Rule for Recognizing Safe Edges (4/7)

- Since the cut respects A, edge (x, y) is not in A.
- To form T' from T:
  - Remove (x, y). Breaks T into two components.
  - Add (u, v). Reconnects.
- So $T' = T - \{(x, y)\} \cup \{(u, v)\}$.
- T' is a spanning tree.
- $$\begin{aligned} w(T') &= w(T) - w(x, y) + w(u, v) \\ &\leq w(T) , \end{aligned}$$

  since $w(u, v) \leq w(x, y)$.

Rule for Recognizing Safe Edges (5/7)

- Since T' is a spanning tree, $w(T') \leq w(T)$, and T is an MST, then T' must be an MST.
- Need to show that (u, v) is safe for A:
  - $A \subseteq T$ and $(x, y) \notin A \Rightarrow A \subseteq T'$.
  - $A \cup \{(u, v)\} \subseteq T'$.
  - Since T' is an MST, (u, v) is safe for A.

Rule for Recognizing Safe Edges (6/7)

- So, in GENERIC-MST:
  - A is a forest containing connected components. Initially, each component is a single vertex.
  - Any safe edge merges two of these components into one. Each component is a tree.
  - Since an MST has exactly $|V| - 1$ edges, the **for** loop iterates $|V| - 1$ times. Equivalently, after adding $|V| - 1$ safe edges, we're down to just one component.

Rule for Recognizing Safe Edges (7/7)

- **Corollary 23.2** Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, and let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.
- **Proof:** Set $S = V_C$ in the theorem.
- This idea naturally leads to the algorithm known as Kruskal's algorithm to solve the minimum-spanning-tree problem.

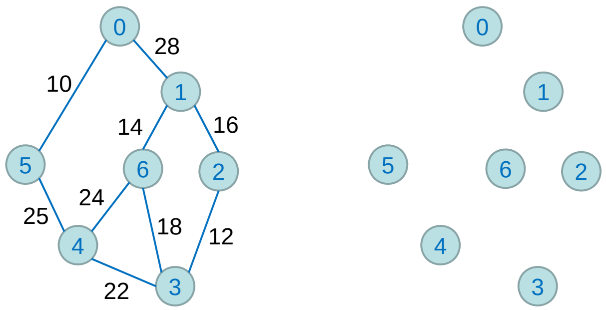
Outline

- Overview
- Growing a Minimum Spanning Tree
- ***The Algorithms of Kruskal and Prim***

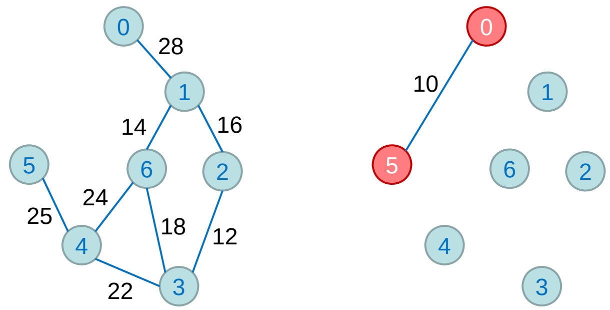
Kruskal's Algorithm

- $G = (V, E)$ is a connected, undirected, weighted graph. $w : E \rightarrow \mathbb{R}$
  - Starts with each vertex being its own component.
  - Repeatedly merges two components into one by choosing the light edge that connects them (i.e., the light edge crossing the cut between them).
  - Scans the set of edges in monotonically increasing order by weight.
  - Uses a disjoint-set data structure to determine whether an edge connects vertices in different components.

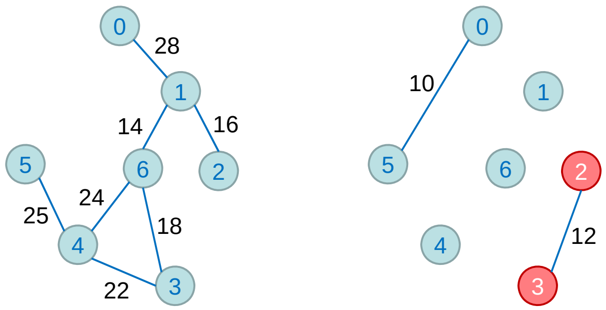
Stages in Kruskal's Algorithm



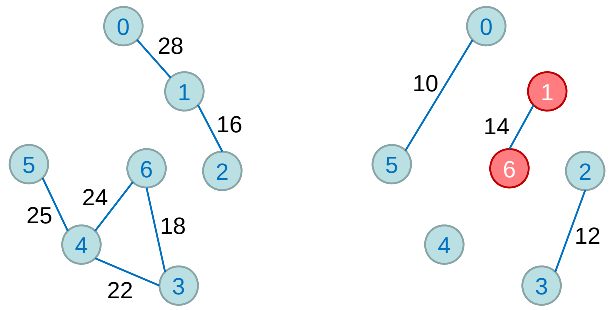
Stages in Kruskal's Algorithm



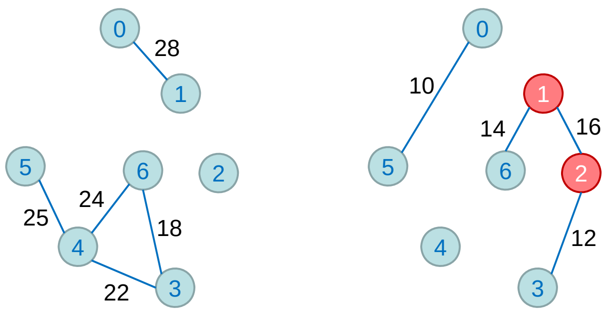
Stages in Kruskal's Algorithm



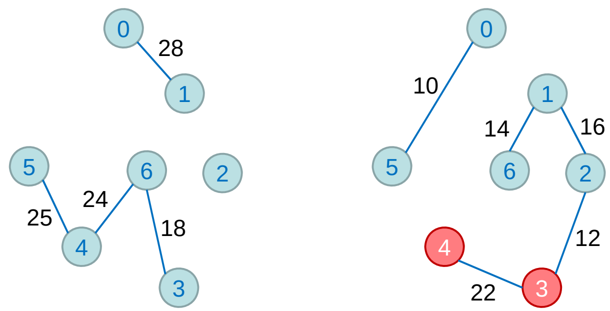
Stages in Kruskal's Algorithm



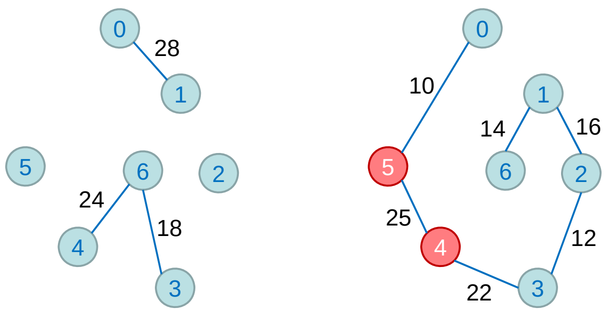
Stages in Kruskal's Algorithm



Stages in Kruskal's Algorithm



Stages in Kruskal's Algorithm

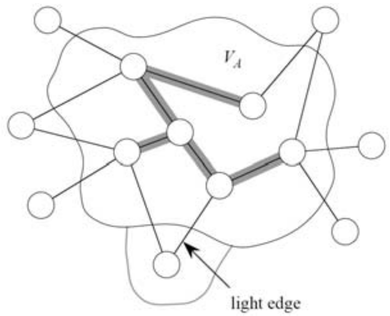


Pseudo Code of Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = Ø
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
```


Prim's Algorithm (1/3)

- Builds one tree, so *A* is always a tree.
- Starts from an arbitrary "root" *r*.
- At each step, find a light edge crossing cut $(V_A, V - V_A)$, where V_A = vertices that *A* is incident on. Add this edge to *A*.



Prim's Algorithm (2/3)

How to Find the Light Edge Quickly?

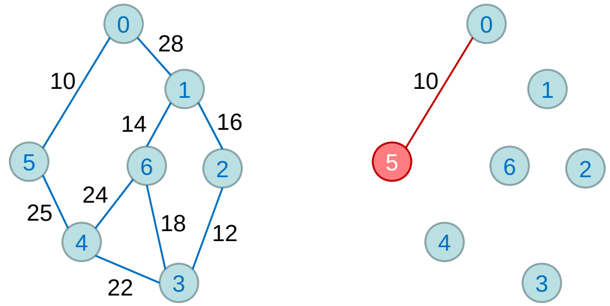
- Use a priority queue Q:
  - Each object is a vertex in $V - V_A$.
  - Key of v is minimum weight of any edge (u, v), where $u \in V_A$.
  - Then the vertex returned by EXTRACT-MIN is v such that there exists $u \in V_A$ and (u, v) is light edge crossing $(V_A, V - V_A)$.
  - Key of v is ∞ if v is not adjacent to any vertices in V_A.

Prim's Algorithm (3/3)

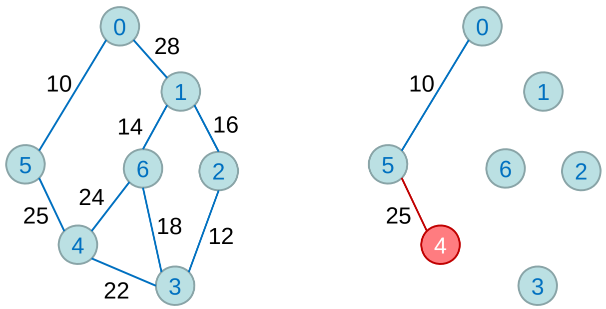
The edges of *A* will form a rooted tree with root r:

- r is given as an input to the algorithm, but it can be any vertex.
- Each vertex knows its parent in the tree by the attribute $v.\pi$ = parent of v. $v.\pi$ = NIL if $v = r$ or v has no parent.
- As algorithm progresses, $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
- At termination, $V_A = V \Rightarrow Q = \emptyset$, so MST is $A = \{(v, v.\pi) : v \in V - \{r\}\}$.

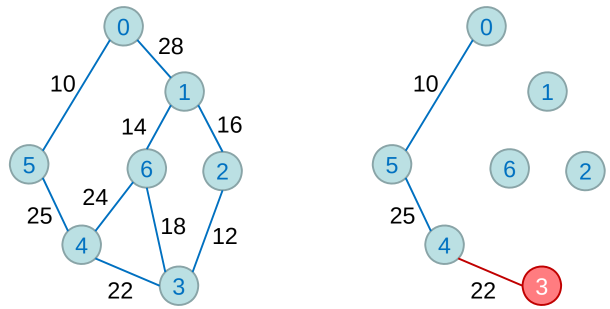
Stages in Prim's Algorithm



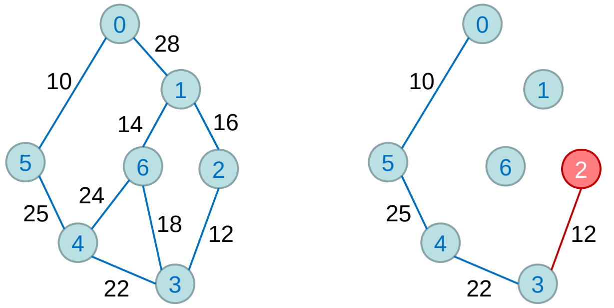
Stages in Prim's Algorithm



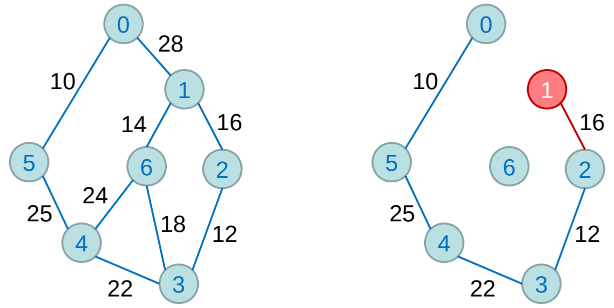
Stages in Prim's Algorithm



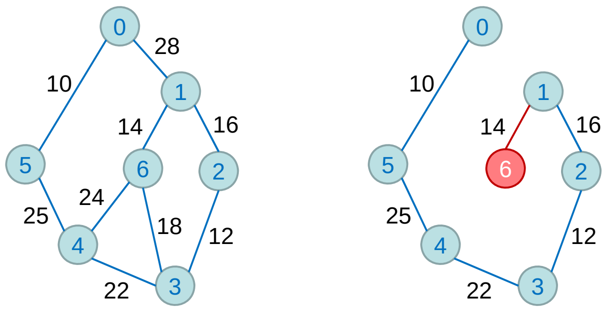
Stages in Prim's Algorithm



Stages in Prim's Algorithm



Stages in Prim's Algorithm



Pseudo Code of Prim's Algorithm

MST-PRIM(G, w, r)

for each $u \in G.V$
  $u.key = \infty$
  $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
  $u = \text{EXTRACT-MIN}(Q)$
  for each $v \in G.Adj[u]$
    if $v \in Q$ and $w(u, v) < v.key$
      $v.\pi = u$
      $v.key = w(u, v)$

Initialize the min-priority queue Q to contain all the vertices.

The key of root is set to 0 so that it will be the first vertex processed.

It identifies a vertex $u \in Q$ incident on a light edge that crosses the cut $(V– Q, Q)$.

Homework Assignment #6

Please implement both Kruskal's Algorithm and Prim's Algorithm.

- TAs will announce the detailed Input/Output format in Moodle.
- Please submit your program to e-Tutor.
- Please submit your README document to Moodle.
- **Due Date: 14 June 2017**.