Minimum Spanning Trees

Kruskal's MST Algorithm

Algorithms: Design and Analysis, Part II

MST Review

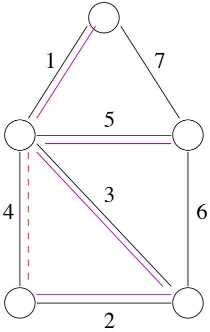
Input: Undirected graph $G = (V, E)$, edge costs c_e.

Output: Min-cost spanning tree (no cycles, connected).

Assumptions: G is connected, distinct edge costs.

Cut Property: If e is the cheapest edge crossing some cut (A, B), then e belongs to the MST.

Example



Kruskal's MST Algorithm

- Sort edges in order of increasing cost
[Rename edges $1, 2, \ldots, m$ so that $c_1 < c_2 < \ldots < c_m$]

- $T = \emptyset$

- For $i = 1$ to m

    - If $T \cup \{i\}$ has no cycles

    - Add i to T

- Return T

Minimum Spanning Trees

Correctness of Kruskal's Algorithm

Algorithms: Design and Analysis, Part II

Correctness of Kruskal (Part I)

Theorem: Kruskal's algorithm is correct.

Proof: Let $T^* =$ output of Kruskal's algorithm on input graph G.

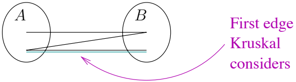
(1) Clearly T^* has no cycles.

(2) T^* is connected. Why?

(2a) By Empty Cut Lemma, only need to show that T^* crosses every cut.

(2b) Fix a cut (A, B). Since G connected at least one of its edges crosses (A, B).

Key point: Kruskal will include first edge crossing (A, B) that it sees [by Lonely Cut Corollary, cannot create a cycle]



Correctness of Kruskal (Part II)

(3) Every edge of T^* satisfied by the Cut Property. (Implies T^* is the MST)

Reason for (3): Consider iteration where edge (u, v) added to current set T. Since $T \cup \{(u, v)\}$ has no cycle, T has no $u - v$ path.

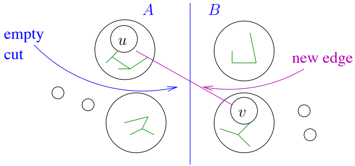
$\Rightarrow$ $\exists$ empty cut (A, B) separating u and v. (As in proof of Empty Cut Lemma)

$\Rightarrow$ By (2b), no edges crossing (A, B) were previously considered by Kruskal's algorithm.

$\Rightarrow$ (u, v) is the first (+ hence the cheapest!) edge crossing (A, B).

$\Rightarrow$ (u, v) justified by the Cut Property. QED

Current State:



Minimum Spanning Trees

Algorithms: Design and Analysis, Part II

Implementing Kruskal's Algorithm via Union-Find

Kruskal's MST Algorithm

- Sort edges in order of increasing cost. ($O(m \log n)$, recall $m = O(n^2)$ assuming nonparallel edges)
- $T = \emptyset$
  - For $i = 1$ to m ($O(m)$ iterations)
    - If $T \cup \{i\}$ has no cycles ($O(n)$ time to check for cycle [Use BFS or DFS in the graph (V, T) which contains $\leq n - 1$ edges])
      - Add i to T
- Return T

Running time of straightforward implementation: ($m = \#$ of edges, $n = \#$ of vertices) $O(m \log n) + O(mn) = O(mn)$

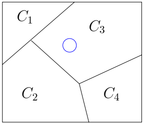
Plan: Data structure for $O(1)$-time cycle checks $\Rightarrow O(m \log n)$ time.

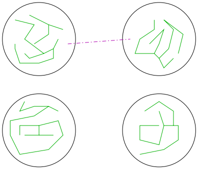
The Union-Find Data Structure

Raison d'être of union-find data structure: Maintain partition of a set of objects.

FIND(X): Return name of group that X belongs to.

UNION(C_i, C_j): Fuse groups C_i, C_j into a single one.





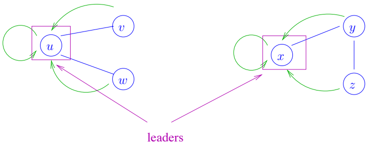
Why useful for Kruskal's algorithm: Objects = vertices

- Groups = Connected components w.r.t. chosen edges T.
- Adding new edge (u, v) to T $\iff$ Fusing connected components of u, v.

Union-Find Basics

Motivation: $O(1)$-time cycle checks in Kruskal's algorithm.

Idea #1: - Maintain one linked structure per connected component of (V, T).
- Each component has an arbitrary <u>leader</u> vertex.



Invariant: Each vertex points to the leader of its component ["name" of a component inherited from leader vertex]

Key point: Given edge (u, v), can check if u & v already in same component in $O(1)$ time. [if and only if leader pointers of u, v match, i.e., FIND(u)=FIND(v)] $\Rightarrow$ $O(1)$-time cycle checks!

Maintaining the Invariant

Note: When new edge (u, v) added to T, connected components of u & v merge.

Question: How many leader pointer updates are needed to restore the invariant in the worst case?

A) $\Theta(1)$

B) $\Theta(\log n)$

C) $\boxed{\Theta(n)}$ (e.g., when merging two components with $n/2$ vertices each)

D) $\Theta(m)$

Maintaining the Invariant (con'd)

Idea #2: When two components merge, have smaller one inherit the leader of the larger one. [Easy to maintain a size field in each component to facilitate this]

Question: How many leader pointer updates are now required to restore the invariant in the worst case?

A) $\Theta(1)$

B) $\Theta(\log n)$

C) $\boxed{\Theta(n)}$ (for same reason as before, i.e., when merging two components with $n/2$ vertices each)

D) $\Theta(m)$

Updating Leader Pointers

But: How many times does a single vertex v have its leader pointer updated over the course of Kruskal's algorithm?

A) $\Theta(1)$

B) $\Theta(\log n)$

C) $\Theta(n)$

D) $\Theta(m)$

Reason: Every time v's leader pointer gets updated, population of its component at least doubles $\Rightarrow$ Can only happen $\leq \log_2 n$ times.

Running Time of Fast Implementation

Scorecard:

$O(m \log n)$ time for sorting

$O(m)$ times for cycle checks [$O(1)$ per iteration]

$O(n \log n)$ time overall for leader pointer updates

$O(m \log n)$ total (Matching Prim's algorithm)

Minimum Spanning Trees

State-of-the-Art and Open Questions

Algorithms: Design and Analysis, Part II

State-of-the-Art MST Algorithms

Question: Can we do better than $O(m \log n)$? (Running time of Prim/Kruskal.)

Answer: Yes!

$O(m)$ randomized algorithm [Karger-Klein-Tarjan JACM 1995]

$O(m\, \alpha(n)\,)$ deterministic [Chazelle JACM 2000]

"Inverse Ackerman Function" : In particular, grows much slower than $\log^* n := \#$ of times you can apply log to n until result drops below 1 (inverse of "tower function" $2^{2^{2^{\cdots^{2}}}}$)

Open Questions

Weirdest of all: [Pettie/Ramachandran JACM 2002] Optimal deterministic MST algorithm, but precise asymptotic running time is unknown! [Between $\Theta(m)$ and $\Theta(m\alpha(n))$, but don't know where]

Open Questions:

- Simple randomized $O(m)$-time algorithm for MST [Sufficient: Do this just for the "MST verification" problem]
- Is there a deterministic $O(m)$-time algorithm?

Further reading: [Eisner 97]