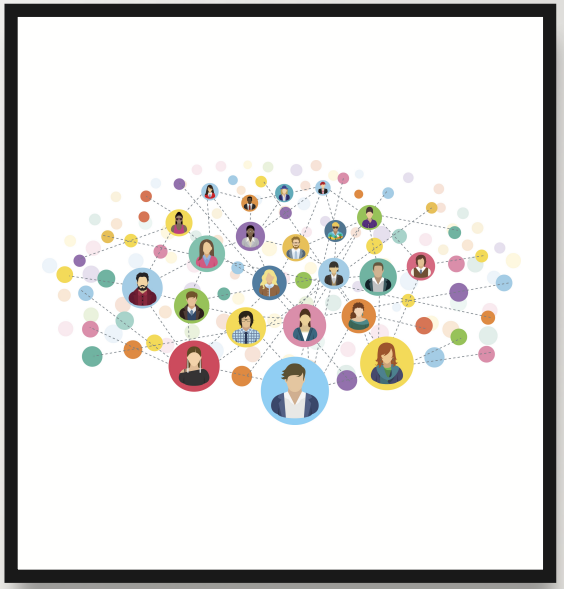
Union-Find

ADITYA ARJUN, KEVIN LI

Introduction

Social Networking, Revisited

- Want to build a social network app with users
- Functions:
  - Add users
  - Connect users
  - Determine if two users are connected directly, or indirectly



Graph Representation

- Treat people as vertices
  - Adding a person is just adding a vertex in the graph
- Treat connections as edges
  - Adding an edge is just adding an edge in the graph
- How to check connectedness?
  - Flood Fill (DFS, BFS)

Graph Representation Runtime

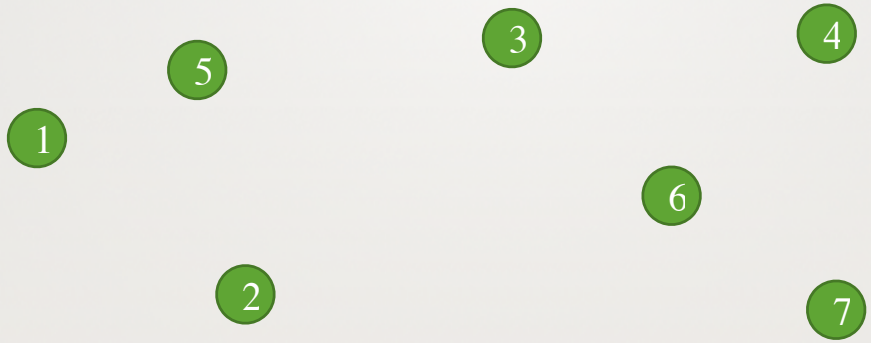
- Adding a person
  - O(1)
- Adding a connection
  - O(1)
- Checking if two people are connected
  - O(N)

Array Representation

- Give each person an ID
  - Initially, each person has a unique ID
- Once two people are merged, they will share the same ID
  - Anyone that is connected to either of these two people will also share that ID
- Let's make the ID of everyone in that group the smallest ID of the members
- How to check if two people are connected?
  - Check if their IDs are the same

Array Representation Visualization

1	2	3	4	5	6	7
1	2	3	4	5	6	7

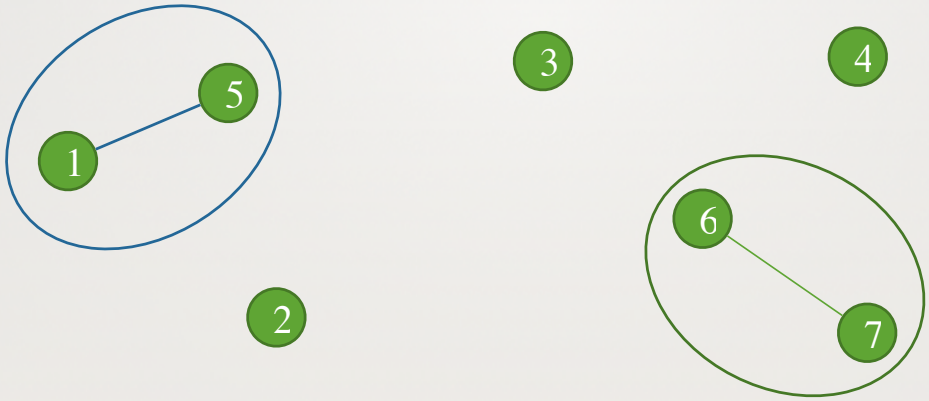


Array Representation Visualization

1	2	3	4	5	6	7
1	2	3	4	5	6	6

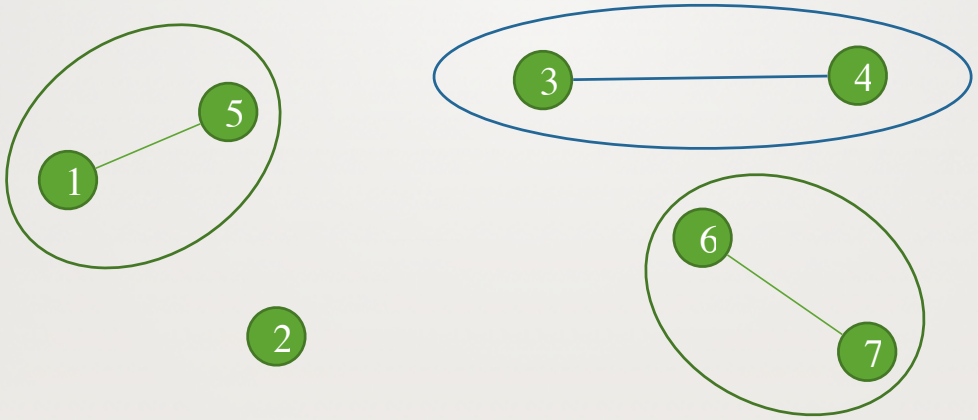
Array Representation Visualization

1	2	3	4	5	6	7
1	2	3	4	1	6	6



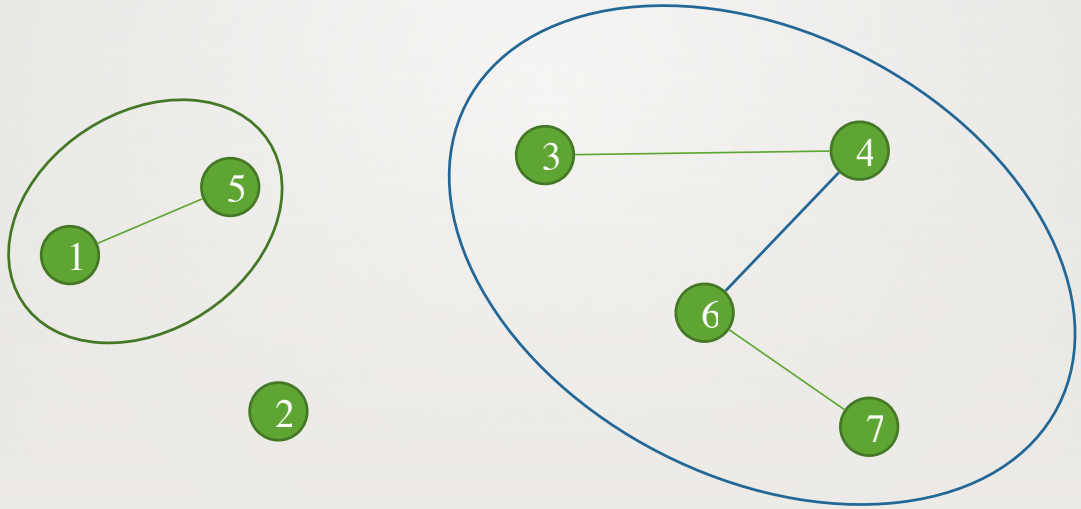
Array Representation Visualization

1	2	3	4	5	6	7
1	2	3	3	1	6	6



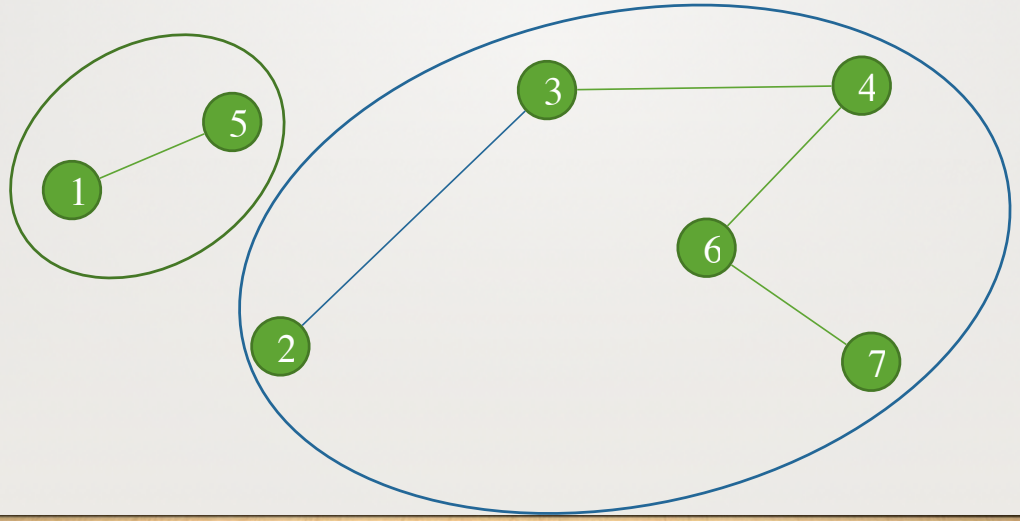
Array Representation Visualization

1	2	3	4	5	6	7
1	2	3	3	1	3	3



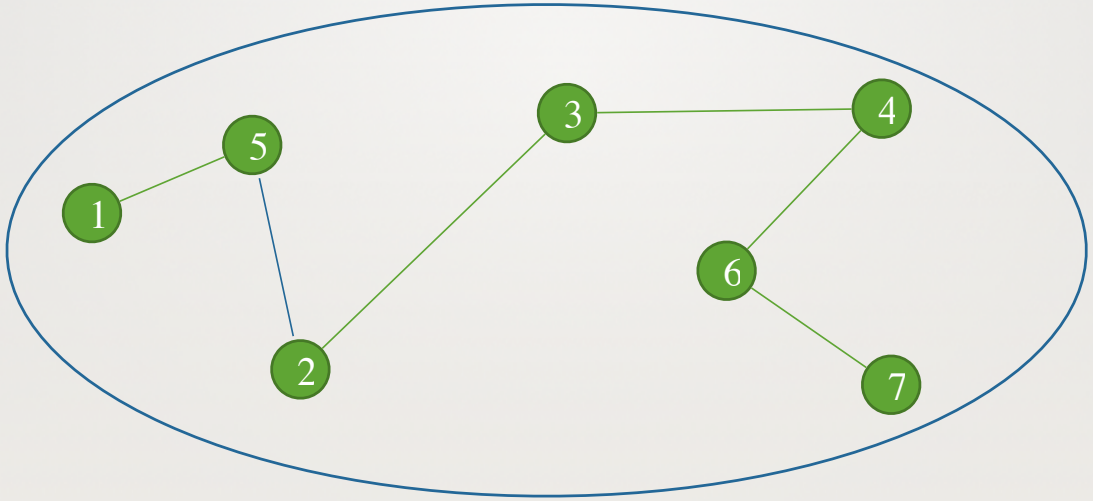
Array Representation Visualization

1	2	3	4	5	6	7
1	2	2	2	1	2	2



Array Representation Visualization

1	2	3	4	5	6	7
1	1	1	1	1	1	1



Array Representation Runtime

- Adding a person
  - O(1)
- Connecting two people (Union)
  - O(N)
- Checking if two people are connected (Find)
  - O(1)

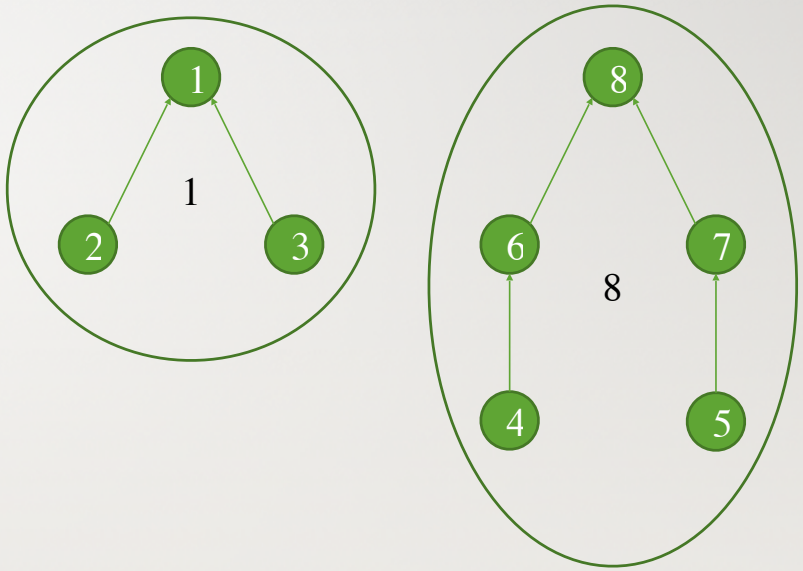
Union-Find

- Data Structure with two primary functions, Union and Find
- Union: Joins two elements together
- Find: Determine the ID of an element
- Two elements are connected if they have the same ID

Optimization

Array Representation Adjustments

- Want to adjust our data structure to allow for optimization
- Be a bit lazy with ID updating
- Turn each group into a tree, where the ID of the tree is the ID of the root



Array Representation Adjustments

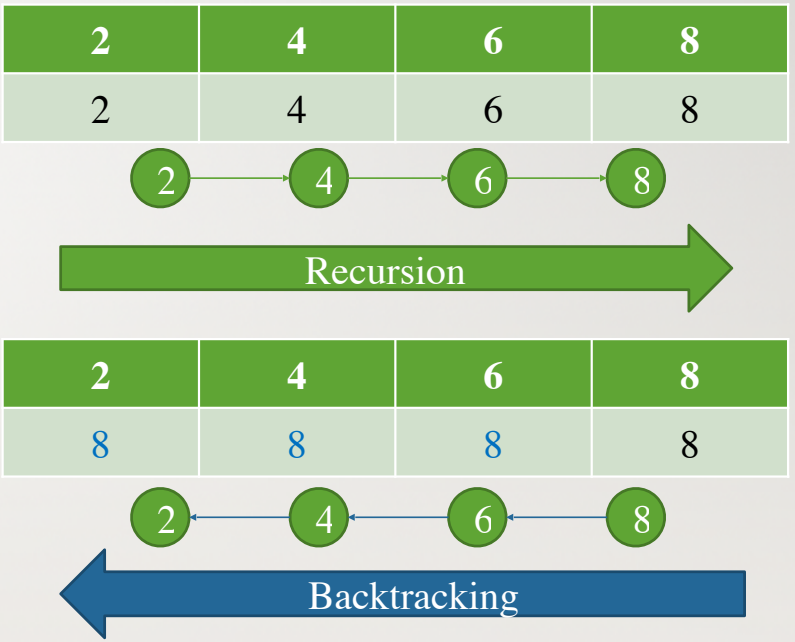
- union(A, B):
  - Find root(A) and root(B)
  - Set either root(A)'s parent to be root(B), or visa versa
- find(A):
  - Look for the ID of the root
  - Done recursively, basically keep calling find(parent(A)) until root is reached
- Note: root and find are the same function, since ID of the root is itself

Adjusted Runtimes

- Adding a person
  - O(1)
- Connecting two people (Union)
  - O(N)
- Checking if two people are connected (Find)
  - O(N)

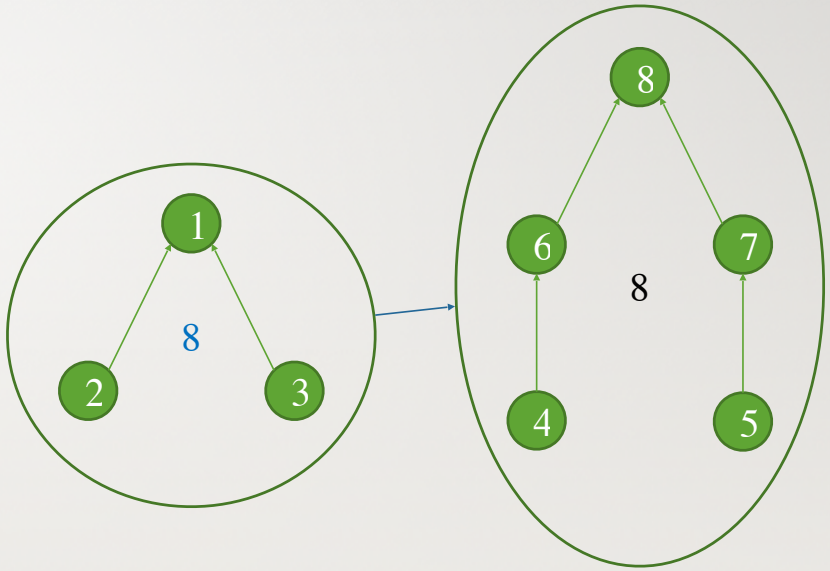
Optimization 1: Path Compression

- Modifying find(A)
- Key Idea: when we get the ID of the root, we can store it at every step along the way
- Now, the path never has to be traversed again (compressed)
- Runtime: O(log N)



Optimization 2: Union by Size

- Modifying union(A, B)
- Keep track of how many nodes are in each group
- Assume size(A) < size(B), then set root(A) to be root(B)'s child
- Runtime: O(log N)



Combined Overall Runtime

- We can combine Path Compression
- Analysis is quite complicated, but here are the runtimes:
  - Adding an element: O(1)
  - Union: O(α(N))
  - Find: O(α(N))
- This is the Inverse Ackermann Function
  - For our purposes, when N $\leq 10^{18}$, $\alpha(N) \leq 5$, so basically constant

Applications

Cycle Detection

- Give an undirected graph G(V, E), determine if a cycle exists using Union-Find
- Iterate through all edges, applying Union the two vertices it connects
- If the two vertices were previously connected, then there is a cycle

Graph Connectedness

- Given a Graph, determine if it is connected using Union-Find
- Iterate through all the edges again, applying Union on both endpoints
- Check that all IDs for vertices in the graph are the same

Minimum Spanning Tree

Fiber Network

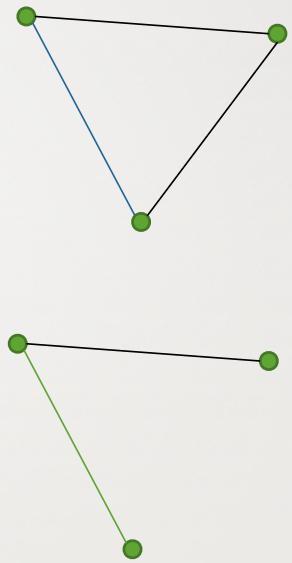
- Google is trying to build a fiber network for N houses. It has determined M possible interhouse connections, each with a cost w_i. The goal is to try to ensure that every house is connected to every other house, directly or indirectly.
- How would we go about solving this?

Greedy Solution

- Always take the shortest edge (Why?)
  - If shortest edge is not in graph, always optimal to swap
- We can keep taking the shortest edge
- Need to check that adding the connection helps
  - Do not want to waste money connecting two houses when they're already connected

Utilizing Union-Find

- Union-Find can help us determine if adding an edge connects two previously nonconnected vertices



Kruskal's Algorithm

- Given a graph G(V, E) of possible connections
  - Sort the edges in ascending order by **weight**
  - Repeatedly add the shortest edge that connects two unconnected vertices
  - Continue until graph is connected
- Overall Runtime
  - $O(E \log E) + O(E \cdot \alpha) = O(E \log V)$

Kruskal's Visualization

Live Coding