Minimum spanning trees

- Given a graph G=<V, E> where V is the set of nodes and E is the set of edges. Each edge has a nonnegative weight.
  - Problem: find a subset T of edges in G such that
    - all the nodes remain connected,
    - and the total weight of the selected edges is minimal
- Let G'=<V, T> be the partial graph satisfying the conditions. G' is a tree, called *Minimum Spanning Tree.*

A generic algorithm for MST

```
genericMST(G)
{
   A = Φ;  // A is a partial solution

   while (A does not form a spanning tree) {
     find a safe edge (u,v) for A;
     A = A ∪ {(u,v)};
   }
   return A;
}
```

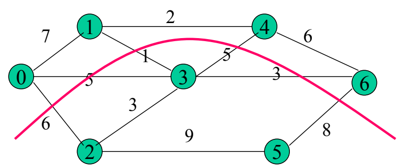
select depends on implementation

We need to find a **safe** edge:

If A is a subset of a minimum spanning tree,
A ∪ {(u,v)} is also a subset of a MST,
then (u,v) is **safe** for A

Some definitions

- A *cut* (S, V-S) of an undirected graph G=(V, E) is a partition of V.



S = {0, 1, 4, 6}
V-S = (2, 3, 5}

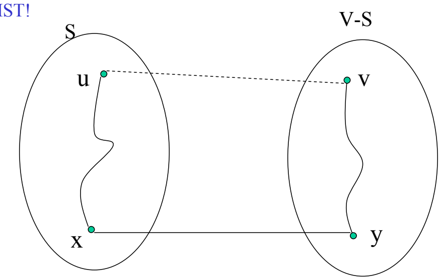
- An edge (u, v) *cross* the cut (S, V-S) is one of its endpoints is in S and the other in V-S
  - For example: (1, 3), (3, 4)
- An edge is a *light* edge crossing a cut if its weight is the minimum of any edge crossing the cut
  - For example: edge (1,3) in the graph above

A safe edge

- A cut *respects* a set *A* of edges if no edge in *A* crosses the cut
- Theorem
  - 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, let $(S, V\text{-}S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V\text{-}S)$. The edge (u,v) is safe for A
- Corollary
  - 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, let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = <V, A>$. If (u, v) be a light edge connecting C to other component in G_A. The edge (u,v) is safe for A

Proof

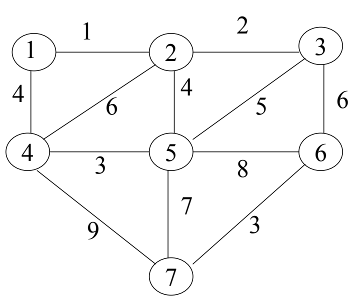
- T is an MST that includes A.
- If (u, v) $\in$ T, done.
- Otherwise, adding (u, v) to T create a cycle
- There exists an edge (x, y) $\in$ T that is a part of the cycle and crosses the cut.
  - We know w(u, v) <= w(x, y)
- T' = T – {(x,y)}$\cup${(u,v)}. w(T') = w(T)-w(x,y)+w(u,v)<=W(T). T' is also an MST!



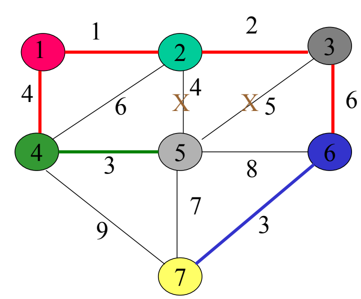
Kruskal's algorithm

- Given G=<V, E>
  - Sort the edges by increasing weight
  - The partial solution is <V, A> , and initially A is empty and each node forms a component
  - Loop: examining the edges in the order of increasing weight
    - Add an edge into the partial solution one by one, reject the edge if its two ends come from the same connected component
    - Stop when finishing one pass to all the edges
- Implementation
  - Disjoint sets for connected components

Example: Kruskal's algorithm

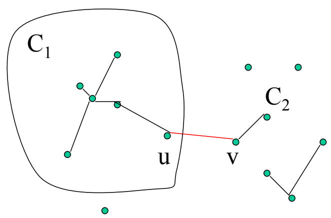


Example: Kruskal's algorithm



Optimality of Kruskal's algorithm

- Let C_1 and C_2 denote the two trees connected by the next lightest edge (u, v). (u,v) is a safe edge for C_1



Kruskal's algorithm

```
Kruskal(Graph G)  // G=<V, E>
{
   sort E by increasing weight;

    A = ϕ ;
    make n initial sets, each contains a node in V;

    for all sorted edges {
       e = <u,v>; // shortest edge not yet considered
       uComponent = find(u);
       vComponent = find(v);

       if (uComponent != vComponent) {
          Union(uComponent, vComponent);
          A = A ∪ {e};
       }
   }
   return A;
}
```

Kruskal's algorithm: cost

O(*E*log *E*)

called makeSet
V times

called at most *E*
times each

called V-1 times

Total:
O(ElogE + Eα(V))
=O(ElogV)

```
Kruskal(Graph G)  // G=<V, E>
{
   sort E by increasing weight;

    A = ϕ ;
    make n initial sets, each contains a node in V;

    for all sorted edges {
       e = <u,v>; // shortest edge not yet considered
       uComponent = find(u);
       vComponent = find(v);

       if (uComponent != vComponent) {
          Union(uComponent, vComponent);
          A = A ∪ {e};
       }
   }
   return A;
}
```

Kruskal's algorithm -- efficiency

Total cost

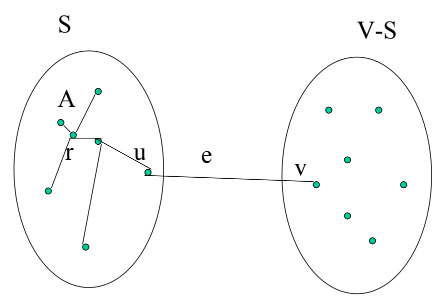
$$
\begin{aligned}
& O(E\log E) + O(E\,\alpha(V)) \\
&= O(E\log V) + O(E\,\alpha(V)) \quad \leftarrow \quad V-1 <= E <= V(V-1)/2 \\
&= O(E\log V) + O(E\log V) \quad \leftarrow \quad O(\alpha(V)) \subset O(\log V) \\
&= O(E\log V)
\end{aligned}
$$

Prim's algorithm

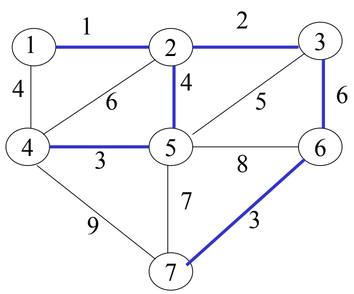
```
Prim(G)
{
    A = ϕ ;
    S = {an arbitrary root node of V};

    while ( S != V) {
        find e=<u,v> of minimum weight
          such that cross the cut (S, V-S);
        A = A ∪ {e};
        S = S ∪ {v};
    }
}
```

Proof Prim's algorithm

- Directly apply the corollary.



Example: Prim's algorithm



S: blue nodes
V-S: white nodes
A: all blue edges

Key to efficiency: find the edge connecting S and V-S with minimum weight

Implementation of Prim's algorithm

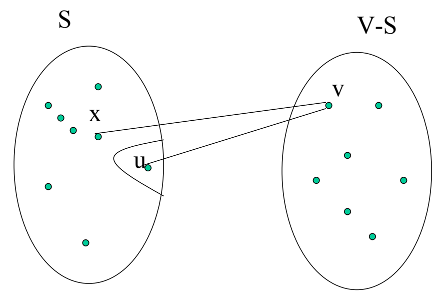
- For each node v
  - key[v] is the minimum weight that connect v to the partial spanning tree A
  - and π[v] is the other endpoint of the minimum edge
- Organize the nodes as a priority list

Prim’s algorithm: an implementation

```
Prim(G, w, r)
{
   for each node u ∈ V {
      key[u] = ∞;
      π[u] = null;
   }
   key[r] = 0;
   Q.build(V); // Q is a priority queue use key[] as keys

    while (!Q.empty()) {
        u = Q.extractMin();
        for each v adjacent to u {
             if (v ∈Q && w(u,v) < key(v)) {
               Q.updateKey(v, w(u,v));
               π[v] = u;
             }
        }
    }
}
```

Prim’s algorithm: an implementation

- A key observation
  - When a node u is added to S, we only update π[v] and key[v] for those nodes adjacent to u.



π[v] and key[v] could change after u is added to S

Cost: use min-heap to implement priority queue

```
Prim(G, w, r)
{
   for each node u ∈ V {
      key[u] = ∞;
      π[u] = null;
   }
   key[r] = 0;
   Q.build(V); // Q is a priority queue use key[] as keys

    while (!Q.empty()) {
        u = Q.extractMin();
        for each v adjacent to u {
             if (v ∈Q && w(u,v) < key(v)) {
               Q.decreaseKey(v, w(u,v));
               π[v] = u;
             }
        }
    }
}
```

O(V)

execute V times

O(VlogV)

execute E times
overall: O(ElogV)

Total: O(ElogV)