Week 5

Trees, Tries, Heaps & Graphs

CSE 492 J

Graphs & Trees

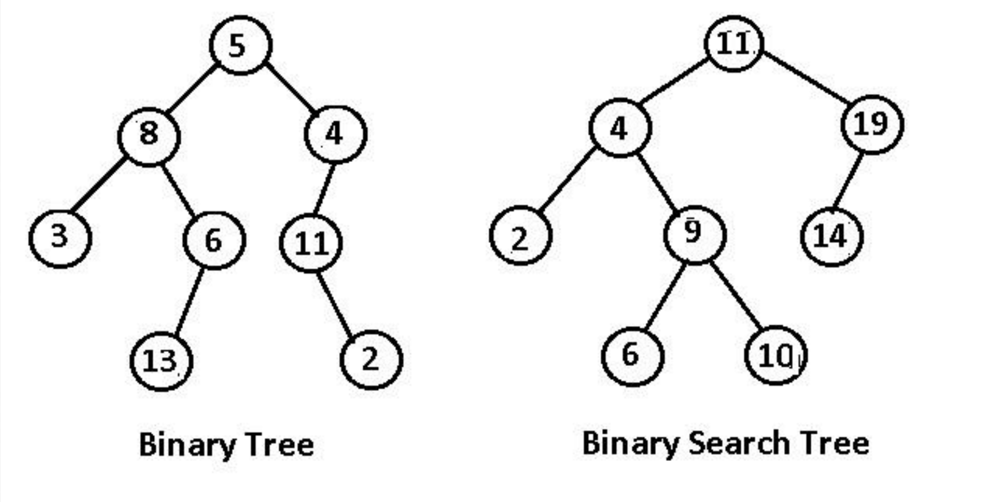
Graphs & Trees

- Trees:
  - Binary Tree, Binary Search Tree, Balanced Binary Tree(AVL), MST, Heaps, Tries.
- Graph Algorithms:
  - Breadth-First Search, Depth-First Search, Dijkstra's, Bellman-Ford, Network Flow.

Graphs & Trees

- Binary Tree, Binary Search Tree, Balanced Binary Tree(AVL)
- MST, Heaps, Tries
- Breadth-First Search, Depth-First Search
- Dijkstra's
- Bellman-Ford, Network Flow

Binary Tree vs Binary Search Tree



AVL Trees

Operation	Case	Runtime
`containsKey(key)`	best	Θ(1)
	worst	Θ(log n)
`insert(key)`	best	Θ(log n)
	worst	Θ(log n)
`delete(key)`	best	Θ(log n)
	worst	Θ(log n)

PROS

All operations on an AVL Tree have a logarithmic worst case

- Because these trees are always balanced!

The act of rebalancing adds no more than a constant factor to insert and delete

➢ Asymptotically, just better than a normal BST!

CONS

- Relatively difficult to program and debug (so many moving parts during a rotation)
- Additional space for the height field
- Though asymptotically faster, rebalancing *does* take some time
  - Depends how important every little bit of performance is to you

Review: Dictionaries

Dictionary ADT

state

Set of items & keys
Count of items

behavior

put(key, item) add item to collection indexed with key
get(key) return item associated with key
containsKey(key) return if key already in use
remove(key) remove item and associated key
size() return count of items

Why are we so obsessed with Dictionaries?

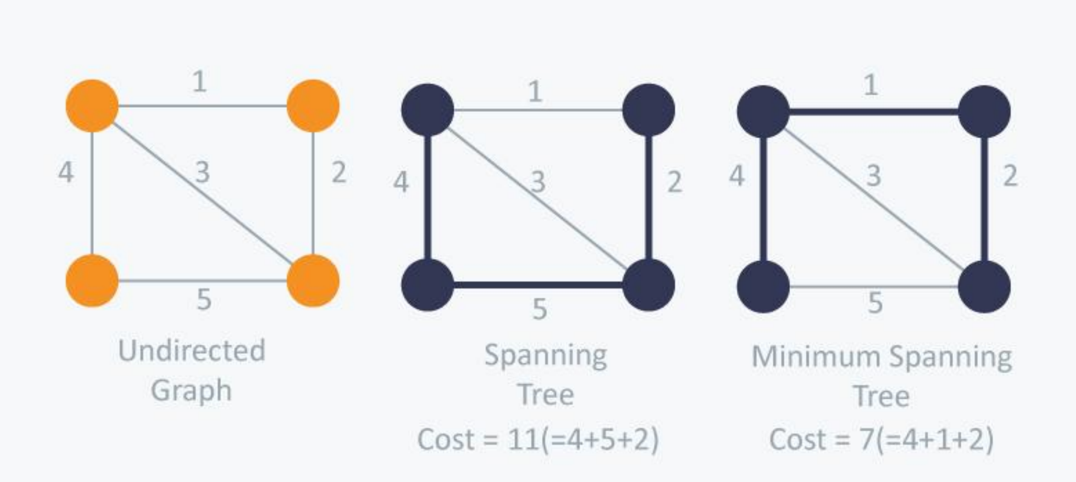
When dealing with data:

- Adding data to your collection
- Getting data out of your collection
- Rearranging data in your collection

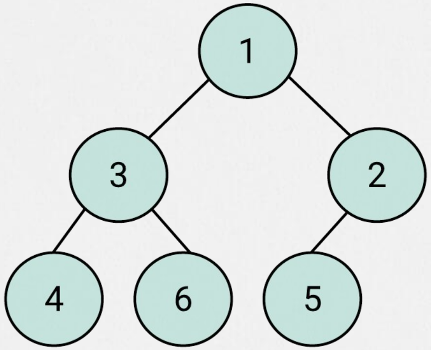
Operation		ArrayList	LinkedList	HashTable	BST	AVLTree
put(key,value)	best	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
	worst	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
get(key)	best	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
	worst	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
remove(key)	best	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(\log n)$
	worst	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$

MST(Minimum Spanning Tree)

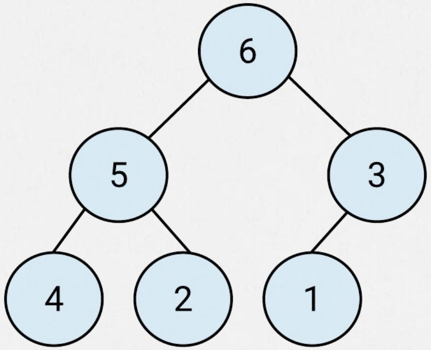
Kruskal's vs Prim's



Heaps

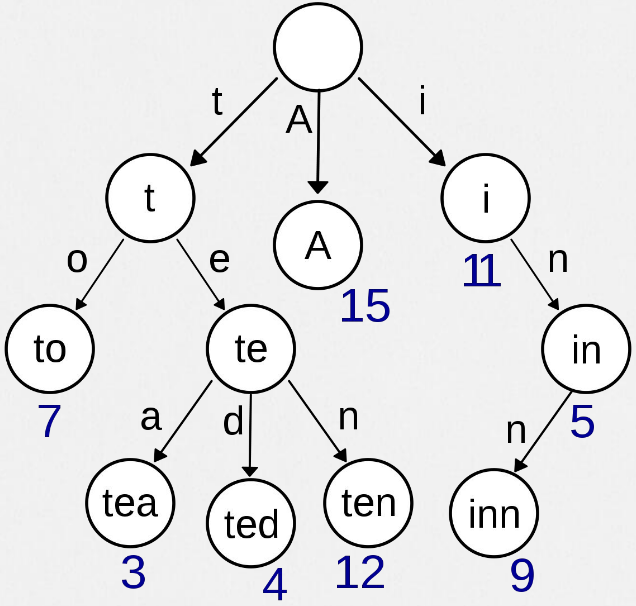


Min heap

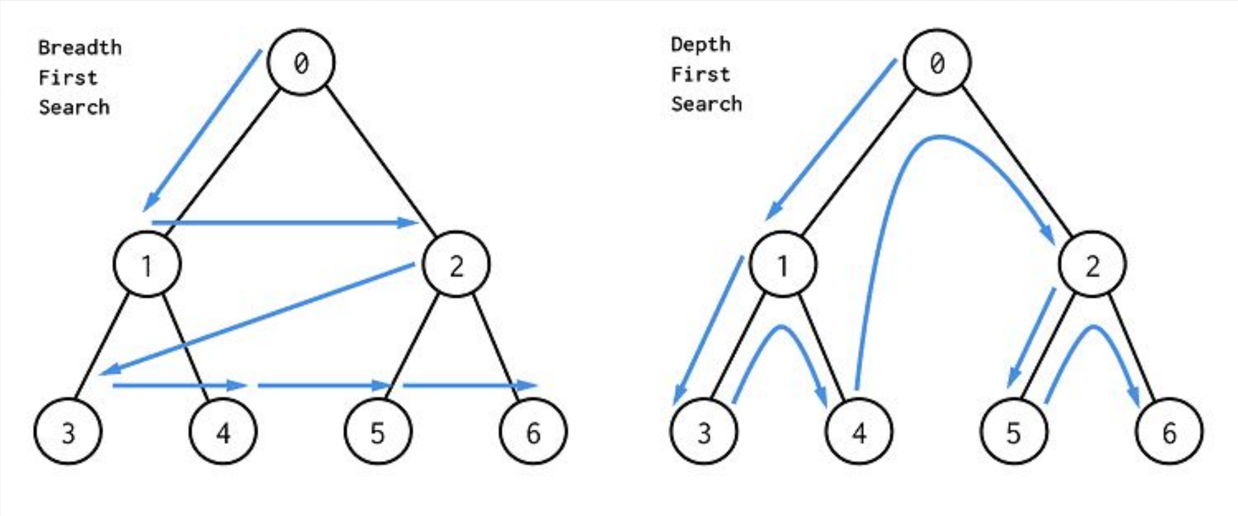


Max Heap

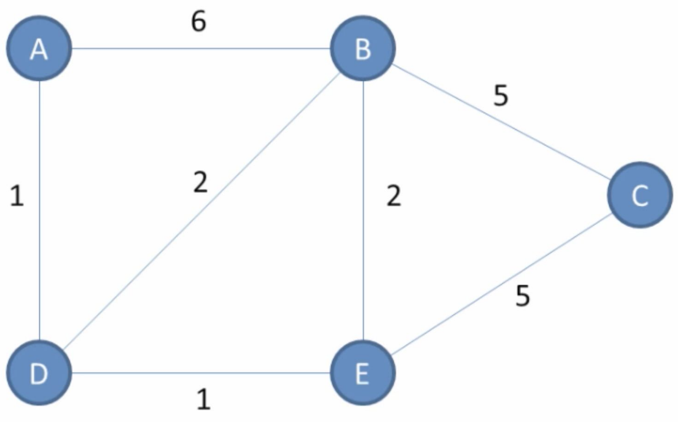
Trie



BFS vs DFS



Dijkstra's

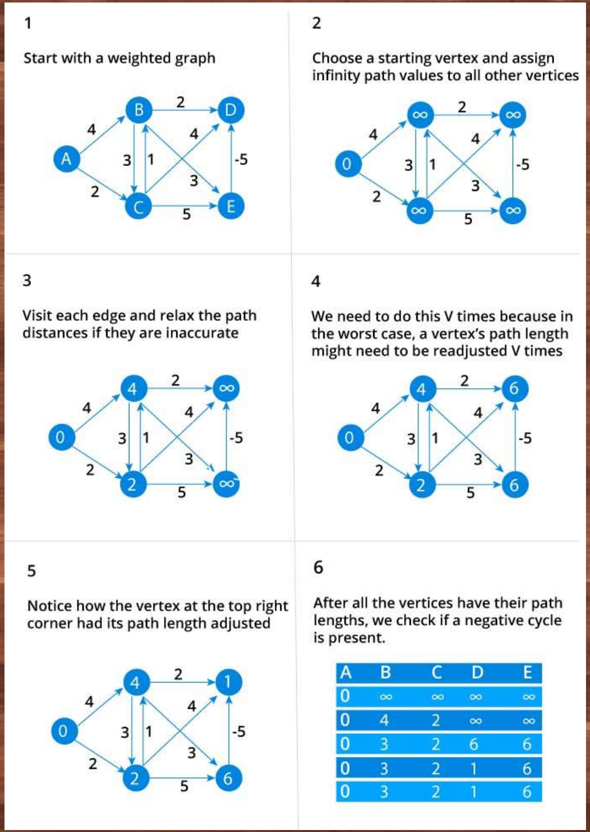


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

Bellman-Ford



1

Start with a weighted graph

2

Choose a starting vertex and assign infinity path values to all other vertices

3

Visit each edge and relax the path distances if they are inaccurate

4

We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times

5

Notice how the vertex at the top right corner had its path length adjusted

6

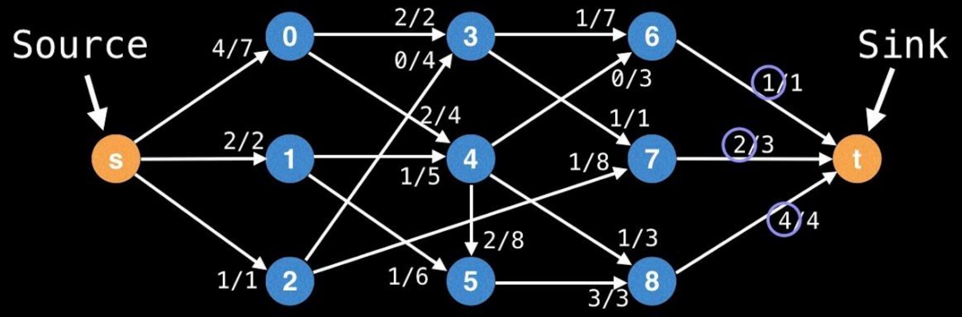
After all the vertices have their path lengths, we check if a negative cycle is present.

A	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

Network Flow

Max flow

Q: With an infinite input source, how much “flow” can we push through the network given that each edge has a certain capacity?



Let's Practice with CEAO-IT!

Question

Given an integer array `nums` and an integer `k`, return the `k`th largest element in the array.

What would we use to solve this problem?

Clarify

1. Does `k` refer to the `k`th largest distinct element?
   a. No, it is the `k`th largest element *after* sorting the input array.
2. Can we use additional data structures?
   a. Yes, assume we want the fastest runtime.
3. Can there be duplicate/non-positive integers in the input array `nums`?
   a. Yes.
4. Can the value of `k` be greater than the length of the input array `nums`?
   a. No. The length of `nums` is greater than `k`, which is greater than 1.

Example

Middle Case:

```
nums = [3,2,1,5,6,4]; k = 2
Output = 5
```

Edge Case 1:

```
nums = [-45]; k = 1
Output = -45
```

Example

Edge Case 2:

```
nums = [3,2,3,1,2,4,5,5,6]; k = 4
Output = 4
```

Notice the duplicates!

(This is why clarifying is important, interviewers may or may not care about duplicates…Ask, do not read minds although that would be cool)

Approach

Brute Force:

1. Use an O(n * logn) sorting algorithm
2. Sort the entire array
3. Traverse through the array and return the `k`th element

There is a faster way using a data structure we discussed today...

Optimize

- Optimized: Use a Heap O(n) for adding elements of input array to Heap + O(logn) for finding Kth largest element in Heap = O(n)
- Optimized no additional data structure: use an in-place sorting algorithm with O(n * logn) runtime
- Note: When you hear "find the largest/smallest.." in an interview, think about using a heap if applicable.

Implement

- There are no sorting algorithms that run in O(n) runtime. So we create a MinHeap using a Priority Queue to have O(n) runtime.
- Iterate through `nums` and add its elements into our heap. For every iteration, make sure our heap size is not greater than `k`, otherwise remove root of MinHeap.
- Our result will be the final root of our MinHeap after iterating through all of `nums`.

Implement – Java Code

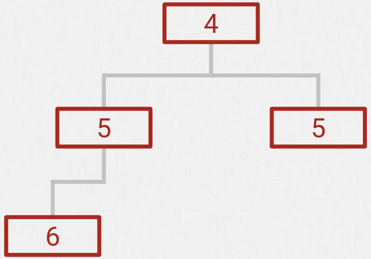
```java
public int findKthLargest(int[] nums, int k) {
    PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>();
    for (int i: nums) {
        minHeap.add(i);
        if (minHeap.size() > k) {
            minHeap.remove();
        }
    }
    return minHeap.remove();
}
```

Test

Test with Edge Case with Duplicates:

```
nums = [3,2,3,1,2,4,5,5,6]; k = 4
```

Resulting MinHeap after for loop ends:



Remove root element to get answer →

```
Output
= 4
```

Extra Practice Problems for Pattern-Building!

Binary Trees (Easy):

- https://leetcode.com/problems/validate-binary-search-tree/
- https://leetcode.com/problems/path-sum/
- https://leetcode.com/problems/maximum-difference-between-node-and-ancestor/

Binary Trees (Medium):

- https://leetcode.com/problems/path-sum-ii/
- https://leetcode.com/problems/binary-search-tree-iterator/

Extra Practice Problems for Pattern-Building!

Graphs (Easy):

> https://leetcode.com/problems/number-of-islands/

> https://leetcode.com/problems/rotting-oranges/

> https://leetcode.com/problems/accounts-merge/

> https://leetcode.com/problems/evaluate-division/

> https://leetcode.com/problems/word-ladder/

> https://leetcode.com/problems/all-paths-from-source-to-target/

Extra Practice Problems for Pattern-Building!

Graphs (Medium):

> https://leetcode.com/problems/reconstruct-itinerary/
> https://leetcode.com/problems/binary-tree-right-side-view/
> https://leetcode.com/problems/pacific-atlantic-water-flow/
> https://leetcode.com/problems/clone-graph/
> https://leetcode.com/problems/course-schedule/
> https://leetcode.com/problems/number-of-provinces/
> https://leetcode.com/problems/symmetric-tree/

Extra Practice Problems for Pattern-Building!

Heaps (Easy):

> https://leetcode.com/problems/kth-largest-element-in-a-stream

> https://leetcode.com/problems/last-stone-weight/

> https://leetcode.com/problems/maximum-product-of-two-elements-in-an-array

Tries (Medium):

> https://leetcode.com/problems/search-suggestions-system/

> https://leetcode.com/problems/map-sum-pairs/