CATALYST

Graphs

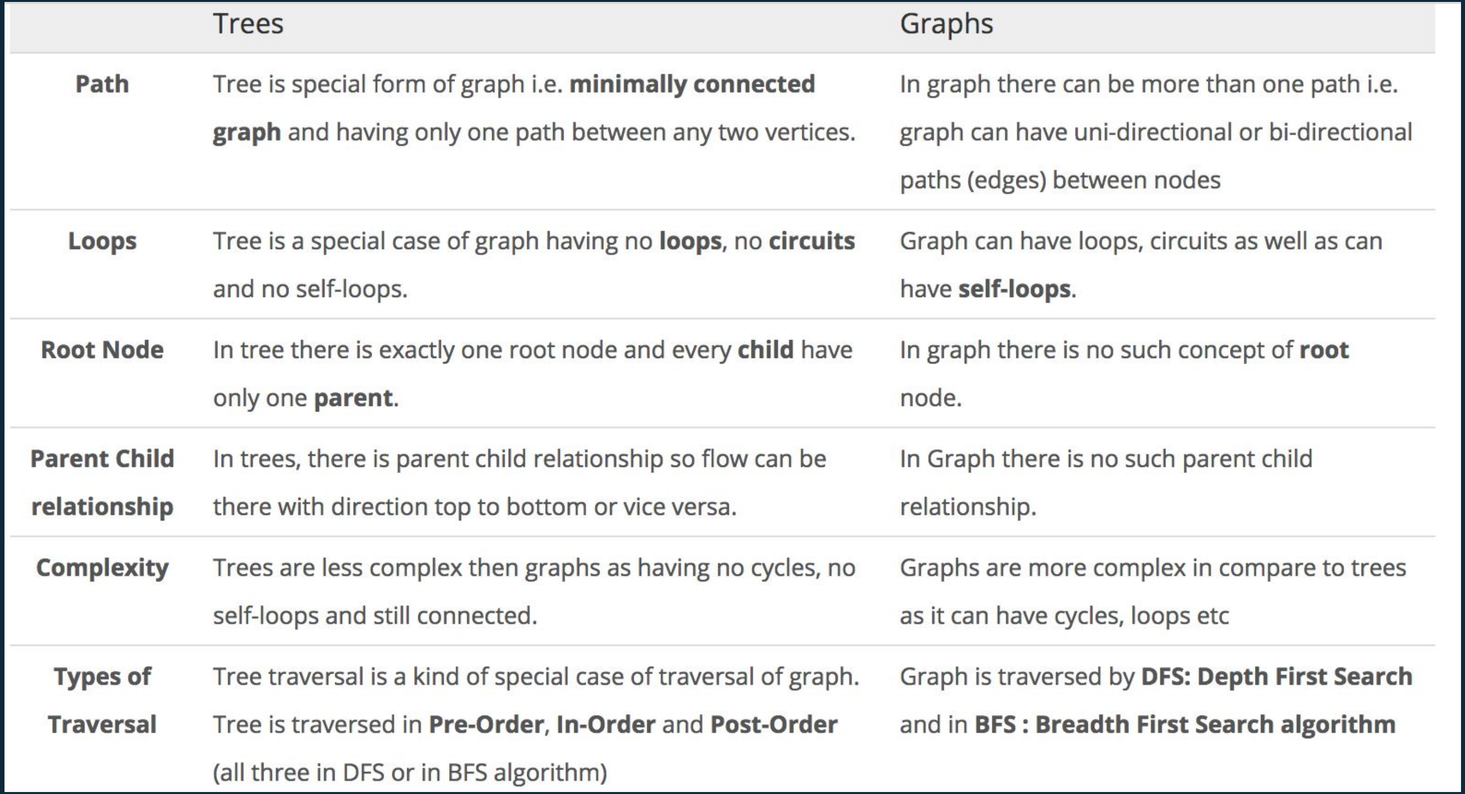
CATALYST

1 What is a Graph?

In a layman's terms:

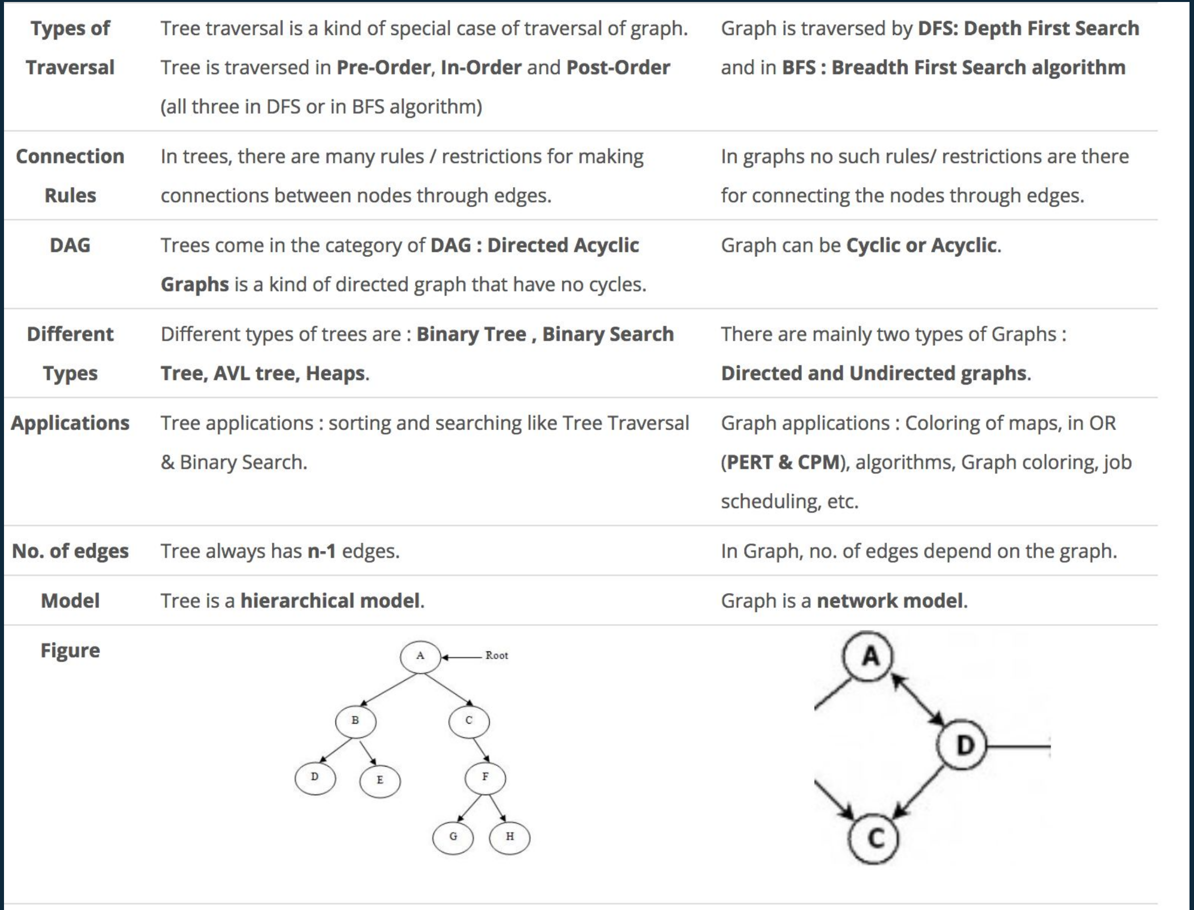
A graph is a set of vertices or nodes which are connected by edges.

Tree v Graph?



	Trees	Graphs
Path	Tree is special form of graph i.e. **minimally connected graph** and having only one path between any two vertices.	In graph there can be more than one path i.e. graph can have uni-directional or bi-directional paths (edges) between nodes
Loops	Tree is a special case of graph having no **loops**, no **circuits** and no self-loops.	Graph can have loops, circuits as well as can have **self-loops**.
Root Node	In tree there is exactly one root node and every **child** have only one **parent**.	In graph there is no such concept of **root** node.
Parent Child relationship	In trees, there is parent child relationship so flow can be there with direction top to bottom or vice versa.	In Graph there is no such parent child relationship.
Complexity	Trees are less complex then graphs as having no cycles, no self-loops and still connected.	Graphs are more complex in compare to trees as it can have cycles, loops etc
Types of Traversal	Tree traversal is a kind of special case of traversal of graph. Tree is traversed in **Pre-Order**, **In-Order** and **Post-Order** (all three in DFS or in BFS algorithm)	Graph is traversed by **DFS: Depth First Search** and in **BFS : Breadth First Search algorithm**

Tree v Graph?



Types of Traversal	Tree traversal is a kind of special case of traversal of graph. Tree is traversed in **Pre-Order**, **In-Order** and **Post-Order** (all three in DFS or in BFS algorithm)	Graph is traversed by **DFS: Depth First Search** and in **BFS : Breadth First Search algorithm**
Connection Rules	In trees, there are many rules / restrictions for making connections between nodes through edges.	In graphs no such rules/ restrictions are there for connecting the nodes through edges.
DAG	Trees come in the category of **DAG : Directed Acyclic Graphs** is a kind of directed graph that have no cycles.	Graph can be **Cyclic or Acyclic**.
Different Types	Different types of trees are : **Binary Tree , Binary Search Tree, AVL tree, Heaps**.	There are mainly two types of Graphs : **Directed and Undirected graphs**.
Applications	Tree applications : sorting and searching like Tree Traversal & Binary Search.	Graph applications : Coloring of maps, in OR (**PERT & CPM**), algorithms, Graph coloring, job scheduling, etc.
No. of edges	Tree always has **n-1** edges.	In Graph, no. of edges depend on the graph.
Model	Tree is a **hierarchical model**.	Graph is a **network model**.
Figure		

> In Short: Tree is a restricted form of a graph

Kinds of Graphs

- Undirected or Directed
- Cyclic or Acyclic
- Weighted or Unweighted
- Simple
- And more!

Some additional vocabulary:

Adjacent/Neighbors, Degree, Loop.

Graph Representation

Adjacency Matrix

V x V boolean (or numerical) matrix where an entry X[a][b] is true (or 1) if there exists an edge from vertex a to b.

SPACE O(V^2)

Adjacency List

Collection of lists (X) where X[i} contains the vertices which are the neighbors of vertex i.

SPACE O(V+E))

BFS / DFS

CATALYST

BFS/DFS

What is BFS?

What is DFS?

BFS

- Boolean visited” array -> why?
- Expand the root and visit all of its children before going to childrens children
- So what data structures would be best for this?

CATALYST

DFS

- Boolean -> why?
- Recursion
- Go deep into a node
- So which is better BFS or DFS?
- Lets try implementing before we move on!

Tries

- A type of tree - used for characters usually
- Allows for specific quick lookups
- Think of this anytime you have lookup problems (prefix based search, store dictionary, etc)
- Let’s see an example.

Insert/Search/Delete

- How would you do each of these?

CATALYST

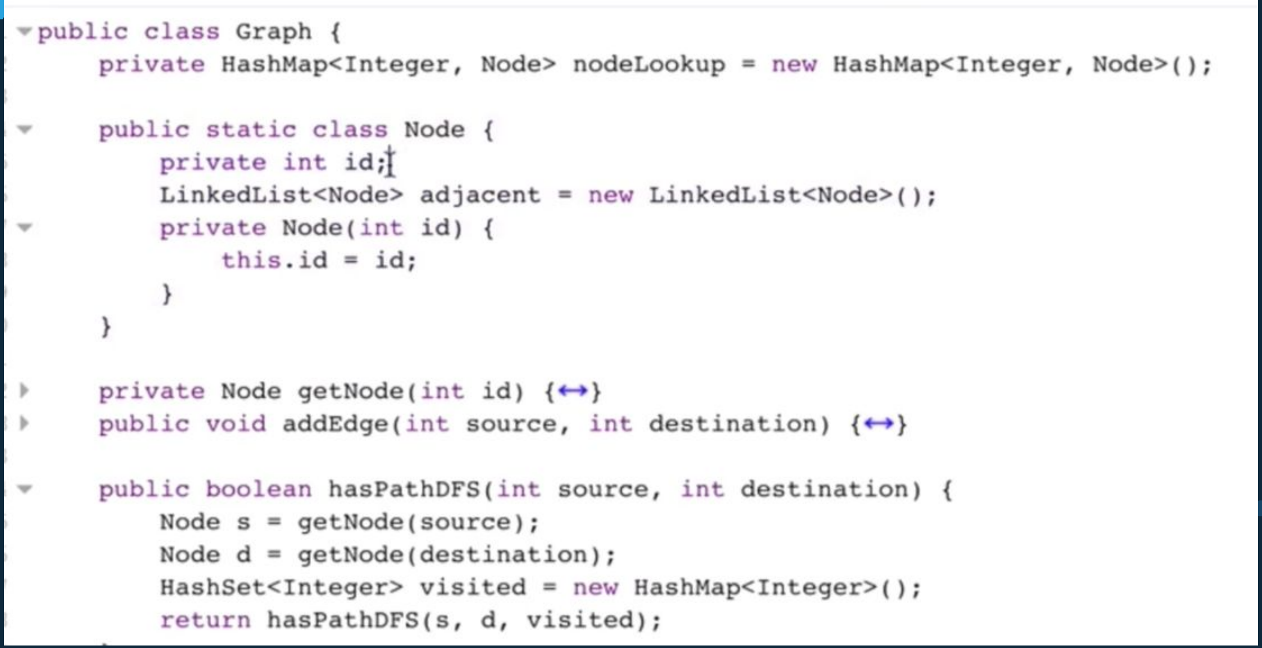
2

Let’s trie some problems!

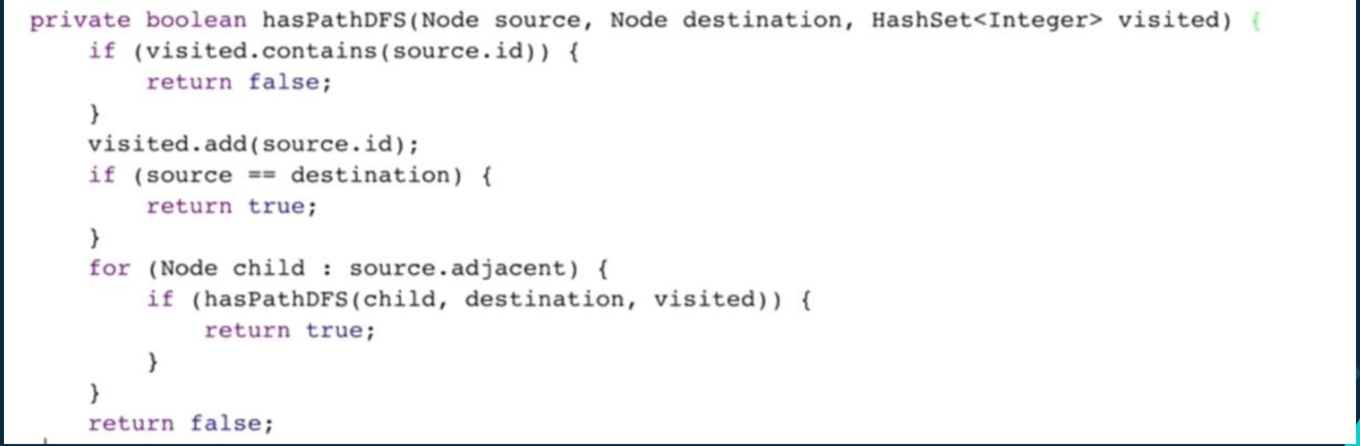
CATALYST

Q1: Implementation

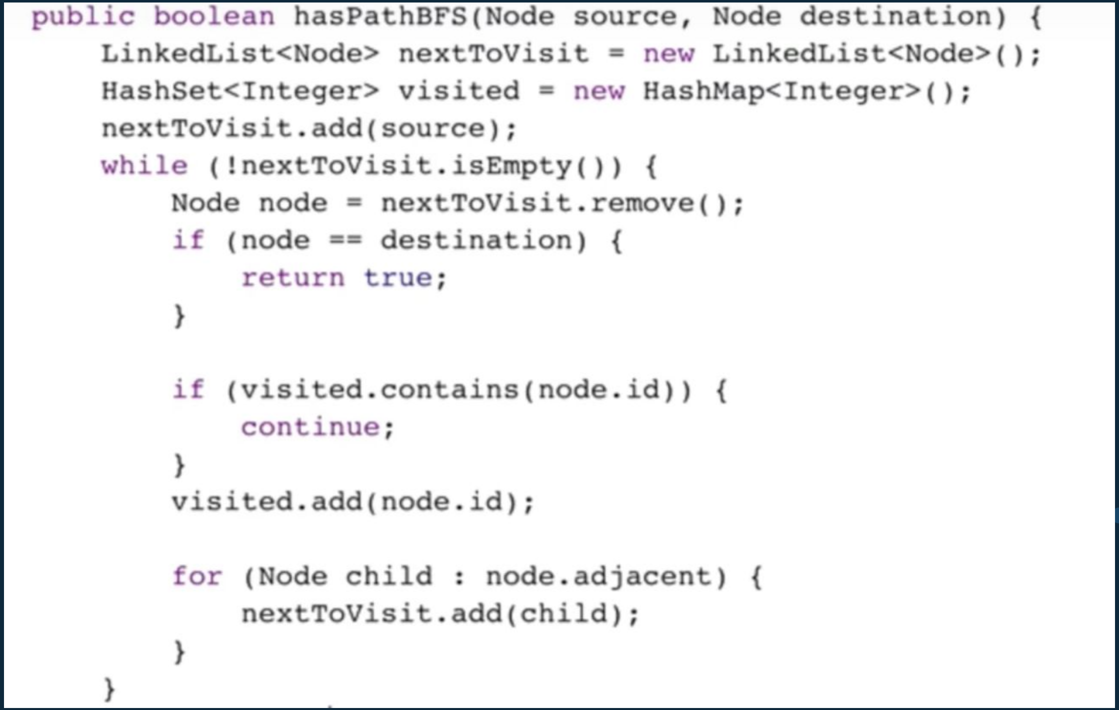
- Implement either BFS or DFS



```
public class Graph {
    private HashMap<Integer, Node> nodeLookup = new HashMap<Integer, Node>();

    public static class Node {
        private int id;
        LinkedList<Node> adjacent = new LinkedList<Node>();
        private Node(int id) {
            this.id = id;
        }
    }

    private Node getNode(int id) {↔}
    public void addEdge(int source, int destination) {↔}

    public boolean hasPathDFS(int source, int destination) {
        Node s = getNode(source);
        Node d = getNode(destination);
        HashSet<Integer> visited = new HashMap<Integer>();
        return hasPathDFS(s, d, visited);
```



```
private boolean hasPathDFS(Node source, Node destination, HashSet<Integer> visited) {
    if (visited.contains(source.id)) {
        return false;
    }
    visited.add(source.id);
    if (source == destination) {
        return true;
    }
    for (Node child : source.adjacent) {
        if (hasPathDFS(child, destination, visited)) {
            return true;
        }
    }
    return false;
```



```
public boolean hasPathBFS(Node source, Node destination) {
    LinkedList<Node> nextToVisit = new LinkedList<Node>();
    HashSet<Integer> visited = new HashMap<Integer>();
    nextToVisit.add(source);
    while (!nextToVisit.isEmpty()) {
        Node node = nextToVisit.remove();
        if (node == destination) {
            return true;
        }

        if (visited.contains(node.id)) {
            continue;
        }
        visited.add(node.id);

        for (Node child : node.adjacent) {
            nextToVisit.add(child);
        }
    }
```

CATALYST

Q2: Detect a Graph Cycle

- Undirected or Directed
- Cyclic or Acyclic
- Weighted or Unweighted
- Simple
- And more!

Some additional vocabulary:

Adjacent/Neighbors, Degree, Loop.

CATALYST

Q3: Perfect Squares

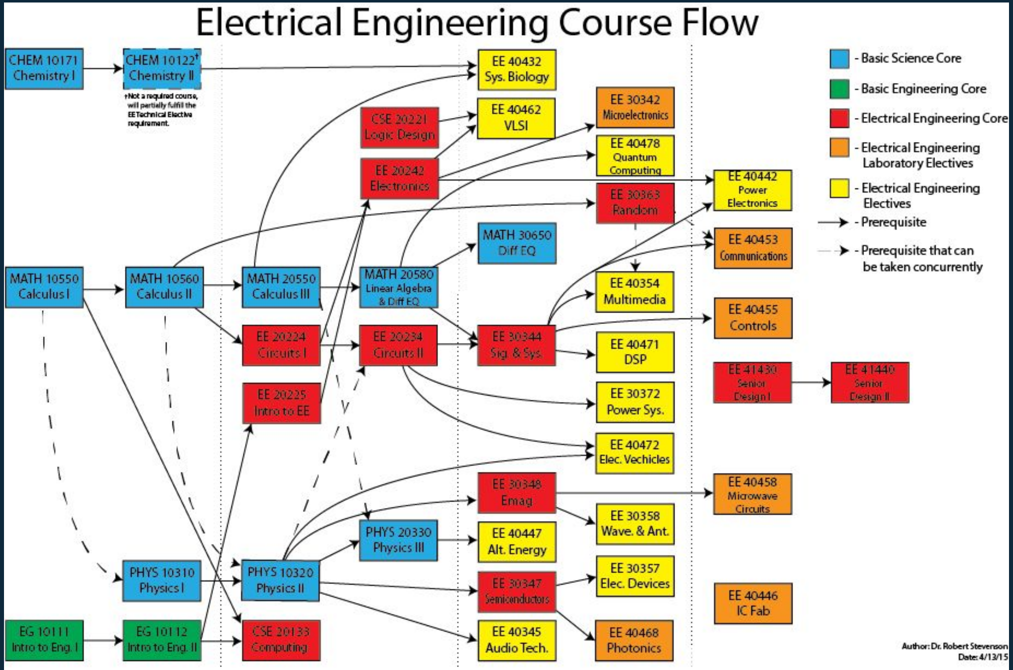
I give you an array of unique integers. Tell me whether it's possible to re-arrange the integers such that every two consecutive elements will sum up to a perfect square.

CATALYST

4

Topological Sorting

An extension of DFS

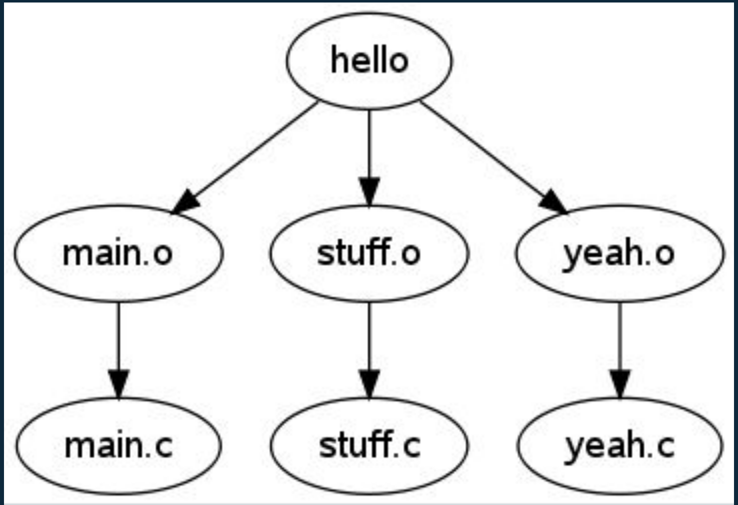


Electrical Engineering Course Flow

†Not a required course, will partially fulfill the EE Technical Elective requirement.

Author: Dr. Robert Stevenson
Date: 4/13/15

CATALYST



CATALYST

5 Heaps

Heap Property

Min heap:

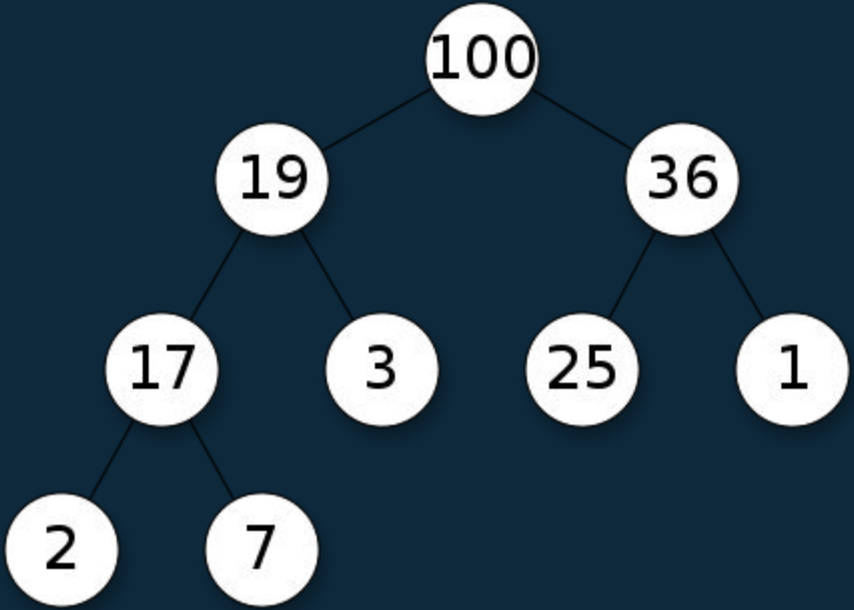
- Item at top of heap is smallest
- The children of any node are smaller than the parent node

Max heap:

- Item at top of heap is largest
- The children of any node are larger

Heap structure: heaps are complete trees (nodes are filled in from left to right - once a level is filled, go to new level)

CATALYST



Heap Uses

- Heapsort!
- Median of stream of integers