Minimum Spanning Tree

G = (V, E) with real-valued edge weights

MST (Minimum Spanning Tree): Subset of edges $T \subseteq E$,

Where T is a tree that connects all nodes whose sum of weights is minimized

Prim's Algorithm:

- Initialize $S = \{$ any one node $\}$.
- Add min cost edge $e = (u, v)$ with $u \in S$ and $v \in V - S$ to T
- Add v to S.
- Repeat until $S = V$

```
Prim(G, r):
for each v ∈ V do:
    v.key ← ∞. v.p ← nil, v.color ← white.
r.key ← 0.
Insert all keys in a min priority queue Q on V
while Q ≠ ∅:
    u ← Extract-min(Q)
    u.color ← black.
    for each v ∈ Adj[u] do:
        if v.color = white and w(u,v) < v.key then.
            v.p ← u
            v.key ← w(u.v)
            Decrease-key(Q. v. w(u.v))
```

Decrease v's key value to w(u.v) and rearrange queue Q

"Not preferred!"

modified

```
Prim(G, r):
for each v ∈ V do:
    v.key ← ∞. v.p ← nil, v.color ← white.
r.key ← 0.
Insert all keys in a min priority queue Q on V
while Q ≠ ∅:
    u ← Extract-min(Q)
    if u.color = black Continue.
    output(u, u.p)
    u.color ← black.
    for each v ∈ Adj[u] do:
        if v.color = white and w(u,v) < v.key then.
            v.p ← u
            v.key ← w(u.v)
            Insert(Q, v)
```

Cut Lemma

Assumption: All edges weights are distinct

Cut Lemma: for any subset S of nodes. Let e be the min-cost edge with one endpoint in S

Then MST Must exist e.

Proof: if not contains e. assume Contains e'. then we can change e' to e and the tree still holds while the total weight is lower. Contradiction. So e is in MST.

Kruskal's Algorithm:

Starts with empty tree T

Consider edges in increasing order of weight.

Case 1: If adding e to T creates a cycle ⇒ delete e

Case 2: if not. insert e = (u,v) to T

Implementation:

① How to detect cycle?

if DFS ⟶ O(|E|·|V|) time

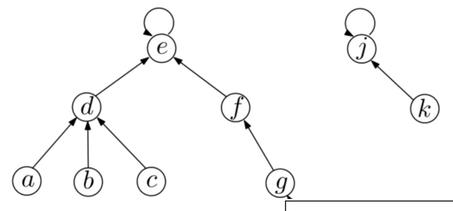
⇒ Want to check in O(log|V|) time

Observation: the actual structure of T does not matter

⇒ "Tree ⇔ A set of nodes"

After adding an edge. two sets are "union" together.

"Union-find" data structure




```
Make_set(x):
x.parent ← x
x.height ← 0.
```

```
Find_set(x):
if x.parent != x then:
    x ← x.parent
return x
```

```
Union(x,y): // Union By Height
a ← find_set(x)
b ← find_set(y)
if a.height ≤ b.height then:
    if a.height = b.height then.
        b.height ← b.height + 1.
    a.parent ← b
else b.parent ← a.
```

Running Time: O(log n) for "find-set" & "union"

proof by induction:

will show that for any tree with height h. its size is at least 2^h

At first h(x) = 0, size(x) = 1 ⇒ $1 \geq 2^0$ (Base case)

Assume, Above hypothesis holds before union(x,y), (inductive step)

Let the resulting tree (After union)'s height and size be h(x'), size(x')

- if h(x) < h(y)
  $size(x') = size(x) + size(y) \geq 2^{h(x)} + 2^{h(y)} \geq 2^{h(y)} = 2^{h(x')}$
- if h(x) = h(y)
  $size(x') = size(x) + size(y) \geq 2^{h(x)} + 2^{h(y)} = 2^{h(y)+1} = 2^{h(x')}$
- if h(x) > h(y)
  $size(x') = size(x) + size(y) \geq 2^{h(x)} + 2^{h(y)} \geq 2^{h(x)} = 2^{h(x')}$

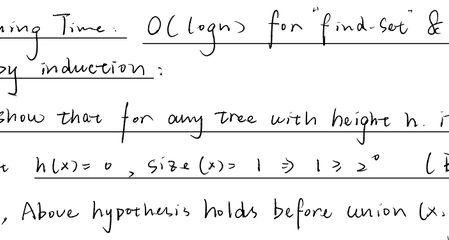
⇒ A tree with height h has size at least 2^h.

⇒ A tree with height log n has size at least n.

⇒ "find-set" is "climb up" whole tree ⇒ O(log n).

⇒ "union"'s running time is dominated by "find-set" ⇒ O(log n)

Path Compression



Idea:

- We have visited a number of nodes after Find-Set(x), and have reached the root r.
- We already know that these nodes belong to the set represented by r.
- Why not just set the parent pointers of these nodes to r directly?
  - Future operations will be faster!

Kruskal's Algorithm

```
MST-Kruskal(G)
for each vertex v ∈ V:
    Make_set(v)
Sort edges of G in order of edge weight
for each edge (u,v) ∈ E taken in above order.
    if find_set(u) != find_set(v) // if (u,v) will not form a cycle
        output edge (u,v)
        union(u,v)
```

Running Time: $O(|E| \cdot \log|E|) + O(|E| \log|V|) = O(|E| \log|V|)$

Sorting "|E| · find_set / union"

Note: If edges are already sorted and we use path compression, then the running time is close to $O(|E|)$.

Correctness of Algorithm ⇒ Cut lemma.

Observations on Prim's and Kruskal's algorithms

Both algorithms are Greedy since they make the choice that looks best at the moment

- Prim adds to MST lightest edge from S
- Kruskal adds to MST lightest edge that does not create a cycle.

For both Prim's and Kruskal's algorithm, we assumed that all the edges have different weights.

If we remove this assumption (and allow some or even all edges to have the same weight) the algorithms still work. The only thing that needs to be changed is that, instead of choosing **the** smallest cost edge, we choose **a** smallest cost edge (breaking ties arbitrarily).