CSE 100: Algorithm Design and Analysis
Chapter 23: Minimum Spanning Trees

Sungjin Im

University of California, Merced

Last Update: 4-14-2023

Minimum Spanning Tree

Problem definition

Input: Undirected graph $G = (V, E)$, with each edge $(u, v) \in E$ having weight/cost $w(u, v)$.

Output: A minimum spanning tree $T \subseteq E$.

Minimum Spanning Tree

Problem definition

Input: Undirected graph $G = (V, E)$, with each edge $(u, v) \in E$ having weight/cost $w(u, v)$.

Output: A minimum spanning tree $T \subseteq E$.

Terminology:

- Tree: a connected graph with no cycles.
- Spanning tree (of G): tree that connects all vertices of G.
- Minimum spanning tree: a spanning tree T whose total edge weight, $\sum_{(u,v)\in T} w(u, v)$, is minimized.

Preliminaries

If a tree T has n vertices, then T must have $n-1$ edges.
True or False?

Any graph $G=(V,E)$ that has $|V|-1$ edges is a tree.
True or False?

Consider any tree $T=(V,E)$. For any pair of vertices $u,v \in V$, there is a unique path from u to v on T.
True or False?

Preliminaries

If a graph G has $|V|$ edges or more, then it must have a cycle.
True or False?

If a graph G is connected and has $|V|$ edges, then it has a unique cycle.
True or False?

Minimum Spanning Tree

We will learn two algorithms, Kruskal's and Prim's, which have a similar framework.

In general, there may exist more than one MSTs. However, we will see that there is a unique MST if all edges have distinct weights. (See Problem 23-1).

Minimum Spanning Tree

An Algorithmic Framework

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

We say edge (u, v) is safe for A if $(u, v) \cup A$ is a subset of a MST.

Definition:

- A cut $(S, V - S)$ of $G = (V, E)$ is a partition of V.
- Edge $(u, v) \in E$ crosses cut $(S, V - S)$ if one of its end points is in S, and the other is in $V - S$.
- An edge (u, v) is a light edge crossing a cut $(S, V - S)$ if its weight is the minimum over all edges crossing the cut.

Minimum Spanning Tree

Definitions

Definition:

- A cut $(S, V - S)$ of $G = (V, E)$ is a partition of V.
- Edge $(u, v) \in E$ crosses cut $(S, V - S)$ if one of its end points is in S, and the other is in $V - S$.
- An edge (u, v) is a light edge crossing a cut $(S, V - S)$ if its weight is the minimum over all edges crossing the cut.

Note: If all edges have distinct weights, then there is a unique light edge for each cut.

Theorem (23.1)

Let $G = (V, E)$ be a connected, undirected graph with weight $w(u, v)$ on each edge (u, v). Let A be a subset of E that is included in some MST for G. Let (u, v) be a light edge crossing a cut $(S, V - S)$ that respects A. Then, (u, v) is safe for A.

Theorem (23.1)

Let $G = (V, E)$ be a connected, undirected graph with weight $w(u, v)$ on each edge (u, v). Let A be a subset of E that is included in some MST for G. Let (u, v) be a light edge crossing a cut $(S, V - S)$ that respects A. Then, (u, v) is safe for A.

Theorem (Safe edges when edges have distinct weights)

Let $G = (V, E)$ be a connected, undirected graph where each edge (u, v) has a distinct weight $w(u, v)$. If (u, v) is the unique light edge crossing some cut $(S, V - S)$, then (u, v) must be included in all MSTs.

Theorem 23.1 becomes simpler under the assumption that edges have distinct weights.

Instead, we will focus on the following simpler theorem assuming that all edge weights are distinct.
Justification: perturb edge weights or break ties consistently.
Throughout, we assume that the graph is connected and all edges have distinct weights.
Under this assumption,

Definition

An edge e is safe if it is the cheapest edge crossing some cut of G.

Theorem (Safe edges can be safely chosen)

A safe edge is included in all MSTs (of G).

Theorem (Safe edges can be safely chosen)

A safe edge is included in all MSTs (of G).

Proof.

Say there is a MST T that doesn't include a safe edge (u, v) w.r.t. some cut $(S, V - S)$.
There is a unique path P between u and v on T.
$P \cup (u, v)$ forms a cycle.
There is another edge (x, y) crossing $(S, V - S)$. Replace (x, y) with (u, v).
Remains connected. Cost decreased, contradiction. $\square$

Minimum Spanning Tree

Finding safe edges

Theorem (Uniqueness of MST when edges have distinct weights)

If $G = (V, E)$ be a connected, undirected graph whose edge weights are distinct, then there is a unique MST for G.

Proof.

(sketch)

Minimum Spanning Tree

Finding safe edges

Theorem (Uniqueness of MST when edges have distinct weights)

If $G = (V, E)$ be a connected, undirected graph whose edge weights are distinct, then there is a unique MST for G.

Proof.

(sketch) Show that the set of safe edges spans all vertices. It suffices to show that every edge e on an arbitrary MST T is safe: Cutting e partitions T into two components, T_1 and T_2. Let $V_1 = V(T_1)$ and $V_2 = V(T_2)$.
Show e is the cheapest edge crossing cut (V_1, V_2). $\square$

Minimum Spanning Tree

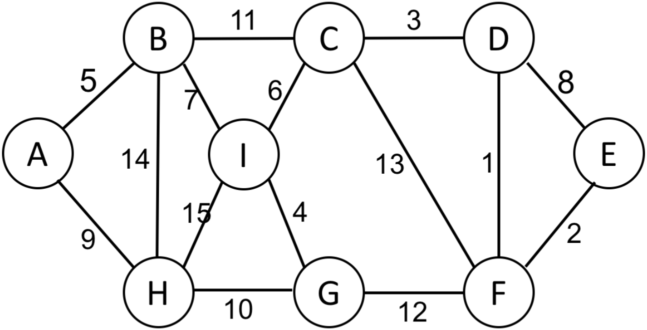
Corollary

An edge $(u, v) \in E$ belongs to the unique MST of G if and only if the edge is safe.

Minimum Spanning Tree

Kruskal's algorithm

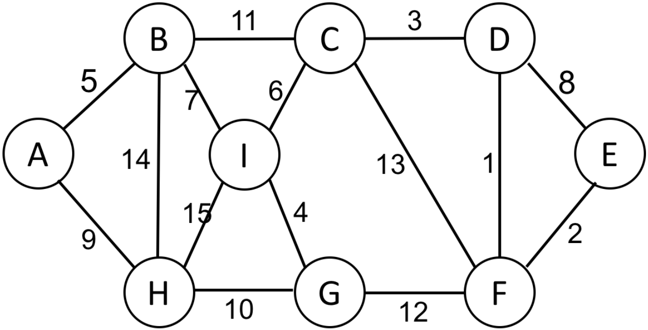
Repeatedly finds and adds the cheapest edge that connects any two trees in the current forest.



Minimum Spanning Tree

Kruskal's algorithm

Repeatedly finds and adds the cheapest edge that connects any two trees in the current forest.



Do you see why all added edges are safe?

Repeatedly finds the cheapest edge that connects any two trees in the current forest.

```
MST-KRUSKAL(G, w)
A = ∅
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
```

Preliminaries

Ch21: Set Operations and Maintaining connected components

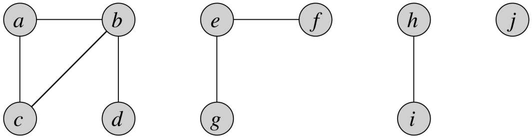
A disjoint-set data structure maintains a collection $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ of disjoint dynamic sets.
Each set is represented by an element in the set.

Three operations:

- Make-Set(x): creates a new set whose only member is x. (x should not appear in other sets in $\mathcal{S}$).
- Union(x,y): Merge two (distinct) sets containing x, y into one.
- Find-Set(x): returns (a pointer to) the representative of the set containing x.

Preliminaries

Ch21: Set Operations and Maintaining connected components



(a)

Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

(b)

Preliminaries

Ch21: Set Operations and Maintaining connected components

Computing connected components.

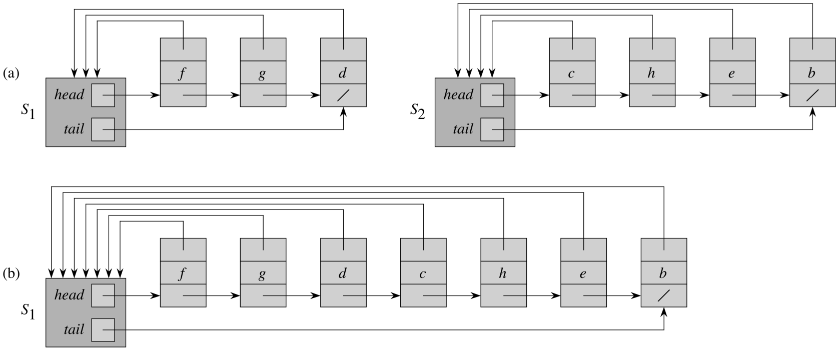
Computing connected components.

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```

Preliminaries

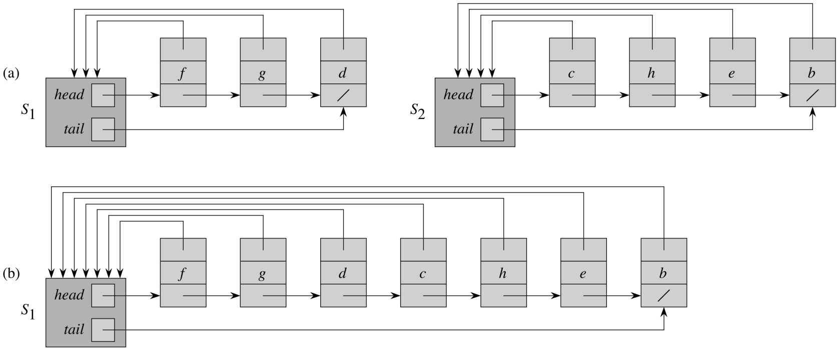
Ch21: Set Operations and Maintaining connected components: implementation



Running time of Find-Set(x)?

Preliminaries

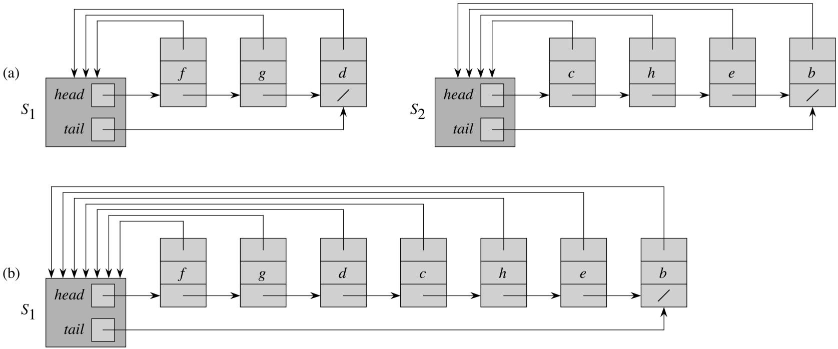
Ch21: Set Operations and Maintaining connected components: implementation



Running time of Find-Set(x)? $O(1)$.

Preliminaries

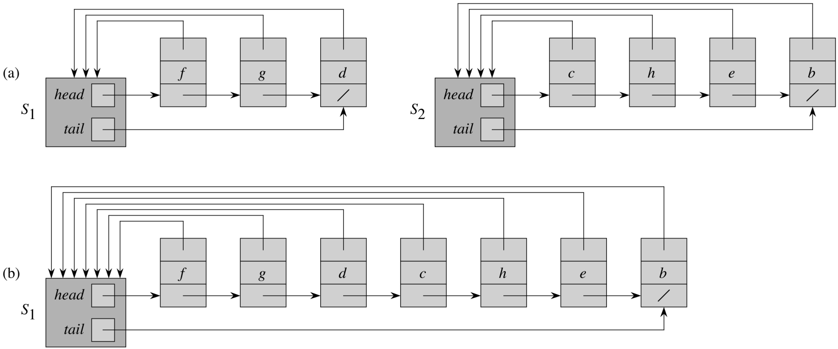
Ch21: Set Operations and Maintaining connected components: implementation



Running time of Find-Set(x)? $O(1)$.
Running time of Make-Set(x)?

Preliminaries

Ch21: Set Operations and Maintaining connected components: implementation



Running time of Find-Set(x)? $O(1)$.
Running time of Make-Set(x)? $O(1)$.

Preliminaries

Ch21: Set Operations and Maintaining connected components: implementation

Linked Lists + Weighted Union

(Amortized) Running time: If we do a sequence of m Make-Set, Union, and Find-Set operations on n elements, it takes $O(m + n \log n)$ time.

Preliminaries

Ch21: Set Operations and Maintaining connected components: implementation

Union by rank $+$ path-compression

(Amortized) Running time: If we do a sequence of m Make-Set, Union, and Find-Set operations on n elements, then it takes $O(m\alpha(n))$ time.

* $\alpha(n) \leq 4$ for all $n \leq 10^{80}$ (ch 21.4)

Minimum Spanning Tree

Kruskal's algorithm

Repeatedly finds the cheapest edge that connects any two trees in the current forest.

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges of $G.E$ into nondecreasing order by weight w
for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A

Minimum Spanning Tree

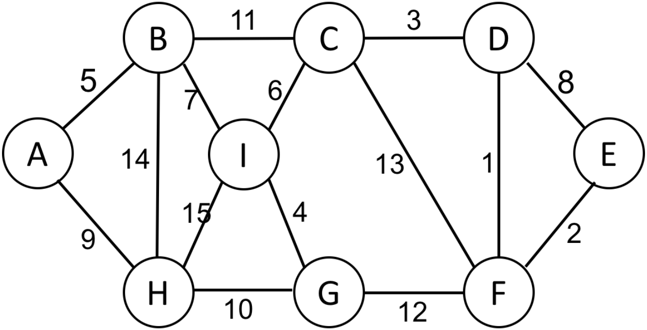
Kruskal's algorithm running time

```
MST-KRUSKAL(G, w)
A = ∅
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
```

Make-Set: $O(V)$.
Sorting: $O(E \log E) = O(E \log V)$
$O(E)$ Find-set and Union operations: $O(E + V \log V)$.
Since $E \geq V - 1$, we have $O(E \log V)$.

Minimum Spanning Tree
Prim's algorithm

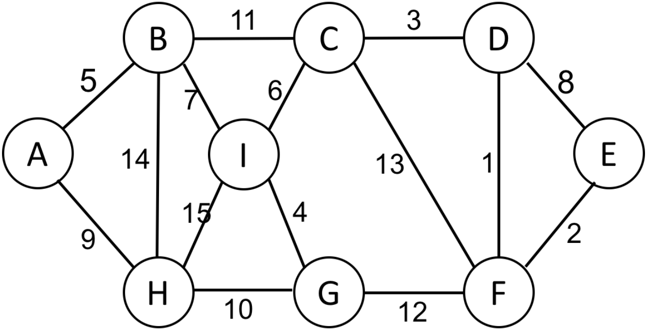
Starts from an arbitrary vertex r (root). Grows a single tree T in each iteration by adding a light edge crossing $(T.V, V - T.V)$.



Minimum Spanning Tree

Prim's algorithm

Starts from an arbitrary vertex r (root). Grows a single tree T in each iteration by adding a light edge crossing $(T.V, V - T.V)$.



Do you see why all added edges are safe?

Minimum Spanning Tree

Prim's algorithm

Starts from an arbitrary vertex r (root). Grows a single tree T in each iteration by adding a light edge crossing $(T.V, V - T.V)$.

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Minimum Spanning Tree

Prim's algorithm

for each $u \in V$,

    $u.key = \infty$, $u.\pi = NIL$. Insert(Q, u)

Decrease-key$(Q, r, 0)$.

while $Q \neq \emptyset$

    $u =$ Extract-Min(Q).

    for ecah $v \in Adj[u]$

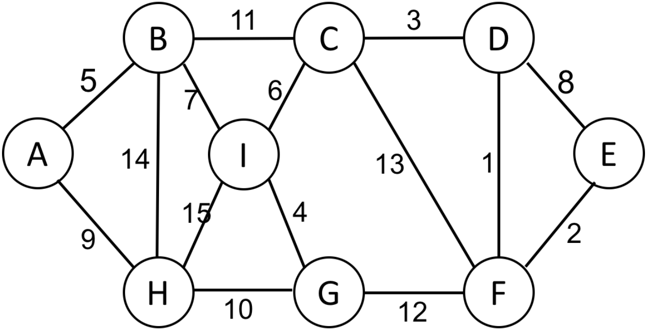
        if $v \in Q$ and $w(u, v) < v.key$

            $v.\pi = u$

            Decrease-Key$(Q, v, w(u, v))$

Minimum Spanning Tree

Prim's algorithm



Minimum Spanning Tree

Prim's algorithm

for each $u \in V$,

  $u.key = \infty$, $u.\pi = NIL$. Insert(Q, u)

Decrease-key$(Q, r, 0)$.

while $Q \neq \emptyset$

  $u =$ Extract-Min(Q).

  for ecah $v \in Adj[u]$

    if $v \in Q$ and $w(u, v) < v.key$

      $v.\pi = u$

      Decrease-Key$(Q, v, w(u, v))$

Running Time. If binary heap was used for min-priority queue:

Total Extract-Min: $O(V \log V)$

Total Decrease-Key: $O(E \log V)$

So, $O(E \log V)$.

Minimum Spanning Tree

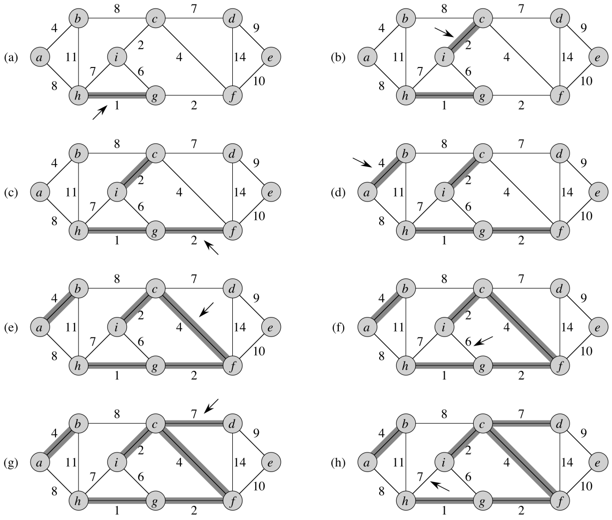
Prim's algorithm

for each $u \in V$,
    $u.key = \infty$, $u.\pi = NIL$. Insert(Q, u)
Decrease-key$(Q, r, 0)$.
while $Q \neq \emptyset$
    $u =$ Extract-Min(Q).
    for ecah $v \in Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            Decrease-Key$(Q, v, w(u, v))$

Running Time. If Fibonacci heap (ch 19) was used for MPQ:
Extract-Min: $O(\log V)$ and Decrease-Key: $O(1)$ (amortized).
Total Extract-Min: $O(V \log V)$
Total Decrease-Key: $O(E)$
So, $O(E + V \log V)$.

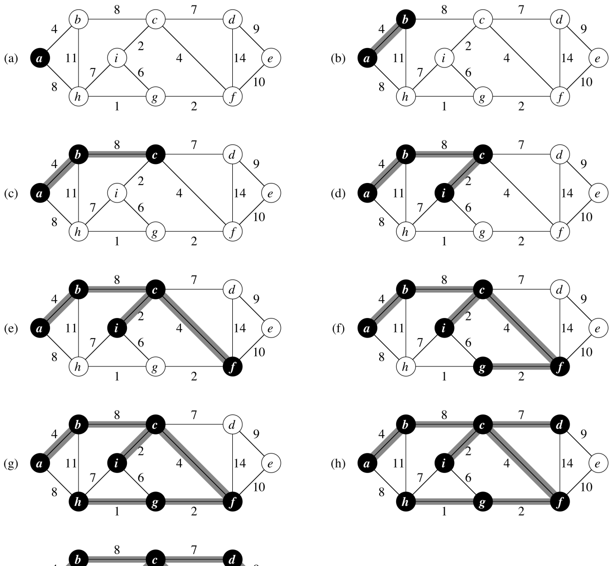
Minimum Spanning Tree

Kruskal vs Prim



Minimum Spanning Tree

Kruskal vs Prim



Minimum Spanning Tree

Other potential MST algorithms

Correct or incorrect?

a. MAYBE-MST-A(G, w)

```
sort the edges into nonincreasing order of edge weights w
T = E
for each edge e, taken in nonincreasing order by weight
    if T - {e} is a connected graph
        T = T - {e}
return T
```

Minimum Spanning Tree

Other potential MST algorithms

Correct or incorrect?

b. MAYBE-MST-B(G, w)

$T = \emptyset$
for each edge e, taken in arbitrary order
    if $T \cup \{e\}$ has no cycles
        $T = T \cup \{e\}$
return T

Minimum Spanning Tree

Other potential MST algorithms

Correct or incorrect?

c. MAYBE-MST-C(G, w)

$T = \emptyset$
for each edge e, taken in arbitrary order
    $T = T \cup \{e\}$
    if T has a cycle c
        let e' be a maximum-weight edge on c
        $T = T - \{e'\}$
return T