Lec02-Kruskal

KT 4.5 / 4.6

Monday, August 28, 2023 10:51 PM

We just saw how the greedy Prim's algorithm for MST always works.
Other greedy algorithms also work.

Greedy MST rules

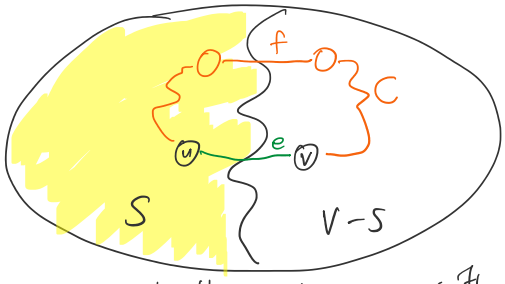
1. Prim's: start with any root, add frontier edge with smallest weight
2. Kruskal's: add edges in increasing weight, skipping any edge that would create a cycle.
3. Reverse-Delete: Start with all edges.
   Delete in decreasing order of weight,
   skipping any that would disconnect the graph.

Reminder: in last section, proved the smallest edge in any cut must be in all MSTs.

Thm (Cycle Property): Let C be a cycle in G. Let $e=(u,v)$ be the edge with maximum weight on C. Then e is NOT in any MST of G.

Intuition: can always replace e with another lower-cost edge in C.

proof. Suppose e is in a MST T.
Deleting e partitions T into two sets $S \ni u$ and $V-S \ni v$.



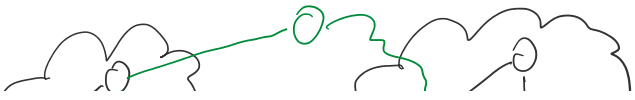
Cycle C must have another edge f that crosses over from S to $V-S$.

$\text{cost}(f) < \text{cost}(e)$, so we should replace e with f in T, contradicting the assumption that T is a MST.

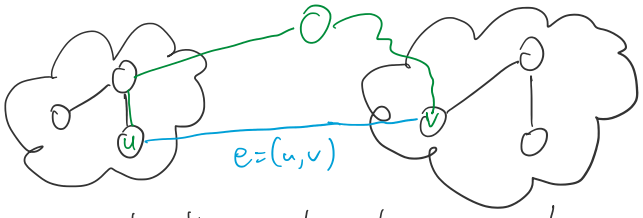
i.e. the largest edge on any cycle is never in any MST, which is what makes Kruskal and Reverse-Delete correct.

Thm. Reverse-Delete produces a MST.

proof



proof

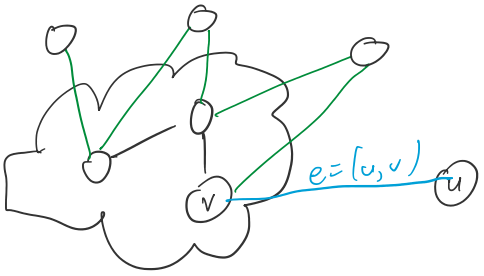


Let e be the next edge removed.
By construction, e's removal won't disconnect the graph,
so there must be another path b/t u and v.

But then we must be removing the largest edge in the cycle, which is never in a MST anyway. ▨

Thm Kruskal produces a MST.

proof. Consider the point when edge $e=(u,v)$ is added.

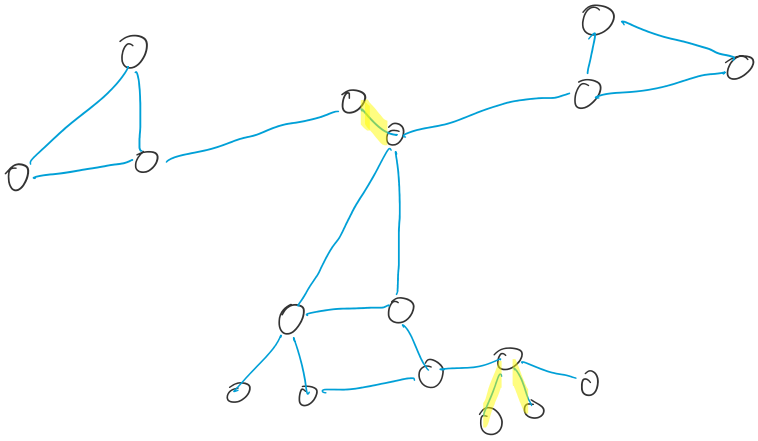


S = nodes in v's connected component before adding e.

u is in V−S (otherwise would be a cycle)

We get a tree because we never create cycles, and if there were a disconnected graph, we could add another edge without creating a cycle.

All rejected edges would be the maximum weight in a cycle, so they will never be in an MST.



How do we check in Kruskal if adding an edge (u, v)

How do we check in Kruskal if adding an edge (u, v) would create a cycle?

- Would create cycle if u, v in same connected component.
- We start with a component for each node
- Components merge when we add an edge
- Need a way to check if u, v are in same component and to merge two components into one.

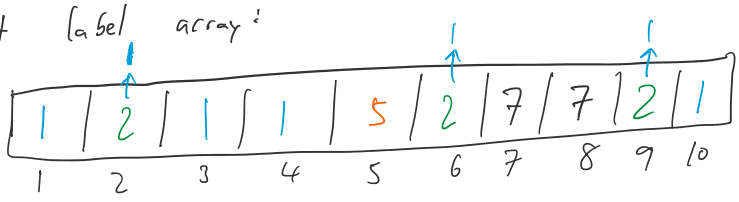
Union-Find Abstract Data Type (HT 46)

Supports following ops:

- MakeUnionFind(S) - create data structure containing $|S|$ sets, each containing one item from S.
- Find(i) - return label of set containing i.
- Union(a, b) - merge sets a and b into single set.

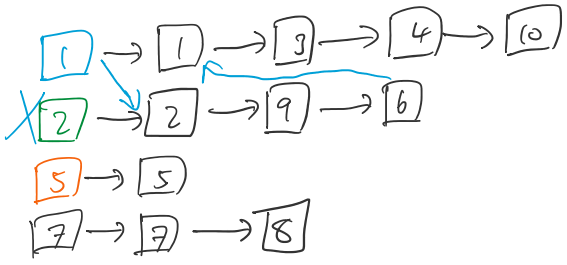
One way to build Union-Find: (array based)

Set label array:

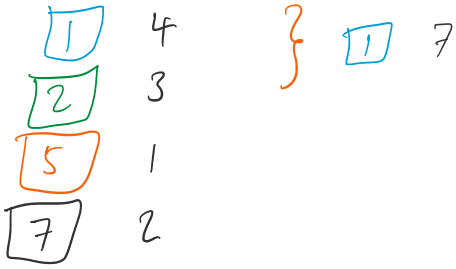


← says which set an item is in.

Items lists:



Sizes



MakeUnionFind(S): create data structures $O(S)$ time (proportional to S time)

Find (i): Return UF.sets[i]

$O(1)$ time (constant time)

Union (x,y): Use size array to decide which set is smaller.

WLOG, assume $|x| \leq |y|$.

Walk down elements i in set x, setting sets[i] = y.

Set size[y] = size[y] + size[x]

(Make y point to start of x list and end of y list point to y
↳ prepend smaller list to larger list

Let's compute runtime of array-based union-find

Thm Any seq of k union operations on a collection of n items takes time proportional to $k \log k$

proof. After k unions, at most 2k items have been involved in a union
(each union can touch at most 2 new items;
if it was a union of 2 singleton items;
often only 1 or 0 new items touched.

For any item v, set[v] changes at most $\log_2(2k)$ times ← total number of items possible in largest set

because each time set[v] changes |set[v]| at least doubles.
(since we update labels of smaller set)

At most 2k items updated at most $\log_2(2k)$ times each

$\Rightarrow 2k \log_2(2k)$ work.

Running time of Kruskal using arrays

Sorting edges $\approx m \log m$ for m edges

$m \leq n^2$, so $\log m \leq \log n^2 = 2 \log n$

$\approx m \log n$

At most 2m "find" operations $\approx 2m$ time

(to check if u, v are in same component)

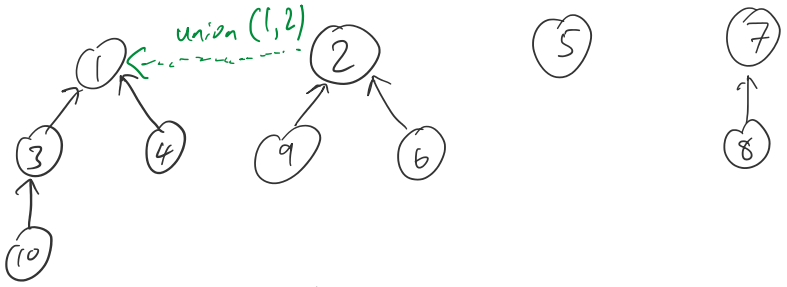
At most $2m$ "find" operations $\approx 2m$ time
(to check if u, v are in same component)

At most $n-1$ union operations $\approx n \log n$ time

$\Rightarrow$ total runtime $\approx m \log n + 2m + n \log n$

largest term since $m \geq n$ if graph is connected and not already a tree

Tree-based Union-Find



Make Union Find (S): create $|S|$ single-node trees. $\Theta(S)$ time

Find (i): Follow pointer from i to root of tree

Union (x,y): If $|x| < |y|$, make x point to y. $\Theta(1)$ time

Thm: Find (i) takes time $\Theta(\log n)$

proof. set $[i]$ is renamed at most $\log_2(n)$ times
because each renaming doubles the size.
The depth of a tree is the max number of renamings for an item.

Runtime of Kruskal using trees:

Sorting edges $\Theta(m \log n)$

$\leq 2m$ "find" ops: $\log n$ time each $\rightarrow 2m \log n$

$\leq n-1$ union ops: $\Theta(n)$ time

Total running time $\approx m \log n + 2m \log n + n = 3m \log n + n$
$= \Theta(m \log n)$