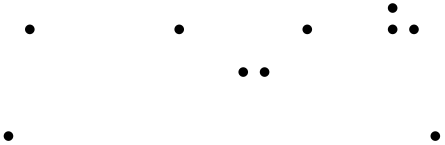
Kruskal Algorithm (1956)

Approach : Maintain a forest. In each step, add an edge of minimum weight that does not create a cycle.

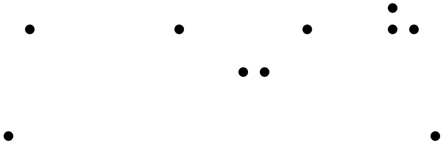
Start : At the beginning, each vertex is a (trivial) tree.



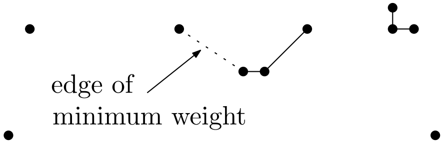
Kruskal Algorithm (1956)

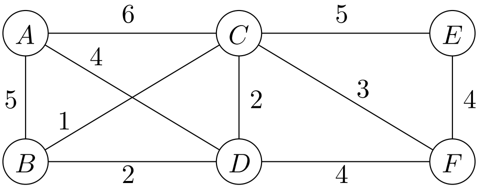
Approach : Maintain a forest. In each step, add an edge of minimum weight that does not create a cycle.

Start : At the beginning, each vertex is a (trivial) tree.



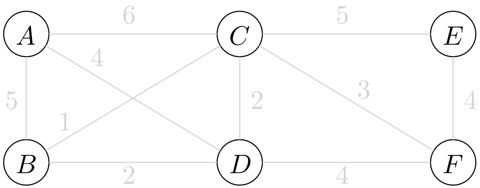
One Iteration : Combine two trees using an edge of minimum weight.

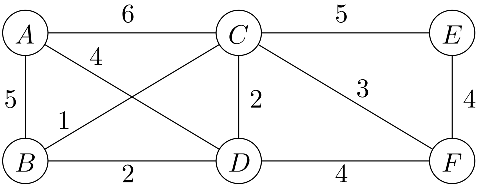




Sor the edges by weight:

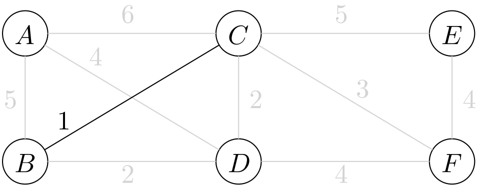
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

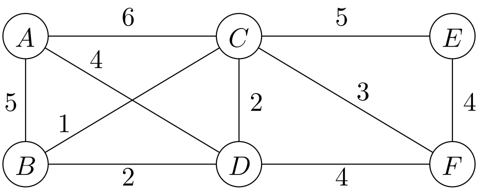




Sor the edges by weight:

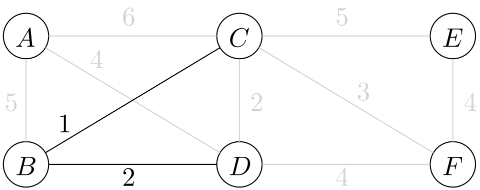
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

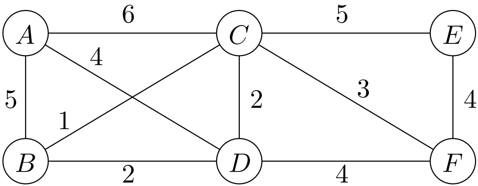




Sor the edges by weight:

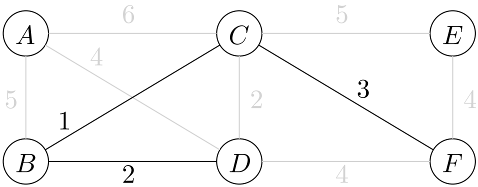
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

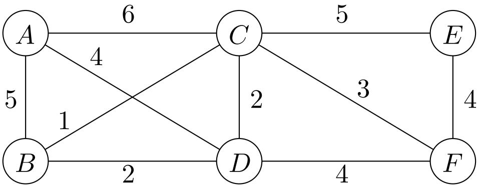




Sor the edges by weight:

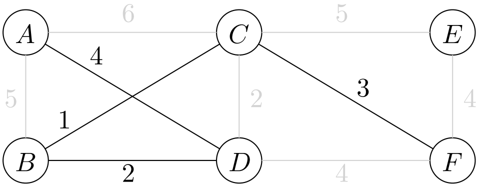
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

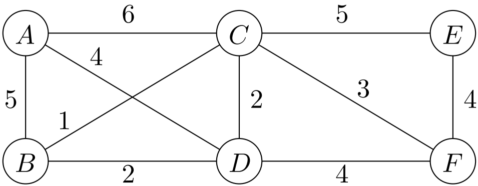




Sor the edges by weight:

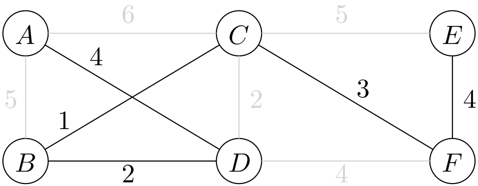
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$





Sor the edges by weight:

$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$



Total weight: 14

Algorithm *Kruskal*(G)

Input: $G = (V, E)$, where $V = \{x_1, x_2, ..., x_n\}$ and $m = |E|$.

Output: A minimum spanning tree of G.

Sort the edges of E by weight using Merge Sort: $e_1, e_2, ..., e_m$
for $i = 1$ to n **do**
    $V_i = \{x_i\}$
end for
$T = \{\ \}$
for $k = 1$ to m **do**
    let u_k and v_k be the vertices of e_k.
    let i be the index such that $u_k \in V_i$
    let j be the index such that $v_k \in V_j$
    if $i \neq j$ **then**
        $V_i = V_i \cup V_j$
        $V_j = \{\ \}$
        $T = T \cup \{\{u_k, v_k\}\}$
    end if
end for
return T

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time
- Second For-loop:
  - Store T in a linked list. Total time to maintain this list: $O(n)$ time
  - Store the sets V_i using the Union-Find data structure.

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time
- Second For-loop:
  - Store T in a linked list. Total time to maintain this list: $O(n)$ time
  - Store the sets V_i using the Union-Find data structure.

  In this second For-Loop, we do

  - $2m$ **Find** operations
  - $n-1$ **Union** operations

  So in total for the second For-Loop:

$$O(n) + O(m + n\log(n)) \text{ time}$$

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time
- Second For-loop:
  - Store T in a linked list. Total time to maintain this list: $O(n)$ time
  - Store the sets V_i using the Union-Find data structure.

  In this second For-Loop, we do
  - $2m$ **Find** operations
  - $n-1$ **Union** operations

  So in total for the second For-Loop:

$$O(n) + O(m + n\log(n)) \text{ time}$$

So the total time is

$$O(m\log(n)) + O(n) + O(m + n\log(n)) = O(m\log(n))$$

Do you see why?

Conclusion: Kruskal computes an MST in $O(m\log(n))$ time.