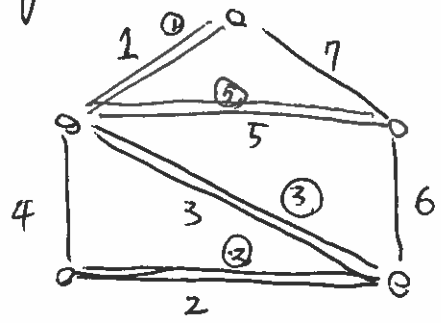
Kruskal's MST Algorithm (Union-find)

→ Solving MST problem in another way, the another potent greedy algorithm.

How Kruskal's algorithm work?

e.g.



① Start with the cheapest edge.
② Get the next cheapest edge.
③ Get the next cheapest edge.
④ Skip the next cheapest edge because it will cause the closed loop.
⑤ Get the next cheapest edge
⑥ Skip rest of the nodes and done.

Psudo code

- Sort edges in order of increasing cost.
  [Rename edges 1, 2, 3 ⋯, m so that $c_1 < c_2 < \cdots < c_m$]
- T = None
- for i = 1 to m
  - if $T_L[i]$ has no cycles
    - add i to T
- return T

What is the way to implement the Kruskal?

What will be the straight forward running time?

① Sorting part : $O(m \log n)$
② Main loop : $O(m)$
③ Check cycle : $O(n)$
Check if there is another path to that node.
[Of course DFS, BFS which is $O(n)$ Algorithm]

all in all : $O(m \log n) + O(m \cdot n) \Rightarrow O(m \cdot n)$

Can we do cycle check faster?

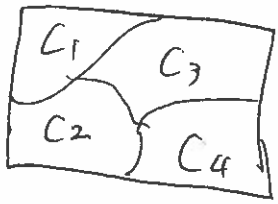
⇒ New data structure "Union-Find" will reduce the cycle check to $O(1)$-time.
then the sorting will be the bottle neck.

$O(m \log n) + O(n) \Rightarrow O(m \log n)$

What is Union-Find data structure?

Raisn d'etre of a union-find :
⇒ Maintain the partition of a set of object.

Operation:
- Find(x) : Return the name of group that x belongs to.
- Union(C_i, C_j) : Fuse groups C_i and C_j into a single one.



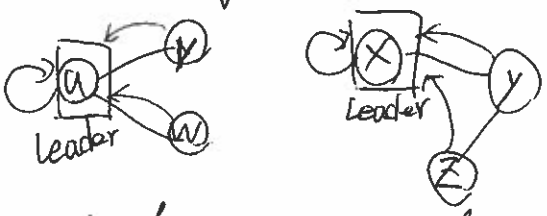
Apply it for Kruskal
- Object = nodes
- Groups = Connected components w.r.t. currently chosen edges T.
- Adding new edge(u,v) to T_L ⇒ Fuse connected components of u,v.

Union-Find Basics

Idea #1 - Maintain one linked structure per connected component of (V, T)
- Each component has an arbitrary leader node.

Invariant: Each node points to the leader of its component.

e.g.



["name" of a component inherited from leader node]

Key point: Given edge (u, v), can check if u and v already in same component in $O(1)$-time.
[Iff leader pointers of u, v match]
ie. Find(u) == Find(v)
⇒ Constant time $O(1)$

Maintain the invariant

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

Idea #2: When two components merge, have smaller one inherit the leader of the larger one.
(Make update minimum)
⇒ It's still takes $\Theta(n)$
⇒ But, the number of chances that node have ~~write~~ leader update through entire Kruskal algorithm is $\Theta(\log n)$

Reason: Everytime node's leader pointer gets updated, population of its component at least doubles.
⇒ Can happen $\leq \log_2 n$ times.

Running time Implementing "Union-find"

Sorting: $O(m \cdot \log n)$

Cyclecheck: $O(m)$ $(m \times O(1))$

Leader node update: $O(n \log n)$

⇒ all: $O(m \log n)$ total

⇒ Close to Prim's algorithm

Other way to solve MST

→ Too difficult to prove.

- $O(m)$ randomized algorithm exist [Karger-Klein 1995]
- $O(m \alpha(n))$ deterministic [Chazelle 2000]
- Optimal algorithm by Pettie 2002 's precised asymptotic running time is unknown. (Between $\theta(m)$ and $\theta(m \log^* n)$)

Why we learn MST?

- Problem itself is interesting.
- Good way to learn data structures.
- "Clustering" is useful technique.

How clustering is useful?

Informal Goal: Given n "points" [Webpages, images, etc.] classify them into "coherent groups".

Assumption: ① As input, given the similarity measure (a distance $d(p,q)$ between each point pair)
② Symmetric [ie. $d(p,q) = d(q,p)$]

Examples: Euclidean distance, genome similarity, etc.

Goal: Same cluster $\Leftrightarrow$ "nearby"

Max-spacing k-clusterings

Assume: We know k = # of clusters desired.
$\Rightarrow$ In practice, can experiment with a range]

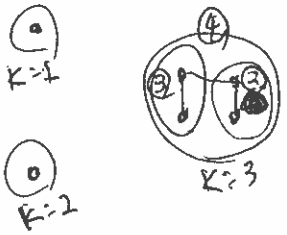
Call points p and q separated if they're assigned to different clusters.

Definition: The spacing of a k-clustering is min separated p,q's $d(p,q)$. (the bigger, the better)

Problem statement: Given a distance measuring d and k, compute the k-clustering with maximum spacing.

A Greedy algorithm to solve k-clustering

e.g.



~~① Con~~
① Set K=3 (Want 3 clusters)
② Connect the closest points. K=5
③ Connect the closest points. K=4
④ Connect the closest points. K=3
⑤ K=3, so stop the loop and done.

Psudo code

- Initially, each point in a seperate cluster.
- Repeat until only k clusters:
  - let p, q = closest pair of separated points (Determines the current spacing)
  - merge the cluster containing p and q into a single cluster.

⇒ Just like Kruskal's MST algorithm, but stopped early.

Vertices ⇒ Points
Edge cost ⇒ Distances

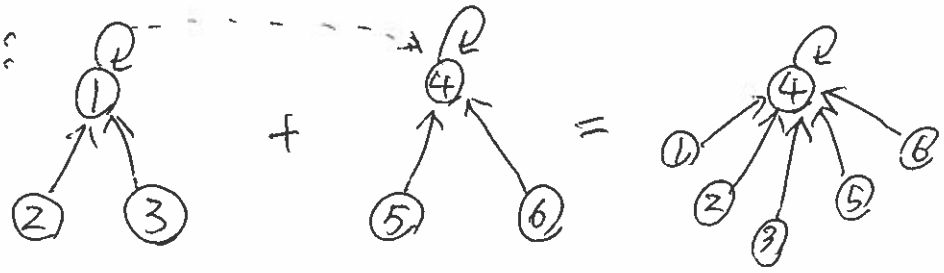
⇒ Called "Single link clustering"

Lazy Union (Other way to update the leader)

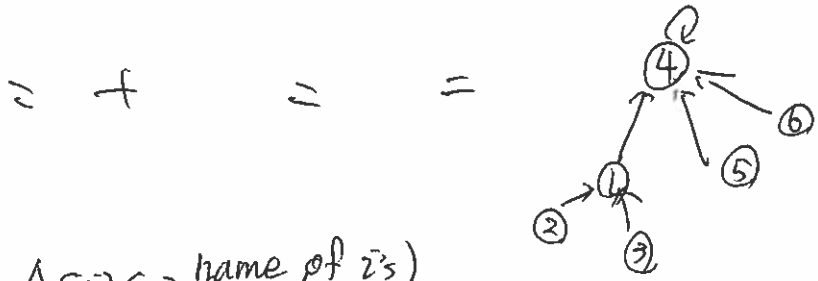
New idea: Update only one pointer each merge!

How?:

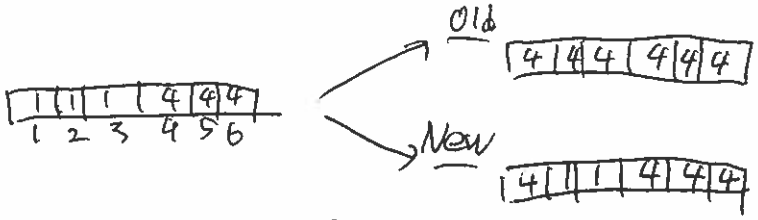
Old solution:



new solution:



In array rep. (Where A[i] ↔ name of i's parent)



Pros: Reduction of update time

Cons: Recovering leader is ~~so~~ more than O(1) time.