Algorithms: Design and Analysis, Part II

Minimum Spanning Trees

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 no parallel edges
  [rename edges $1, 2, 3, \ldots, m$ so that $c_1 < c_2 < \cdots < c_m$]
- $T = \emptyset$
- 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)] contains $\leq n-1$ edges
    - add i to T
- return T

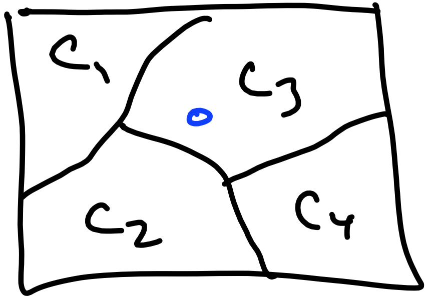
Running time of straightforward implementation: (m = # of edges, n = # of vertices)

$O(m \log n) + O(mn) = \boxed{O(mn)}$

Plan: data structure for $O(1)$-time cycle checks ⇒ $O(m \log n)$ time

The Union-Find Data Structure

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

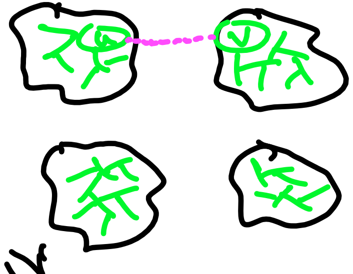


FIND(x): return name of group that x belongs to.

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

Why useful for Kruskal's algorithm: objects = vertices

- groups = connected components w.r.t. currently chosen edges T
- adding new edge (u,v) to T $\Leftrightarrow$ fusing connected components of u,v

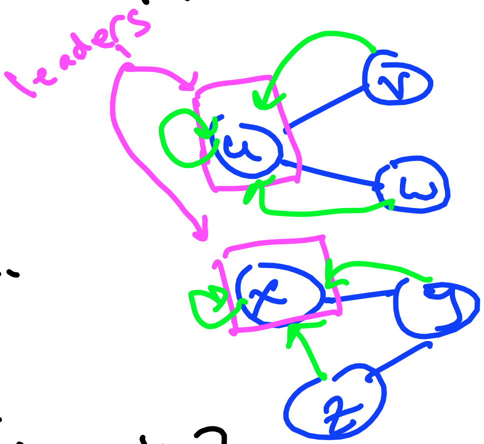


Union-Find Basics

Motivation: O(1)-time cycle checks in Kruskal's alg

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)

=> 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) $\Theta(n)$

(D) $\Theta(m)$

e.g., when merging two components with $n/2$ vertices each

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) $\Theta(n)$

(D) $\Theta(m)$

for same reason as before (might be merging two components with $n/2$ vertices each)

Updating Leader Pointers

But: how many times does a single vertex have its leader pointer update 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

$O(m)$ time 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)