CS 61B Spring 2025

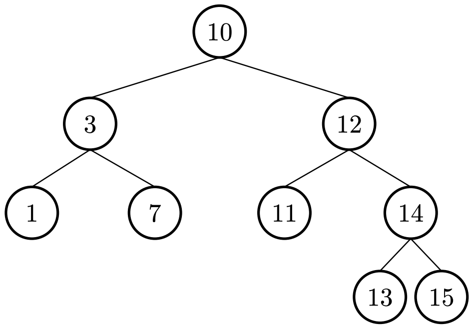
Graphs, Heaps

Discussion 08: March 17, 2025

1 Trees, Graphs, and Traversals, Oh My!

(a) Write the following traversals of the BST below.

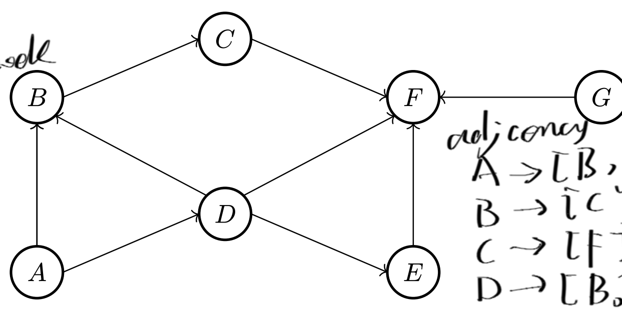
Pre-order: 10 3 1 7 12 11 14 13 15
In-order: 1 3 7 10 11 12 13 14 15
Post-order: 1 7 3 11 13 15 14 12 10
Level-order (BFS): 10 3 12 1 7 11 14 13 15



(b) Write the graph below as an adjacency matrix, then as an adjacency list. What would be different if the graph were undirected instead?

matrix

start-node \ end-node	A	B	C	D	E	F	G
A		1		1			
B			1				
C						1	
D		1			1	1	
E						1	
F							
G						1	



adjacency list
A → [B, D]
B → [C]
C → [F]
D → [B, E, F]
E → [F]
F
G → [F]

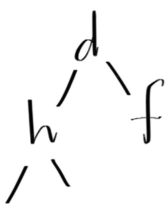
(c) Write the order in which (1) DFS pre-order, (2) DFS post-order, and (3) BFS would visit nodes in the same directed graph above, starting from vertex A. Break ties alphabetically.

Pre-order: A B C F D E (G)
Post-order: F C B E D A (G)
BFS: A B D C E F (G)

2 Absolutely Valuable Heaps

(a) Assume that we have a binary min-heap (smallest value on top) data structure called **`MinHeap`** that has properly implemented the **`insert`** and **`removeMin`** methods. Draw the heap and its corresponding array representation after each of the operations below:

```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');     [f]
h.insert('h');     [f, h]
h.insert('d');     [d, h, f]
h.insert('b');     [b, d, f, h]
h.insert('c');     [b, c, f, h, d]
h.removeMin();     [c, d, f, h]
h.removeMin();     [d, h, f]
```



(b) Your friendly TA Mihir challenges you to create an integer max-heap without writing a whole new data structure. Can you use your min-heap to mimic the behavior of a max-heap? Specifically, we want to be able to get the largest item in the heap in constant time, and add things to the heap in $\Theta(\log n)$ time, as a normal max heap should.

Hint: You should treat the **`MinHeap`** as a black box and think about how you should modify the arguments/return values of the heap functions.

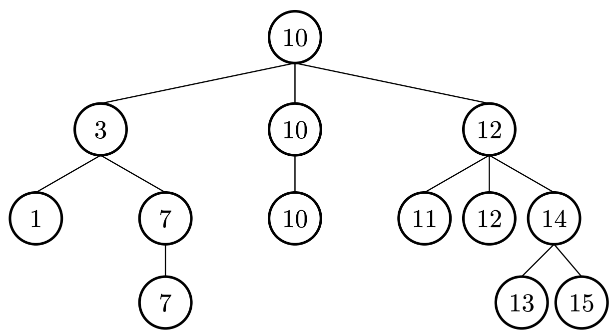
```
Integer Get( )
    return -minHeap.get( );
void insert(Integer a)
    minHeap.insert(-a);
```

3 Trinary Search Tree

We'd like a data structure that acts like a BST (Binary Search Tree) in terms of operation runtimes but allows duplicate values. Therefore, we decide to create a new data structure called a TST (Trinary Search Tree), which can have up to three children, which we'll refer to as **`left`**, **`middle`**, and **`right`**. In this setup, we have the following invariants, which are very similar to the BST invariants:

1. Each node in a TST is a root of a smaller TST
2. Every node to the **`left`** of a root has a value "lesser than" that of the root
3. Every node to the **`right`** of a root has a value "greater than" that of the root
4. **Every node to the `middle` of a root has a value equal to that of the root**

Below is an example TST to help with visualization.



Describe an algorithm that will print the elements in a TST in **descending** order. *(Hint: recall that an in-order traversal for a BST gives elements in increasing order.)*

```
List result = new List;
function(node)
    if node == null
        return;
    function(node.left);
    result.add(node);
    while node.middle != null
        result.add(node);
        node = node.middle;
    function(node.right);

Collectio...reverse(result);
```

```
reverse(tst):
    if tst is null:
        return
    reverse(tst.right)
    print(tst.value)
    reverse(tst.middle)
    reverse(tst.left)
```