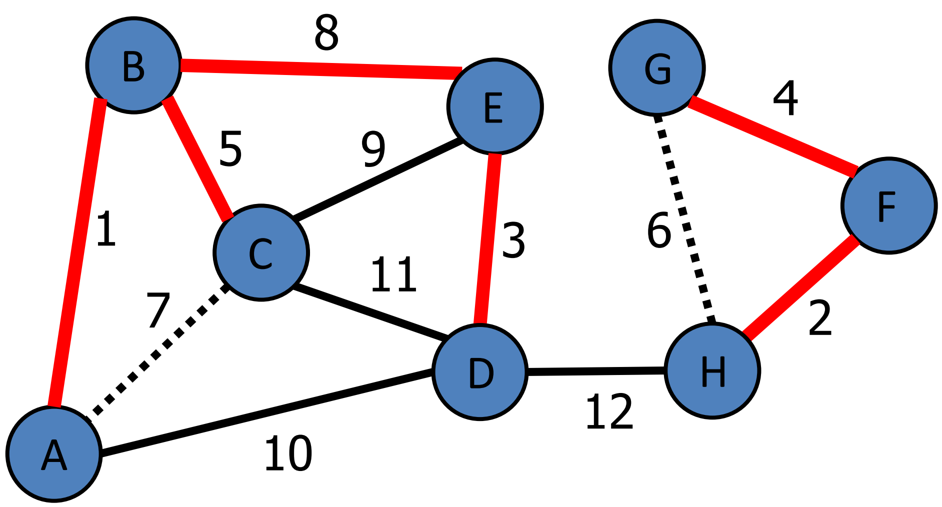
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

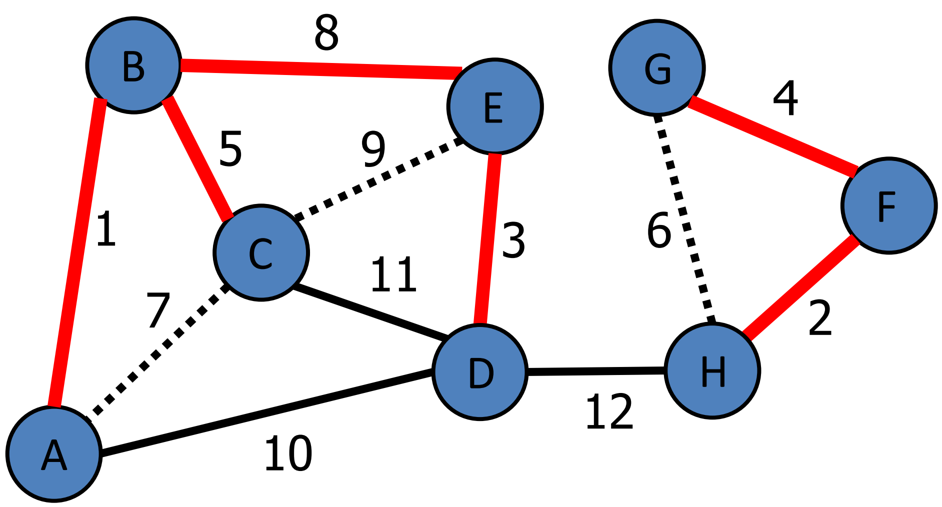
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

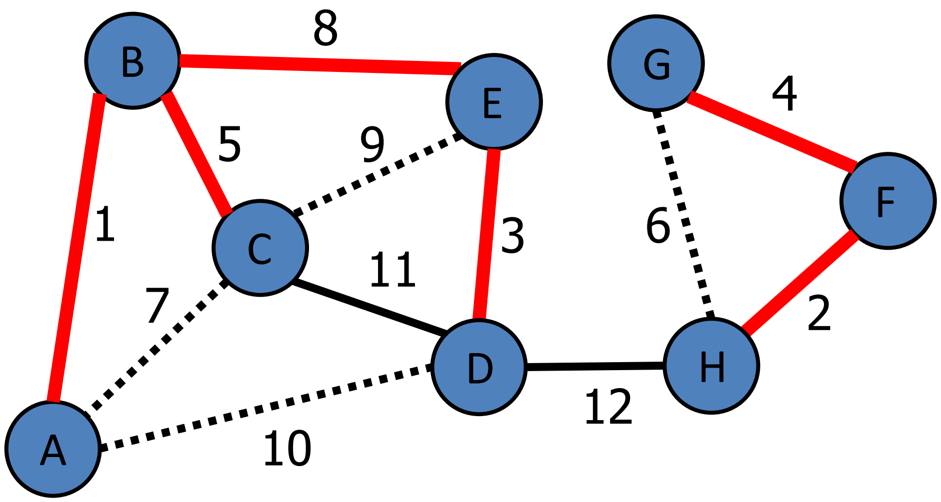
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

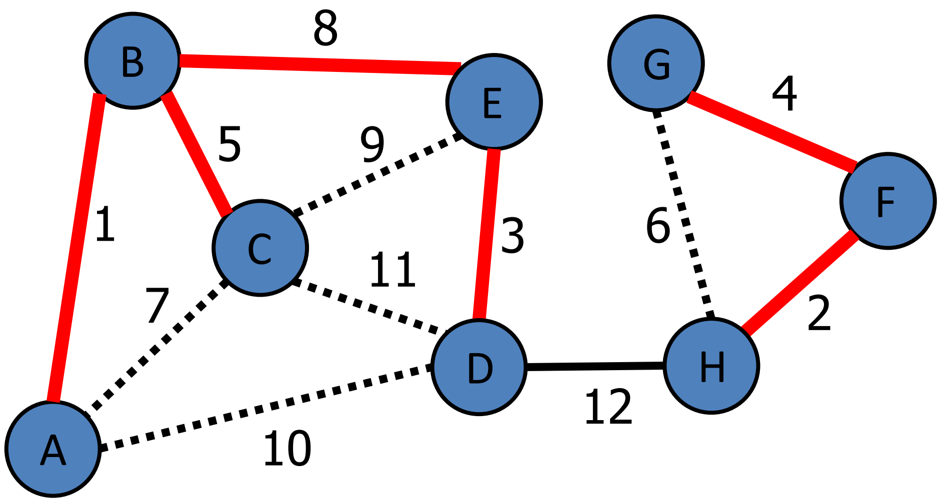
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

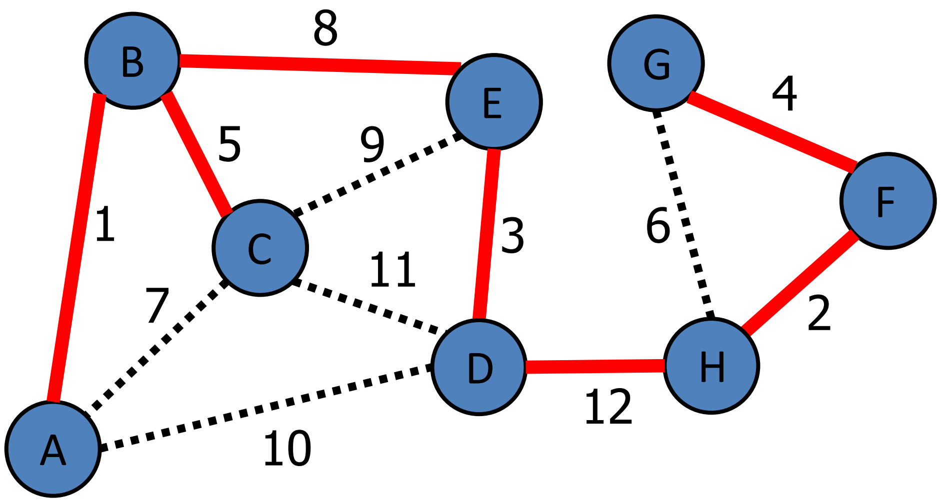
Example:



Kruskal's Algorithm for MSTs

Idea 1: Greedy approach. Consider the edges from smaller weight to larger. Include each edge in the current solution as long as it does not create a cycle, otherwise discard it.

Example:



Total cost

$1 + 2 + 3 + 4 + 5 + 8 + 12 = 35$

Kruskal's Algorithm for MSTs

Why Kruskal's algo works: General argument. Suppose there is a better solution. Assume the m edges of G are ordered in increasing order of weights, i.e., $\mathrm{w}_1 \leq w_2 \ldots \leq w_m$. G has also n vertices.

- Let $x_1, \ldots, x_{n-1}$ be the weight values of the edges in increasing order of the minimum spanning tree T'.
- Let $y_1, \ldots, y_{n-1}$ be the weight values of the edges in increasing order of Kruskal's spanning tree T.

Kruskal's Algorithm for MSTs

Why Kruskal's algo works: General argument. Suppose there is a better solution. Assume the m edges of G are ordered in increasing order of weights, i.e., $\mathrm{w}_1 \leq w_2 \ldots \leq w_m$. G has also n vertices.

- Let $x_1, \ldots, x_{n-1}$ be the weight values of the edges in increasing order of the minimum spanning tree T'.
- Let $y_1, \ldots, y_{n-1}$ be the weight values of the edges in increasing order of Kruskal's spanning tree T.
- There is an index i, so that $y_i < x_i$. We add edge with value y_i in T', we create a cycle C.

Kruskal's Algorithm for MSTs

Why Kruskal's algo works: General argument. Suppose there is a better solution. Assume the m edges of G are ordered in increasing order of weights, i.e., $\mathrm{w}_1 \leq w_2 \ \ldots \leq w_m$. G has also n vertices.

- Let $x_1, \ldots, x_{n-1}$ be the weight values of the edges in increasing order of the minimum spanning tree T'.
- Let $y_1, \ldots, y_{n-1}$ be the weight values of the edges in increasing order of Kruskal's spanning tree T.
- There is an index i, so that $y_i < x_i$. We add edge with value y_i in T', we create a cycle C.
  - If x_i is in C, we remove it and create a spanning tree smaller than T' (contradiction).

Kruskal's Algorithm for MSTs

Why Kruskal's algo works: General argument. Suppose there is a better solution. Assume the m edges of G are ordered in increasing order of weights, i.e., $\mathrm{w}_1 \leq w_2 \ldots \leq w_m$. G has also n vertices.

- Let $x_1, \ldots, x_{n-1}$ be the weight values of the edges in increasing order of the minimum spanning tree T'.
- Let $y_1, \ldots, y_{n-1}$ be the weight values of the edges in increasing order of Kruskal's spanning tree T.
- There is an index i, so that $y_i < x_i$. We add edge with value y_i in T', we create a cycle C.
  - If x_i is in C, we remove it and create a spanning tree smaller than T' (contradiction).
  - If x_i not in C, by cycle property, y_i is the largest value from edges in C. Kruskal would not have chosen y_i (contradiction).

Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. We build the **tree** by adding one new vertex at a time. Each vertex v has label $d[v] := $ smallest weight of an edge connecting v to a vertex in the built tree.

At each step:

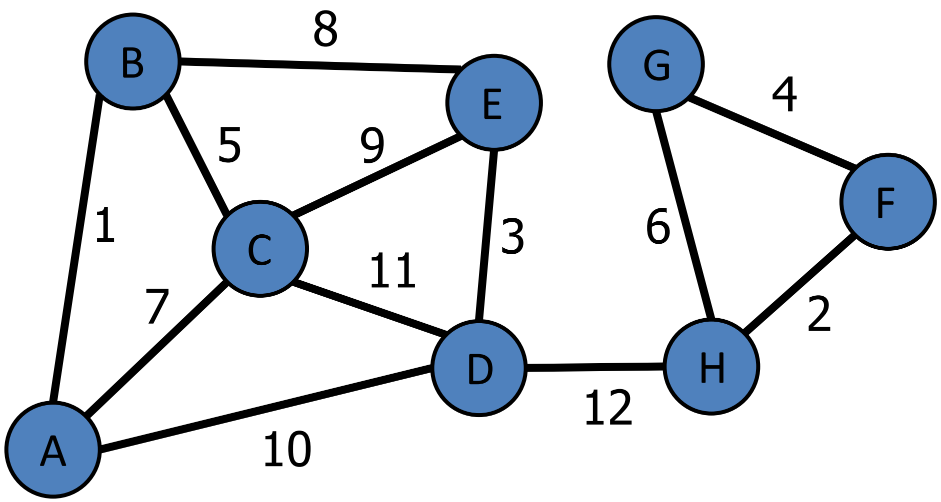
- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = \infty$
$d[C] = \infty$
$d[D] = \infty$
$d[E] = \infty$
$d[F] = \infty$
$d[G] = \infty$
$d[H] = \infty$

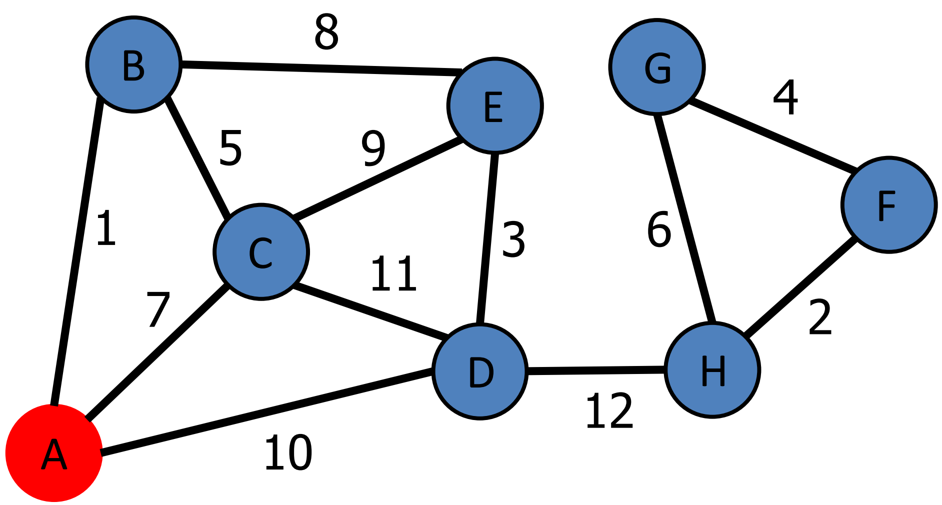


Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$\mathbf{d[B]=1}$
$\mathbf{d[C]=7}$
$\mathbf{d[D]=10}$
$d[E] = \infty$
$d[F] = \infty$
$d[G] = \infty$
$d[H] = \infty$

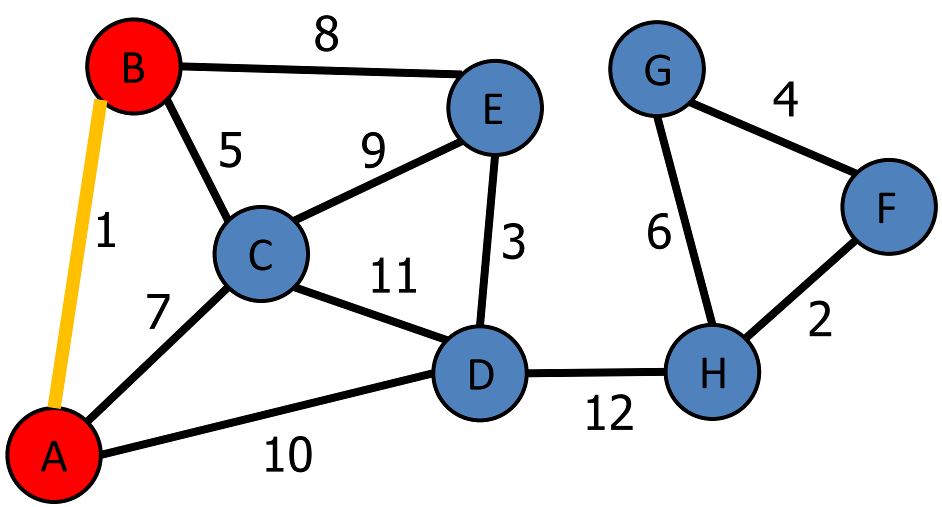


Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = 1$
$\mathbf{d[C]=5}$
$d[D] = 10$
$\mathbf{d[E]=8}$
$d[F] = \infty$
$d[G] = \infty$
$d[H] = \infty$

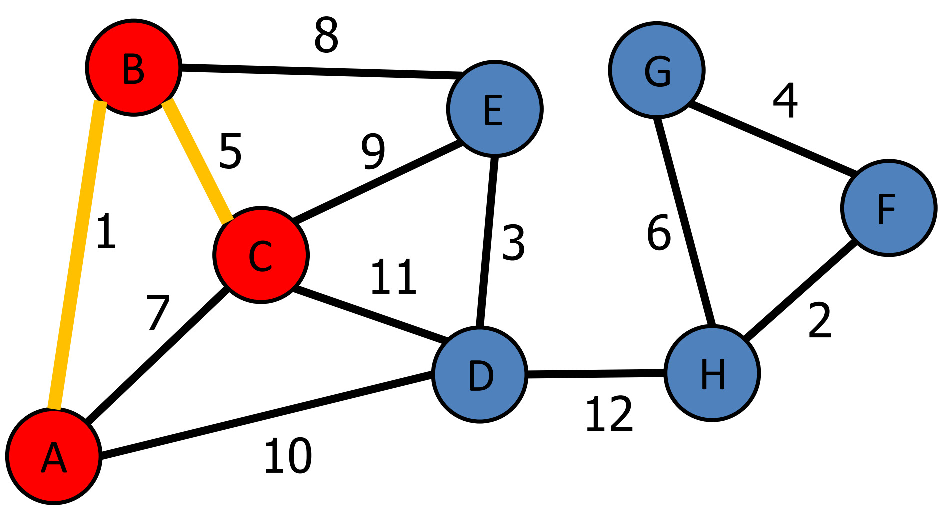


Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = 1$
$d[C] = 5$
$d[D] = 10$
$d[E] = 8$
$d[F] = \infty$
$d[G] = \infty$
$d[H] = \infty$

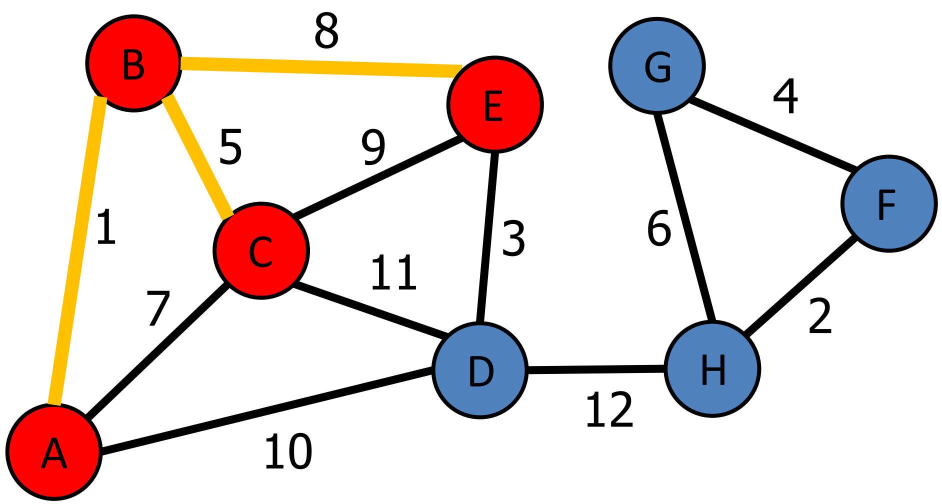


Prim’s Algorithm for MSTs

Idea 2: Similar to Dijkstra’s algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = 1$
$d[C] = 5$
$\mathbf{d[D]=3}$
$d[E] = 8$
$d[F] = \infty$
$d[G] = \infty$
$d[H] = \infty$

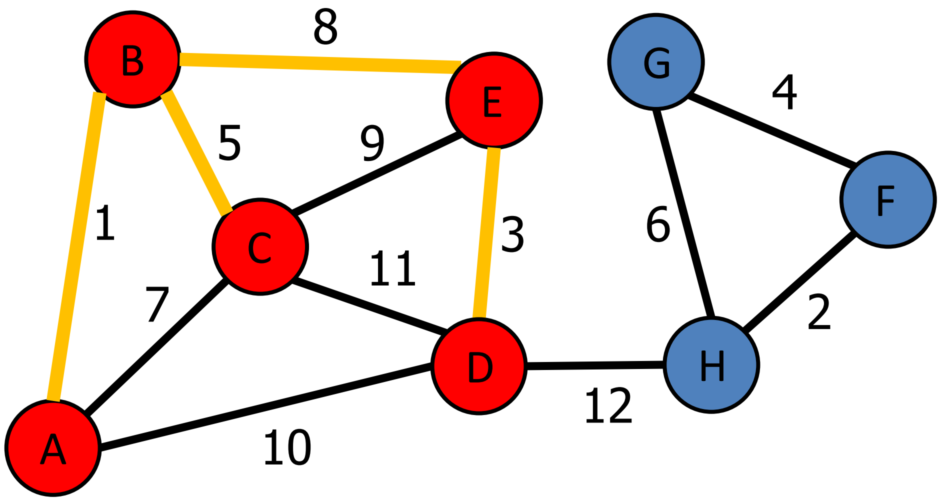


Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = 1$
$d[C] = 5$
$d[D] = 3$
$d[E] = 8$
$d[F] = \infty$
$d[G] = \infty$
$\mathbf{d[H]=12}$

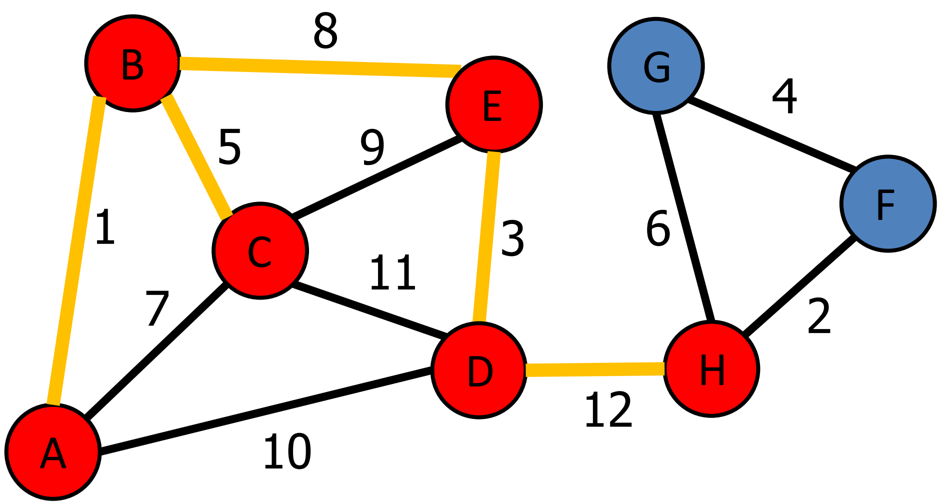


Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = 1$
$d[C] = 5$
$d[D] = 3$
$d[E] = 8$
$\mathbf{d[F]=2}$
$\mathbf{d[G]=6}$
$d[H] = 12$

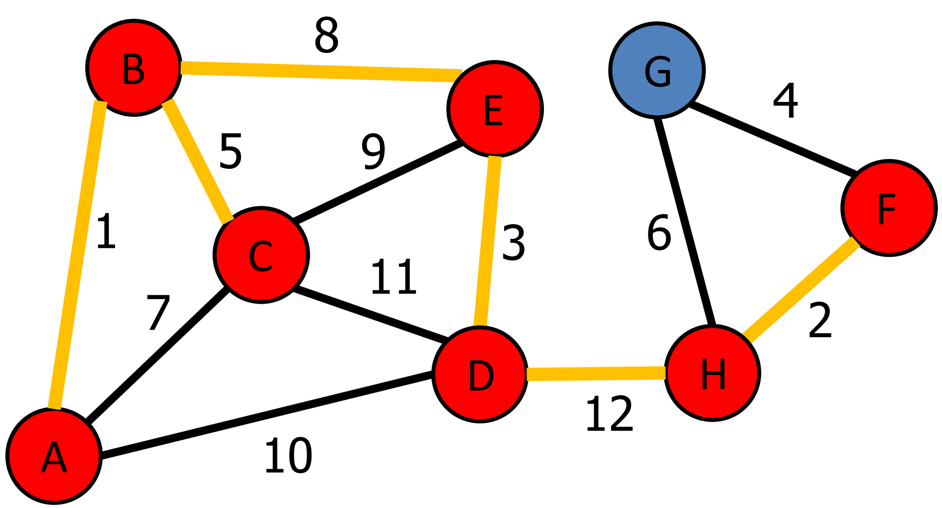


Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = 1$
$d[C] = 5$
$d[D] = 3$
$d[E] = 8$
$d[F] = 2$
$\mathbf{d[G]=4}$
$d[H] = 12$

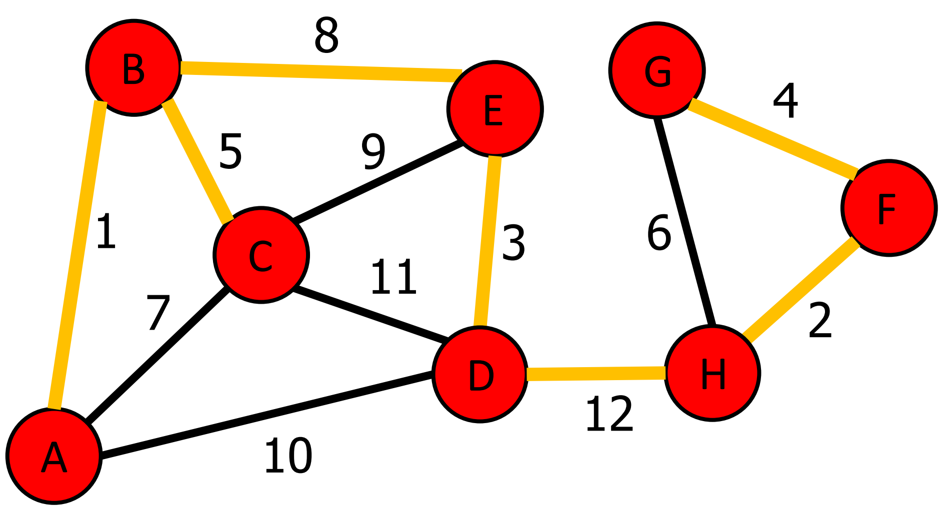


Prim's Algorithm for MSTs

Idea 2: Similar to Dijkstra's algorithm. We pick an arbitrary vertex s. At each step:

- We add to the current tree the vertex u with the smallest $d[u]$ and the corresponding incident to u edge.
- We update the labels of the vertices adjacent to u.

$d[A] = 0$
$d[B] = 1$
$d[C] = 5$
$d[D] = 3$
$d[E] = 8$
$d[F] = 2$
$d[G] = 4$
$d[H] = 12$



Prim's Algorithm for MSTs

Pseudocode:

Starting vertex

Initialization

Relaxation

Pick any vertex v of G
$D[v] \leftarrow 0$
for each vertex $u \neq v$ **do**
    $D[u] \leftarrow +\infty$
Initialize $T \leftarrow \emptyset$.
Initialize a priority queue Q with an item $((u, \text{null}), D[u])$ for each vertex u, where (u, null) is the element and $D[u]$ is the key.
while Q is not empty **do**
    $(u, e) \leftarrow Q.\text{removeMin}()$
    Add vertex u and edge e to T.
    for each vertex z adjacent to u such that z is in Q **do**
        // perform the relaxation procedure on edge (u, z)
        if $w((u, z)) < D[z]$ **then**
            $D[z] \leftarrow w((u, z))$
            Change to $(z, (u, z))$ the element of vertex z in Q.
            Change to $D[z]$ the key of vertex z in Q.
return the tree T

Prim's Algorithm for MSTs

Pseudocode:

Pick any vertex v of G
$D[v] \leftarrow 0$
for each vertex $u \neq v$ **do**
  $D[u] \leftarrow +\infty$
Initialize $T \leftarrow \emptyset$.
Initialize a priority queue Q with an item $((u, \text{null}), D[u])$ for each vertex u, where (u, null) is the element and $D[u]$ is the key.
while Q is not empty **do**
  $(u, e) \leftarrow Q$.removeMin()
  Add vertex u and edge e to T.
  for each vertex z adjacent to u such that z is in Q **do**
    // perform the relaxation procedure on edge (u, z)
    if $w((u, z)) < D[z]$ **then**
      $D[z] \leftarrow w((u, z))$
      Change to $(z, (u, z))$ the element of vertex z in Q.
      Change to $D[z]$ the key of vertex z in Q.
return the tree T

Starting vertex

Initialization

Relaxation

Running time: If extractmin in $\Theta(|V|)$, update in $\Theta(1)$ then $|V|^2 + |E|$.

Prim's Algorithm for MSTs

Pseudocode:

Pick any vertex v of G
$D[v] \leftarrow 0$
for each vertex $u \neq v$ **do**
    $D[u] \leftarrow +\infty$
Initialize $T \leftarrow \emptyset$.
Initialize a priority queue Q with an item $((u, \text{null}), D[u])$ for each vertex u, where (u, null) is the element and $D[u]$ is the key.
while Q is not empty **do**
    $(u, e) \leftarrow Q.\text{removeMin}()$
    Add vertex u and edge e to T.
    for each vertex z adjacent to u such that z is in Q **do**
        // perform the relaxation procedure on edge (u, z)
        if $w((u, z)) < D[z]$ **then**
            $D[z] \leftarrow w((u, z))$
            Change to $(z, (u, z))$ the element of vertex z in Q.
            Change to $D[z]$ the key of vertex z in Q.
return the tree T

Starting vertex

Initialization

Relaxation

Running time: If extractmin in $\Theta(|V|)$, update in $\Theta(1)$ t

$\Theta(|V|^2)$

Prim's Algorithm for MSTs

Pseudocode:

Pick any vertex v of G
$D[v] \leftarrow 0$
for each vertex $u \neq v$ **do**
  $D[u] \leftarrow +\infty$
Initialize $T \leftarrow \emptyset$.
Initialize a priority queue Q with an item $((u, \text{null}), D[u])$ for each vertex u, where (u, null) is the element and $D[u]$ is the key.
while Q is not empty **do**
  $(u, e) \leftarrow Q.\text{removeMin}()$
  Add vertex u and edge e to T.
  for each vertex z adjacent to u such that z is in Q **do**
    // perform the relaxation procedure on edge (u, z)
    if $w((u, z)) < D[z]$ **then**
      $D[z] \leftarrow w((u, z))$
      Change to $(z, (u, z))$ the element of vertex z in Q.
      Change to $D[z]$ the key of vertex z in Q.
return the tree T

Starting vertex

Initialization

Relaxation

Running time: If extractmin, update in $\Theta(\log |V|)$ then $|E| \log |V|$.