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) $O(m \log n) + O(mn) = O(mn)$

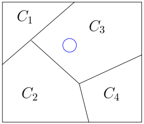
Plan: Data structure for $O(1)$-time cycle checks $\Rightarrow O(m \log n)$ time.

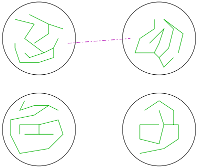
The Union-Find Data Structure

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.





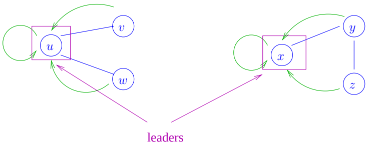
Why useful for Kruskal's algorithm: Objects = vertices

- 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 leader 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) $\Theta(n)$ (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)