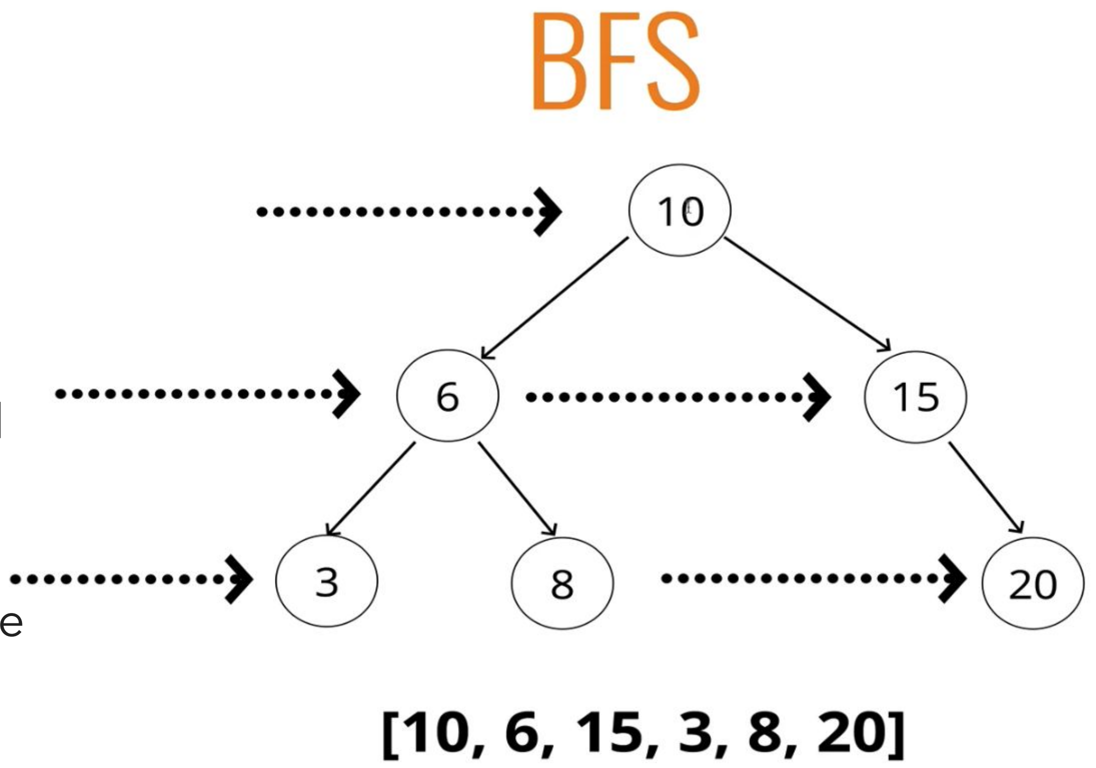
App Academy

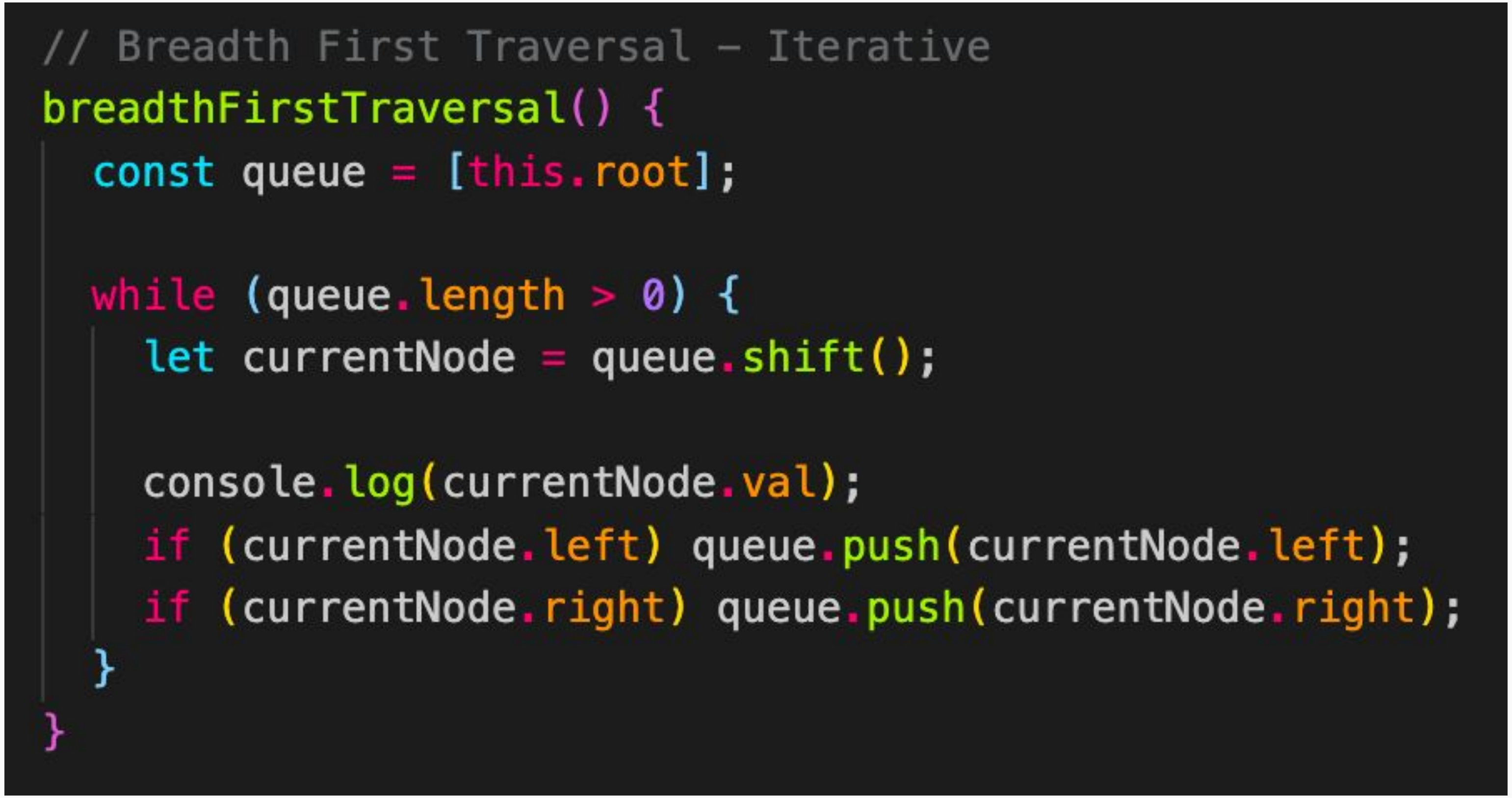
BST & Intro to Graphs

Breadth First Traversal (BFT)

- Visits each level before moving downward
- Use a Queue
- Can take up more memory in a fully flushed out, balanced tree than DFT
- Better for an unbalanced BST (close to a Linked List)



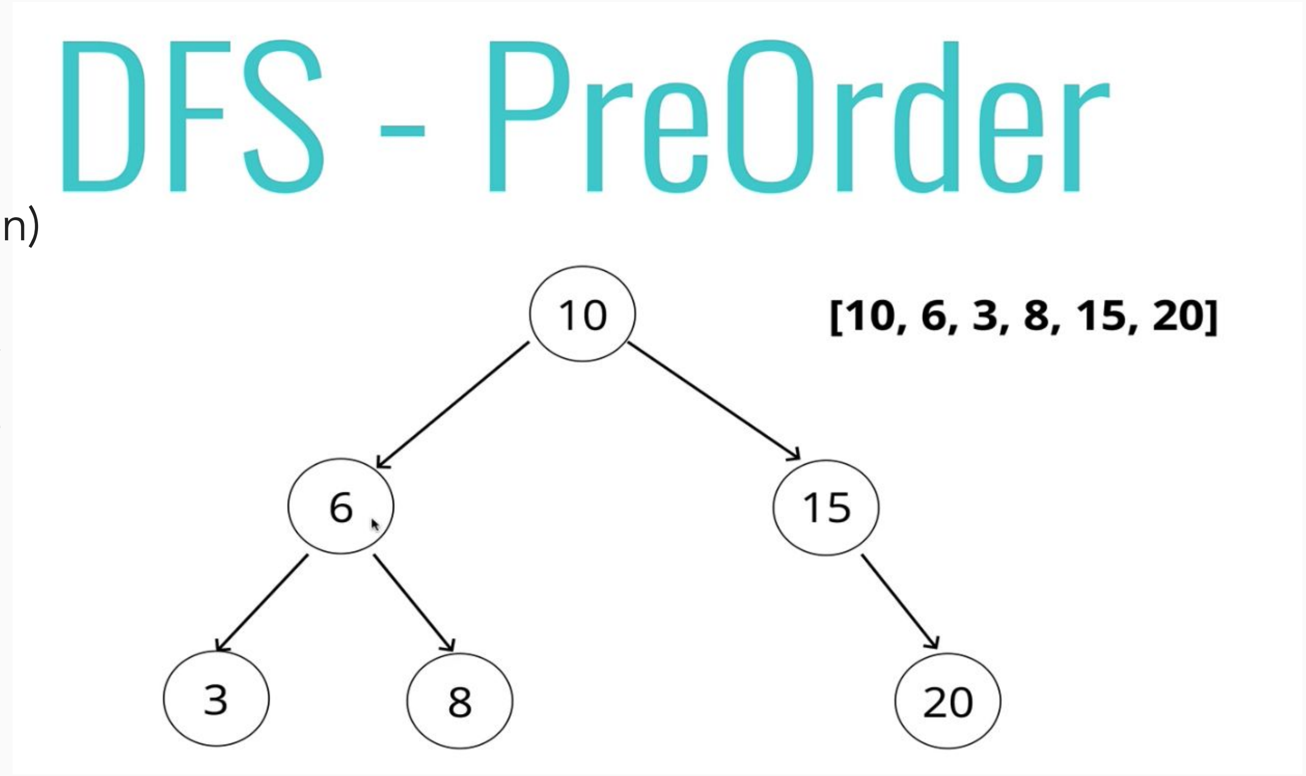
Breadth First Traversal (BFT) cont



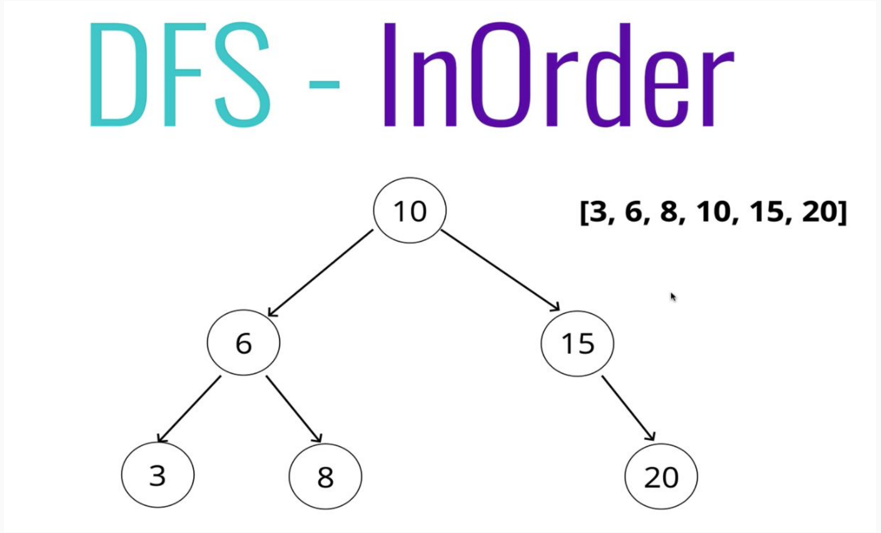
```javascript
// Breadth First Traversal - Iterative
breadthFirstTraversal() {
  const queue = [this.root];

  while (queue.length > 0) {
    let currentNode = queue.shift();

    console.log(currentNode.val);
    if (currentNode.left) queue.push(currentNode.left);
    if (currentNode.right) queue.push(currentNode.right);
  }
}
```

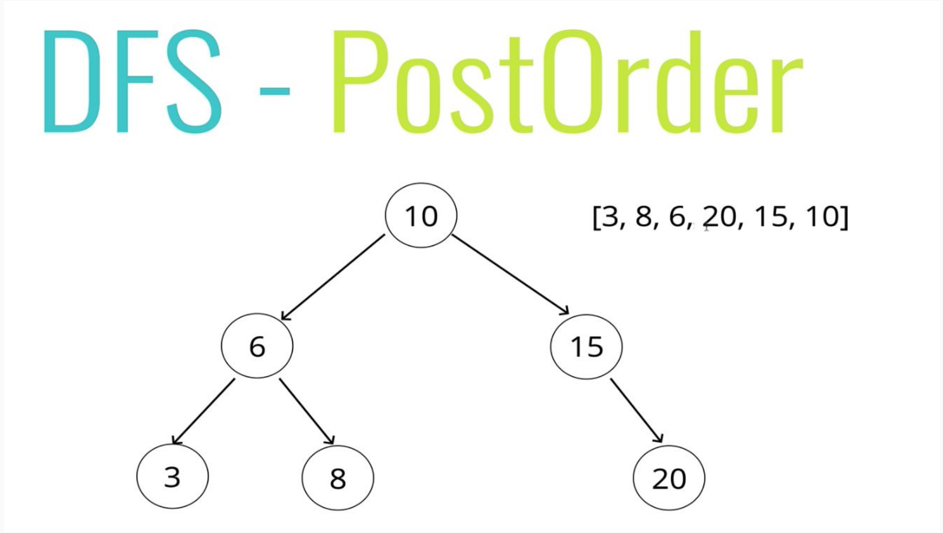
Depth First Traversal (DFT)

- Visits nodes all the way to the bottom left, then right
- Use a Stack (recursion)
- 3 Ways, Pre-Order, In-Order, Post Order
- You are traveling the tree the same way, difference is what order you print/push or consider a node "visited"



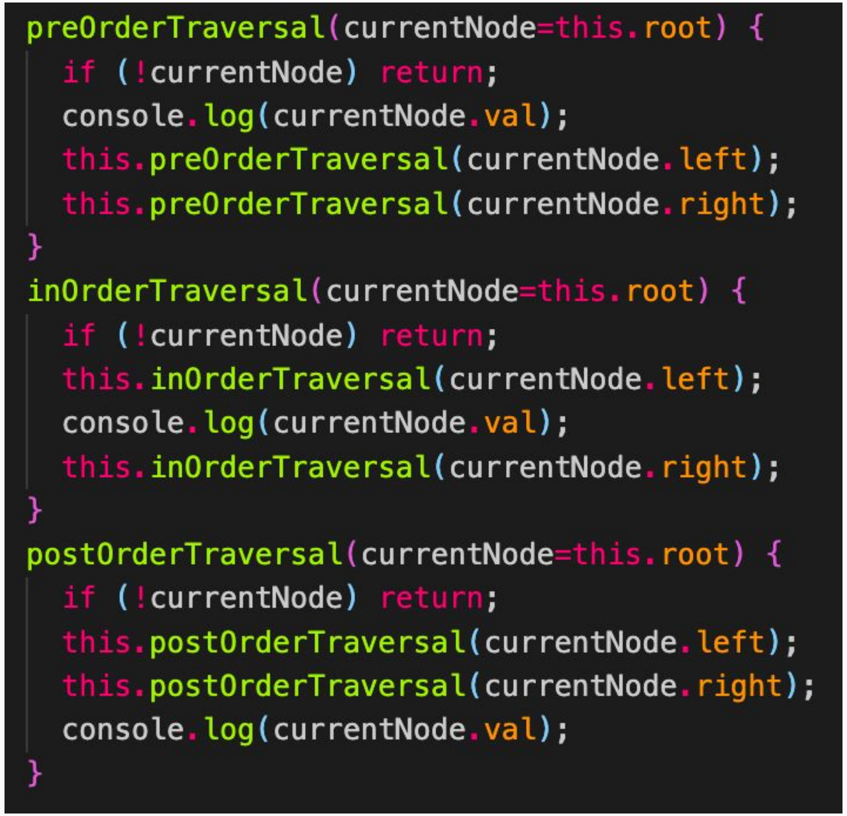
Depth First Traversal (DFT) cont



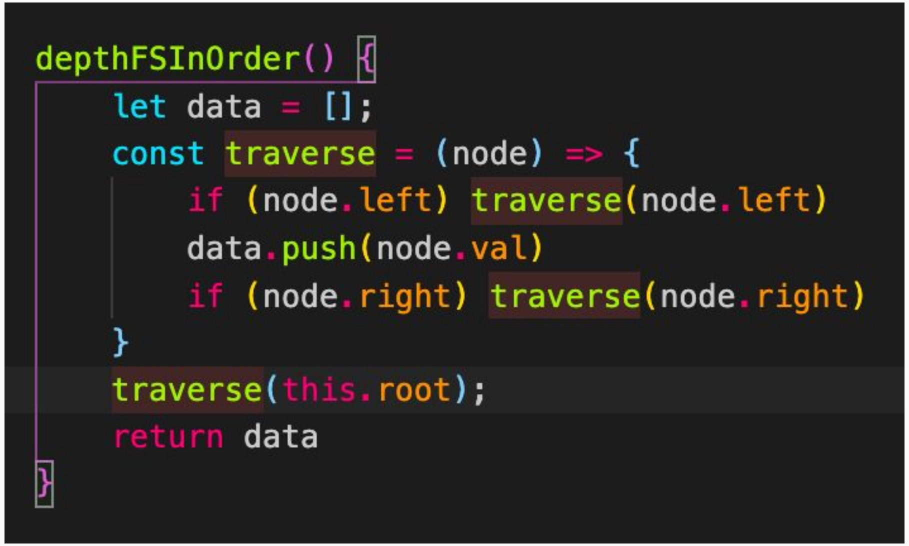


- DFS InOrder - Array is actually in order, thats nice!
- DFS PreOrder - If you wanted to construct a tree from the array, you could use the output array
- DFS PostOrder - fun to talk about at parties? (used for deleting a tree)

Depth First Traversal (DFT) cont



```
preOrderTraversal(currentNode=this.root) {
  if (!currentNode) return;
  console.log(currentNode.val);
  this.preOrderTraversal(currentNode.left);
  this.preOrderTraversal(currentNode.right);
}
inOrderTraversal(currentNode=this.root) {
  if (!currentNode) return;
  this.inOrderTraversal(currentNode.left);
  console.log(currentNode.val);
  this.inOrderTraversal(currentNode.right);
}
postOrderTraversal(currentNode=this.root) {
  if (!currentNode) return;
  this.postOrderTraversal(currentNode.left);
  this.postOrderTraversal(currentNode.right);
  console.log(currentNode.val);
}
```

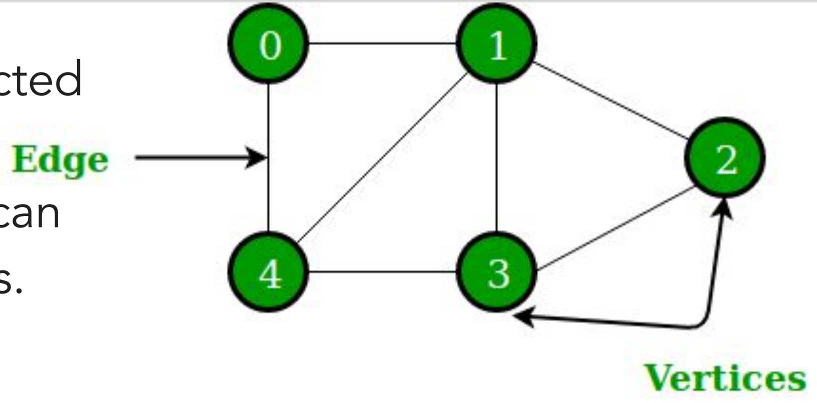


```
depthFSInOrder() {
    let data = [];
    const traverse = (node) => {
        if (node.left) traverse(node.left)
        data.push(node.val)
        if (node.right) traverse(node.right)
    }
    traverse(this.root);
    return data
}
```

Let’s Talk about Graphs!

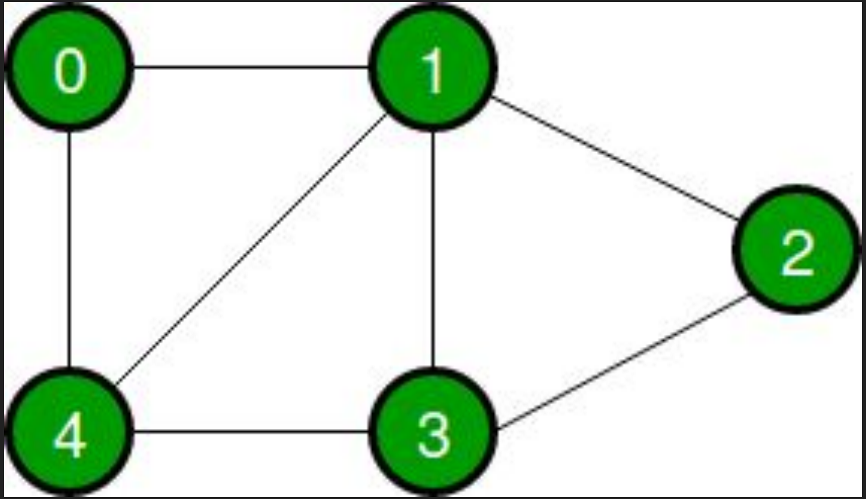
Graphs!

- A graph is a collection of nodes connected by edges. Unlike trees, graphs do not necessarily start from a root node and can have any number of neighboring edges.
- Can be weighted or unweighted
- Can be directed or undirected
- Can be cyclic or acyclic
- Often represented as an Adjacency List - an object with a key for each node, and arrays of the nodes connected to them or a Adjacency Matrix - a 2d array that represents edges.

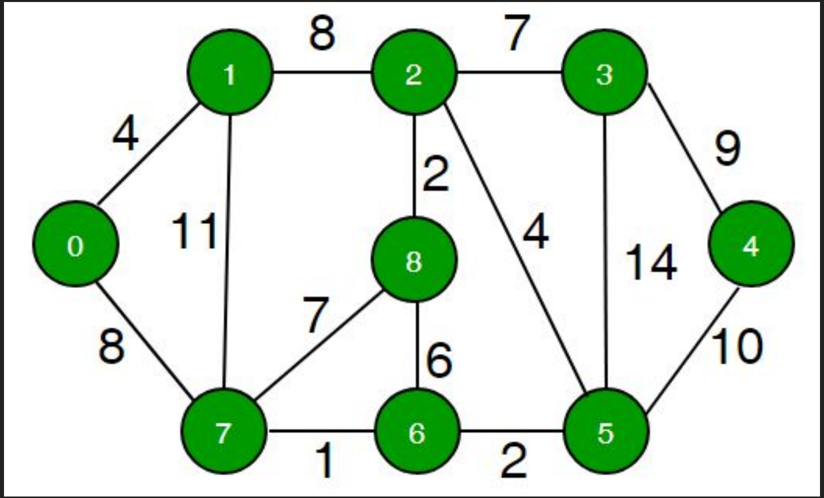


Unweighted vs Weighted Graphs

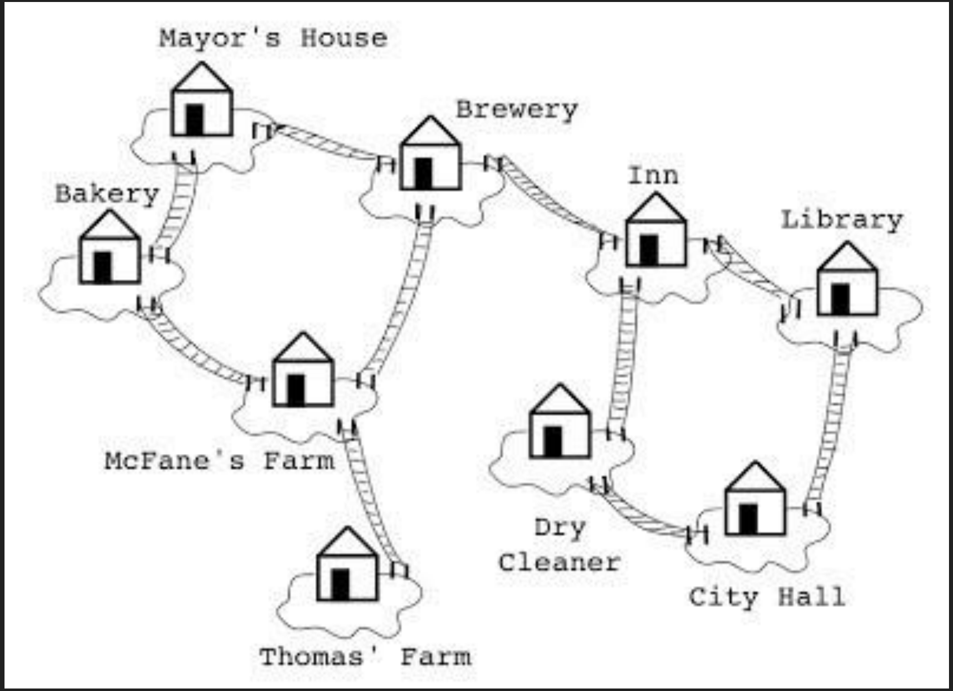
UNWEIGHTED



WEIGHTED



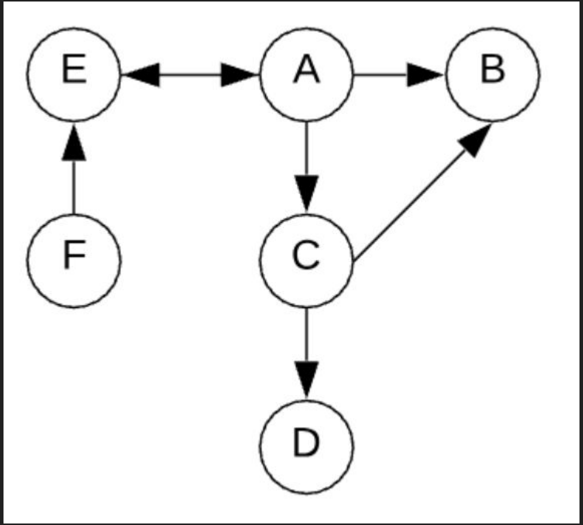
Example of an Unweighted Graph



Island Town Map

Shows you how to get from one location to the other. Distances are not being tracked, so you can't determine the fastest way from one point to another, but you know how they are connected

Example of a Unweighted Graph - Adjacency Matrix

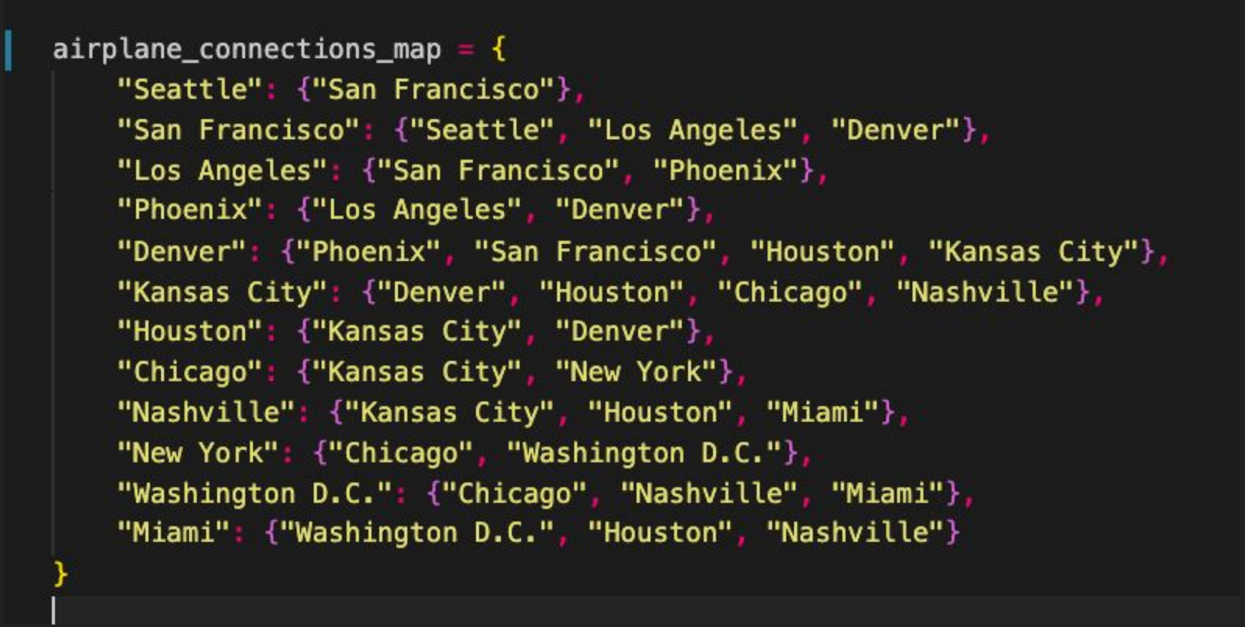


If this was weighted we would use weight values instead of boolean values in the below matrix

```
let matrix = [
  /*          A    B    C    D    E    F   */
  /*A*/    [  1,   1,   1,   0,   1,   0   ],
  /*B*/    [  0,   1,   0,   0,   0,   0   ],
  /*C*/    [  0,   1,   1,   1,   0,   0   ],
  /*D*/    [  0,   0,   0,   1,   0,   0   ],
  /*E*/    [  1,   0,   0,   0,   1,   0   ],
  /*F*/    [  0,   0,   0,   0,   1,   1   ]
  ];
```

Example of a Unweighted Graph - Sneak Peak!

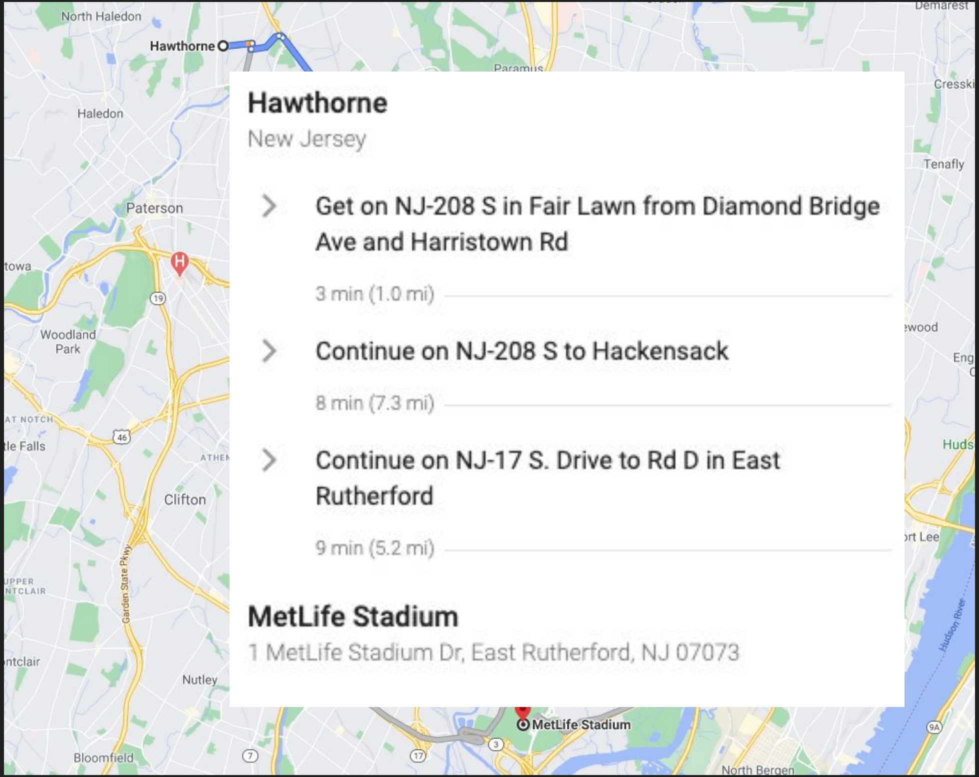
Adjacency List from Python Project



```
airplane_connections_map = {
    "Seattle": {"San Francisco"},
    "San Francisco": {"Seattle", "Los Angeles", "Denver"},
    "Los Angeles": {"San Francisco", "Phoenix"},
    "Phoenix": {"Los Angeles", "Denver"},
    "Denver": {"Phoenix", "San Francisco", "Houston", "Kansas City"},
    "Kansas City": {"Denver", "Houston", "Chicago", "Nashville"},
    "Houston": {"Kansas City", "Denver"},
    "Chicago": {"Kansas City", "New York"},
    "Nashville": {"Kansas City", "Houston", "Miami"},
    "New York": {"Chicago", "Washington D.C."},
    "Washington D.C.": {"Chicago", "Nashville", "Miami"},
    "Miami": {"Washington D.C.", "Houston", "Nashville"}
}
```

Not weighted, so like previous example it shows you how to get from one location to the other. Without distances we don't know which path with be shortest!

Example of an Weighted Graph



Hawthorne
New Jersey

> Get on NJ-208 S in Fair Lawn from Diamond Bridge Ave and Harristown Rd

3 min (1.0 mi)

> Continue on NJ-208 S to Hackensack

8 min (7.3 mi)

> Continue on NJ-17 S. Drive to Rd D in East Rutherford

9 min (5.2 mi)

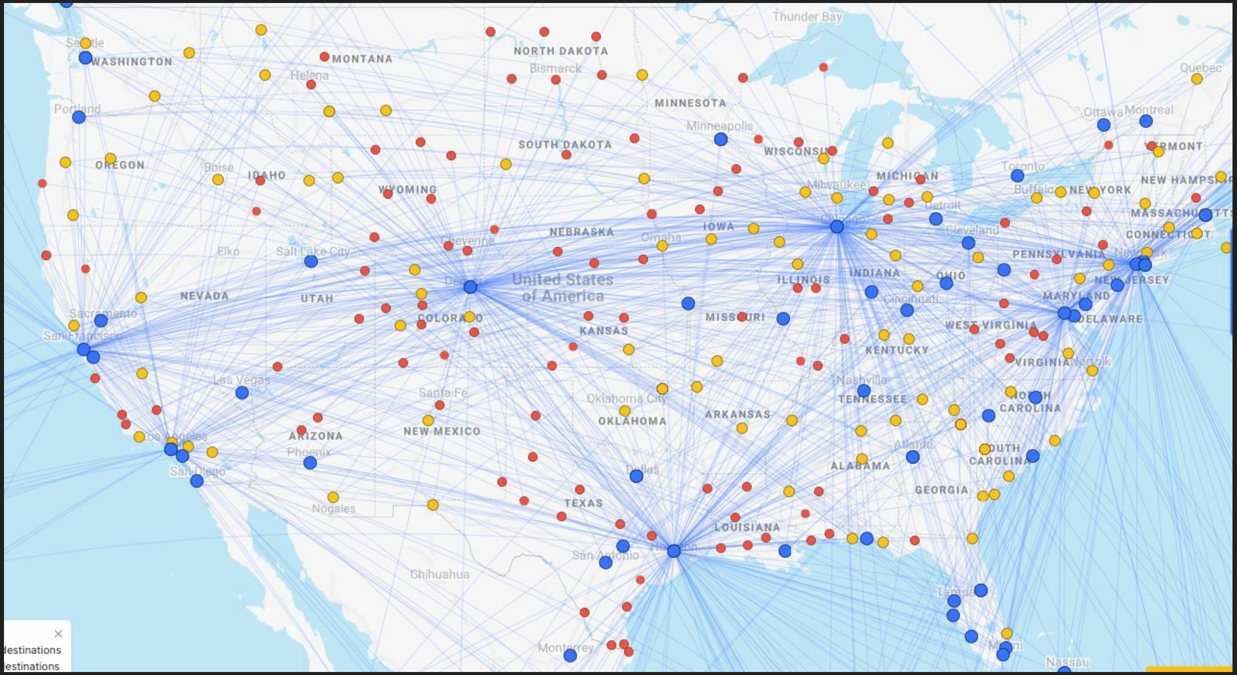
MetLife Stadium
1 MetLife Stadium Dr, East Rutherford, NJ 07073

Google Maps

Roads are edges and the intersections are vertices. Distance between vertices are important to determine shortest routes

Fun Fact!
The route directions are a linked list! And if you want roundtrip directions thats a double linked list!

Another example of an Weighted Graph

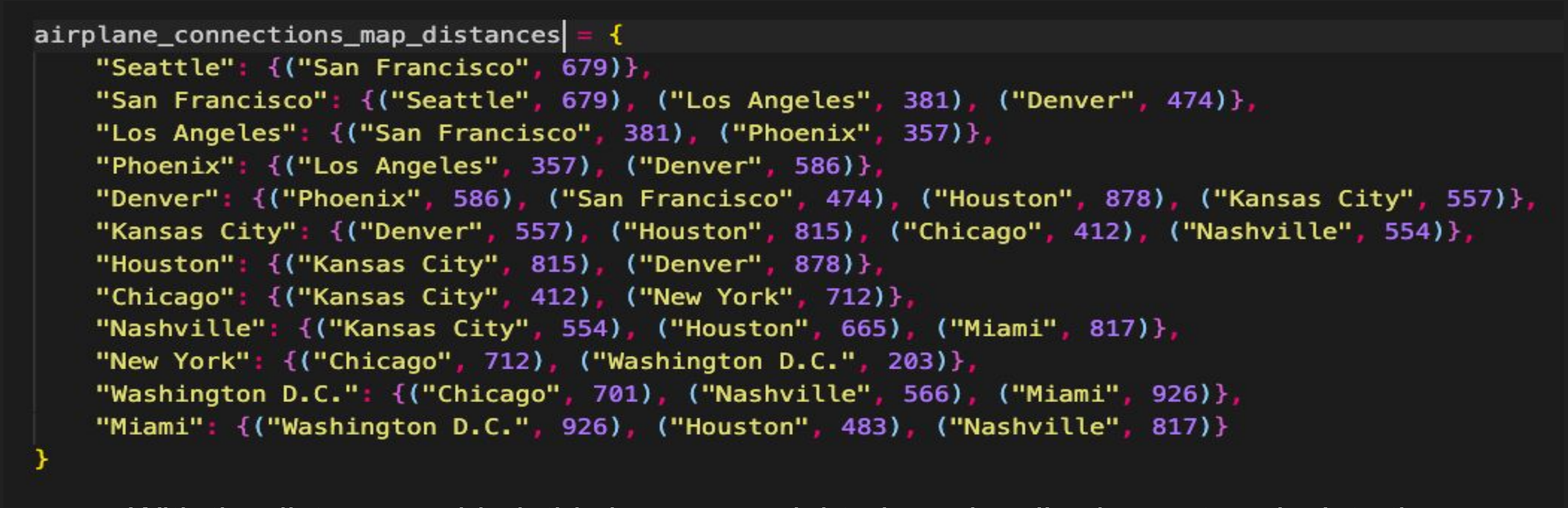


Flight Maps - United

Airports are the vertices and flight paths the edges. Distance between vertices are important to determine shortest routes

Example of a Weighted Graph - Sneak Peak!

Adjacency List from Python Project

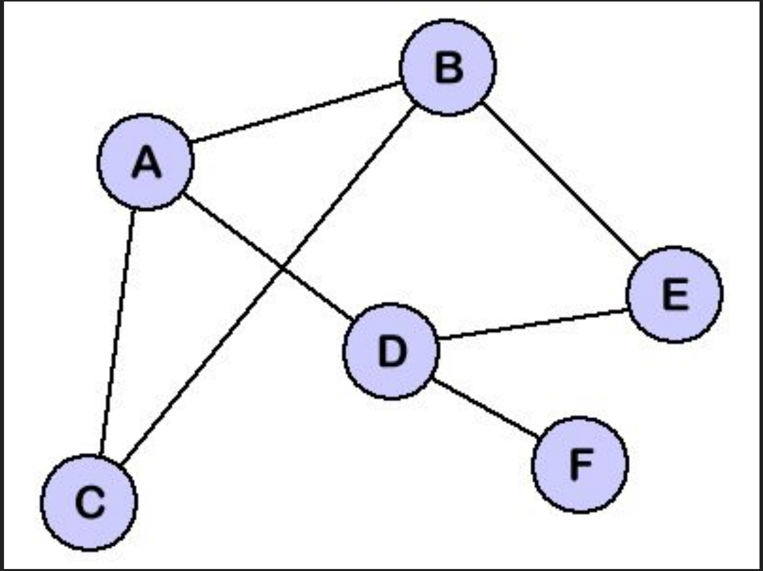


```
airplane_connections_map_distances = {
    "Seattle": {("San Francisco", 679)},
    "San Francisco": {("Seattle", 679), ("Los Angeles", 381), ("Denver", 474)},
    "Los Angeles": {("San Francisco", 381), ("Phoenix", 357)},
    "Phoenix": {("Los Angeles", 357), ("Denver", 586)},
    "Denver": {("Phoenix", 586), ("San Francisco", 474), ("Houston", 878), ("Kansas City", 557)},
    "Kansas City": {("Denver", 557), ("Houston", 815), ("Chicago", 412), ("Nashville", 554)},
    "Houston": {("Kansas City", 815), ("Denver", 878)},
    "Chicago": {("Kansas City", 412), ("New York", 712)},
    "Nashville": {("Kansas City", 554), ("Houston", 665), ("Miami", 817)},
    "New York": {("Chicago", 712), ("Washington D.C.", 203)},
    "Washington D.C.": {("Chicago", 701), ("Nashville", 566), ("Miami", 926)},
    "Miami": {("Washington D.C.", 926), ("Houston", 483), ("Nashville", 817)}
}
```

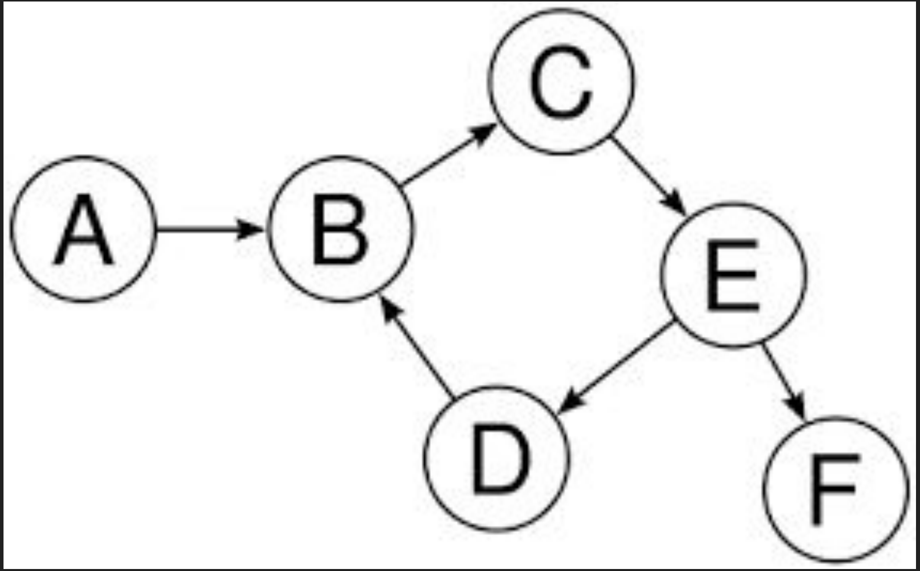
With the distances added, this is now a weighted graphs allowing us to calculate the shortest distance between 2 points (using dark magic and Dijkstra’s Algorithm)

Undirected vs Directed Graphs

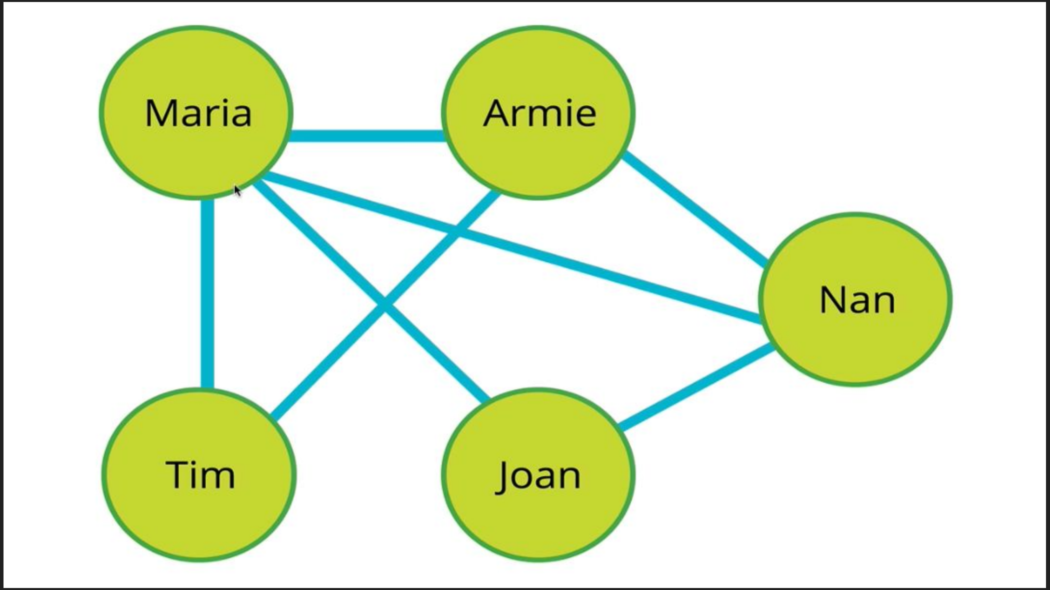
UNDIRECTED



DIRECTED



Example of an Undirected Graph - Facebook



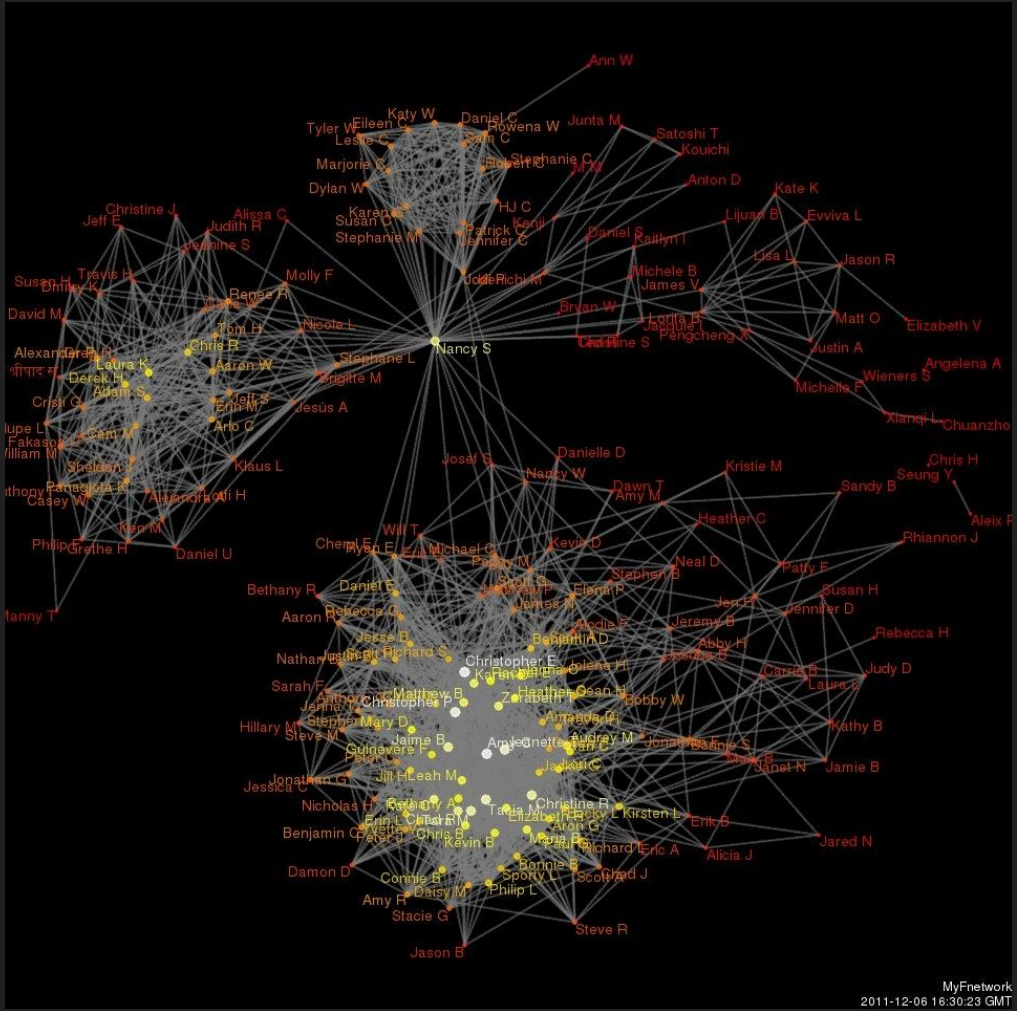
Facebook Friends List

Connections (edges) between Friends (vertexes) are 2 directional, you see all your Friends content, and they see yours

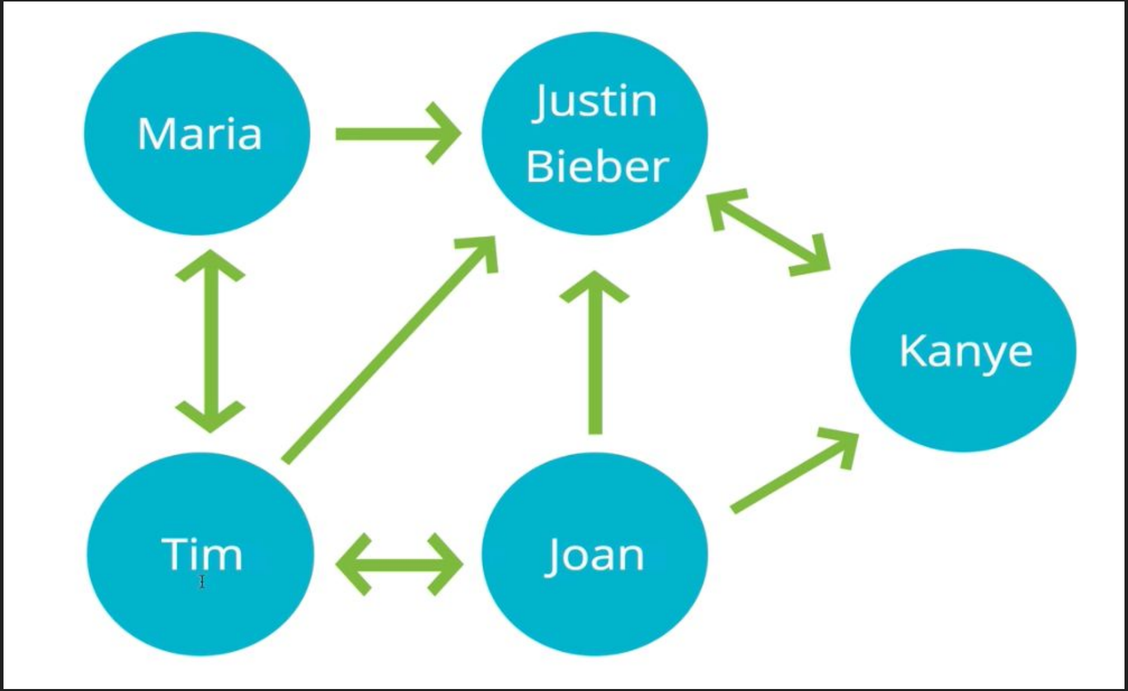
Example of an Undirected Graph - Facebook

Facebook Friends List

Graphs representation of a friends list for a single person, showing their friends and friends friends. Clusters are likely coworkers or high school / college friends



Example of Directed Graph - Instagram



Instagram Followers

Connections (edges) between Friends (vertexes) are not guaranteed to be 2 directional (but can be) Maria follows Tim and Justin Bieber. Tim follows Maria back, but Justin does not (Maria really wished Justin did)

Other IRL Graphs

Knowledge Graph -

Wikipedia

Recommendation Engine -

People you may know…, Frequently bought with…, People also watched...