Slides by Kevin Miao (kevinmiao@berkeley.edu)

CS61BL – Tutoring Section 11

Disjoint Sets and MSTs

- Review
- Quiz Review (Optional)
- Worksheet (20 min)
- Questions (5 min)

Friendly reminder: Please participate in order to get marked as 'present'

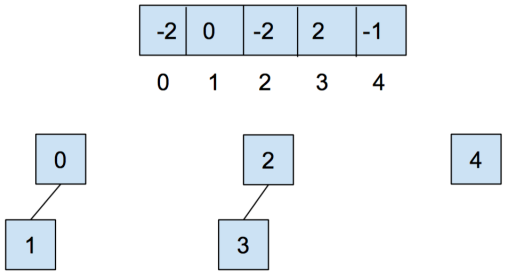
Resources:

- www.cs61bl.org/su20/resources

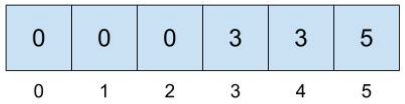
Disjoint Sets

- **Objective:** Quickly determine whether items belong to the same set or not.
- **Functions:**
  - ***Find****(E element):* What set does my *element* belong to?
  - ***Union****(E element1, E element2):* Combine the two sets that element1 and element 2 belong to.
  - **Naïve implementations**

QuickUnion



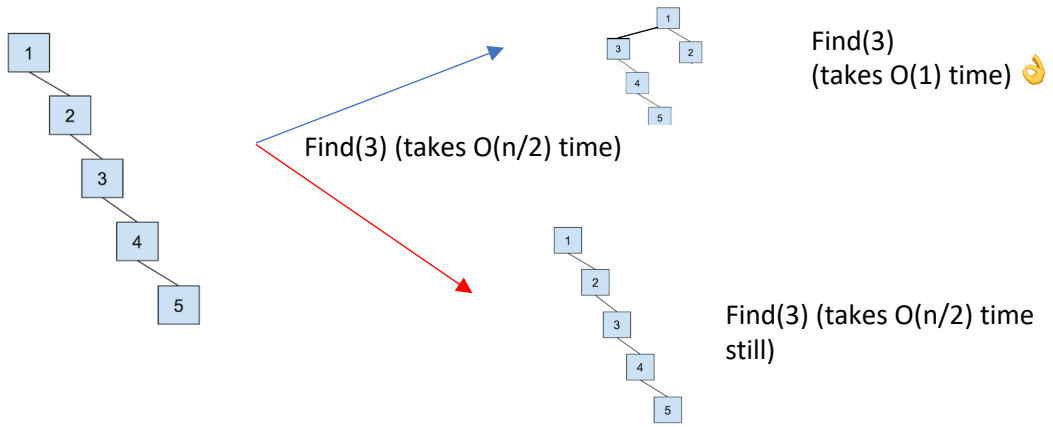
QuickFind



Disjoint Sets: Tweaks

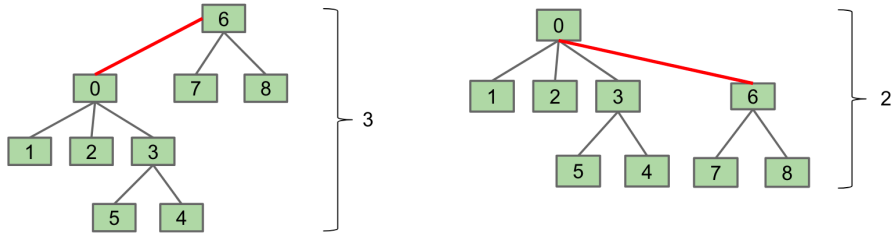
Path Compression

Idea: Doing extra work to make finding faster



Union by Height

Idea: We don't want to increase the height of the tree



Option 1

Option 2

Runtime: $O(Mlog^*(N))$ for **M** union/find calls on a set of **N** elements (implementing both Path Compression & Weighted Quick Union)

Minimum Spanning Tree

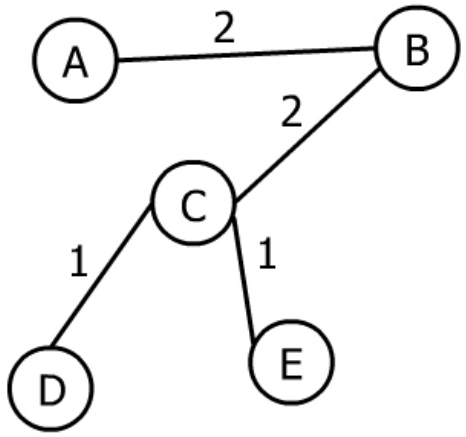
- **Minimum spanning tree**
  - **Minimum:** Lowest total edge weight
  - **Spanning:** All vertices are included
  - **Tree:** Connected acyclic graph

  - *Greedy Algorithms* where we will utilize **UnionFind**
    - *A greedy algorithm is an algorithm that makes the optimal choice at each step to create an optimal solution.*
    - **Prim's**
    - **Kruskal's**

Prim, Prim, Prim

Algorithm:

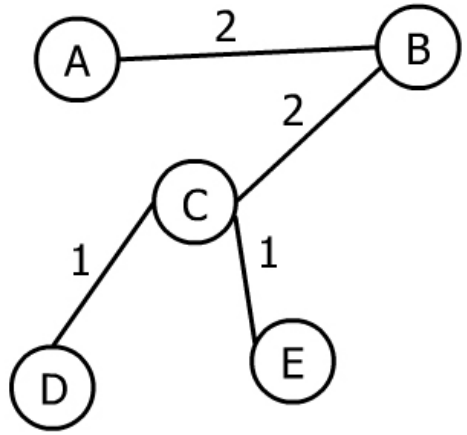
1. Start with empty Tree **T**
2. Choose an arbitrary starting vertex in **G**, add vertex to **T**
3. Repeatedly
    1. Add the minimum edge that is across the cut (one vertex in **G** but **not T** and one vertex in **T**)
4. Continue until T has V – 1 edges.



Kruskal's gone crazy

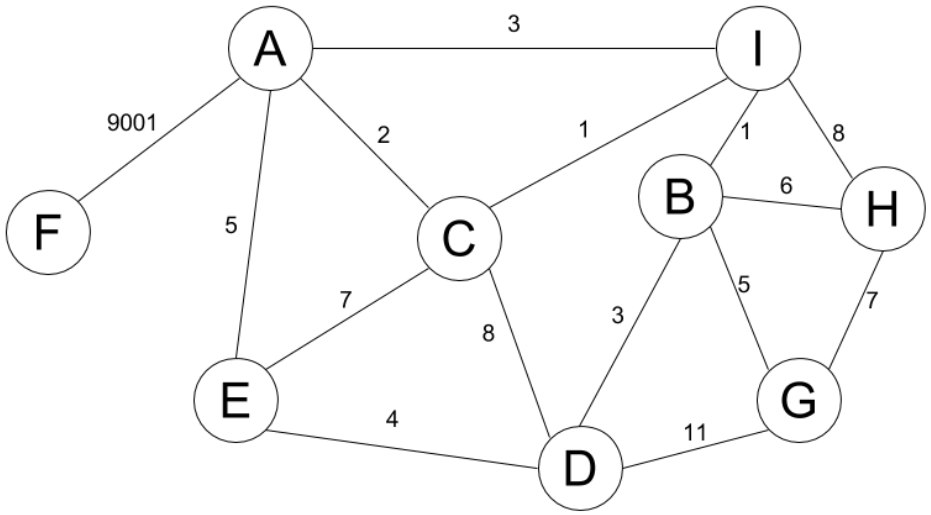
Algorithm

1. Create a new tree **T** with the same vertices as **G** with no edges
2. Make a **list** of all the **edges** in **G**
3. Sort the **edges** from **smallest** weight to **largest** weight.
4. Iterate through the edges in sorted order. For each edge (u, w), if u and w are not connected by a path in T, add (u, w) to T.



Quiz Q1: MST

Run Kruskal's starting at *F*



Quiz Q2: Potpourri

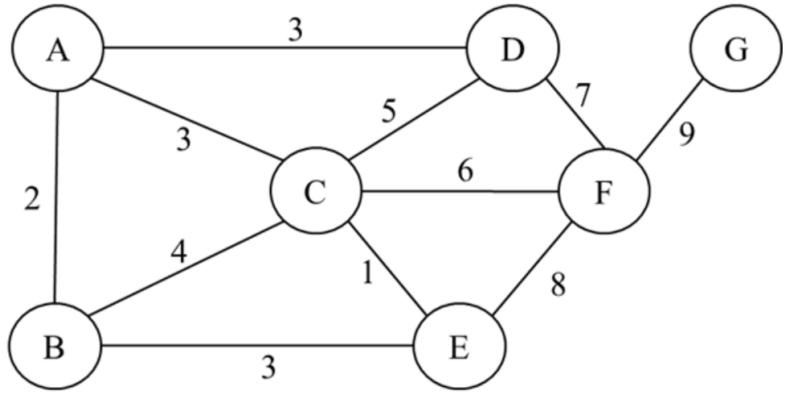
If we add a constant amount to the weight, would our original MST be still valid?

Does the MST also return the shortest path for any vertices u, v?

If there is one unique MST, that means all edge weights are unique.

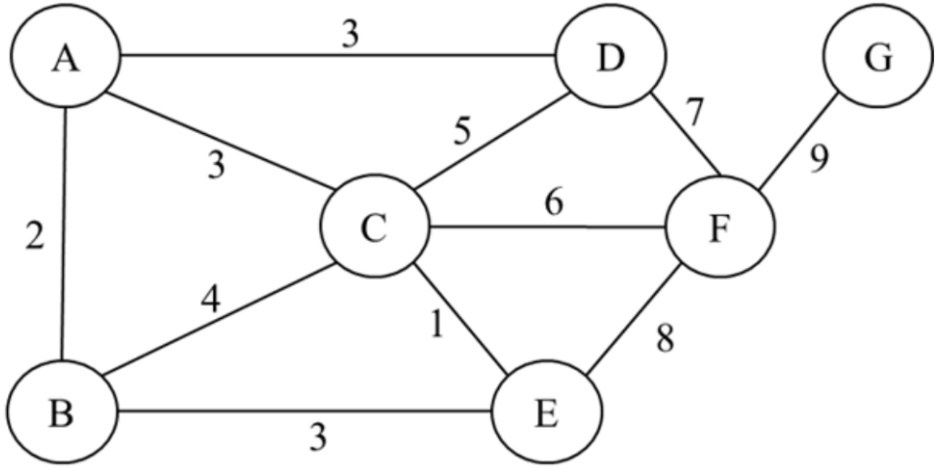
Kruskal's works on negative edge weights

1 Minimum Spanning Tree



(a) Given the graph above, run Prim's algorithm to determine the minimum spanning tree of this graph. For Prim's algorithm, assume we start at node A and fill in the following chart including the value cost(v) for all vertices v for that iteration as well as which node was popped off of the fringe for that iteration. (Note: Ties are broken in alphabetical order.)

v	init	Pop __	Pop __	Pop __	Pop __	Pop __	Pop __	Pop __
cost(a)	**0**							
cost(b)	∞							
cost(c)	∞							
cost(d)	∞							
cost(e)	∞							
cost(f)	∞							
cost(g)	∞							



(b) Run Kruskal's algorithm on the same graph.

(c) Does Kruskal's algorithm for finding the minimum spanning tree work on graphs with negative edge

2 Fiat Lux

After graduating from Berkeley with solid understanding of CS61B topics, Josh became a billionaire and wants to build power stations across Berkeley campus to help students survive from PG&E power outages. Josh want to minimize his cost, but due to the numerous power outages when he took CS61B, he did not learn anything about Prim's or Kruskal's algorithm and he is asking for your help! We must meet the following constrains to power the whole campus:

- There are V locations where Josh can build power stations, and it costs v_i dollars to build a power station at the i^{th} position.
- There are E positions we can build wires and it cost e_{ij} to build a wire between location i and j.
- All locations must have a power station itself or be connected to another position with power station.
- $e_{ij} << v_i, \forall i, j$

Modify the Prim's or Kruskal's algorithm taught in class that will minimize the cost Josh will spend while still fulfilling the constrains above.