Minimum Spanning Tree Problem

Input: A connected undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.

For simplicity, assume that all edge weights are distinct. (the algorithms will still work but the analysis is more difficult)

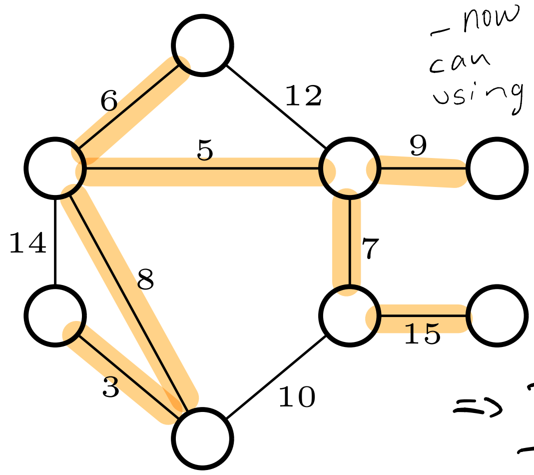
Output: A spanning tree T (a tree that connects all the vertices) of minimum weight:

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

cost of Tree

sum of all edges weights in the tree

Example: MST



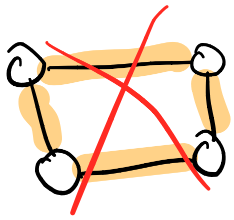
– now all nodes can be reached using these edges

=> Tree Weight

W(T) = 6+5+9+7+15+3+8

W(T) = 53

=>



∴ Cant happen if looking for minimum cost to connect.

How do we find the minimum spanning tree of a particular graph?

Useful theorem:

A is subset of set of vertices (V)

Theorem.
Let T be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V \backslash A$. Then, $(u, v) \in T$.

"not in A"

(Proof skipped: see page 626 for additional details)

Remember, "A locally optimal choice is globally optimal" is the hallmark of a **greedy algorithm**.

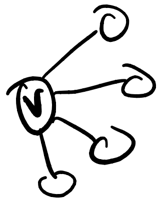
From this theorem, create a greedy algorithm to compute the MST. Roughly the idea is that we can repeatedly choose the least-weight edge that connects a new vertex to our tree.

Prim's Algorithm

"Edges not in A"

Idea: Maintain $V \backslash A$ as a priority queue Q. Key each vertex in Q with the weight of the least-weight edge connecting it to a vertex in A.

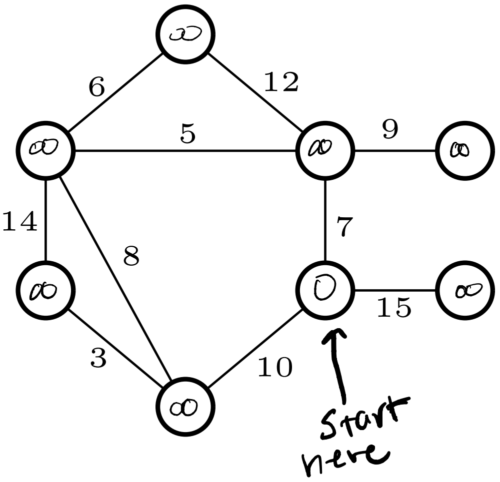
$Q = V$
Set $key[v] = \infty$ for all $v \in V$
Set $key[s] = 0$ for some arbitrary $s \in V$
while Q is not empty **do**
  $u = ExtractMin(Q)$
  for each $v \in Adj[u]$ **do**
    if $v \in Q$ and $w(u, v) < key[v]$ **then**
      $key[v] = w(u, v)$ //Decreases key in priority queue Q
      $\pi[v] = u$
    end if
  end for
end while



Keep track of the Tree (reached v from u)

// Start anywhere in Tree always results in same tree
// all nodes are ∞ initially

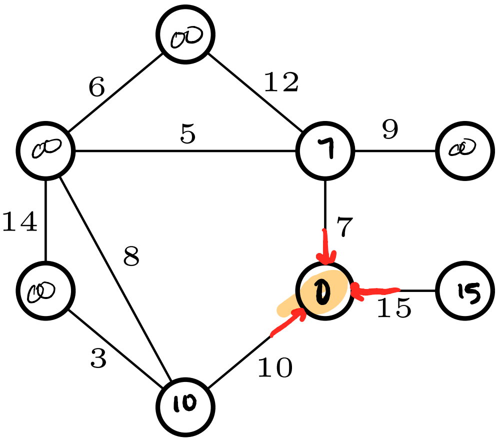
Prim's Example: (1)



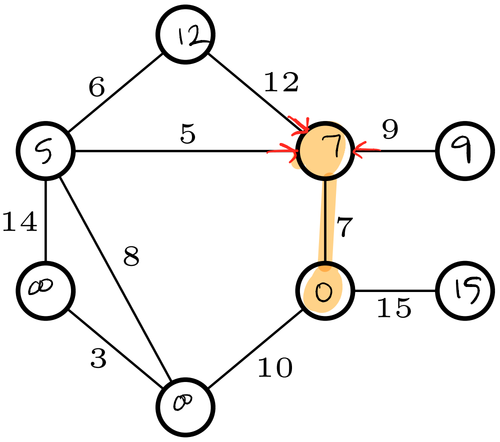
min prio. que.

front rear

Q: $\cancel{0}$, ∞, ∞, ∞, ∞, ∞, ∞, ∞



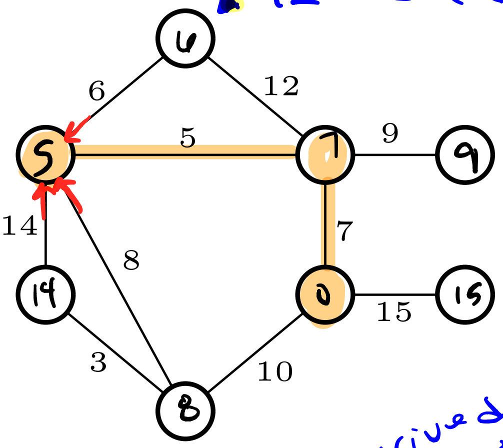
Q: $\cancel{7}$, 10, 15, ∞, ∞, ∞, ∞



Q: $\cancel{5}$, 9, 10, 12, 15, ∞, ∞

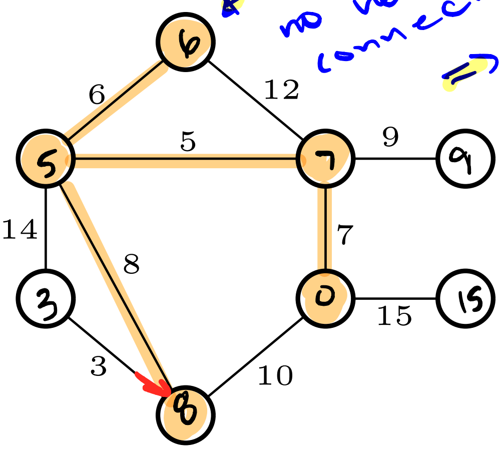
Prim's Example: (2)

change to 6 from 12 as (6 < 12)

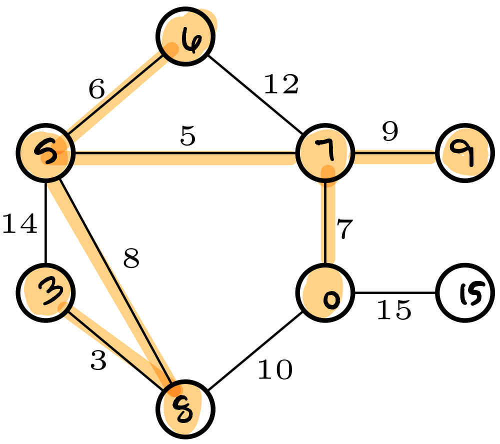


Q: ~~6~~, ~~8~~, 9, 14, 15

we arrived at 6, but no new minimum connections. ⟹ remove next element in queue I.E: 8

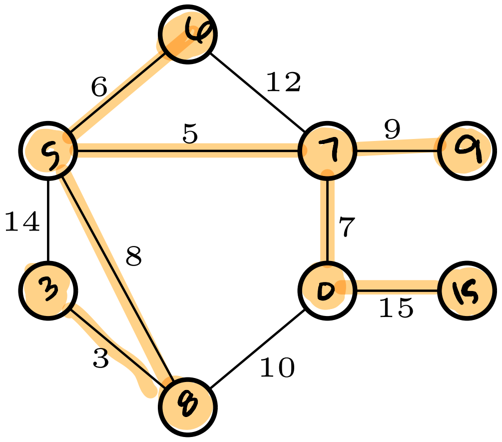


Q: ~~3~~, 9, 15



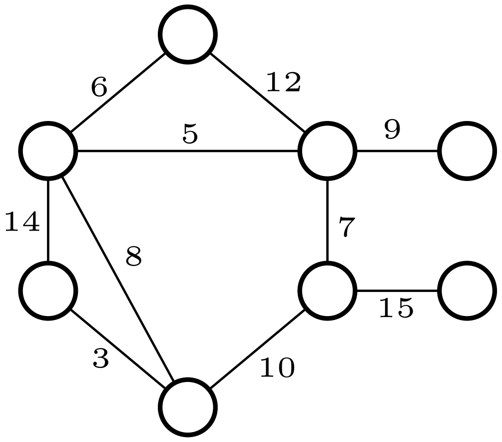
Q: ~~9~~, 15

Prim's Example: (3)

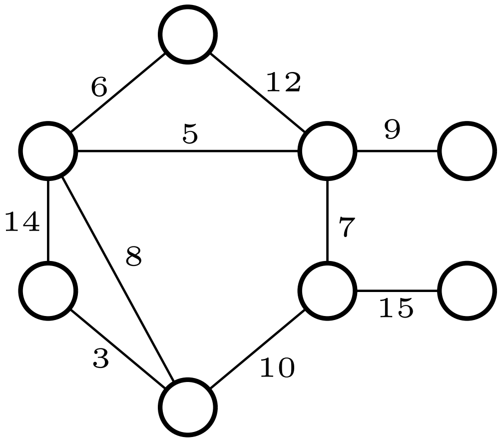


Q: ~~15~~

complete MST



Q:



Q:

Prim's Algorithm Runtime

$Q = V$
Set $key[v] = \infty$ for all $v \in V$
Set $key[s] = 0$ for some arbitrary $s \in V$
while Q is not empty **do**
    $u = ExtractMin(Q)$
    for each $v \in Adj[u]$ **do**
        if $v \in Q$ and $w(u, v) < key[v]$ **then**
            $key[v] = w(u, v)$ //Decreases key in priority queue Q
            $\pi[v] = u$
        end if
    end for
end while

Q	$T_{EXTRACT-MIN}$	$T_{DECREASE-KEY}$	Total Runtime
array			
binary heap			
Fibonacci heap (not covered)			

Kruskal's Algorithm

Another greedy algorithm for computing a MST is Kruskal's.

Idea: Repeatedly pick edge with smallest weight as long as it does not form a cycle.

- The algorithm creates a set of trees (a forest)
- During the algorithm the added edges merge the trees together, such that in the end only one tree remains