Graphs, Heaps

Discussion 08

Announcements

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			3/13 Mid-semester Survey Due		3/15 Lab 8 Due Project 2B/C Checkpoint and Design Doc Due MT2 Review Session	
	3/18 Homework 3 Due			3/21 **Midterm 2**		

Content Review

Trees, Revisited (and Formally Defined)

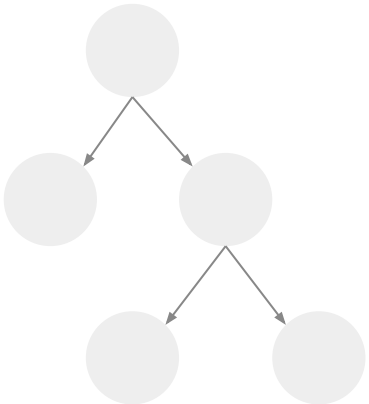
Trees are structures that follow a few basic rules:

1. If there are N nodes, there are N-1 edges
2. There is exactly 1 path from root to every other node
3. The above two rules means that trees are fully connected and contain no cycles

A parent node points towards its child.

The root of a tree is a node with no parent nodes.

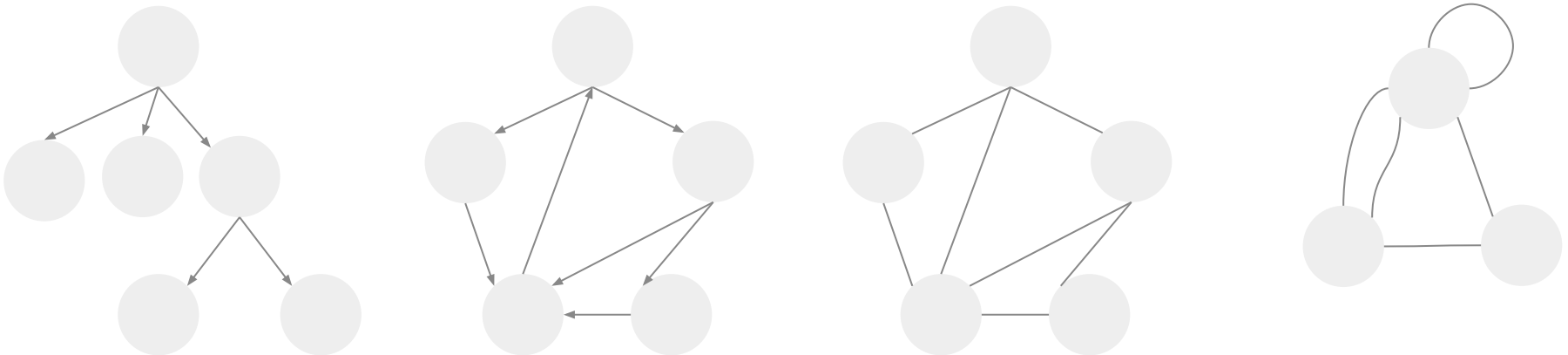
A leaf of a tree is a node with no child nodes.



Graphs

Trees are a specific kind of **graph,** which is more generally defined as below:

1. Graphs allow cycles
2. Simple graphs don't allow parallel edges (2 or more edges connecting the same two nodes) or self edges (an edge from a vertex to itself)
3. Graphs may be directed or undirected (arrows vs. no arrows on edges)

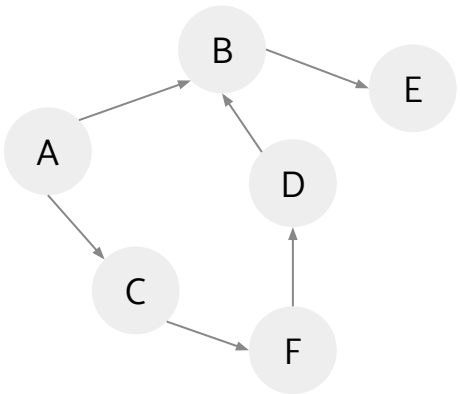


Check! How would you describe each of these graphs (in terms of directedness and cycles)?

Graph Representations

Adjacency lists list out all the nodes connected to each node in our graph:

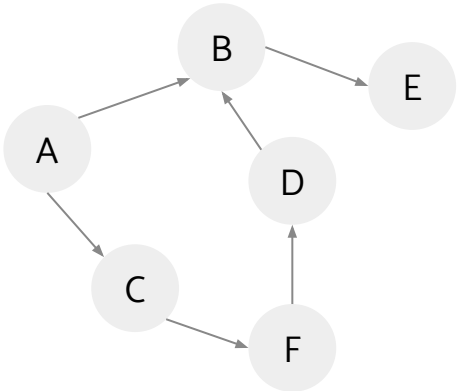
A	B , C
B	E
C	F
D	B
E	
F	D



Graph Representations

Adjacency matrices are true if there is a line going from node A to B and false otherwise.

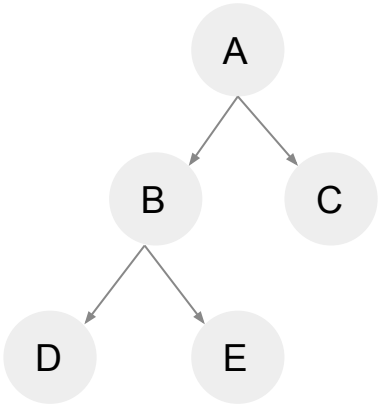
	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	0	0	1	0
C	0	0	0	0	0	1
D	0	1	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	1	0	0



Breadth First Search

Breadth First Search means visiting nodes based off of their distance to the source, or starting point. For trees, this means visiting the nodes of a tree level by level. Breadth first search is one way of traversing a graph.

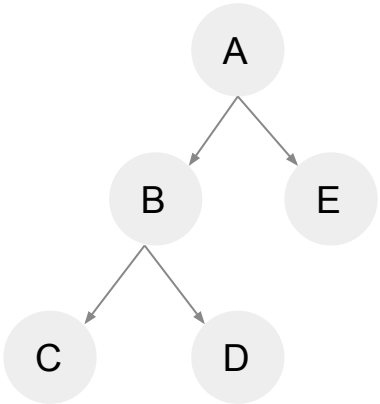
BFS is usually done using a queue.



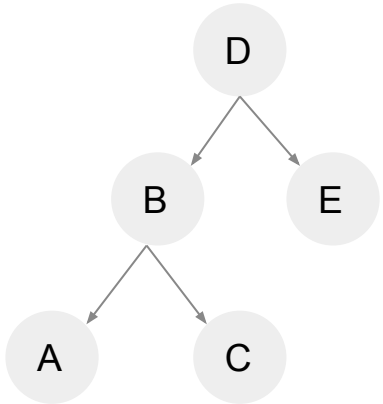

```
BFS(G):
      Add G.root to queue
      While queue not empty:
            Pop node from front of queue and visit
            for each immediate neighbor of node:
                  Add neighbor to queue if not
                  already visited
```


Depth First Search

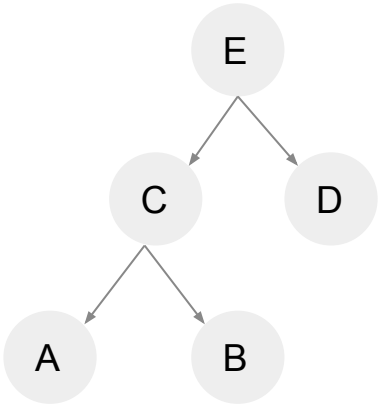
Depth First Search means we visit each subtree (subgraph) in some order recursively. DFS is usually done using a stack. Note that for graphs more generally, it doesn't really make sense to do in-order traversals.



Pre-order traversals visit the parent node before visiting child nodes.*



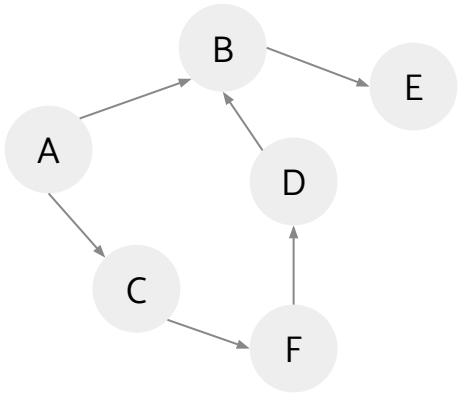
In-order traversals visit the left child, then the parent, then the right child.



Post-order traversals visit the child nodes before visiting the parent nodes.*

* in binary trees, we visit the left child before right child

General Graph DFS Pseudocode (Stack)




```
DFS(start):
      stack = {start}, visited = {}
      while stack not empty:
            n = top node in stack
            visited.add(n)
            preorder.add(n)
            if n has unvisited neighbors:
                  push n's next unvisited
                  neighbor onto stack
            else:
                  pop n off top of stack
                  postorder.add(n)
      return preorder, postorder
```

Preorder: "Visit the node as soon as it enters the stack: myself, then all my children"

Postorder: "Visit the node as soon as it leaves the stack: all my children, then myself"

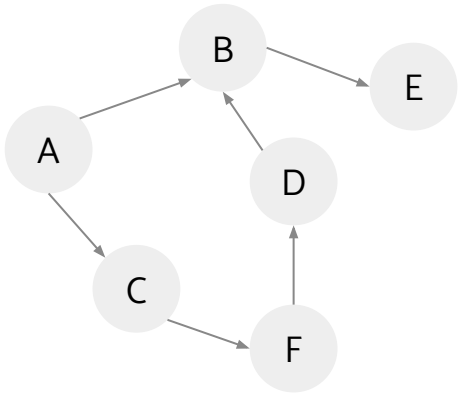
* in-order for binary trees:

```
DFSInorder(T):
      DFSInorder(T.left)
      visit T.root
      DFSInorder(T.right)
```

"Visit my left child, then myself, then my right child"*

* can be done with a stack, but usually easier with recursive

General Graph DFS Pseudocode (Recursive)




```
DFS(start):
      preorder.add(start)
      visited.add(start)
      for each neighbor of start:
            if neighbor not visited:
                  DFS(neighbor)
      postorder.add(start)
      return preorder, postorder
```

Note: technically can add:

```
if start.neighbors is empty
  preorder.add(start)
  visited.add(start)
  postorder.add(start)
```

as base case, but the code on the left will skip the loop if neighbors is empty.

* in-order for binary trees:

```
DFSInorder(T):
      DFSInorder(T.left)
      visit T.root
      DFSInorder(T.right)
```

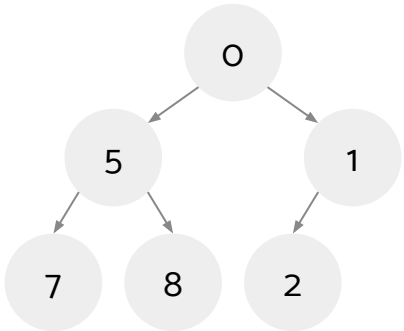
"Visit my left child, then myself, then my right child"*

* can be done with a stack, but usually easier with recursive

Heaps

Heaps are special trees that follow a few invariants:

1. Heaps are complete - the only empty parts of a heap are in the bottom row, to the right
2. In a min-heap, each node must be *less than or equal to* all of its child nodes. The opposite is true for max-heaps: each node must be *greater than or equal to* all of its child nodes.

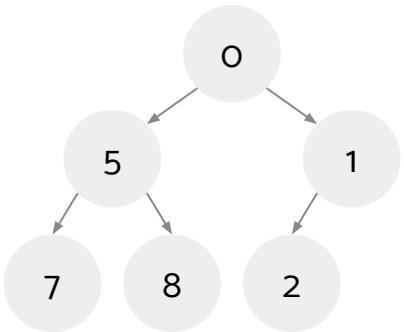


Check! What makes a binary min-heap different from a binary search tree?

Heap Representation

We can represent binary heaps as arrays with the following setup:

1. The root is stored at index 1 (not 0 - see points 2 and 3 for why)
2. The left child of a binary heap node at index i is stored at index 2i
3. The right child of a binary heap node at index i is stored at index 2i + 1

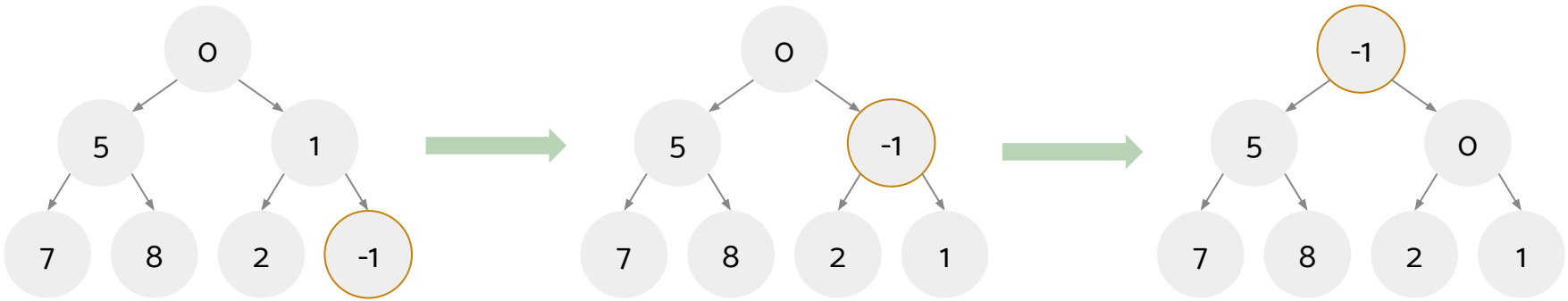



```
[-, 0, 5, 1, 7, 8, 2]
```

Check! What kind of graph traversal does the ordering of the elements in the array look like starting from the root at index 1?

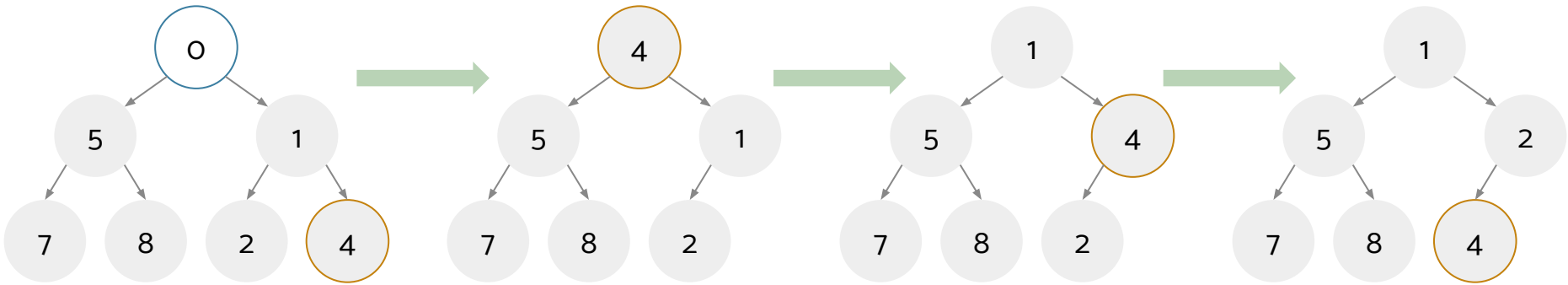
Insertion into (Min-)Heaps

We insert elements into the next available spot in the heap and **bubble up** as necessary: if a node is smaller than its parent, they will swap. (Check: what changes if this is a max heap?)



Root Deletion from (Min-)Heaps

We swap the last element with the root and **bubble down** as necessary: if a node is greater than its children, it will swap with the lesser of its children. (Check: what changes if this is a max heap?)

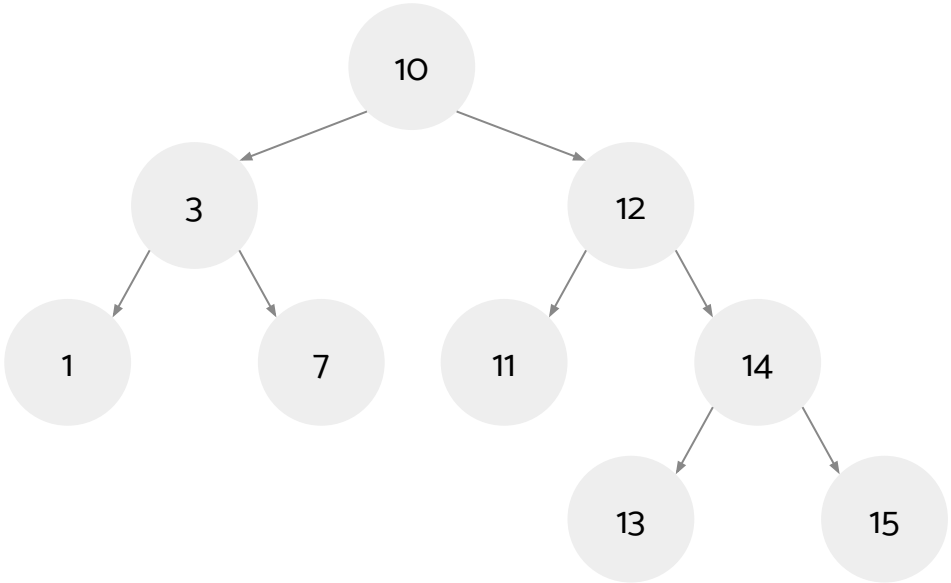


Heap Asymptotics (Worst case)

Operation	Runtime
insert	Θ(logN)
findMin	Θ(1)
removeMin	Θ(logN)

Worksheet

1a Trees, Graphs, and Traversals, Oh My! Write out the following traversals of this BST.



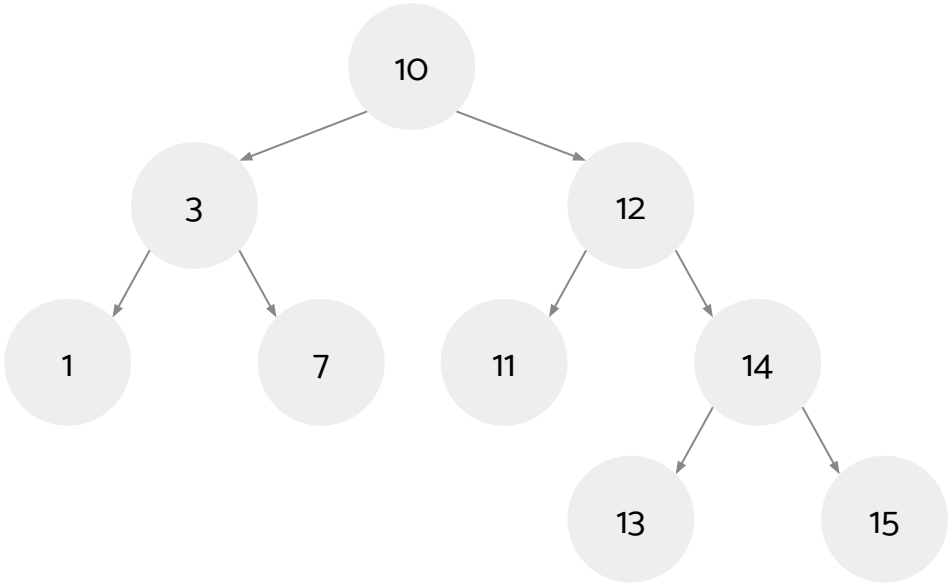
Preorder:

Inorder:

Postorder:

BFS:

1a Trees, Graphs, and Traversals, Oh My! Write out the following traversals of this BST.



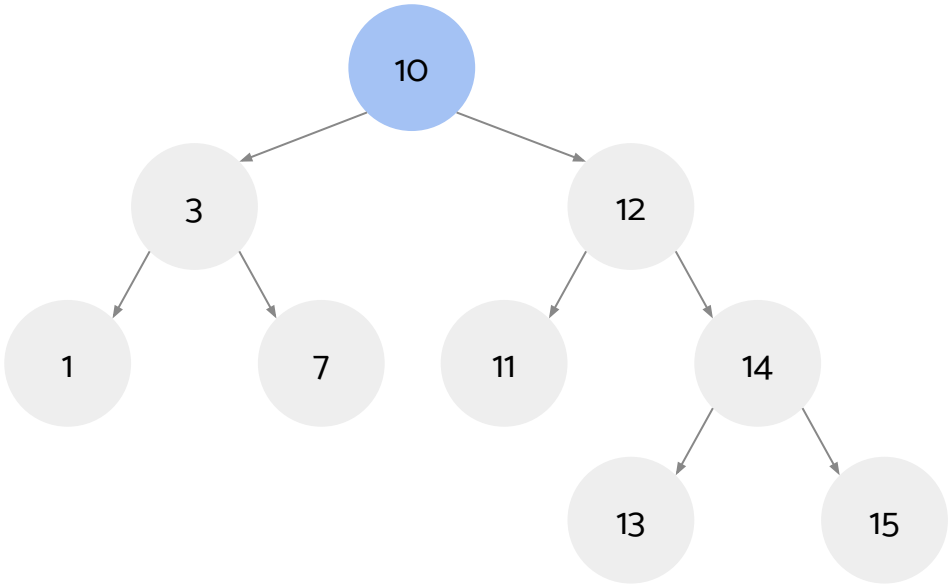
Preorder: 10 3 1 7 12 11 14 13 15

Inorder: 1 3 7 10 11 12 13 14 15

Postorder: 1 7 3 11 13 15 14 12 10

BFS: 10 3 12 1 7 11 14 13 15

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

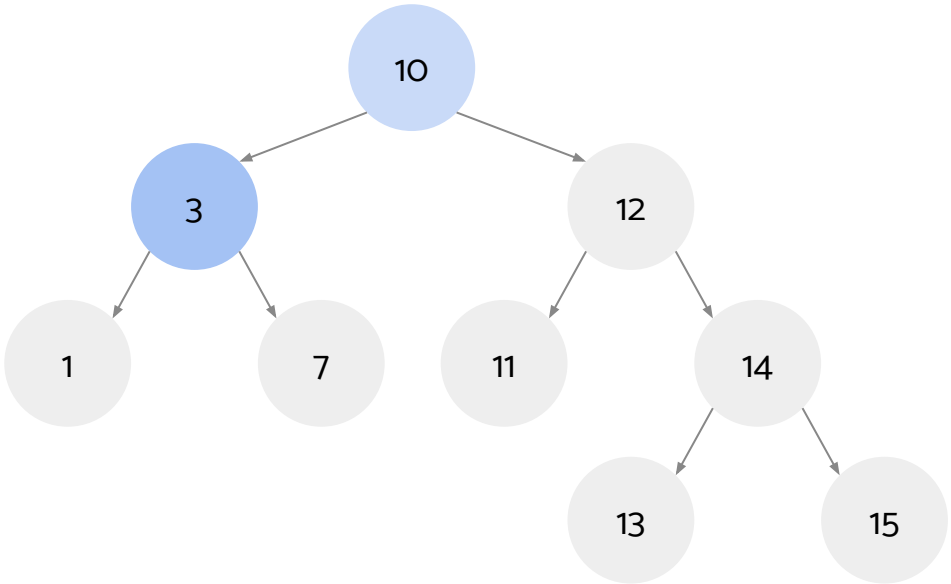


Inorder:

BFS:

Stack: 10

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

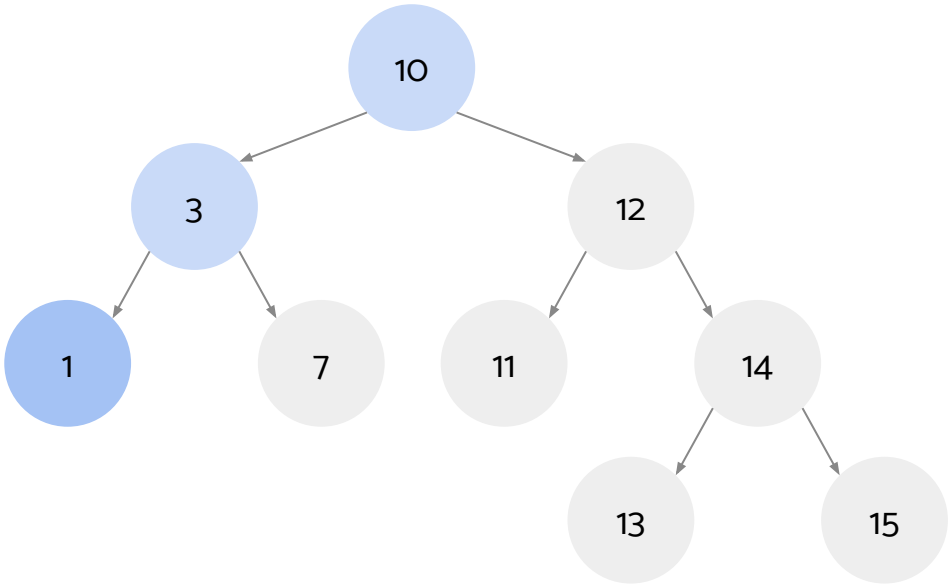


Inorder:

BFS:

Stack: 10 3

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

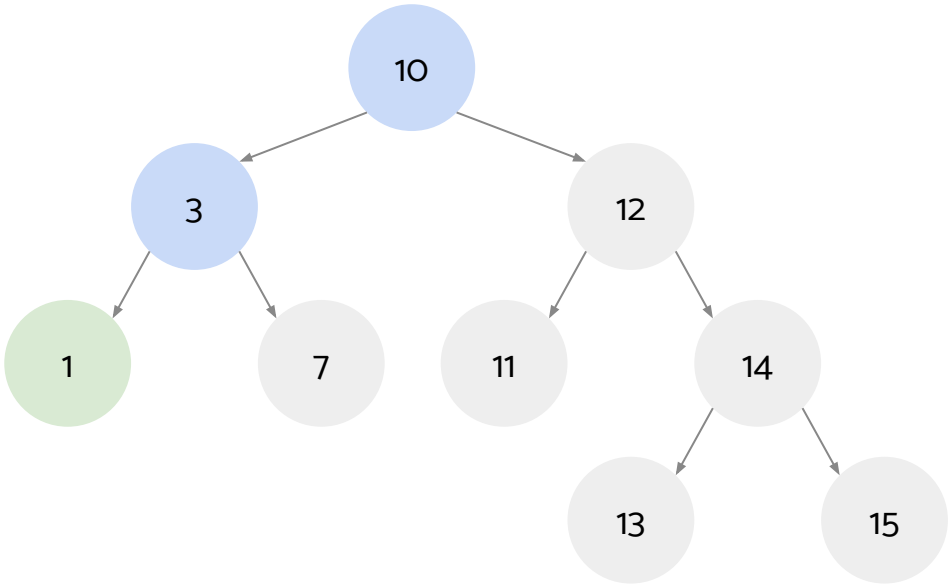


Inorder:

BFS:

Stack: 10 3 1

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

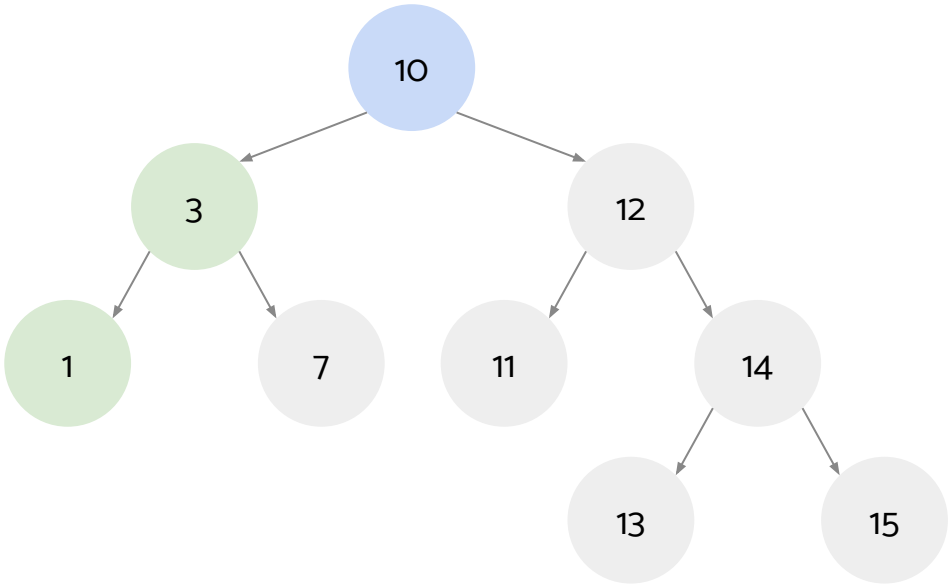


Inorder: 1

BFS:

Stack: 10 3

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

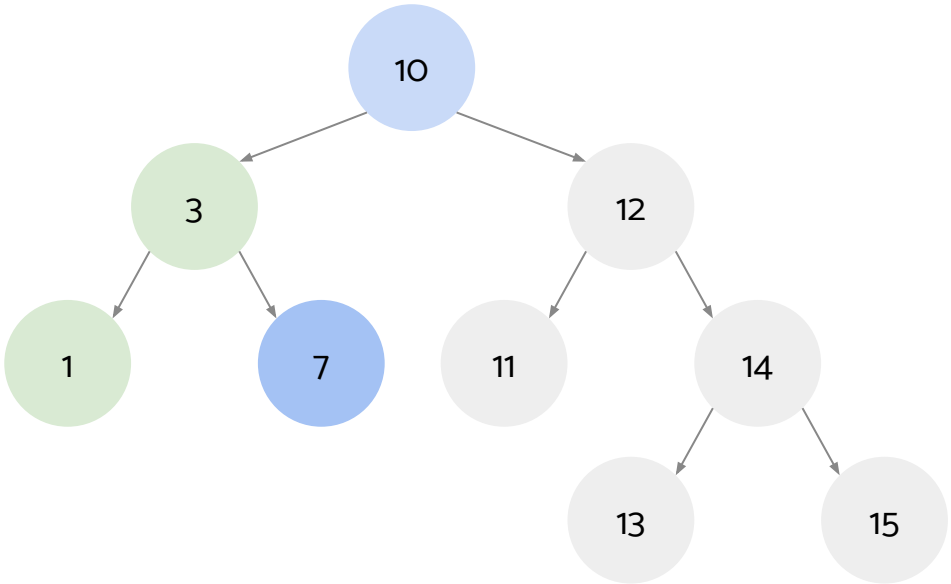


Inorder: 1 3

BFS:

Stack: 10

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

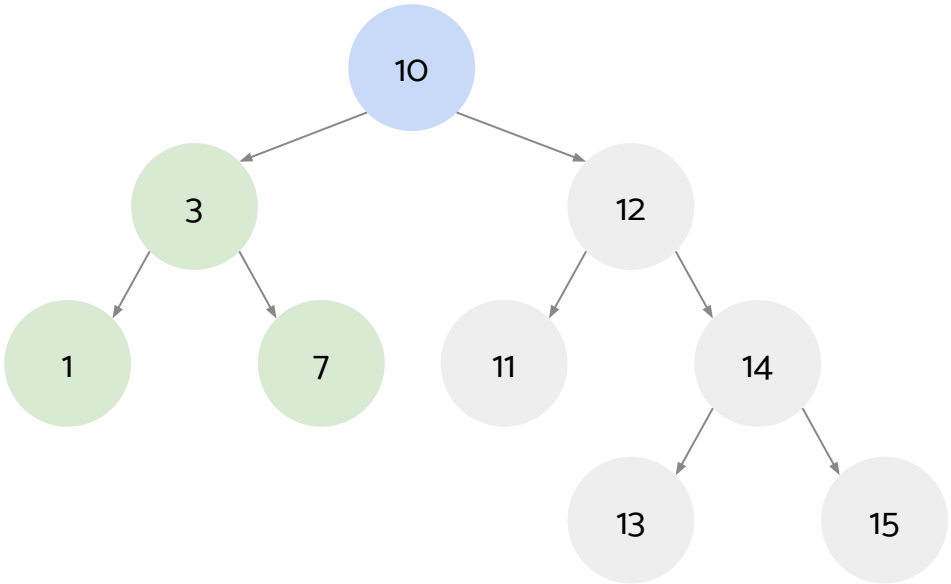


Inorder: 1 3

BFS:

Stack: 10 7

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

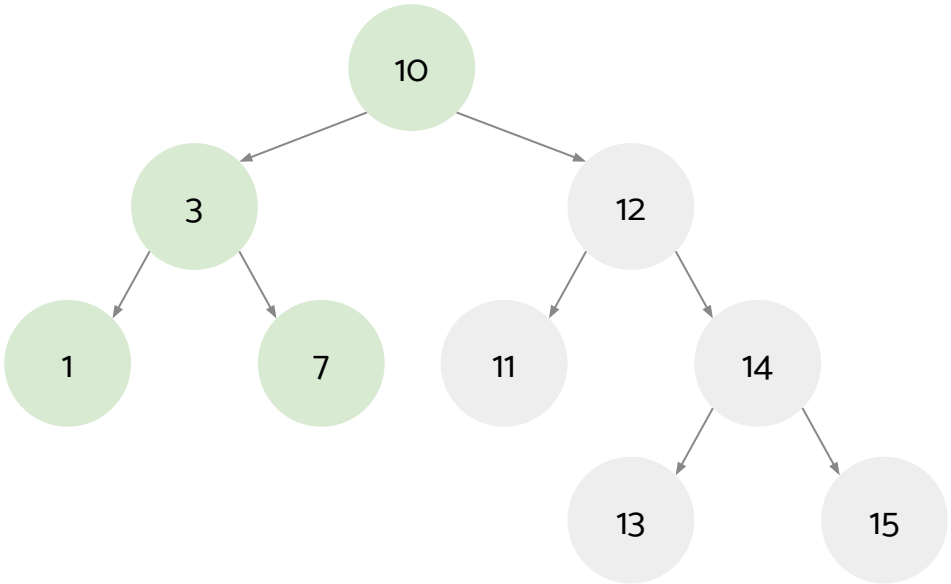


Inorder: 1 3 7

BFS:

Stack: 10

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

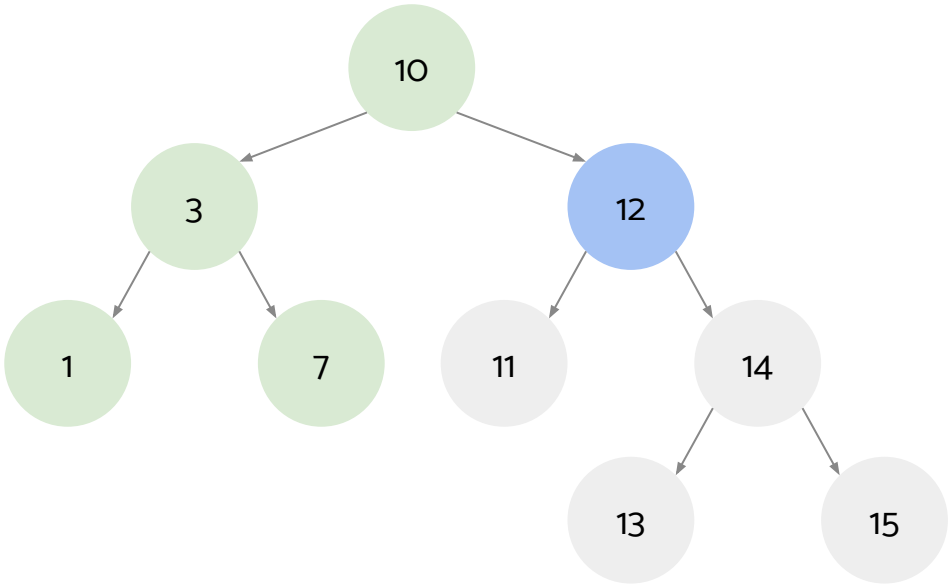


Inorder: 1 3 7 10

BFS:

Stack:

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

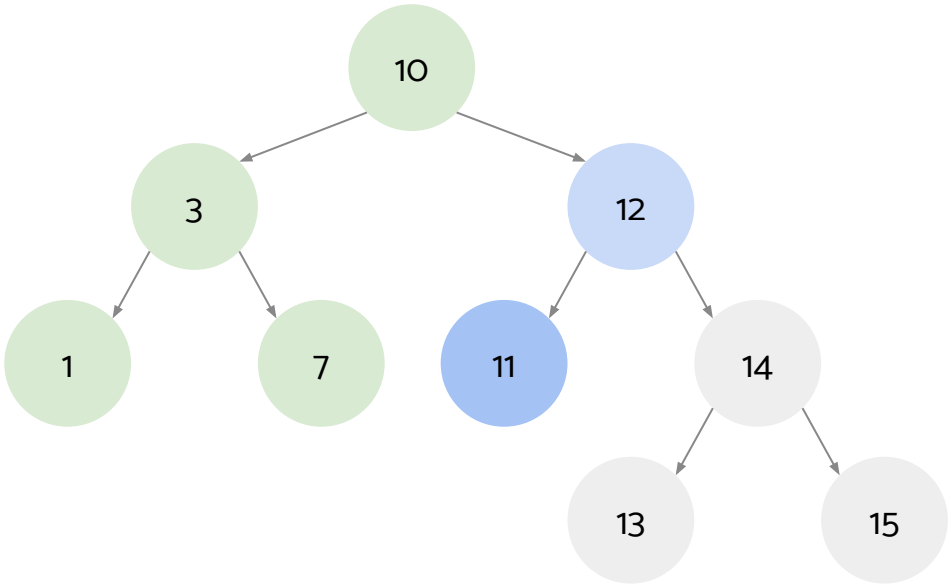


Inorder: 1 3 7 10

BFS:

Stack: 12

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

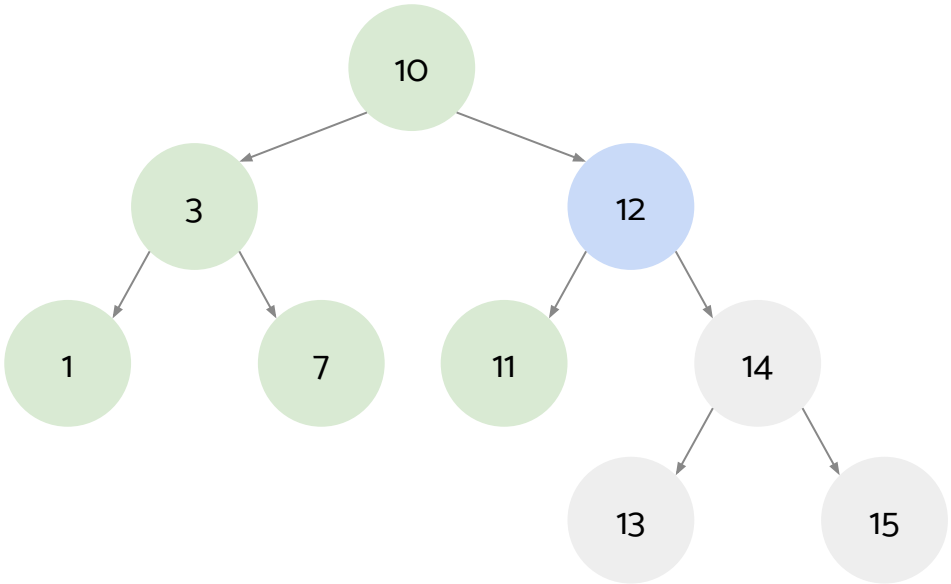


Inorder: 1 3 7 10

BFS:

Stack: 12 11

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

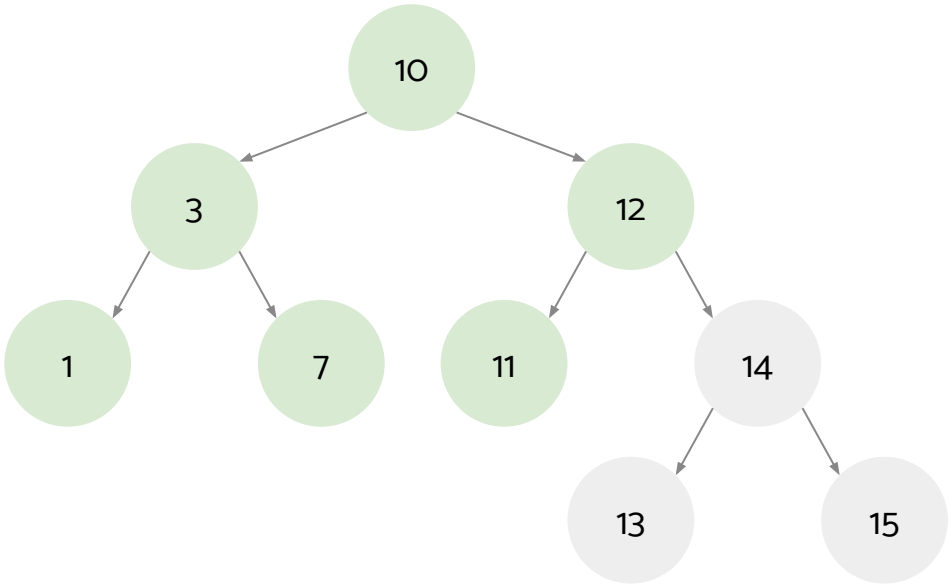


Inorder: 1 3 7 10 11

BFS:

Stack: 12

1a Trees, Graphs, and Traversals, Oh My! Write out the inorder and BFS traversals of this BST.

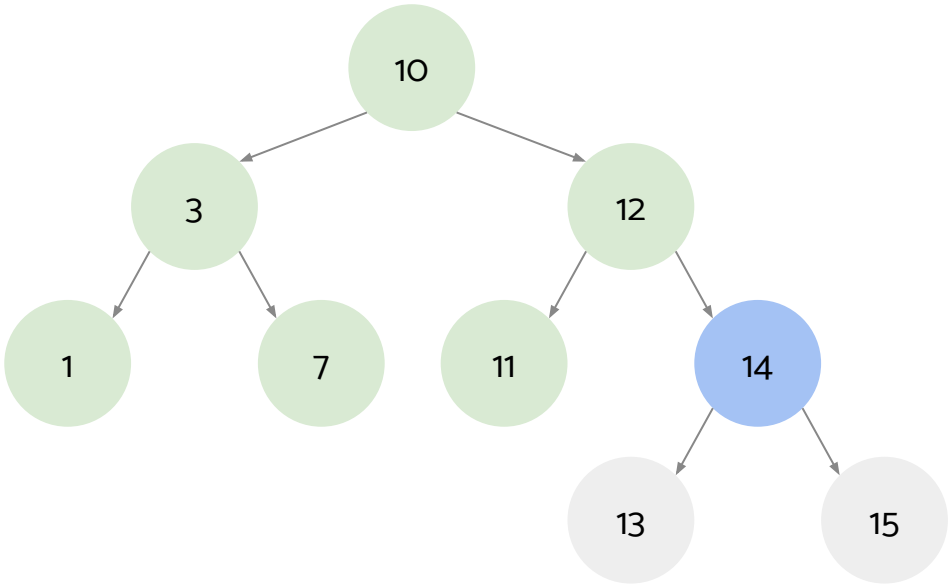


Inorder: 1 3 7 10 11 12

BFS:

Stack:

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

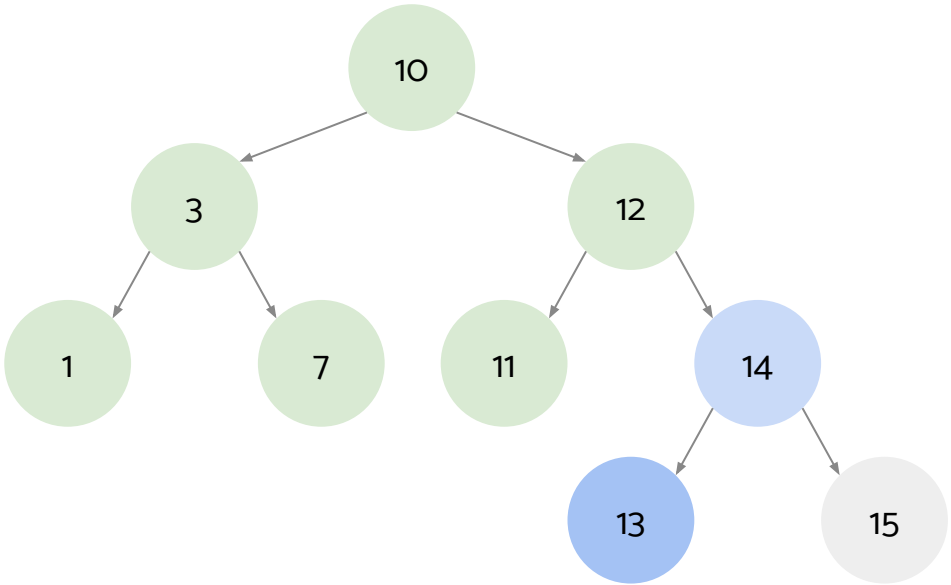


Inorder: 1 3 7 10 11 12

BFS:

Stack: 14

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

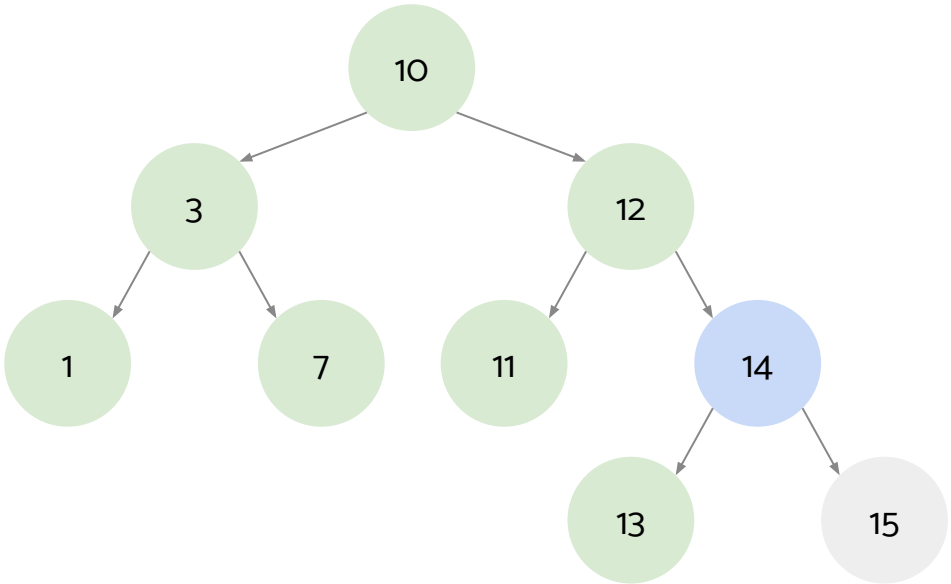


Inorder: 1 3 7 10 11 12

BFS:

Stack: 14 13

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

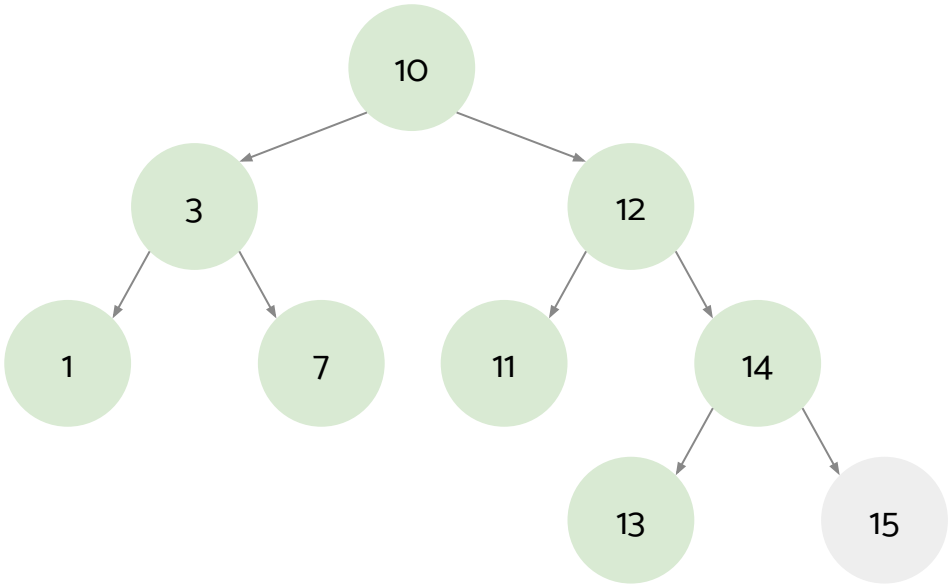


Inorder: 1 3 7 10 11 12 13

BFS:

Stack: 14

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

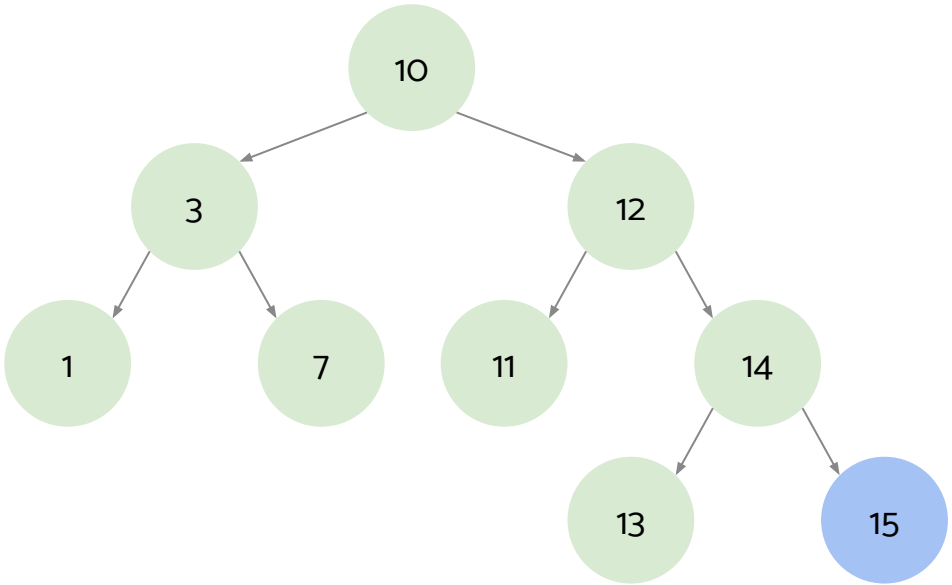


Inorder: 1 3 7 10 11 12 13 14

BFS:

Stack: 14

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

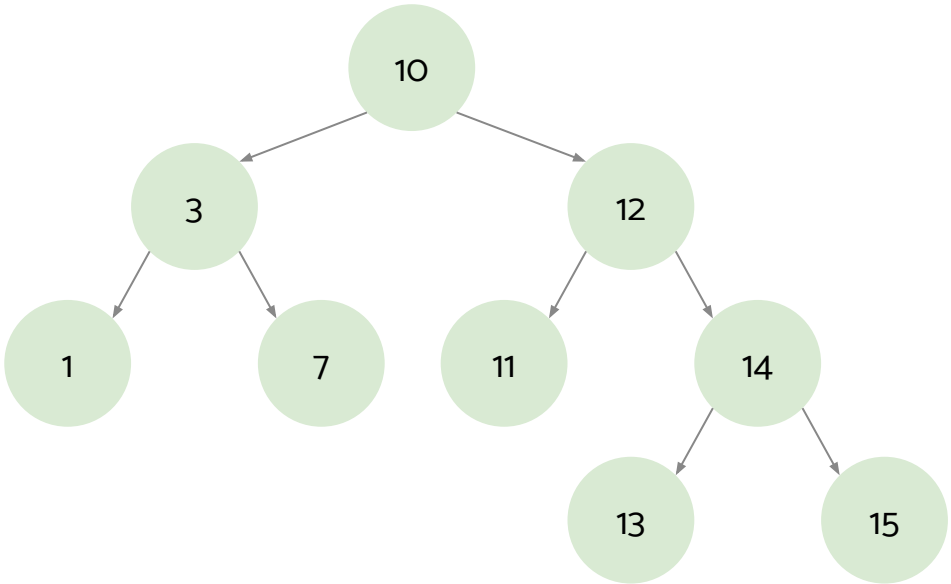


Inorder: 1 3 7 10 11 12 13 14

BFS:

Stack: 15

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

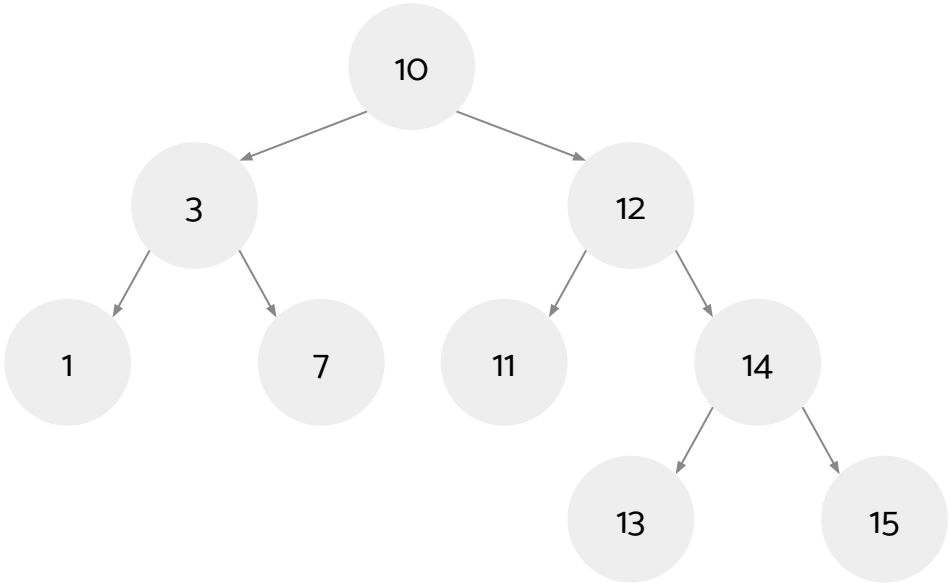


Inorder: 1 3 7 10 11 12 13 14 15

BFS:

Stack:

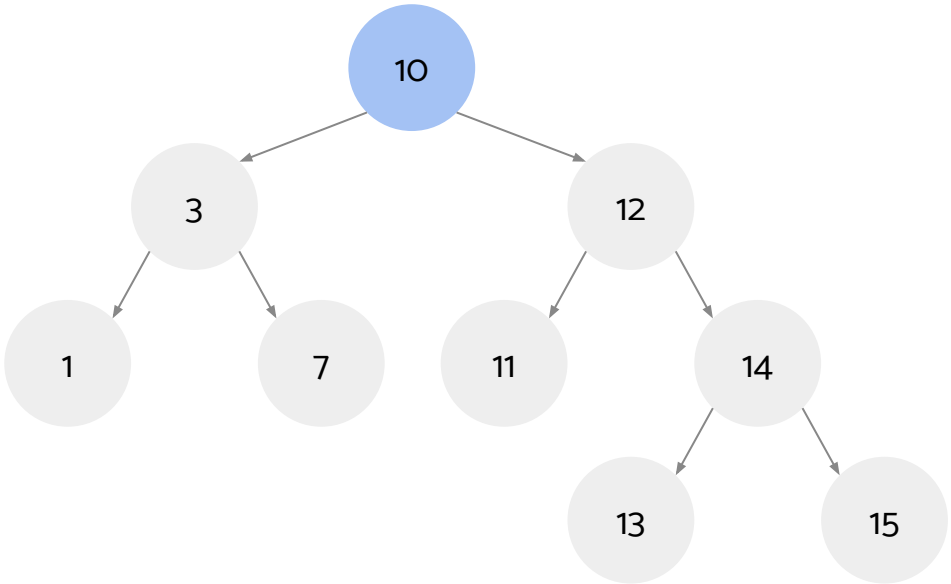
1a Trees, Graphs, and Traversals, Oh My! Write out the inorder and BFS traversals of this BST.



Inorder: 1 3 7 10 11 12 13 14 15

BFS:

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

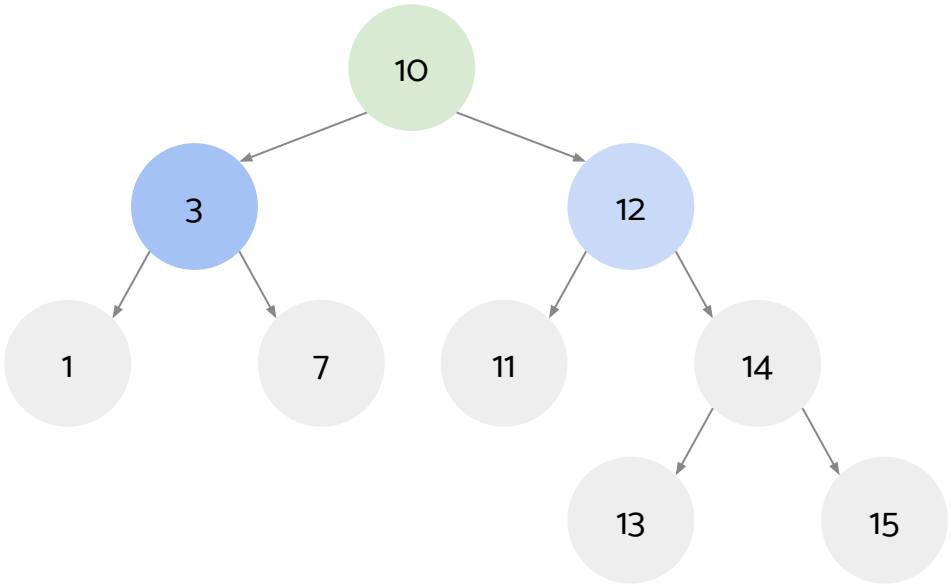


Inorder: 1 3 7 10 11 12 13 14 15

BFS:

Queue: 10

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

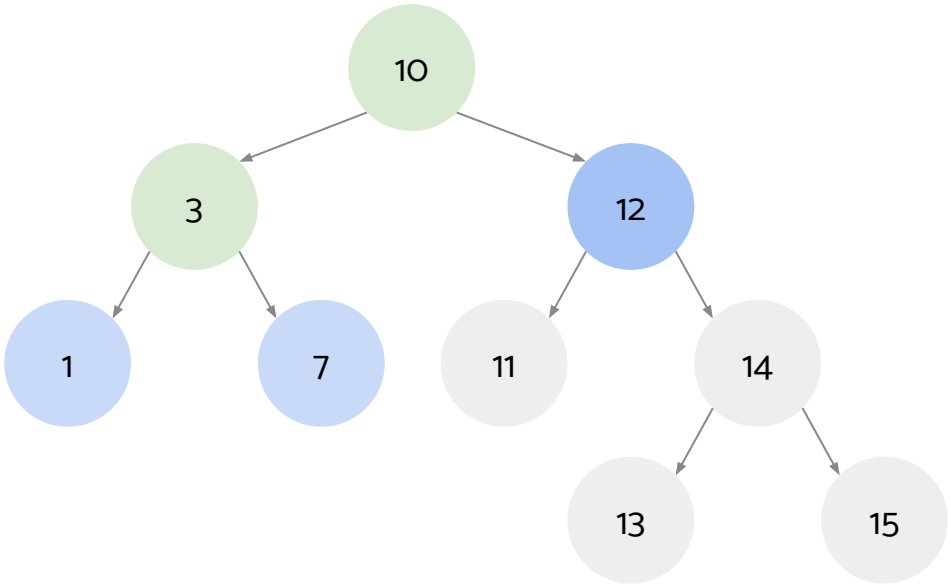


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10

Queue: 3, 12

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

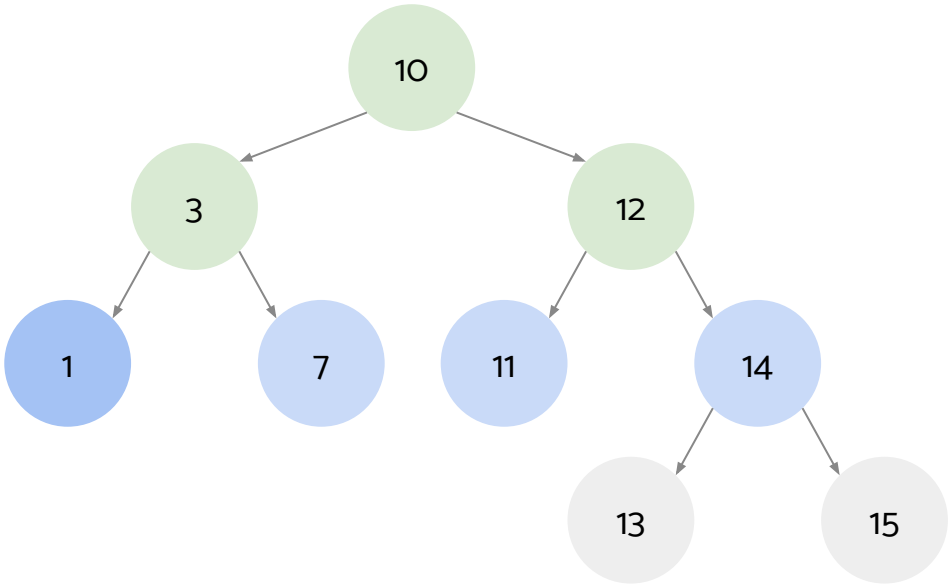


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10 3

Queue: 12, 1, 7

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

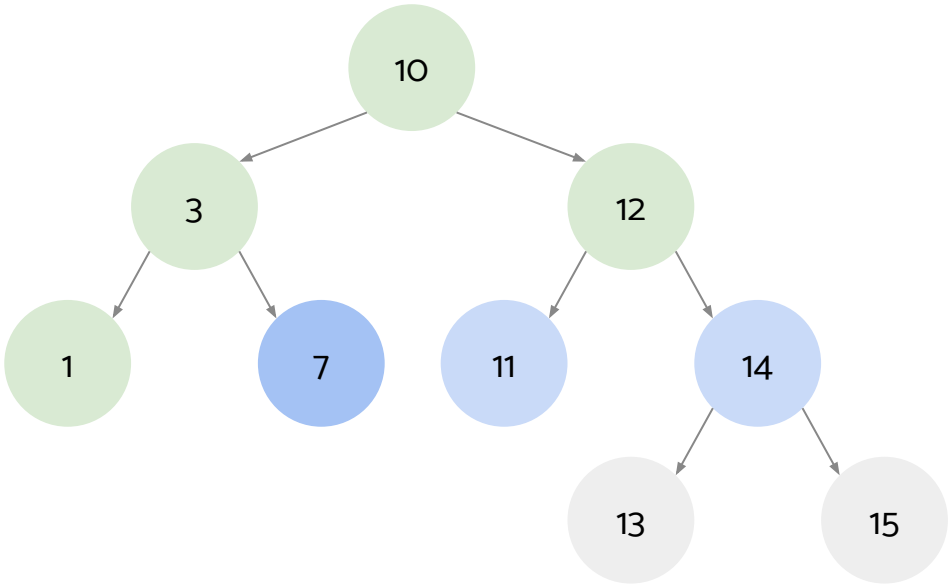


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10 3 12

Queue: 1, 7, 11, 14

1a Trees, Graphs, and Traversals, Oh My! Write out the inorder and BFS traversals of this BST.

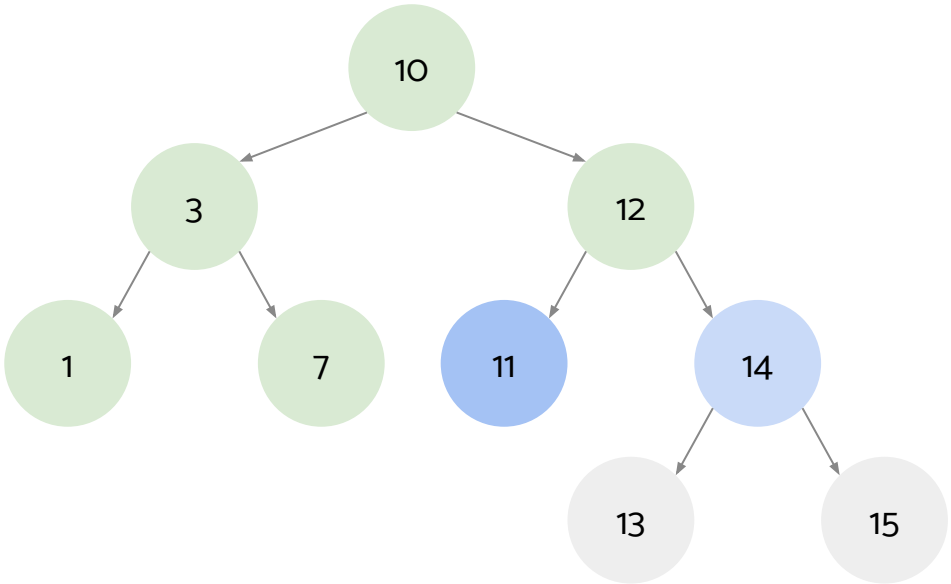


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10 3 12 1

Queue: 7, 11, 14

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

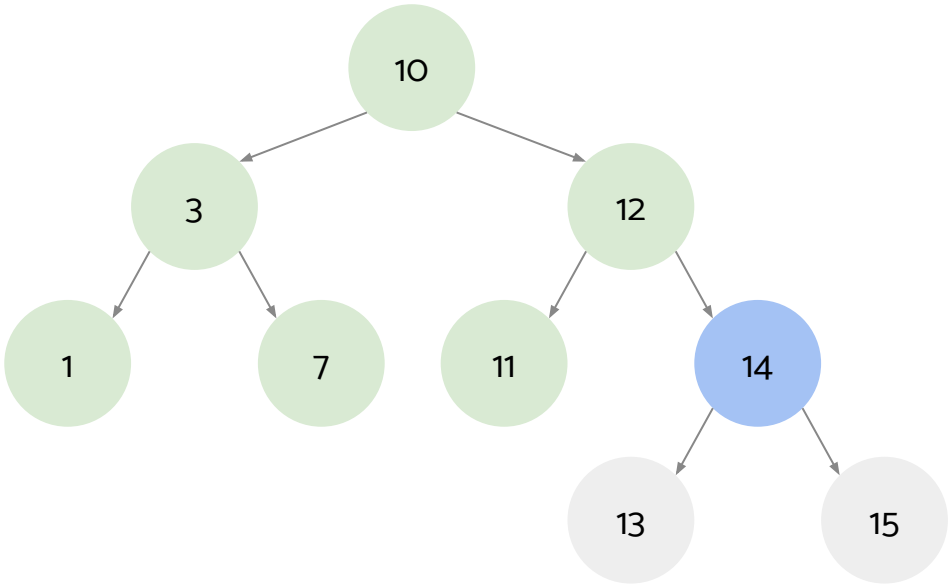


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10 3 12 1 7

Queue: 11, 14

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

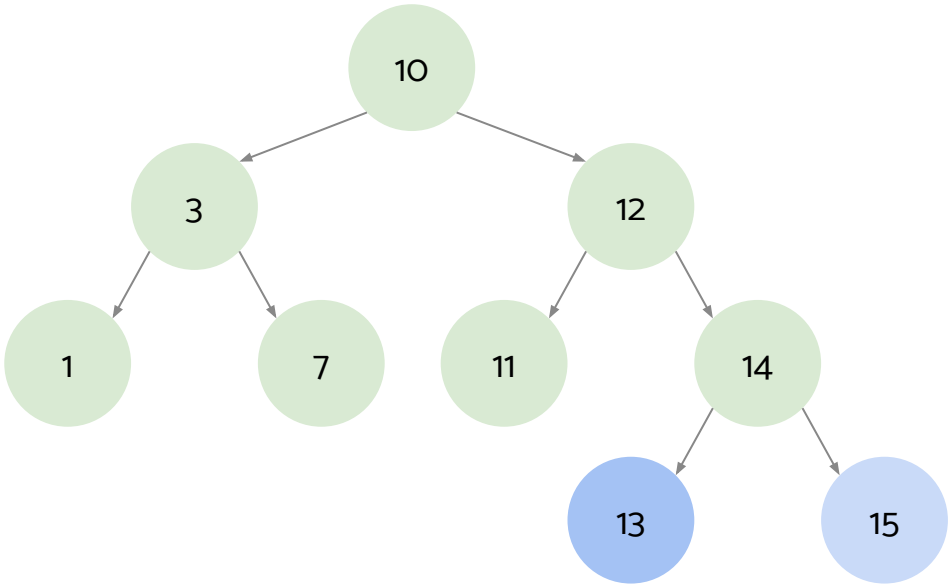


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10 3 12 1 7 11

Queue: 14

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

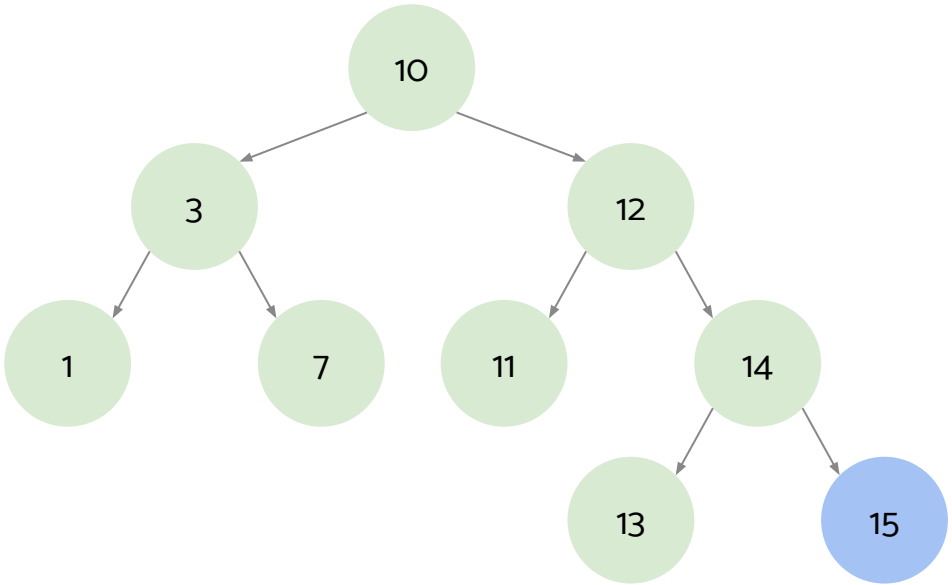


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10 3 12 1 7 11 14

Queue: 13, 15

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

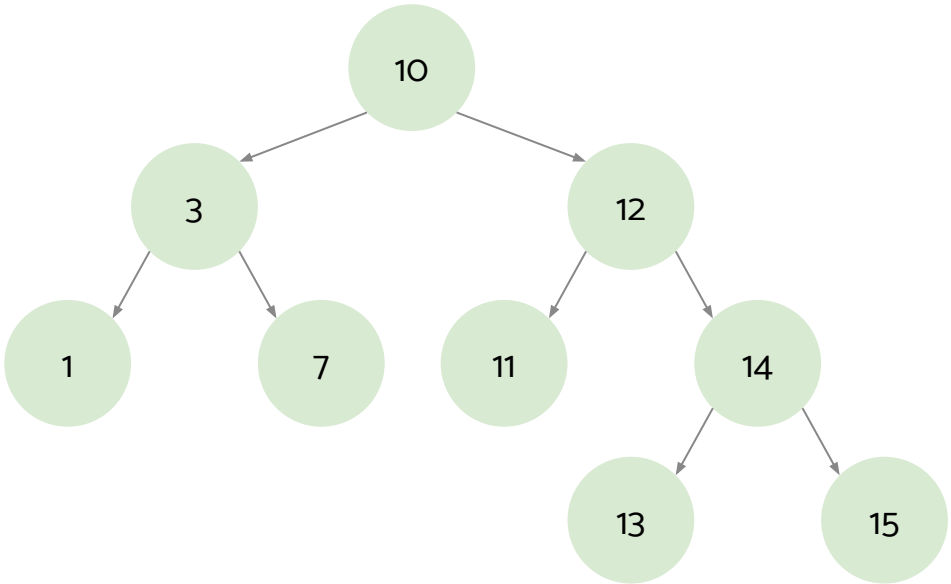


Inorder: 1 3 7 10 11 12 13 14 15

BFS: 10 3 12 1 7 11 14 13

Queue: 15

1a Trees, Graphs, and Traversals, Oh My! **Write out the inorder and BFS traversals of this BST.**

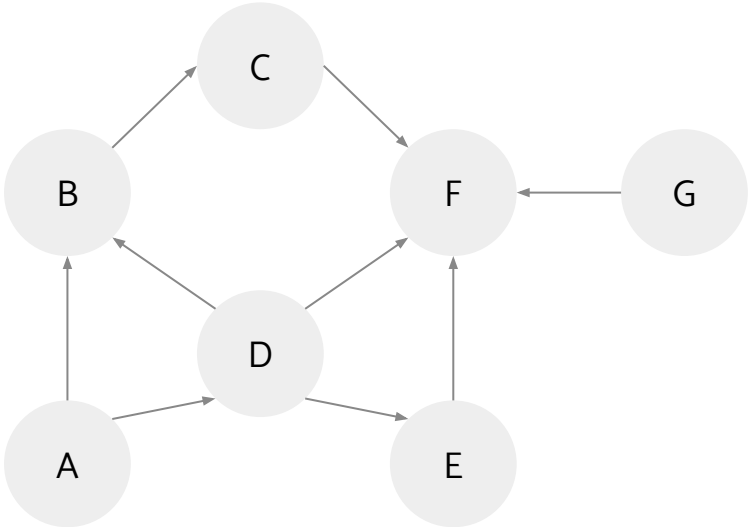


Inorder: 1 3 7 10 11 12 13 14 15

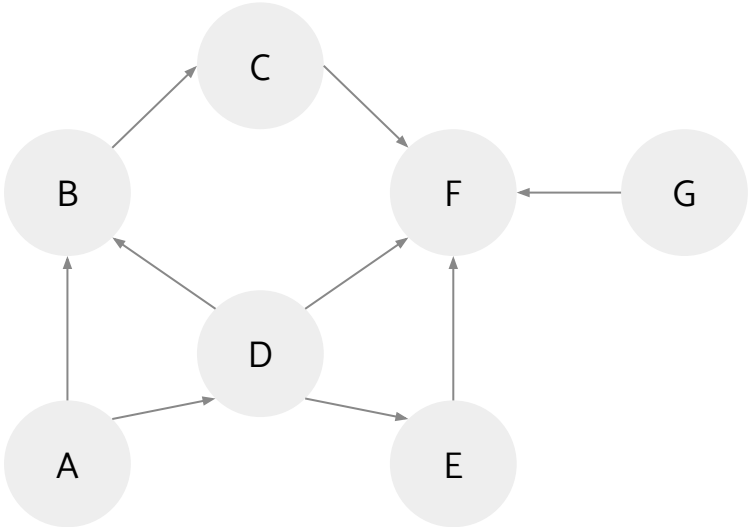
BFS: 10 3 12 1 7 11 14 13 15

Queue:

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.



1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.

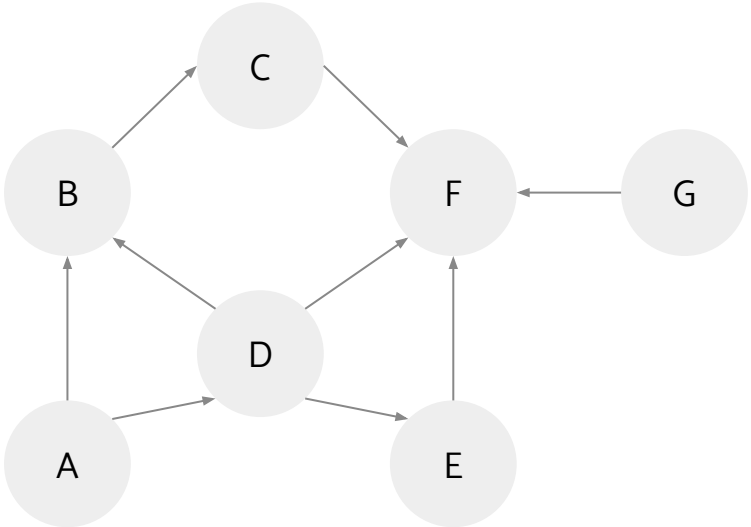


	A	B	C	D	E	F	G
A							
B							
C							
D							
E							
F							
G							

To

From

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.

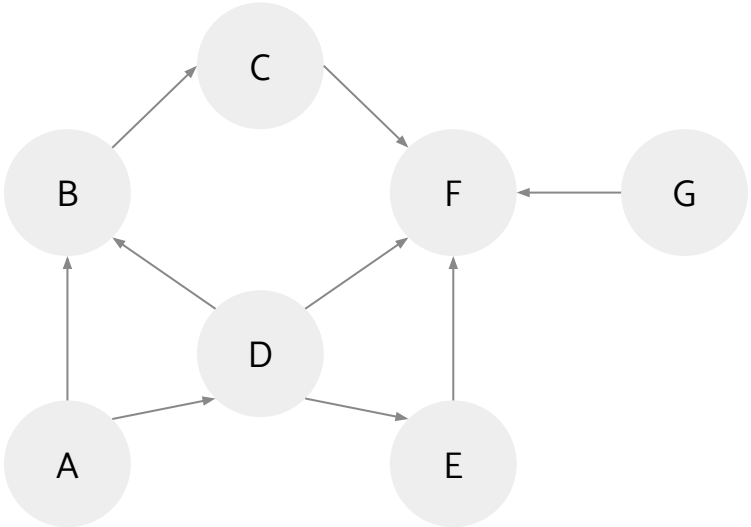


	A	B	C	D	E	F	G
A		✓		✓			
B							
C							
D							
E							
F							
G							

To

From

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.

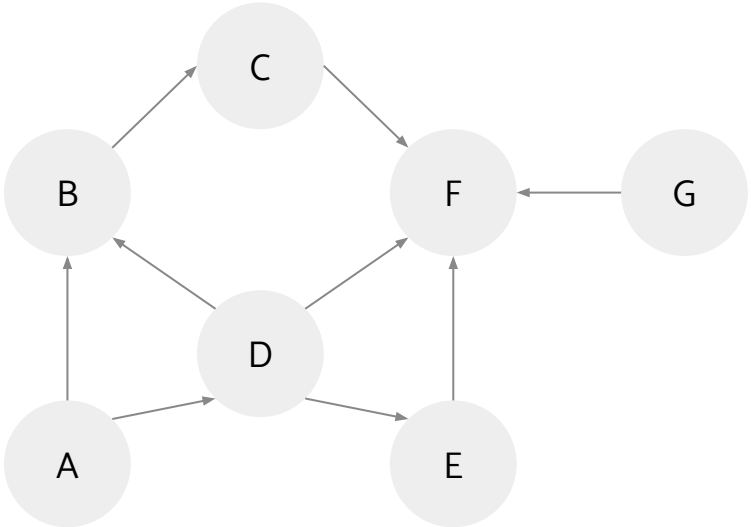


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C							
D							
E							
F							
G							

To

From

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.

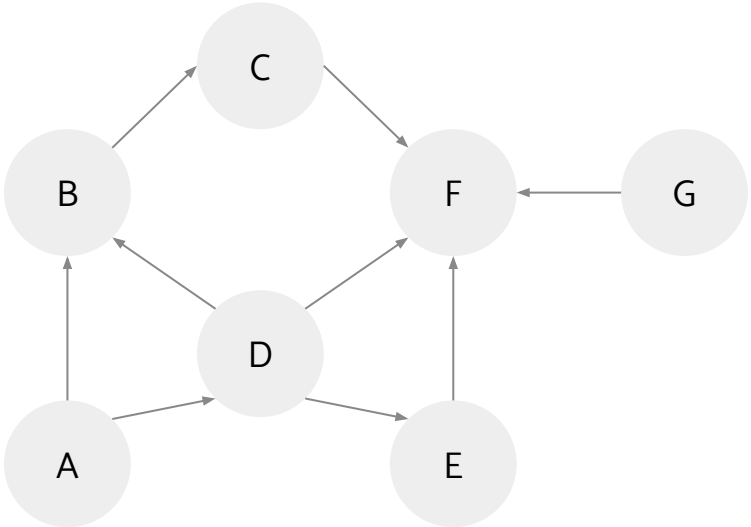


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C						✓	
D							
E							
F							
G							

To

From

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.

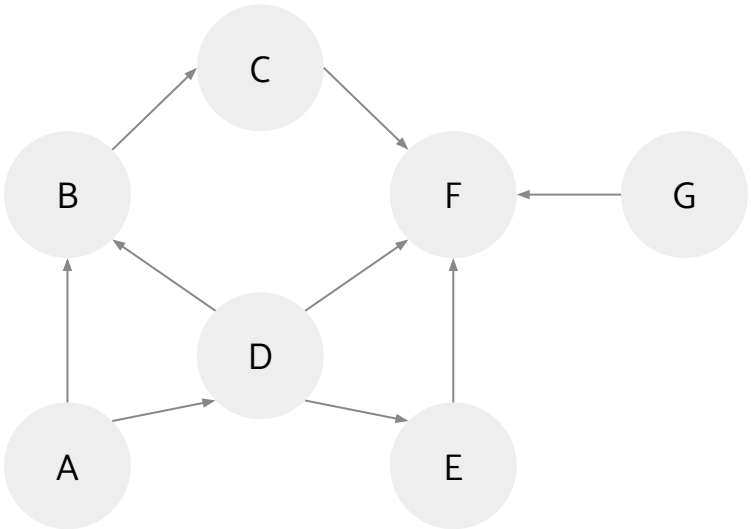


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C						✓	
D		✓			✓	✓	
E							
F							
G							

To

From

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.

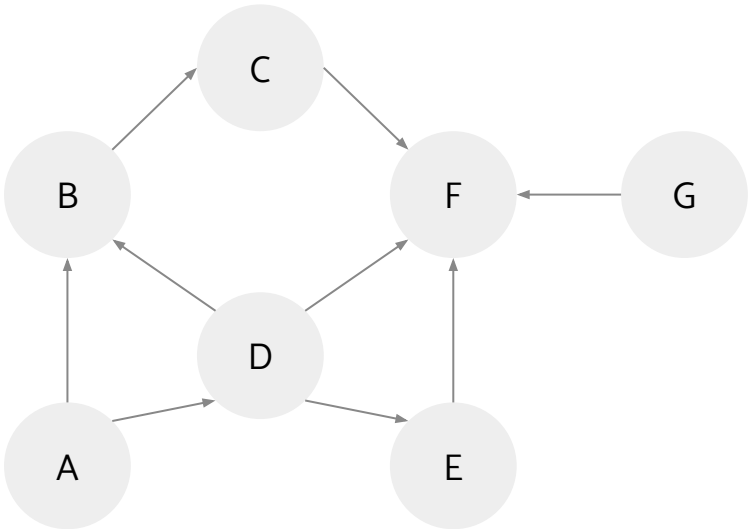


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C						✓	
D		✓			✓	✓	
E						✓	
F							
G							

To

From

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.

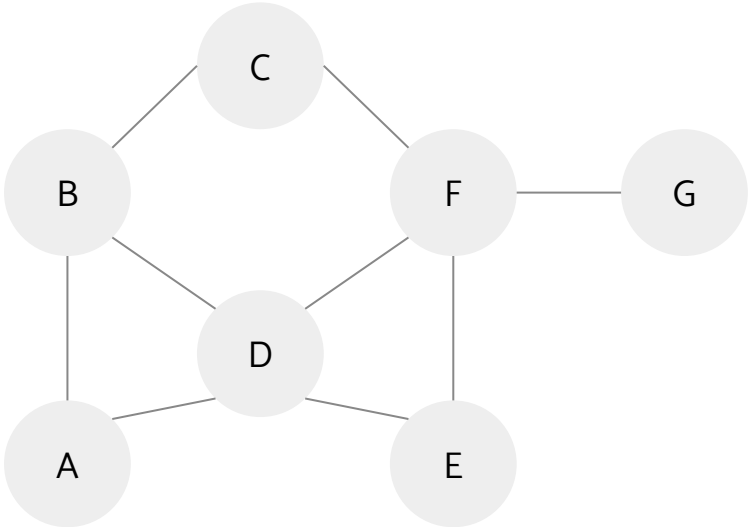


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C						✓	
D		✓			✓	✓	
E						✓	
F							
G						✓	

To

From

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list. What if the graph is undirected?

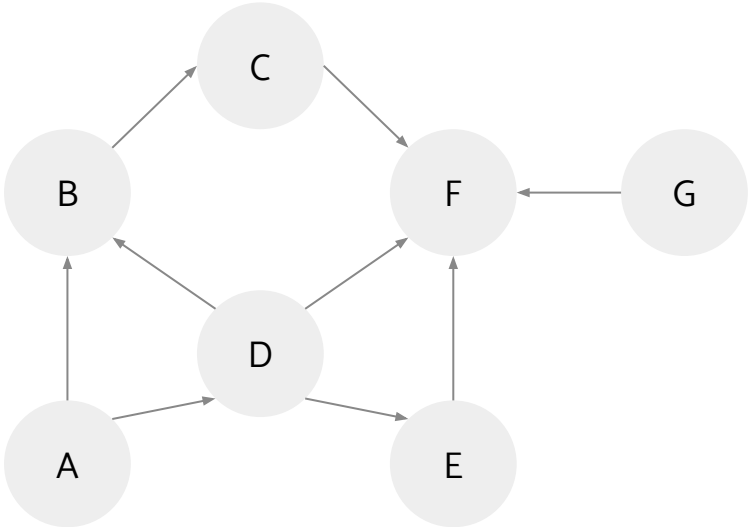


	A	B	C	D	E	F	G
A		✓		✓			
B	✓		✓	✓			
C		✓				✓	
D	✓	✓			✓	✓	
E				✓		✓	
F			✓	✓	✓		✓
G						✓	

To

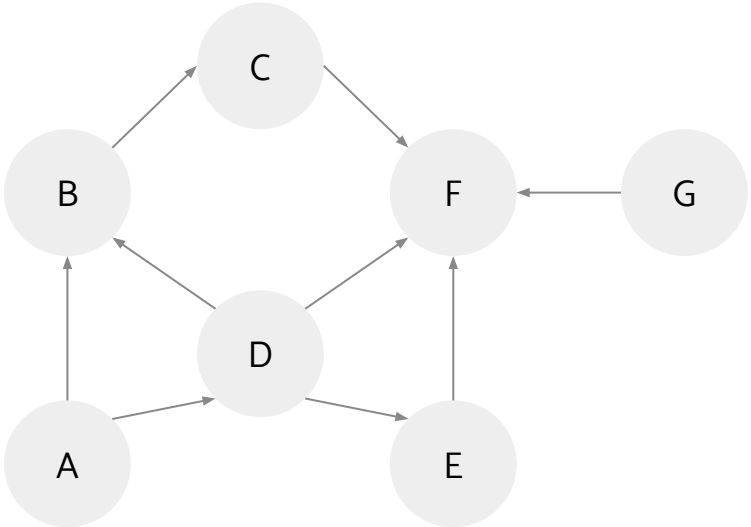
From

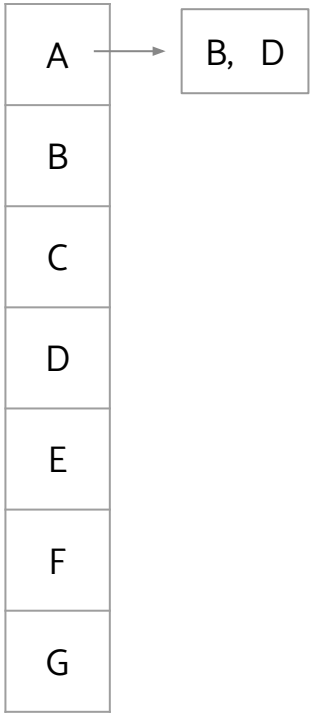
1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.



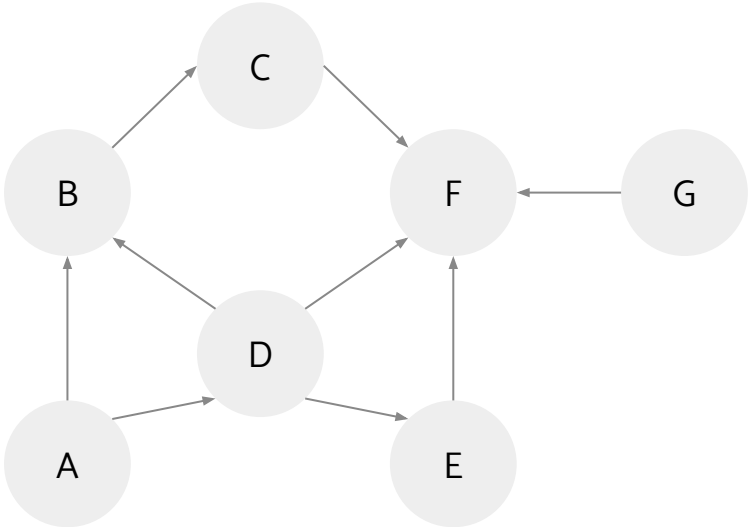
A
B
C
D
E
F
G

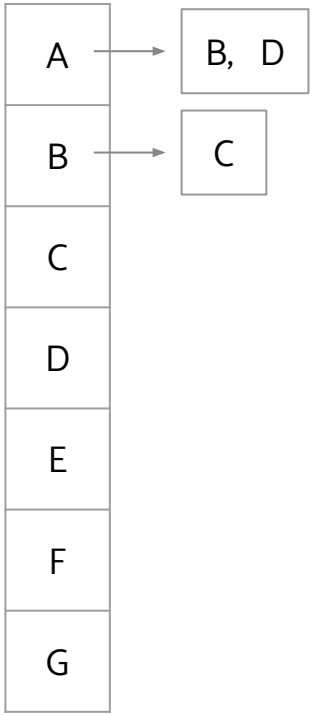
1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.



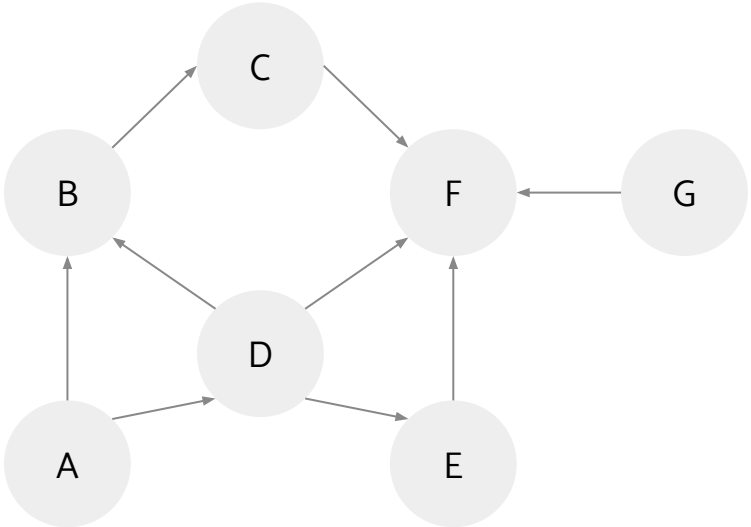


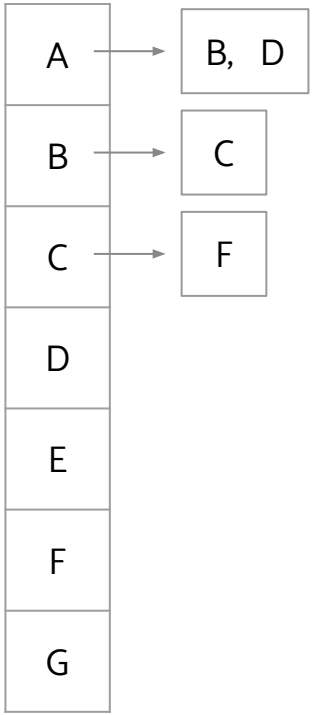
1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.



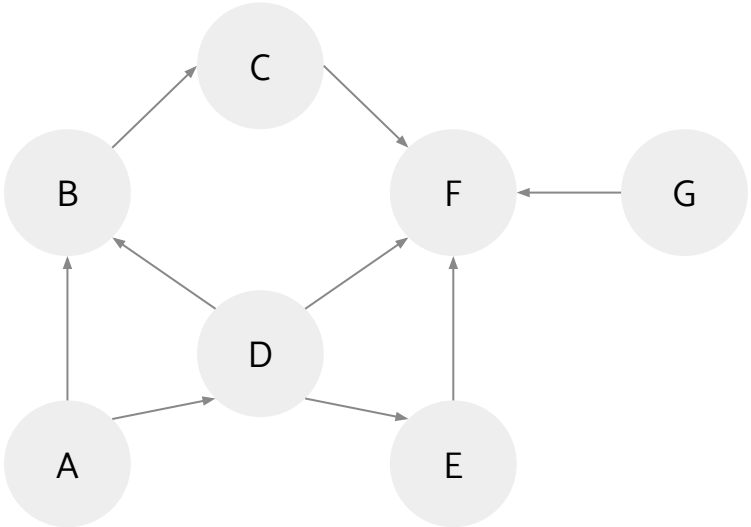


1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.



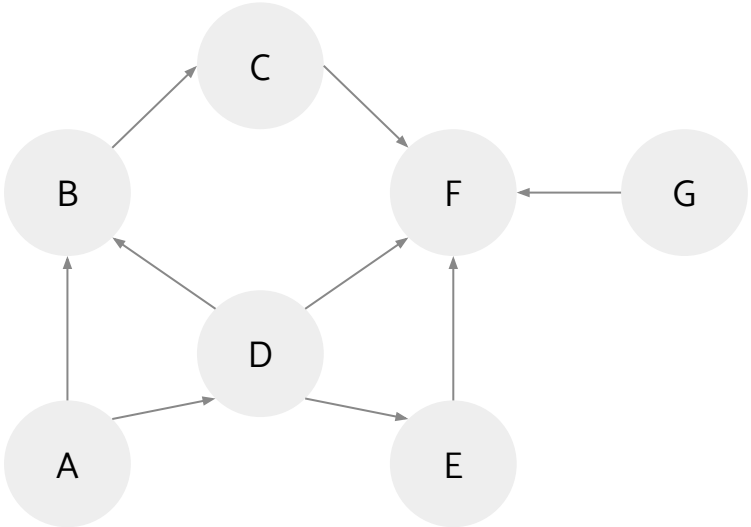


1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list.



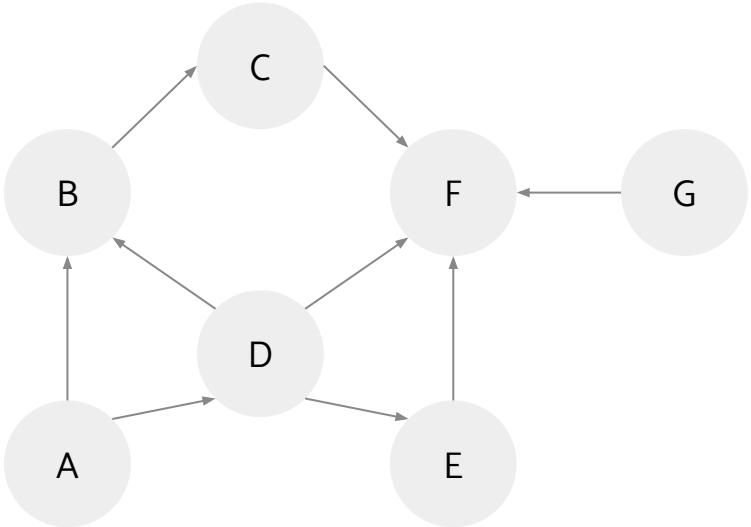
A	B, D
B	C
C	F
D	B, F, E
E	F
F	
G	F

1b Trees, Graphs, and Traversals, Oh My! Write out the adjacency matrix and adjacency list. What if the graph is undirected?



A	B, D
B	A, C, D
C	B, F
D	A, B, E, F
E	D, F
F	C, D, E, G
G	F

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

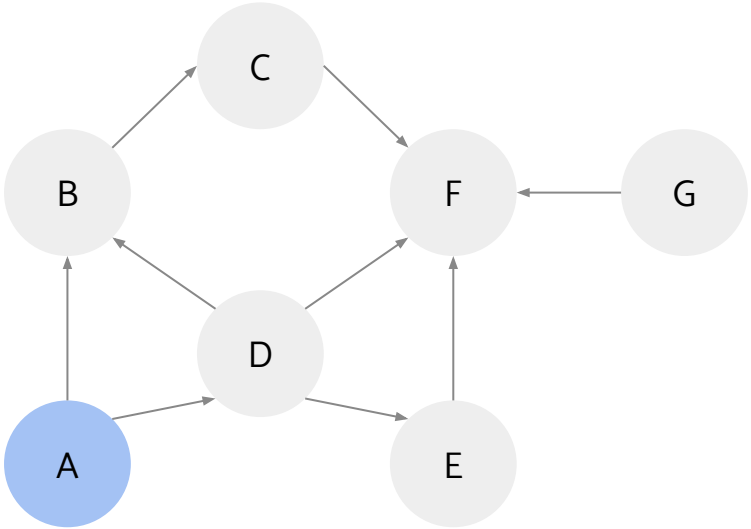


DFS Pre-Order:

DFS Post-Order:

Stack:

1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.

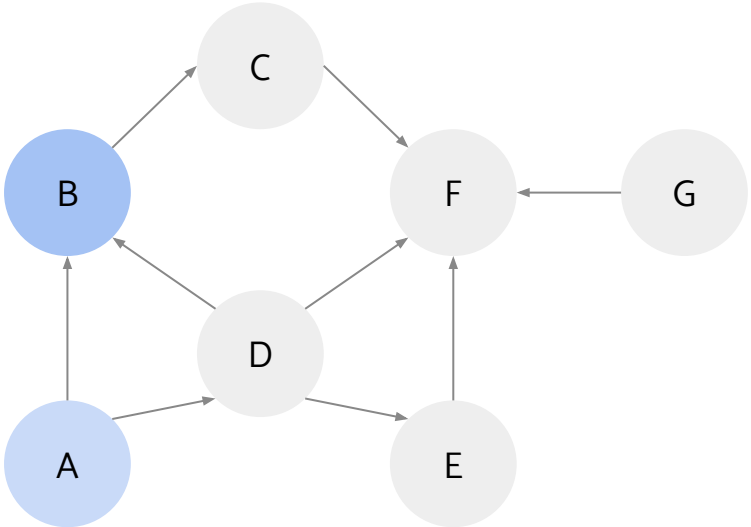


DFS Pre-Order:
A

DFS Post-Order:

Stack: A

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

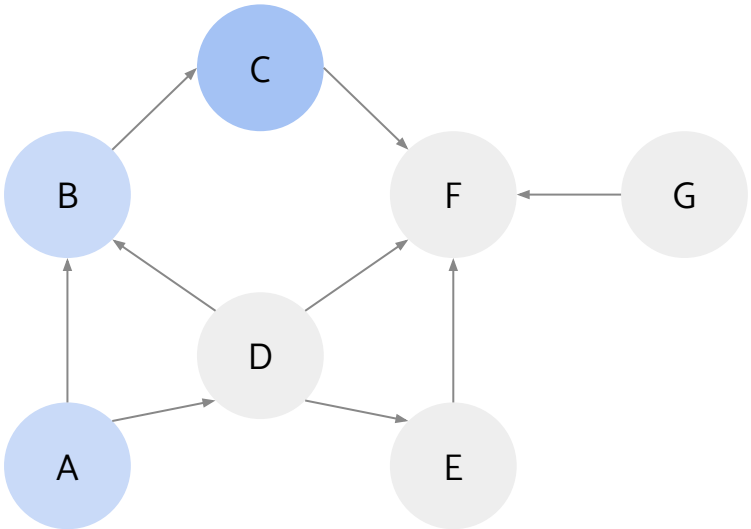


DFS Pre-Order:
A, B

DFS Post-Order:

Stack: A, B

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

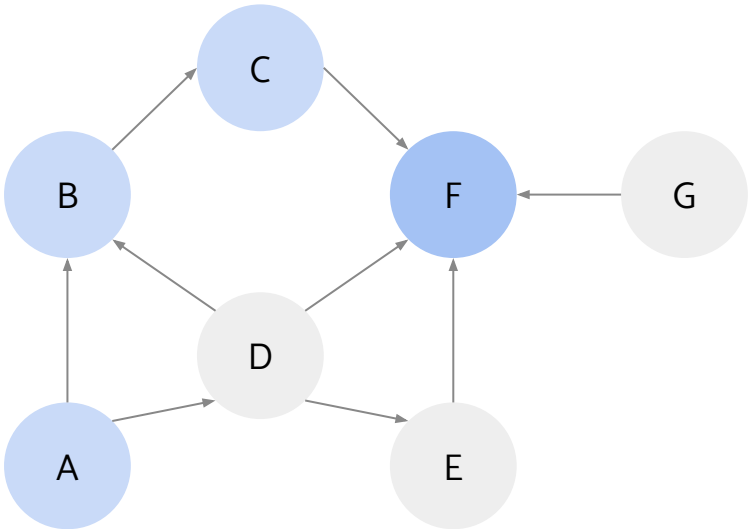


DFS Pre-Order:
A, B, C

DFS Post-Order:

Stack: A, B, C

1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.

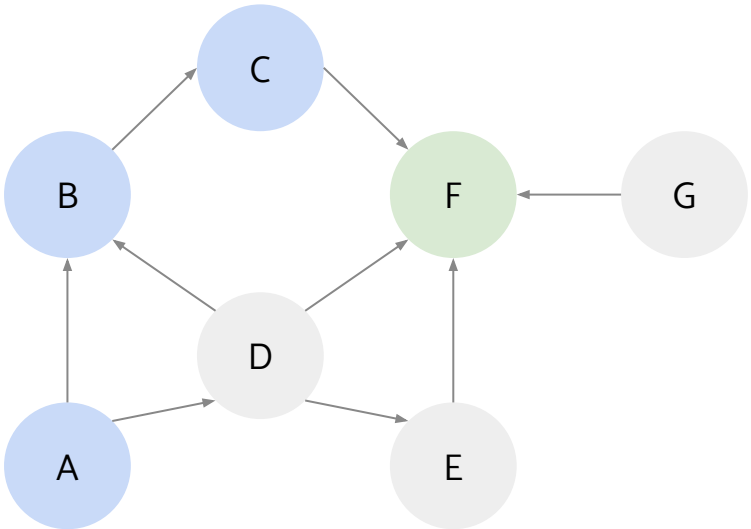


DFS Pre-Order:
A, B, C, F

DFS Post-Order:

Stack: A, B, C, F

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

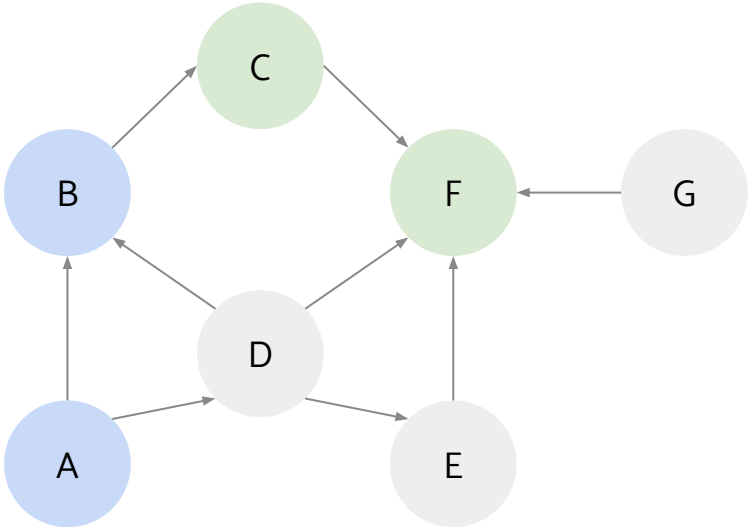


DFS Pre-Order:
A, B, C, F

DFS Post-Order:
F

Stack: A, B, C

1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.

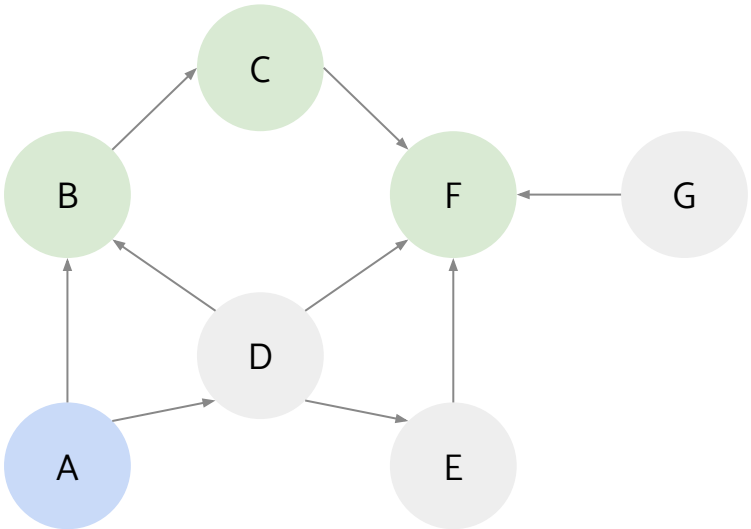


DFS Pre-Order:
A, B, C, F

DFS Post-Order:
F, C

Stack: A, B

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

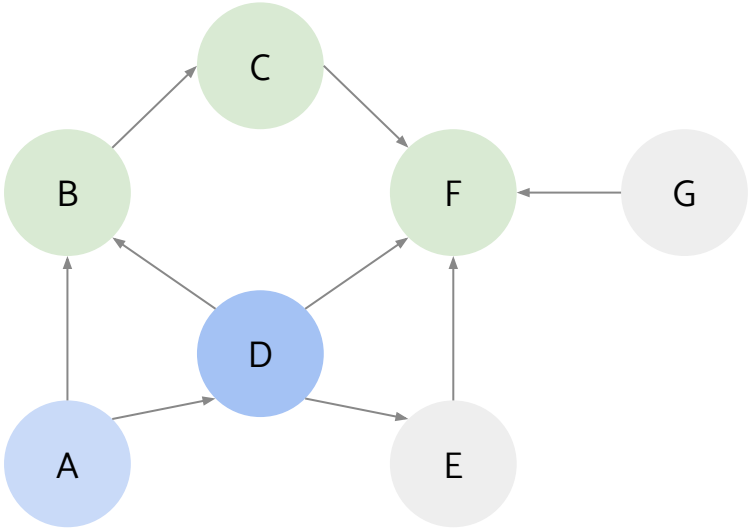


DFS Pre-Order:
A, B, C, F

DFS Post-Order:
F, C, B

Stack: A

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

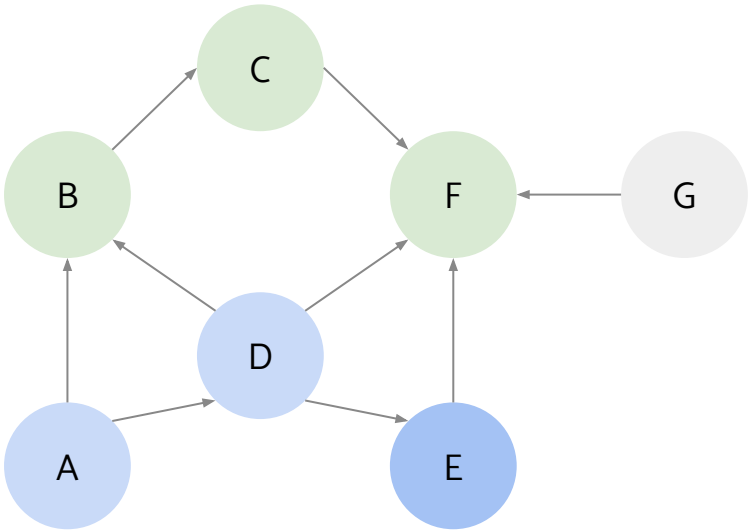


DFS Pre-Order:
A, B, C, F, D

DFS Post-Order:
F, C, B

Stack: A, D

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

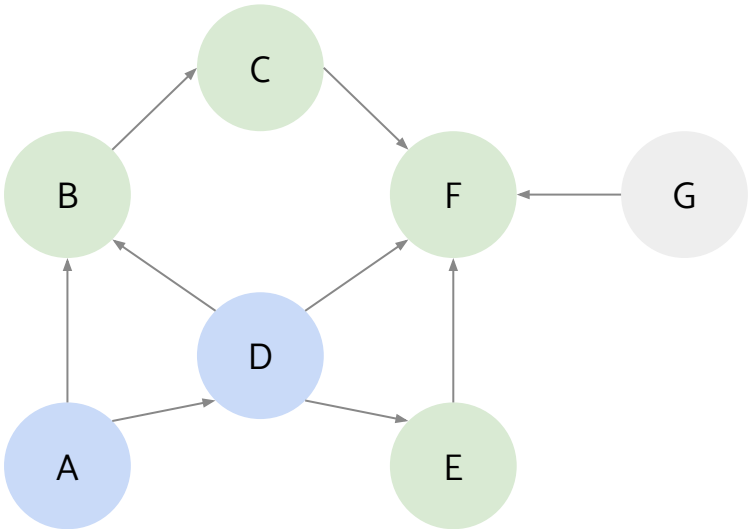


DFS Pre-Order:
A, B, C, F, D, E

DFS Post-Order:
F, C, B,

Stack: A, D, E

1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.

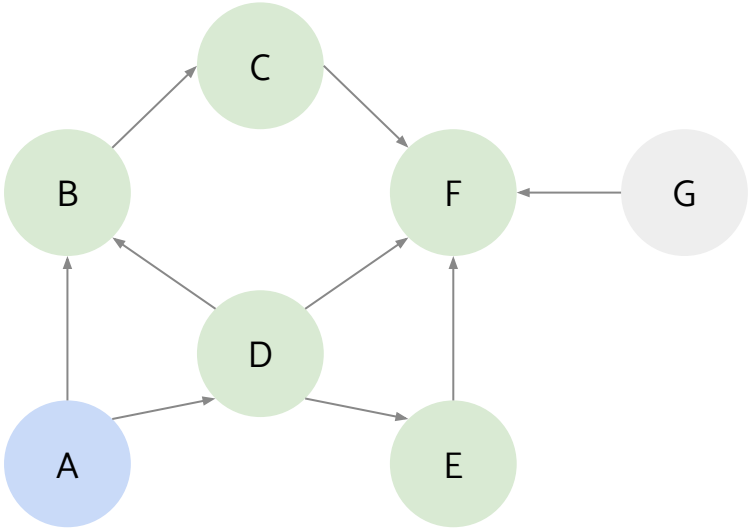


DFS Pre-Order:
A, B, C, F, D, E

DFS Post-Order:
F, C, B, E

Stack: A, D

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

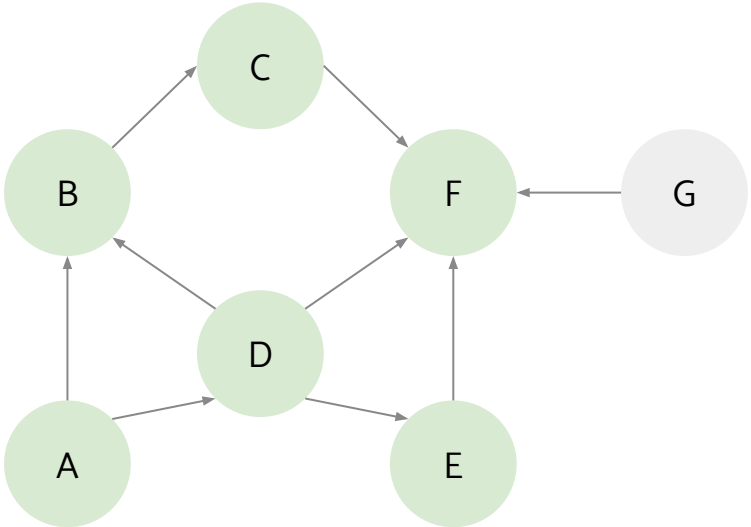


DFS Pre-Order:
A, B, C, F, D, E

DFS Post-Order:
F, C, B, E, D

Stack: A,

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**

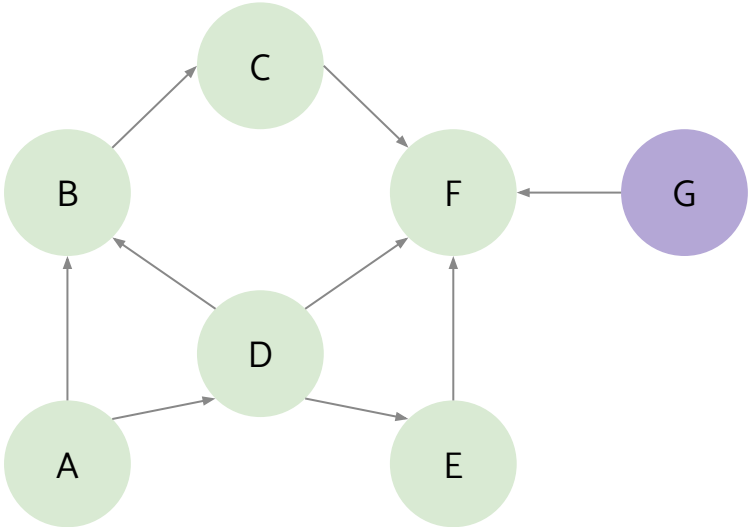


DFS Pre-Order:
A, B, C, F, D, E

DFS Post-Order:
F, C, B, E, D, A

Stack:

1c Trees, Graphs, and Traversals, Oh My! **Write out the order in which nodes are visited by DFS pre-order and post-order.**



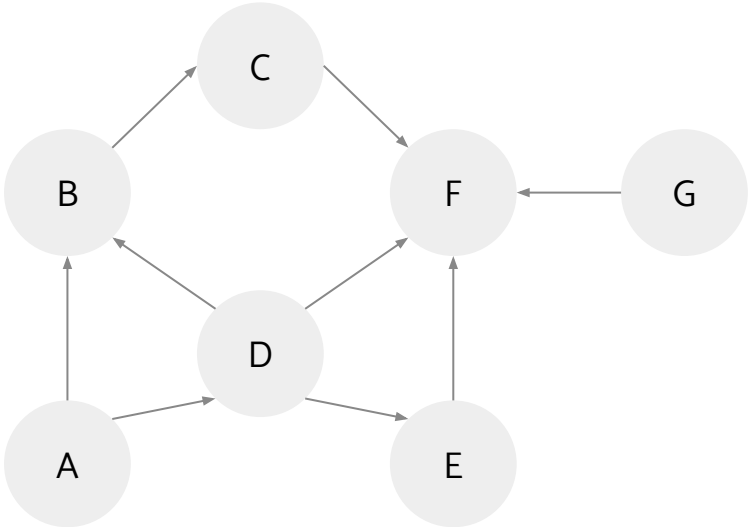
DFS Pre-Order:
A, B, C, F, D, E, G

DFS Post-Order:
F, C, B, E, D, A, G

* if we allow DFS to restart on unmarked nodes, G would be added to the stack (and ultimately the last element in both the preorder and postorder traversals)

Stack:

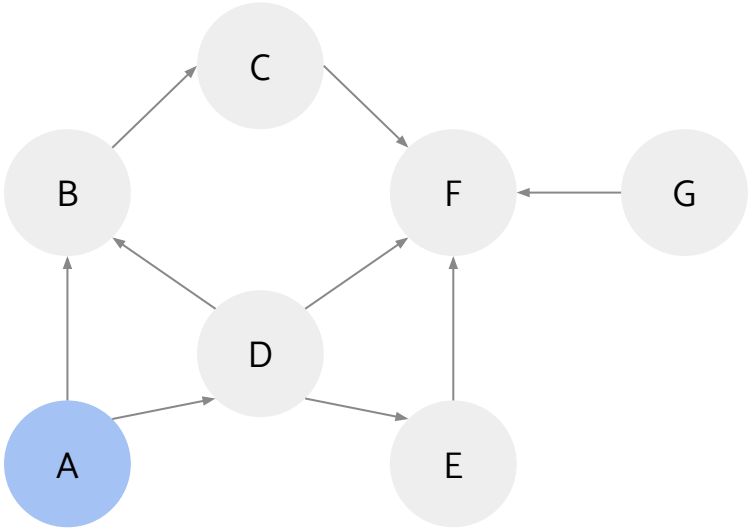
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:

Queue: A

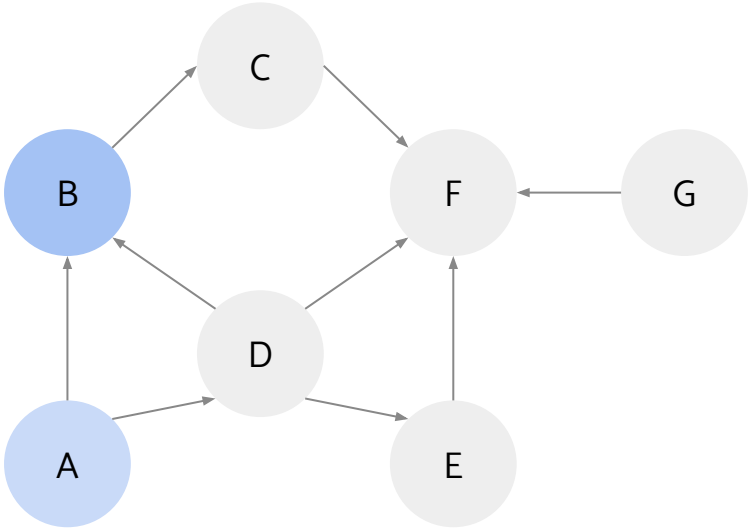
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A

Queue: B D

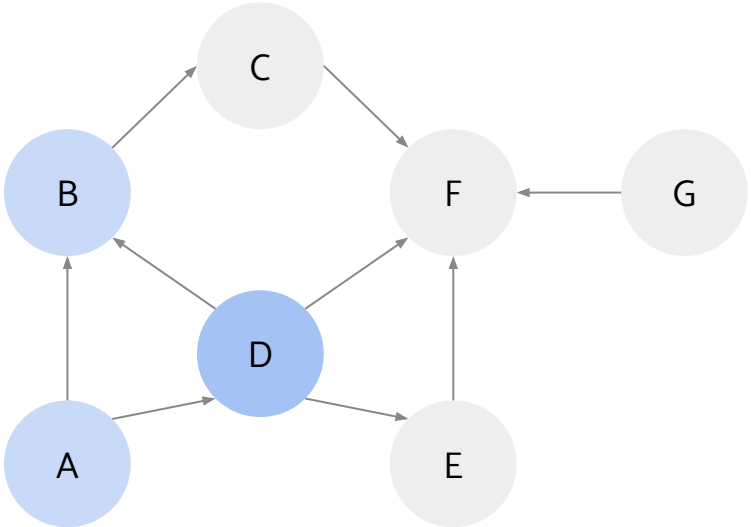
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A B

Queue: D C

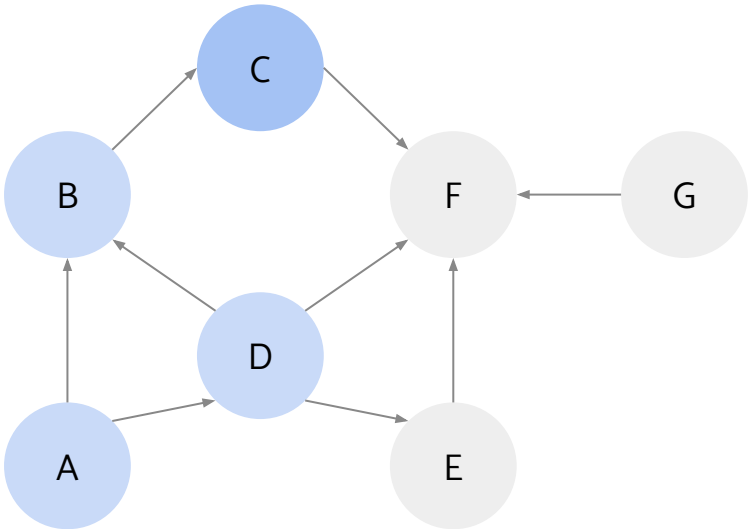
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A B D

Queue: C E F

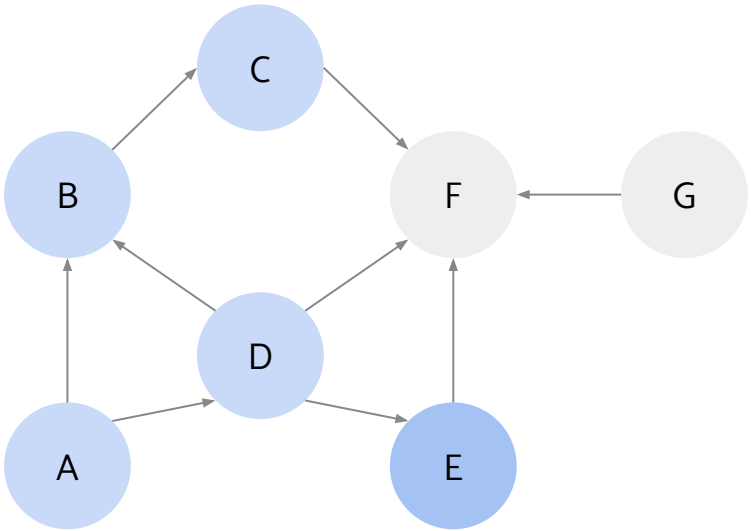
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A B D C

Queue: E F

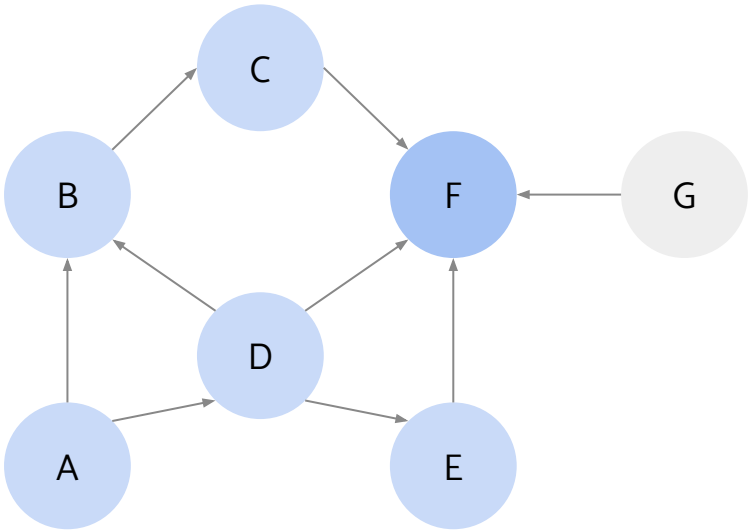
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A B D C E

Queue: F

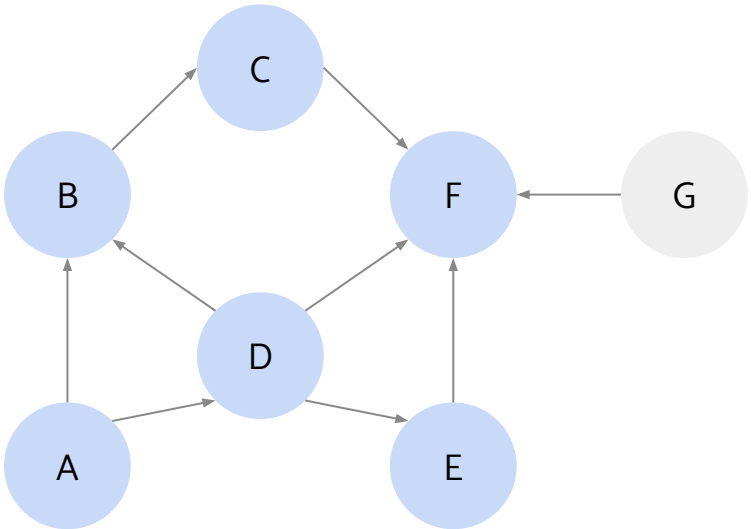
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A B D C E F

Queue:

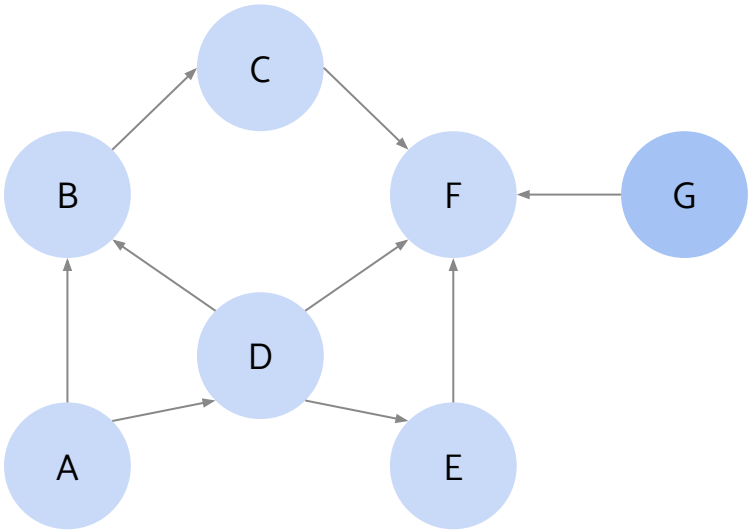
1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A B D C E F

Queue: G

1c Trees, Graphs, and Traversals, Oh My! Write out the order in which nodes are visited by DFS pre-order and post-order.



BFS:
A B D C E F G

Queue:

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

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

Underlying array: [-]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

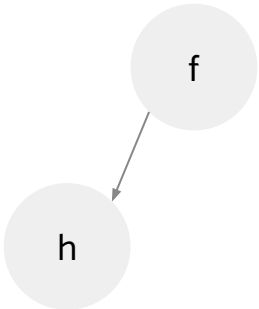
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```

f

Underlying array: [- f]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

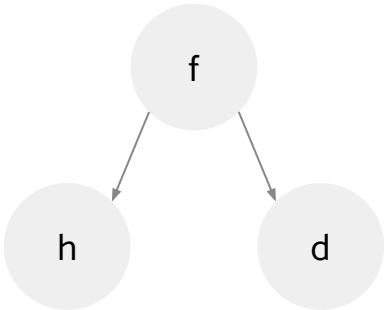
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- f h]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

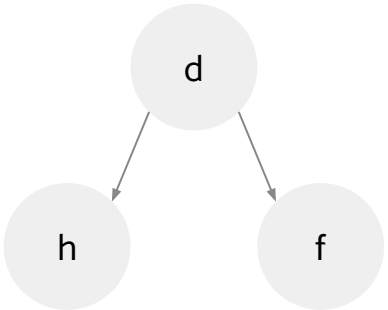
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- f h d]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

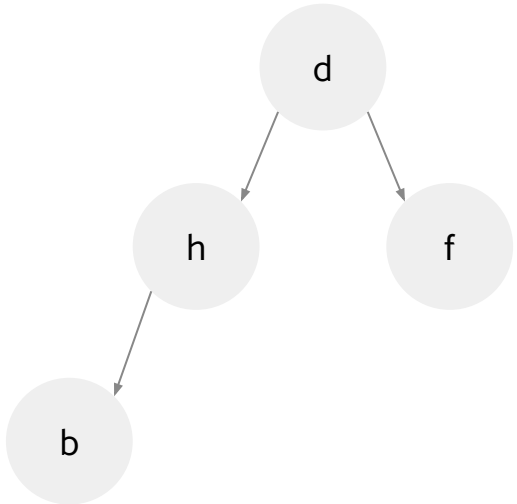
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- d h f]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

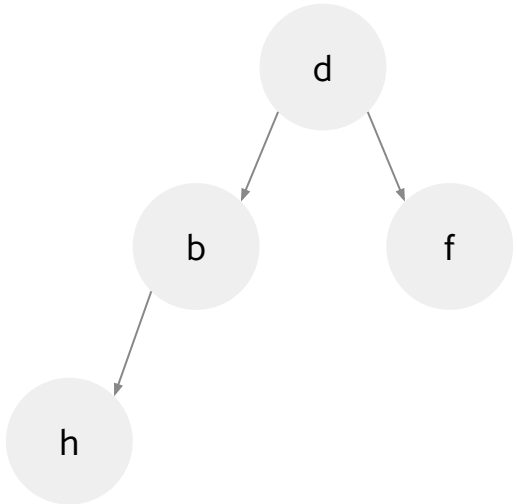
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- d h f b]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

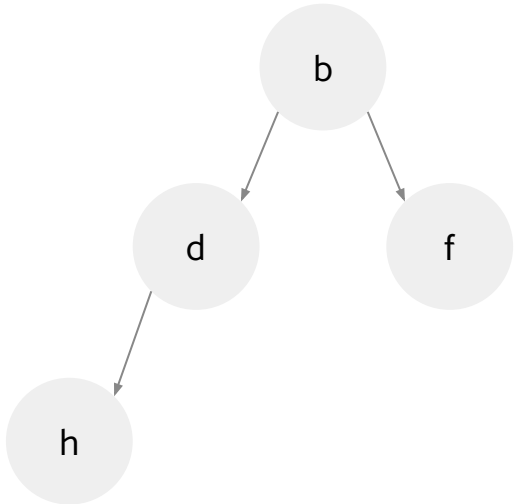
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- d b f h]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

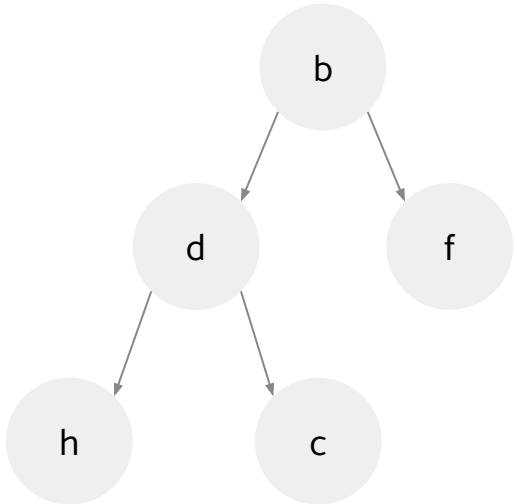
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- b d f h]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

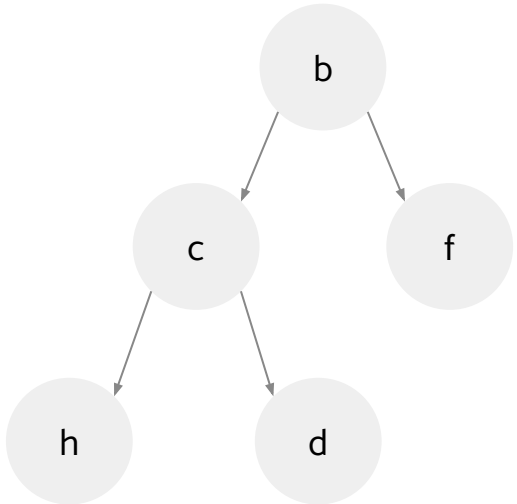
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- b d f h c]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

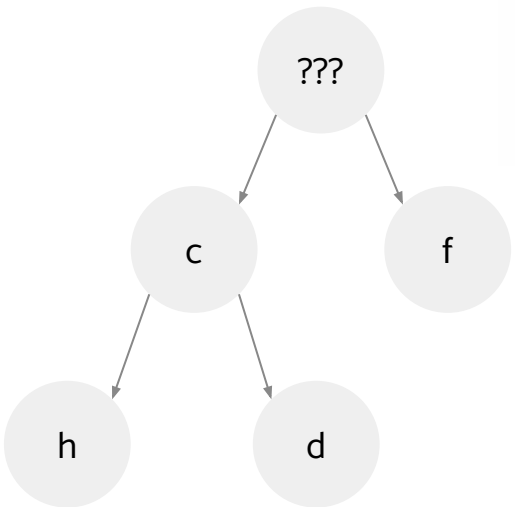
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- b c f h d]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

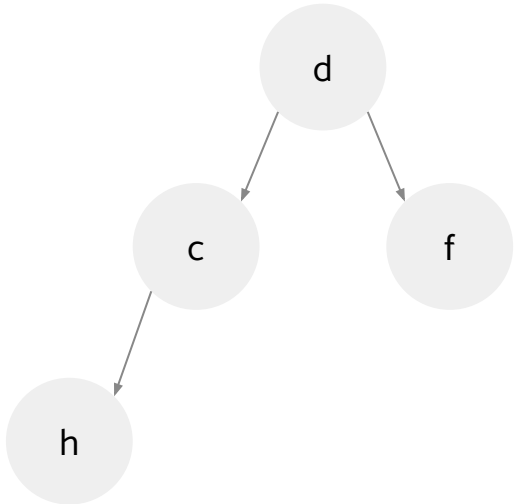
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- ? c f h d]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

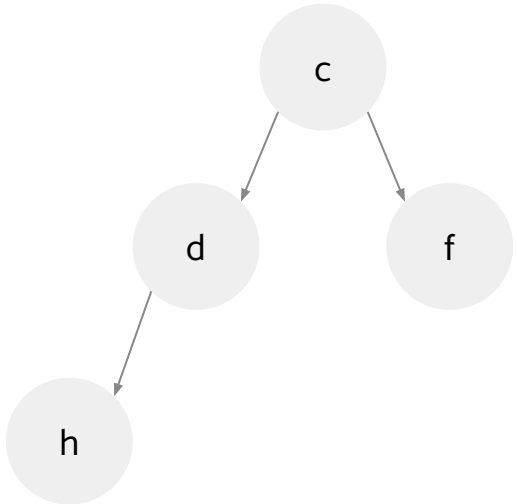
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- d c f h]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

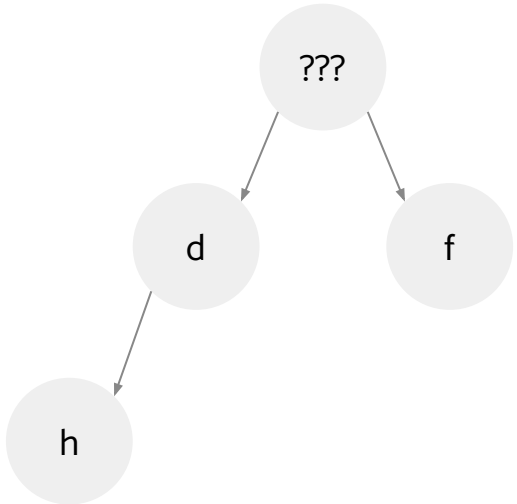
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- c d f h]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

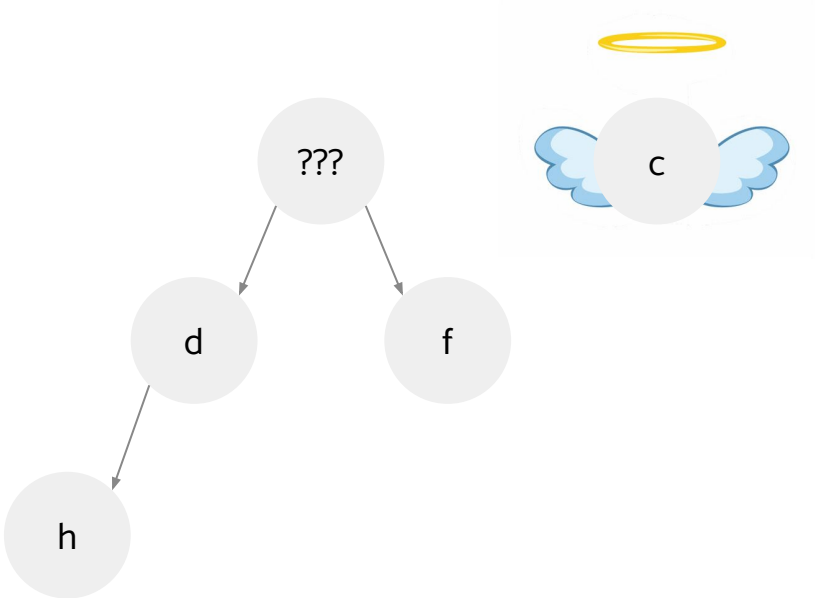
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- ? d f h]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

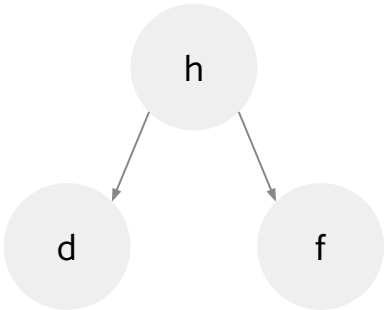
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- ? d f h]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

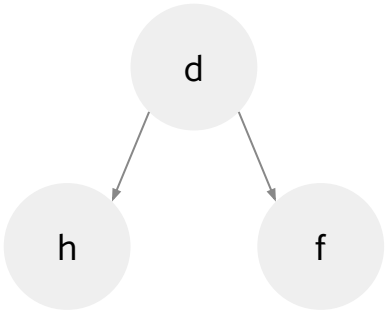
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [- h d f]

2a Absolutely Valuable Heaps **Draw the heap and its corresponding array after each operation below.**

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



Underlying array: [- d h f]

2b Absolutely Valuable Heaps

Your friendly TA Allen challenges you to quickly implement an integer max-heap data structure. However, you already have written a min-heap and you don't feel like writing a whole second 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.

2b Absolutely Valuable Heaps

Your friendly TA Allen challenges you to quickly implement an integer max-heap data structure. However, you already have written a min-heap and you don't feel like writing a whole second 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.

For every insert operation, negate the number and add it to the min-heap.

For a removeMax operation call removeMin on the min-heap and negate (or take the absolute value of) the number returned. Any number negated twice is itself, and since we store the negation of numbers, the order is now reversed (what used to be the max is now the min).

3 Trinary Search Trees

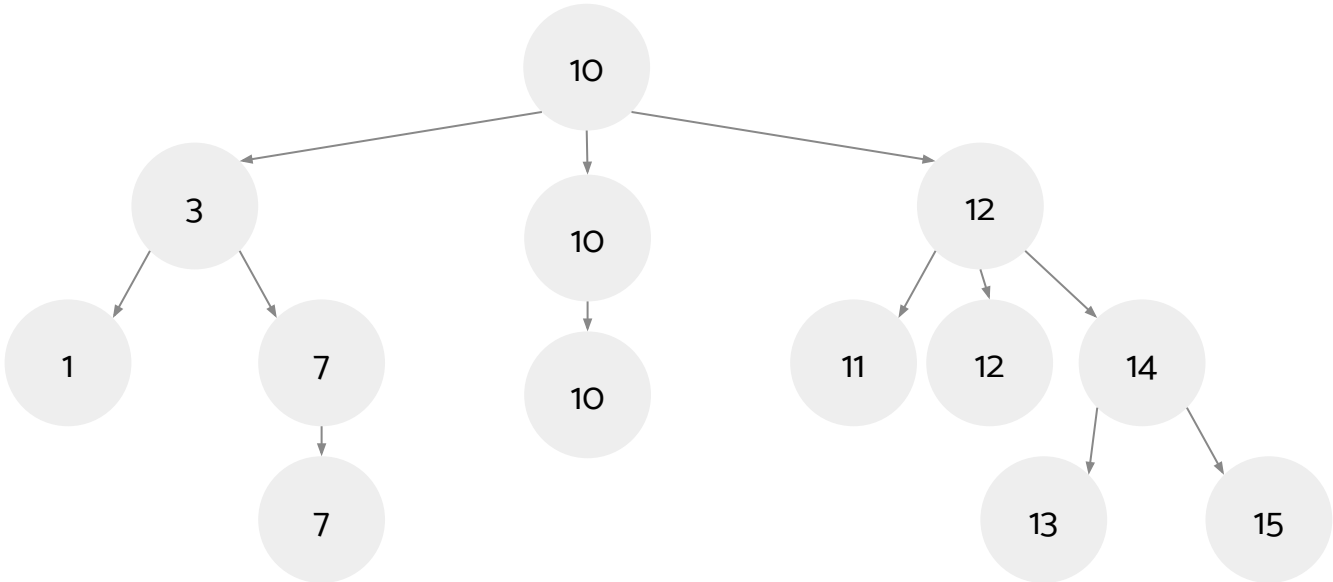
Suppose we build a Trinary Search Tree (TST), which behaves like a BST but allows duplicates, with the following BST-like invariants:

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 (only new rule)

Describe an algorithm that will print the elements in a TST in **descending** order. *Hint: you might find one of the traversals we used in Question 1 to be a good starting point to your algorithm here.*

3 Trinary Search Trees

Example TST:



3 Trinary Search Trees

Describe an algorithm that will print the elements in a TST in **descending** order.

Reverse inorder traversal: given the root of some TST, we

1. reverse onto the right child subtree
2. print the root's value
3. reverse onto the middle child subtree
4. reverse onto the left subtree.

*The print root value (step 2) and traverse onto the middle child (step 3) steps can be swapped, because overall the order of the printed values should be the same

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