Minimum Spanning Tree (MST)

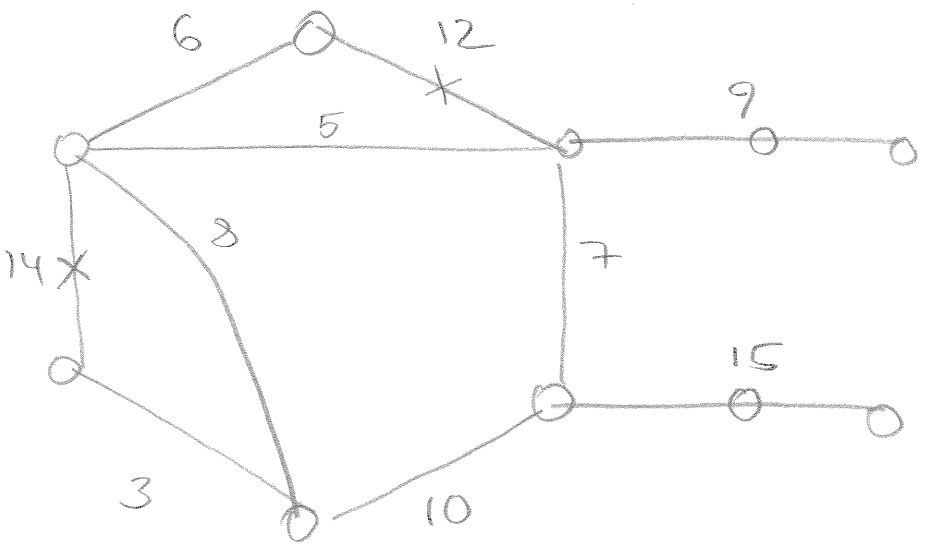
Input: connected, undirected graph $G = (V, E)$ with edge weight function $w : E \longrightarrow \mathbb{R}$

Output: a spanning tree T of minimum weight, i.e., $T \subseteq E$, (V, T) is a tree, and

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

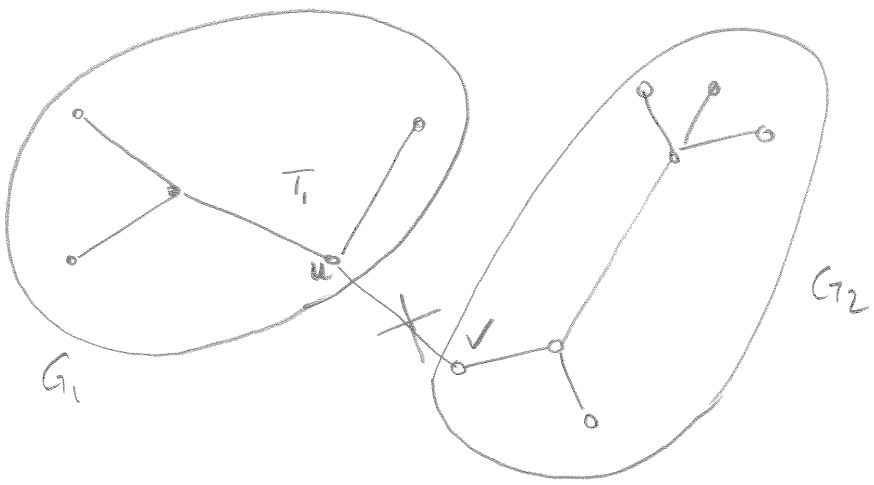
is minimal for all such T

Example



Optimal Substructure

T minimal spanning tree of G



Remove some edge (u,v)

$\Rightarrow$ T is partitioned into two subtrees T_1, T_2

$\Rightarrow$ G is partitioned into subgraphs G_1, G_2

where $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and

V_i = nodes of T_i $(i = 1,2)$

Theorem T_1 is a MST for G_1

(and T_2 for G_2)

Proof: Cut & Paste

$$w(T) = w(u,v) + w(T_1) + w(T_2)$$

Assume: T_1' is a better spanning tree for G_1 than T_1

Then: $T' := \{(u,v)\} \cup T_1' \cup T_2$ is a better ST for G than T ↯

Overlapping Subproblems

- Idea: Split G in two: G_1, G_2
  find MSTs T_1, T_2
  combine to T
- Overlapping subproblems exist

Better:

Hallmark for greedy algorithms:

"A locally optimal choice is globally optimal"

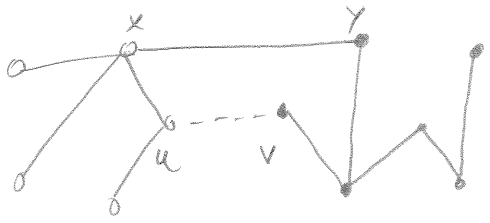
Greedy choice property

Definition: Let $G=(V,E)$, $V = V_1 \dot{\cup} V_2$ (i.e., $V_1 \cap V_2 = \emptyset$, $V = V_1 \cup V_2$), G_1, G_2 subgraphs induced by V_1, V_2. Then G_1, G_2 is a cut of G.

Theorem: Let G_1, G_2 be a cut of G.
Let $(u,v) \in E$, $u \in V_1$, $v \in V_2$, be the least-weight edge connecting V_1 and V_2.
Then:

- $(u,v) \in T$ for every MST T
- T_1, T_2 MSTs for G_1, G_2, resp.
  $\Rightarrow T_1 \cup T_2 \cup \{(u,v)\}$ MST for G

Proof: Assume T is MST and $(u,v) \notin T$.
There is another edge (x,y) connecting V_1, V_2.
$\Rightarrow w(x,y) > w(u,v)$



Replace (x,y) with (u,v) in T
$\Rightarrow$ ST with lower weight results

Corollary: Let all weights of edges in G be different. Then there is a unique MST of G

Proof: Assume T, T' are two MSTs.

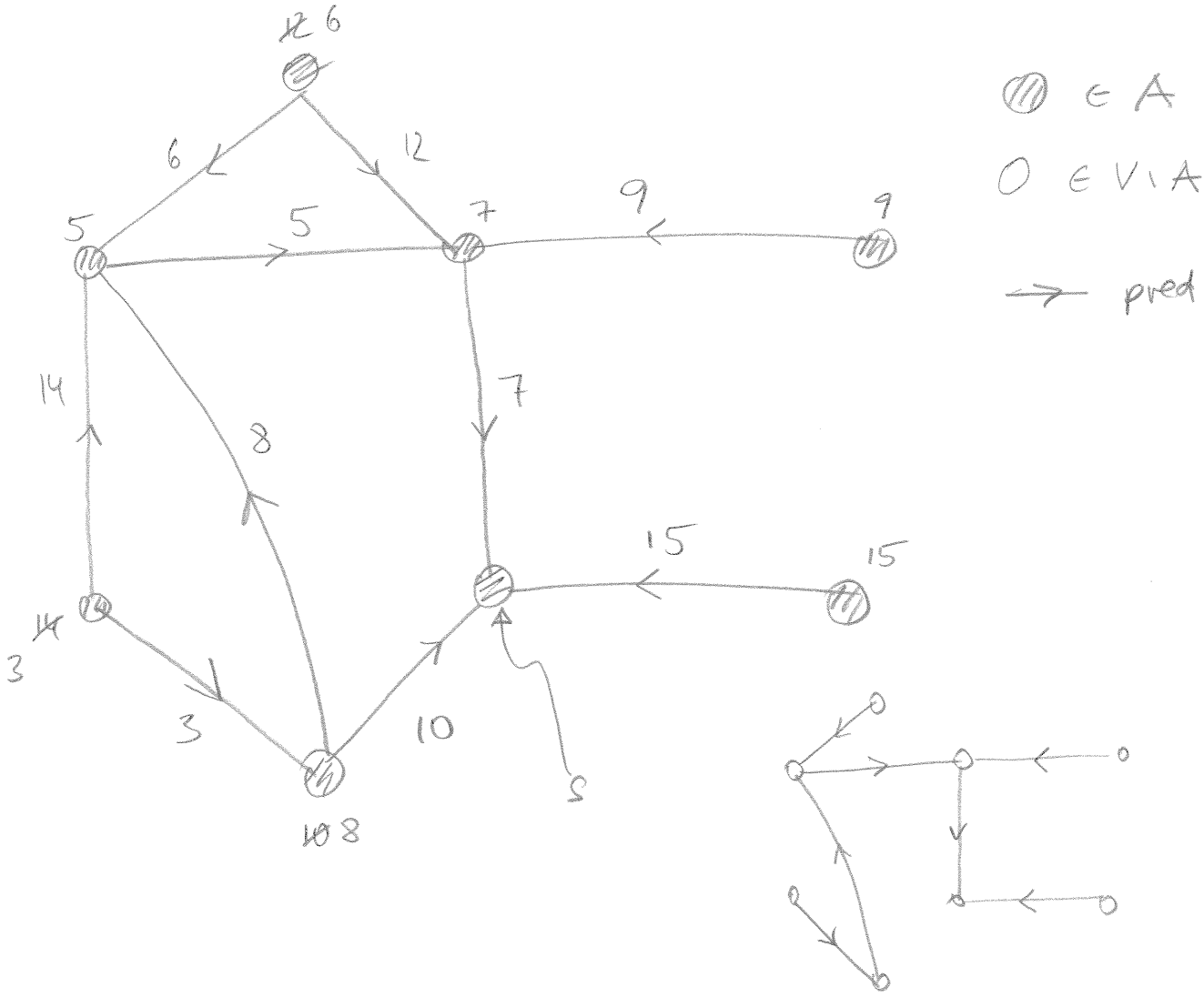
Ex. edge $(u,v) \in T \setminus T'$.

Removing (u,v) splits T into T_1, T_2 with vertices V_1, V_2. In T' must exist edge (x,y) connecting V_1, V_2. Replacing (x,y) with (u,v) gives a better tree $\Rightarrow$ T' is not MST ↯

From now on, assume distinct weights

Prim's algorithm (Idea):

- Start with arbitrary node s
- Grow tree T for node set $A \subseteq V$:
  add [least-weight edge]
  connecting A and $V \setminus E$ to T
  ↑ must be in T



Prim's algorithm (code)

Idea:

- Maintain $V \setminus A$ as a priority queue Q
- Key each vertex in Q with weight of least-weight edge connecting it to a vertex in A

$\Theta(V)$
- Q.init(V)
- for v in V do v.key := ∞
- s.key := 0

|V| times
- while Q.notempty() do
  - u := Q.extractMin()
  - $\deg(u)$ for v ∈ u.adj() do
    - if v ∈ Q and w(u,v) < v.key
      - then v.key := w(u,v)
        - v.pred := u

return $\{(v, v.\text{pred}) \mid v \in V\}$

Q.decreaseKey(v, w(u,v))

Priority Queues (Abstract Data Type)

set S of items

function key: $S \longrightarrow \mathbb{R}$

Operations

- Q.init(S)
- Q.extractMin()
- Q.decreaseKey(x, val)

Implementations

- array
- binary heap
- doubly linked list

Analysis

Depends on operations in priority queue

- extractMin()
- decreaseKey(v, w(u,v))
  moves v up in queue

$$\text{Time} = \Theta(V \cdot T_{\text{extractMin}} + E \cdot T_{\text{decreaseKey}})$$

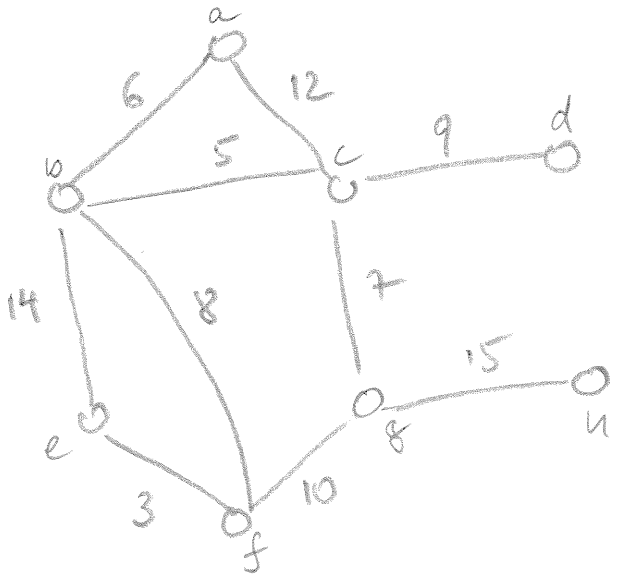
Q	$T_{\text{extractMin}}$	$T_{\text{decreaseKey}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \cdot \lg V)$
(Fibonacci heap)	$O(\lg V)$	$O(1)$	$O(E + V \cdot \lg V)$

Randomized alg $\Theta(V+E)$ expected time

Kruskal's algorithm

- edge based
- sort edges according to weight
- check edges
  - add if useful
  - ignore otherwise
- Idea: forest grows into single tree

Example



Edges:

3, 5, 6, 7, 8, 9, 10, 12, 14, 15

Nodes

a, b, c, d, e, f, g, h

Operations on disjoint sets

P partition of V

($P \subseteq 2^V$; $\bigcup_{X \in P} X = V$; sets in P are mutually disjoint)

- find (P, x)

  return X in P s.th. $x \in X$

- union (P, x, y)

  $P := (P \setminus \{X, Y\}) \cup \{X \cup Y\}$

  where $X = \text{find}(P, x)$, $Y = \text{find}(P, y)$

- addSingleton (P, x)

  $P := P \cup \{\{x\}\}$

Also, OO notation, e.g.

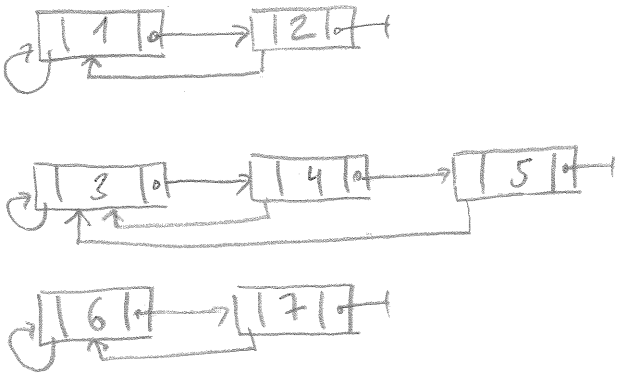
P.find(x)

Implementation of disjoint sets

$$P = \{ \{1,2\}, \{3,4,5\}, \{6,7\} \}$$

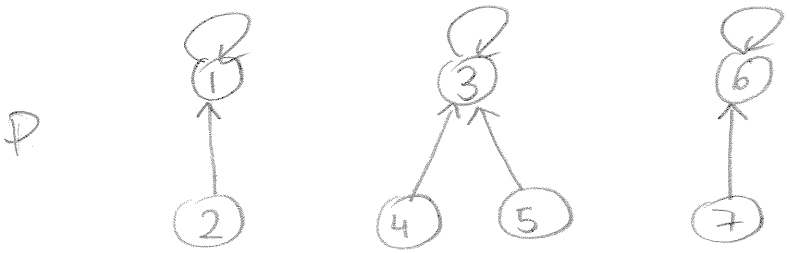
List based

- set is list
- set identified by first element

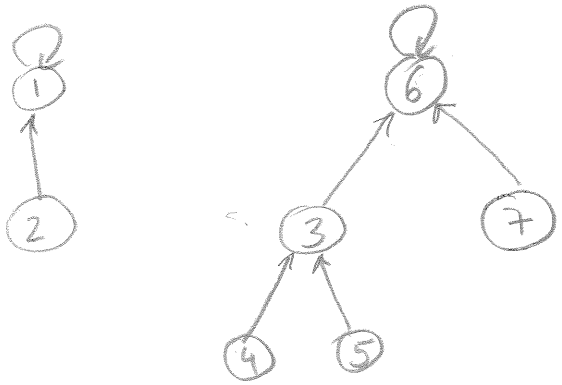


- P.find(x)
  go to first element : $O(1)$
- P.addSingleton(x)
  add to tail: $O(1)$
- P.union(X,Y)
  add shorter set to longer (counter!); $O(1)$
  reset head pointers of shorter:
  $O(\min\{|P.find(x)|, |P.find(y)|\})$

Tree-based

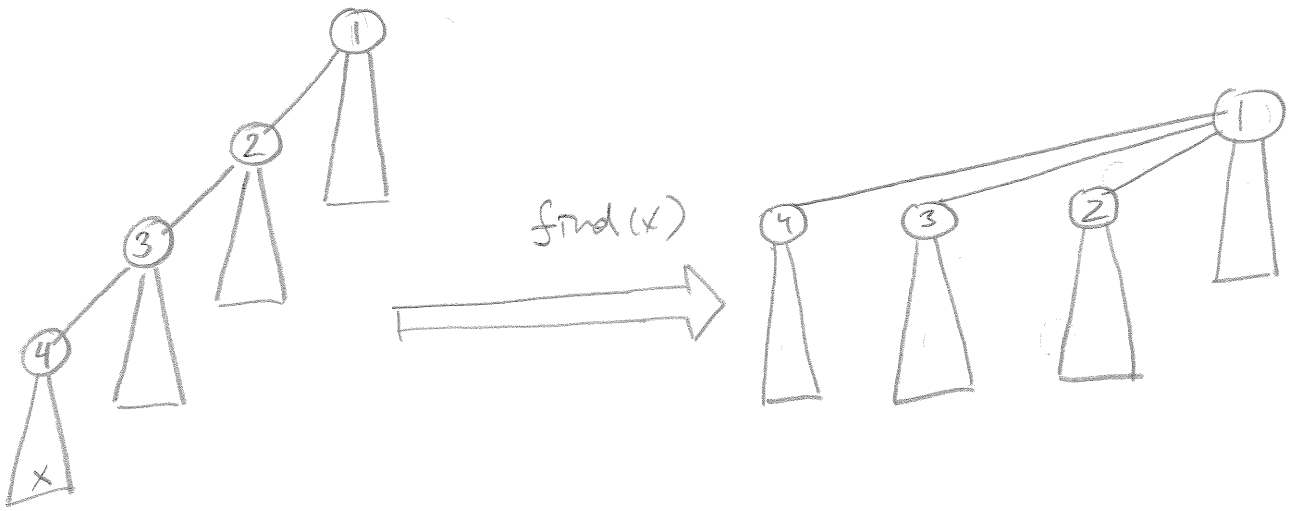


p.union(5, 7)



Union fast, find slow

Improve by path compression during find



Kruskal Code

```
MSTK(G)
    T := ∅
    P.init()
    for v in G.V do
        P.addSingleton(v)
    EdgeList := sorted list of edges in G.E
    for (u,v) in EdgeList do
        if P.find(u) ≠ P.find(v)
            then P.union(u,v)
                 T := T ∪ {(u,v)}
    return T
```

Kruskal Analysis

- Initialization : $O(V)$
- Sorting edges: $\sigma(E \cdot \log E)$
  $= \sigma(E \cdot \log V)$
- Find(·): $\sigma(E)$ calls

Analysis of union

- Charge Union(u, v)
  - to v, if v's set is smaller
  - to u else
- Let $t(v) :=$ # unions charged to v
  $\Rightarrow t(v) \leq \log_2 V$
  (v's set at least doubles its size)
- Cost of union: $\sum_{v \in V} t(v) \leq V \cdot \log_2 V$

Total time:

$$\sigma(V) + \sigma(E \cdot \log V) + \sigma(E) + \sigma(V \cdot \log V)$$

$$= \Theta(E \cdot \log V)$$