Lecture 10 Greedy Algorithms: Minimum Spanning Trees	L10.1 6.046 3/12/2020

Today:
- MST Problem
- MST Property and more
- Kruskal's Algorithm
- Prim's Algorithm

Reading:
- CLRS Chpt 23
- CLRS §§ 16.1, 16.2
- (CLRS Apdx B.4, B.5)

Greedy Algorithm —
- Overall problem solved in a series of steps
- Choice made at each step "looks best at the moment" (makes the most progress toward overall goal), without explicit reference to overall problem.

[Sometimes called "locally" best or optimal, while the overall problem is called "global"]

In lecture 01 applied greedy algorithm to scheduling problem — today will apply greedy algorithm to a graph problem.

Tree: A connected graph with no cycles

Spanning Tree: A subset of a graph's edges that forms a tree containing all vertices

Minimum Spanning Tree (MST) Problem:

Given: Graph $G=(V,E)$ & edge weights $w: E \to \mathbb{R}$

Find: Spanning tree $T \subseteq E$ of minimum weight

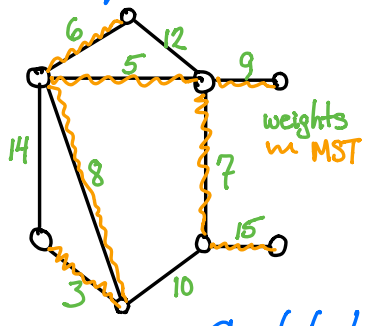
$$w(T) = \sum_{e \in T} w(e)$$

Applications:

- Planning minimum-length networks for connecting cities or stations with fiber optics, pipelines, or other infrastruct.
- Clustering through cutting longest edges in MST
- Finding higher-order correlative relationships in large datasets from MST in mutual information graph

⋮

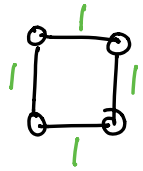
Example:



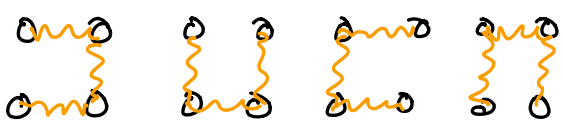
heuristics:

- avoid large weights (14, 15) ✓ ✗
- include small weights (3, 5) ✓ ✓
- some edges are inevitable (9, 15)
  - "only route"
- cycles forbidden (not a tree)

Could be multiple MSTs:



has 4 MSTs:

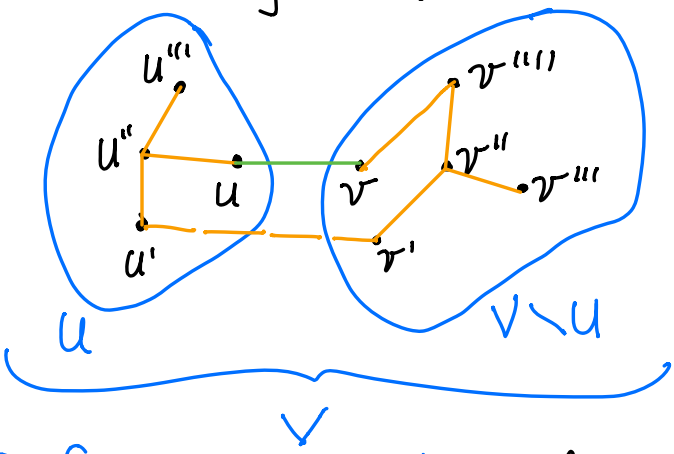


The MST Property

– Somewhat different from book

L10.3

Theorem: $G = (V, E)$ is connected graph with cost function defined on edges. U is proper subset of V & $U \neq \emptyset$. If (u,v) is an edge of lowest cost ("light edge") with $u \in U$ & $v \in (V \setminus U)$, then there is an MST containing (u,v).



- This partition of V into U & $V \setminus U$ is called a "cut."
- An edge "crosses" the cut if one vertex is in U and the other in $V \setminus U$.
- A cut "respects" a set of edges if no edge in the set crosses the cut.
- If (u,v) is the unique lightest edge, then (u,v) is in all MSTs.

Proof (by "cut-and-paste"): Assume the theorem is false, and there is no MST that includes (u,v). Let T be an MST.

- Adding (u,v) to T introduces a cycle, because T is an MST and so already contains a path connecting u & v.
- There must be at least one other edge connecting U and $V \setminus U$ in the cycle. WLOG call this (u',v').
- Deleting (u',v') breaks cycle, giving tree T'.
- Because no MST includes (u,v), T' can not be an MST and $w(T') > w(T)$.
- This contradicts the given statement that (u,v) is a light edge crossing the cut $\rightarrow$ $w(T') \leq w(T)$
- Thus assumption wrong $\Rightarrow$ (u,v) is in an MST.

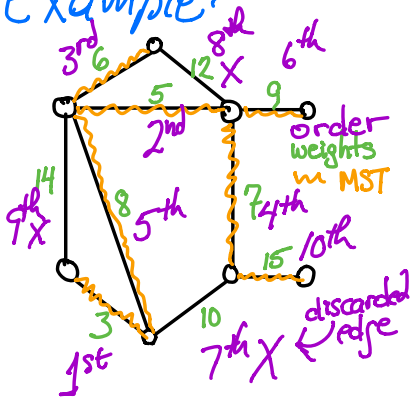
So how might a greedy alg. work here? | L10.4

- What if we sequentially add lowest cost edges that don't form cycles?

Kruskal's Algorithm

- Initially $T = (V, \emptyset)$ ← vertices but no edges
- Examine edges in increasing weight order ← arbitrarily break ties!
  - If edge connects two unconnected components then add edge to T
  - Else discard edge and continue ← edge forms cycle
- Can terminate when all vertices in single connected tree, or can continue through all edges.

Example:



Correctness (by loop invariant):

- Invariant: Prior to each iteration, T is a subset of an MST.
- Initialization: T has no edges, and so is trivially satisfied.
- Maintenance: Edges that are added to T in the loop are light edges crossing a cut of V (otherwise the edge would create a cycle). By the MST property, these edges are in MST.
- Termination: All edges are examined and added to T if in MST, so T must be MST.

Note:

① For distinct weights, MST is unique and proof okay.

② For non-distinct weights, proof only shows edges are in an MST, but not necessarily the same MST. Book has stronger proof.

Implementation: Use Union-Find DS to maintain connected components of MST

- $T = \emptyset$ — $\Theta(1)$
- for $v \in V$: MAKE-SET(v) — $\Theta(|V|) \cdot T_{\text{Make-Set}}$
- sort E by w — $O(|E| \log |E|)$
- for $e = (u, v) \in E$ (in non-decreasing weight order) — $O(|E|)(T_{\text{FIND-SET}} + T_{\text{UNION}})$
  - if FIND-SET(u) $\neq$ FIND-SET(v) ← u & v are in different components $\Rightarrow$ e won't create cycle
    - add e to T
    - UNION(u, v)

Time:

$$T = O(E \log E) + O(V) \cdot T_{\text{Make-Set}} + O(E)(T_{\text{Find-Set}} + T_{\text{Union}})$$

($T_{\text{Make-Set}} = \Theta(1)$; $T_{\text{Find-Set}} + T_{\text{Union}} = O(\lg(V))$ inverse Ackermann, amortized)

$$= O(E \log E) + O((V+E)\lg(V))$$

($|E| \geq |V| - 1$, $|E| < |V|^2$)

$$= O(E \log V)$$

CLRS Theorem 23.1

Given:

1. $G = (V, E)$ is connected, undirected graph with real-valued weight function w on E
2. A is a subset of E included in some MST for G
3. $(U, V \setminus U)$ is a cut of G that respects A
4. (u, v) is a light edge crossing the cut

Then: Edge (u, v) is "safe" for A

↳ edge can be added to A and the new edge set is still included in some MST of G

Proof: Cut-and-paste (see CLRS)

CLRS Corollary 23.2

Given: Same setup as Theorem 23.1 (①+②), plus
- $C = (V_C, E_C)$ is a connected component in forest $G_A = (V, A)$
- (u,v) is light edge connecting C to some other component in $G_A = (V, A)$

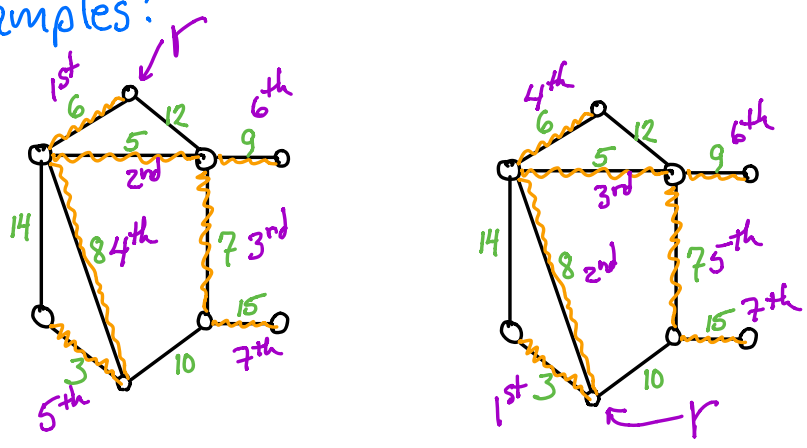
Then: (u,v) is safe for A

Proof: Examine the cut $(V_C, V \setminus V_C)$
- This cut respects A
- (u,v) is a light edge for this cut
- $\Rightarrow$ (u,v) is safe for A by Theorem 23.1

Prim's Algorithm

Select a vertex to start, r
$T \leftarrow \{r\}$
until done
  select light edge connecting T to isolated vertex, (u,v)
  $T \leftarrow T \cup \{(u,v)\}$

Examples:



Correctness (by loop invariant):

- Invariant: Prior to each loop iteration, T is a connected component and a subset of an MST
- Initialization: T has no edges; trivially satisfied
- Maintenance: New edge added to T in loop is light edge on a cut respecting connected component T, and so is safe for T, maintaining T as a subset of an MST. ~~New edge connects isolated node to a connected component, maintaining T as a connected component.~~ (Corollary 23.2)
- Termination: All vertices have been added to create a single connected component that is a subset of an MST at termination. Thus, T is an MST at termination. (resolves uniqueness issue)

Implementation: Use MIN-PRIORITY Queue data structure

Recall (from 6.006):

INSERT (S, x) – Inserts element x into set of elements S
MINIMUM (S) – Returns element of S with smallest key
EXTRACT-MIN (S) – Removes & returns element w/smallest key
DECREASE-KEY (S, x, k) – Decrease value of element x's key to the new value k

- Alg connects vertices individually to growing connected MST subtree
- Priority queue tracks vertices not yet connected and their lightest weight to growing component
- Lightest edge joining an isolated vertex to growing MST component identified with EXTRACT-MIN

```
T = ∅                                   initially each vertex is infinitely far from growing MST
for each u ∈ G.V : u.key = ∞ & u.π = NIL   & has no light edge partner in growing MST
r.key = 0                               ← start with arbitrary vertex r
Q = G.V                                 ← put all vertices into queue
while Q ≠ ∅                                                          } loop |V| times
    u = EXTRACT-MIN(Q)                  ← pull lightest-edged vertex from Q   } T_extract-min
    T = T ∪ {(u, u.π)} if u.π ≠ NIL     ← add edge to tree
    for each v ∈ G.Adj[u]               } O(E) times total
        if v ∈ Q and w(u,v) < v.key     } Update "weight priority" of vertices adjacent to u to give least weight edge between v and growing MST
            v.π = u
            v.key = w(u,v)              ← DECREASE-KEY   } T_decrease-key
```

initialize: $O(V)$

$$\text{Time} = O(V) \cdot T_{\text{EXTRACT-MIN}} + O(E) \cdot T_{\text{DECREASE-KEY}}$$

Min-Priority Queue	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
Array	$O(V)$	$O(1)$	$O(V^2)$
Binary Heap	$O(\log V)$	$O(\log V)$	$O(E \log V)$
Fibonacci Heap	$O(\log V)$	$O(1)$ amortized	$O(E + V \log V)$

Best MST Algorithm to date:

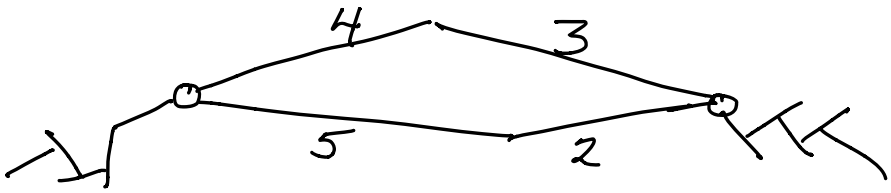
[Karger, Klein, & Tarjan, 1993]

$O(V+E)$ expected time (randomized)

Appendix:

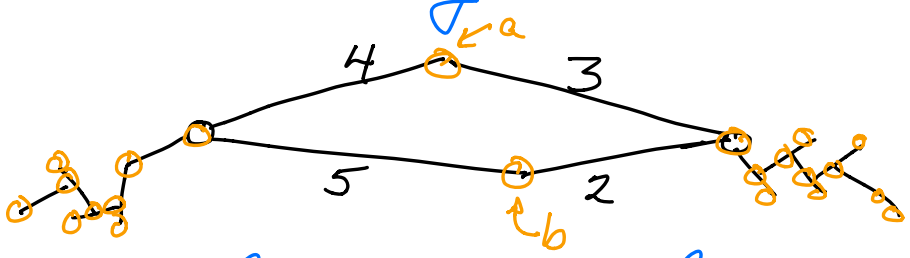
A student once asked the following question in class:

Although a connected graph with unique edge weights should have only one MST, isn't the following a counterexample,



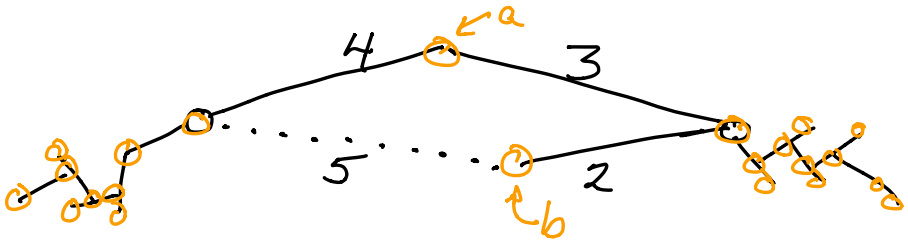
where one could accept the upper or lower path to connect the left & right portions of the graph.

This is a great question (especially as asked without a drawing). Upon drawing it out and noting all vertices (○)



we see that neither the upper or lower path is an MST; one misses vertex a and the other b.

The correct MST must include the following midsection, which is unique.



(with the dotted edge excluded).

Thanks for the thoughtful question.