COMPSCI 311: Introduction to Algorithms

Lecture 8: Minimum Spanning Trees

Marius Minea

University of Massachusetts Amherst

slides credit: Dan Sheldon

Spanning Trees and Cuts

- **Given**: an undirected graph $G = (V, E)$ with edge costs (weights) $c_e > 0$ (distinct for now).
- **Find**: subset of edges $T \subseteq E$ such that (V, T) is connected and the total cost of edges in T is as small as possible. Call $T \subseteq E$ a **spanning tree** if (V, T) is a tree (*connected*, no cycles).
- **Claim**: in a minimum-cost solution, T is a spanning tree.
- This is the **minimum spanning tree (MST) problem**.
- **Definition**: A cut in G is a partition of the nodes into two nonempty subsets $(S, V - S)$.
- **Definition**: Edge $e = (v, w)$ crosses cut $(S, V - S)$ if $v \in S$ and $w \in V - S$
  The cutset of a cut is the set of edges that cross the cut.

Minimum spanning trees: quiz 1

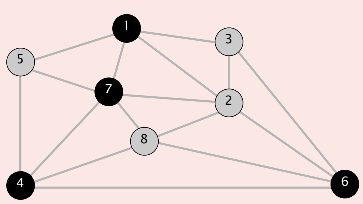
Consider the cut S = { 1, 4, 6, 7 }. Which edge is in the cutset of S?

A. S is not a cut (not connected)

B. 1–7

C. 5–7

D. 2–3



24

slide credit: Kevin Wayne / Pearson

Cut Property (IMPORTANT)

- **Theorem (cut property)**: Let $e = (v, w)$ be the minimum-weight edge crossing cut $(S, V - S)$ in G.
  Then e belongs to every minimum spanning tree of G.
- Terminology:
  - e is the cheapest or lightest edge across the cut
  - It is safe to add e to a MST
- We will see two different greedy algorithms based on the cut property: Kruskal's algorithm and Prim's algorithm.

Proof of Cut Property

- Let T be a spanning tree that doesn't include $e = (u, v)$.
  We'll construct a different spanning tree T' such that $w(T') < w(T)$ and hence T can't be the MST.
- Since T is a spanning tree, there's a $u \rightsquigarrow v$ path P in T.
  Since the path starts in S and ends up outside S, there must be an edge $e' = (u', v')$ on this path where $u' \in S, v' \notin S$.
- Let $T' = T - \{e'\} + \{e\}$.
  This is still connected, since any path in T that needed e' can be routed via e instead, and it has no cycles, so it is a spanning tree.
- But since e was the lightest edge between S and $V \setminus S$,

  $$w(T') = w(T) - w(e') + w(e) \leq w(T) - w(e') + w(e') = w(T)$$

What's wrong with the following proof ?

- Let T be a spanning tree so the cheapest edge e is not in T
- T must contain an edge f that links S to $V - S$
- e is the cheapest such edge, so $T - \{f\} \cup \{e\}$ is a cheaper tree.

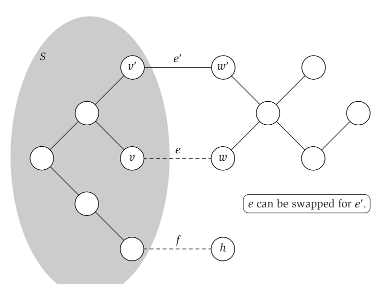


Figure: Kleinberg & Tardos

- $T - \{f\} \cup \{e\}$ may not be a tree!

Clicker Question 2: Properties of Spanning Trees

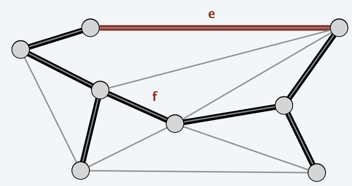
Which of the following is not true ?

- A: A spanning tree must intersect any cutset
- B: Adding an edge to a spanning tree produces at least two cycles
- C: Swapping a tree edge with another edge from the same cutset may produce a cycle
- D: The union of two different spanning trees produces a cycle

Fundamental cycle

Fundamental cycle. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any non tree-edge $e \in E$: $T \cup \{ e \}$ contains a unique cycle, say C.
- For any edge $f \in C$: $T \cup \{ e \} - \{ f \}$ is a spanning tree.



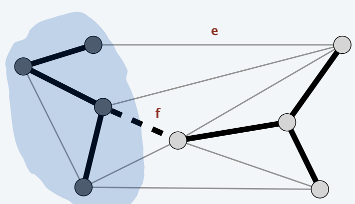
graph G = (V, E)
spanning tree H = (V, T)

Observation. If $c_e < c_f$, then (V, T) is not an MST.

Fundamental cutset

Fundamental cutset. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any tree edge $f \in T$: $T - \{ f \}$ contains two connected components. Let D denote corresponding cutset.
- For any edge $e \in D$: $T - \{ f \} \cup \{ e \}$ is a spanning tree.



graph G = (V, E)
spanning tree H = (V, T)

Observation. If $c_e < c_f$, then (V, T) is not an MST.

Kruskal's algorithm

- Armed with the cut property, how can we find a MST?
  - Starting no edges, which edge do we add first? How can we prove it is safe to add?
  - What edge do we add next? How do we prove it is safe?
  - Next?
  - Does this get stuck? If so, how to fix? Need to prove.
- **Kruskal's algorithm**: add edges in order of *increasing weight*, as long as they *don't cause a cycle*.

Kruskal's algorithm

Assume edges are numbered $e = 1, \ldots, m$
Sort edges by weight so $c_1 \leq c_2 \leq \ldots \leq c_m$
Initialize $T = \{\}$
for $e = 1$ to m **do**
  if adding e to T does not form a cycle **then**
    $T = T \cup \{e\}$
  end if
end for

Exercise: argue correctness (use cut property)

Kruskal's algorithm proof

- Let T be the partial spanning tree just before adding edge $e = (u, v)$ – the cheapest one not causing a cycle
  - Let S be the connected component of T that contains u
  - Then e crosses the cut $(S, V - S)$, otherwise it would create a cycle when added to T
  - No other edge crossing $(S, V - S)$ has been considered yet; it could have been added without creating a cycle, and would have connected S to $V - S$
  - Therefore, e is the cheapest edge across $(S, V - S)$, so it belongs to every MST
- So, every edge added belongs to the MST
- The final T is a spanning tree, since the algorithm won't stop until the graph is connected, and by design it creates no cycles
- Therefore, the output is a MST

Prim's Algorithm

- What if we want to grow a tree as a *single* connected component starting from some vertex s?
  - Which edge should we add first? How can we prove it is safe?
  - Which edge should we add next? How can we prove it is safe?
- **Prim's algorithm**: Let S be the connected component containing s. Add the cheapest edge from S to $V \setminus S$.

Prim's Algorithm

Initialize $T = \{\}$
Initialize $S = \{s\}$
while $|S| < n$ **do**
  Let $e = (u, v)$ be the minimum-cost edge from S to $V - S$
  $T = T \cup \{e\}$
  $S = S \cup \{v\}$
end while

Exercise: prove correctness

Clicker Question 3

Which of the following is always true?

A: Kruskal's algorithm creates disconnected trees and links them

B: Prim's and Kruskal's algorithm choose edges in different order

C: Prim's and Kruskal's algorithm choose the same set of edges

D: Only one of the algorithms is greedy

Exercise: Prove that the minimum spanning tree is unique (C).

Prim's algorithm proof

- Let T be the partial spanning tree just before adding edge e
  - Let S be the connected component containing s
  - By construction, e is the cheapest edge across the cut $(S, V - S)$
  - Therefore, e belongs to every MST
- So, every edge added belongs to the MST
- The algorithm creates no cycles and does not stop until the graph is connected, therefore, the final output is a spanning tree
- The final output is a minimum-spanning tree

Remove Distinctness Assumption?

- Hack: break ties in weights by perturbing each edge weight by a tiny unique amount.
- Implementation: break ties in an arbitrary but consistent way (e.g., lexicographic order)
- This is correct. There is a slightly more principled way that requires a stronger cut property.

Implementation of Prim's algorithm

Initialize $T = \{\}$
Initialize $S = \{s\}$
while T is not a spanning tree **do**
  Let $e = (u, v)$ be the minimum-cost edge from S to $V - S$
  $T = T \cup \{e\}$
  $S = S \cup \{s\}$
end while

What does this remind you of?

Prim Implementation

```
Set A = V                                          ▷ Unattached nodes
Set a(v) = ∞ for all nodes                         ▷ Attachment cost
Set a(s) = 0
Set edgeTo(s) = null                               ▷ Attachment edge
while A not empty do                               ▷ Nodes left to attach
    Extract node v ∈ A with smallest a(v) value
    Set T = T ∪ edgeTo(v)
    for all edges (v, w) where w ∈ A do
        if c(v, w) < a(w) then                     ▷ Cheaper edge to w?
            a(w) = c(v, w)
            edgeTo(w) = (v, w)
        end if
    end for
end while
```

Nearly identical to Dijkstra. Priority queue for $A \to O(m \log n)$

Kruskal Implementation?

```
Sort edges by weight so c_1 ≤ c_2 ≤ ... ≤ c_m
Initialize T = {}
for e = 1 to m do
    if adding e = (u, v) to T does not form a cycle then
        T = T ∪ {e}
    end if
end for
```

Ideas?

BFS to check if u and v in same connected component: $O(mn)$.

(Each BFS is $O(n)$: why?)

Can we do better?

Kruskal Implementation: Union-Find

Idea: use clever data structure to maintain connected components of growing spanning tree. Should support:

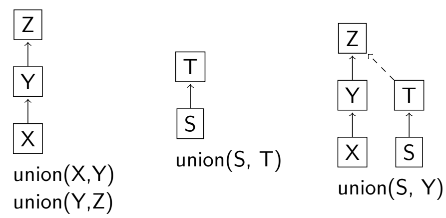
- find(v): return name of set containing v
- Union(A, B): merge two sets

```
for e = 1 to m do
    Let u and v be endpoints of e
    if find(u) != find(v) then                     ▷ Not in same component?
        T = T ∪ {e}
        Union(find(u), find(v))                    ▷ Merge components
    end if
end for
```

Goal: union $= O(1)$, find $= O(\log n) \Rightarrow O(m \log n)$ overall

Union-Find Data Structure

- Forest of trees with links to parent



- Find(X): follow pointers to root (equivalence class representative)
- Union(X, Y): links root of X to root of Y
- How to avoid trees degenerating to lists?
  Link smaller equivalence class (tree) to larger one

Union-Find Complexity

- Union is $O(1)$: update one pointer
- Find is $O(\log n)$:
  follow at most $\log_2(n)$ pointers to find representative of set
- Better: path compression (*Find* links all traversed nodes to root)
  essentially constant time