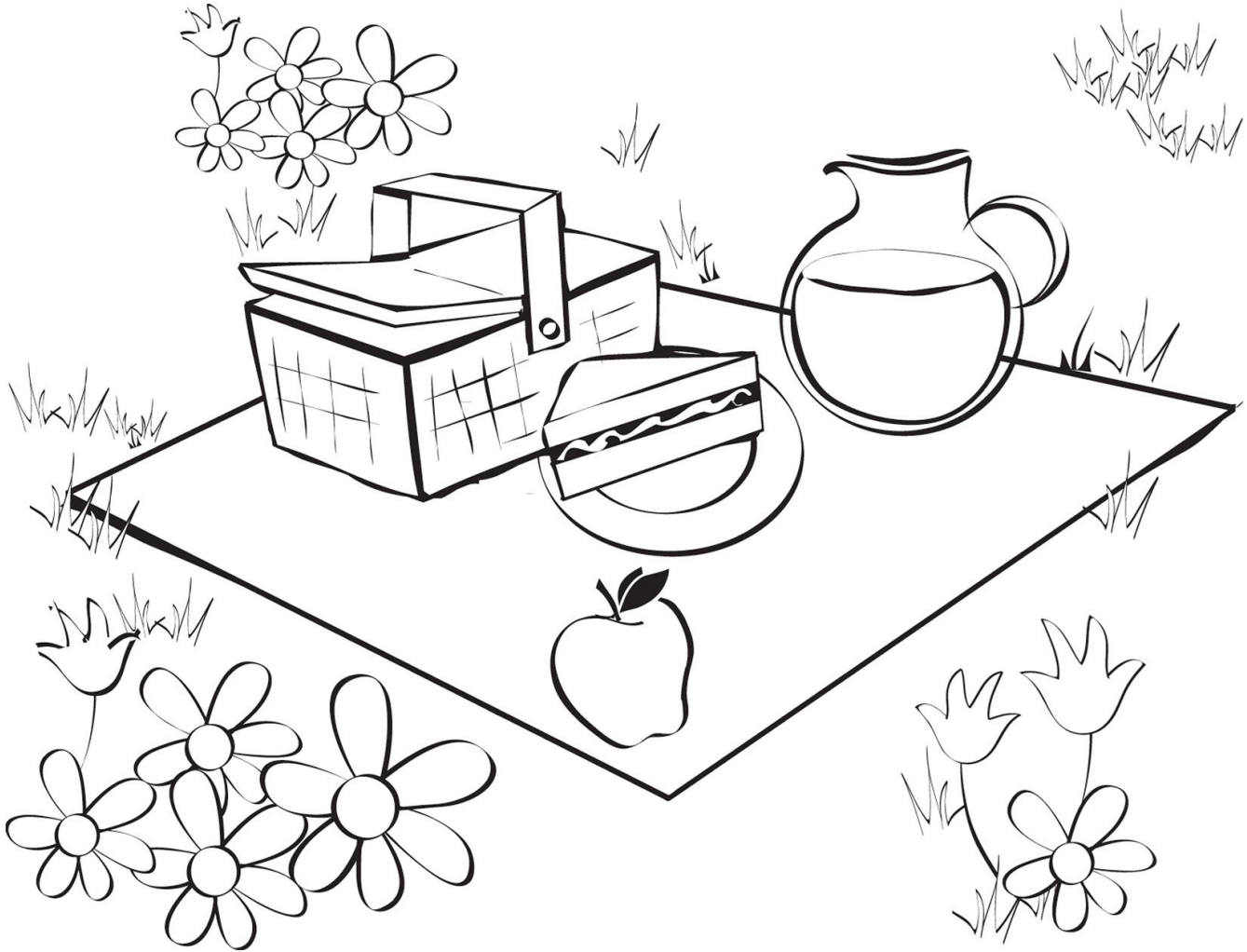


We'll start at Berkeley time!

Till then, feel free to ask me any questions (we've got a midterm coming up!) or keep yourself entertained with this riddle:

Imagine you are in a room with no windows or doors. How will you get out?

Trees, Graphs, & Heaps

Discussion 09

Announcements

Week 9

- ❏ Homework 2 - due on Monday 3/15
- ❏ No lab this week
- ❏ Mid Semester Survey - due on Monday 3/15 (32 extra credit points)
- ❏ Midterm 2 - happening Wednesday (3/17) 7:10 - 9:00 PM

Content Review

Trees

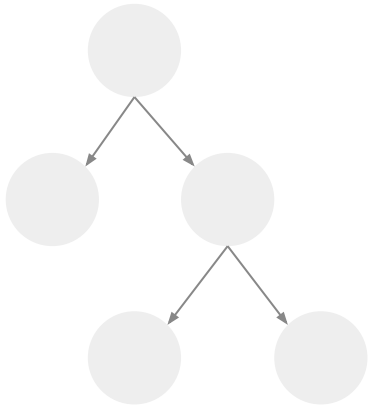
Trees are structures that follow a few basic rules:

1. If there are N nodes, there are N-1 edges
2. There is exactly 1 path from every node 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.

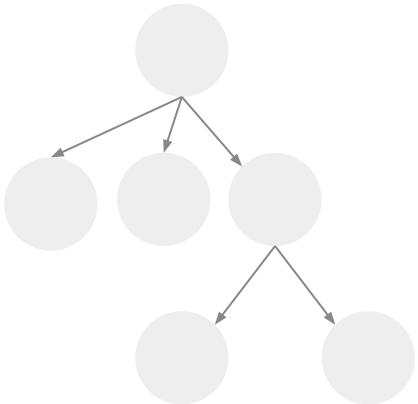
A leaf of a tree is a node with no child nodes.

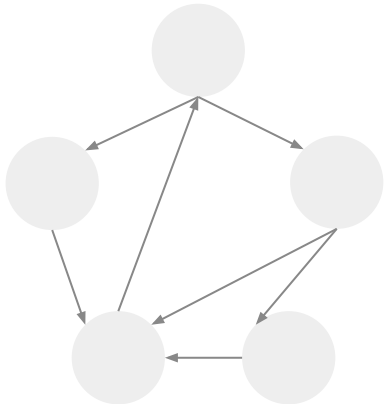


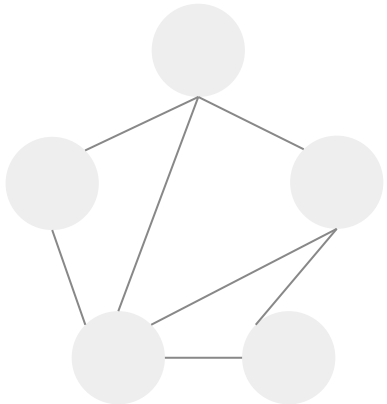
Graphs

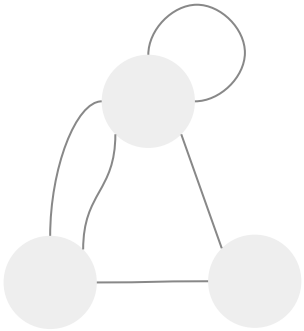
Graphs are similar to what we know about trees but more general!

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





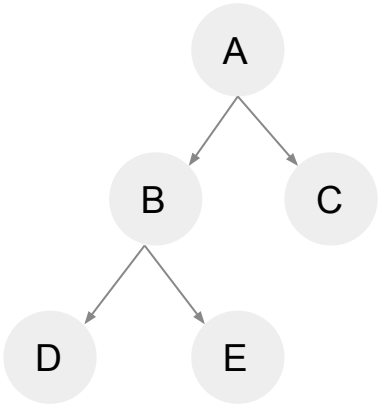




Breadth First Search

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 tree.

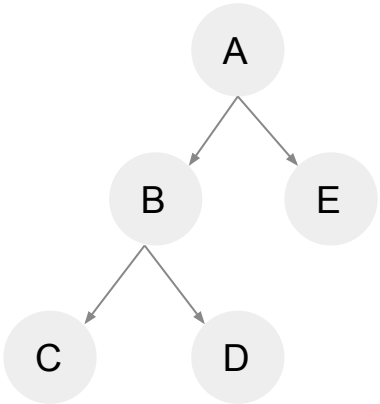
BFS is usually done using a queue.



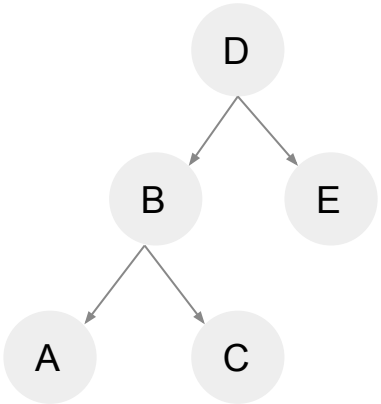
Depth First Search

Depth First Search means we visit each subtree in some order recursively.

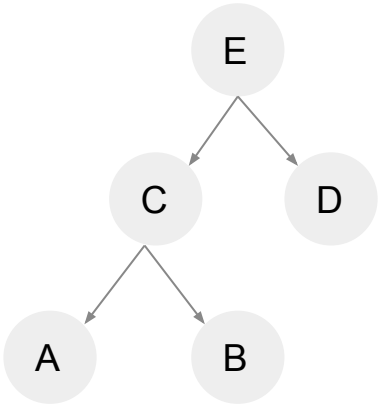
DFS is usually done using a stack.



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.

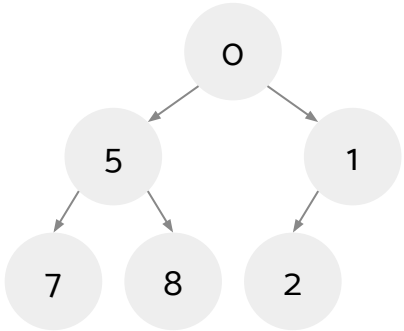


Post-order traversals visit the child nodes before visiting the parent nodes

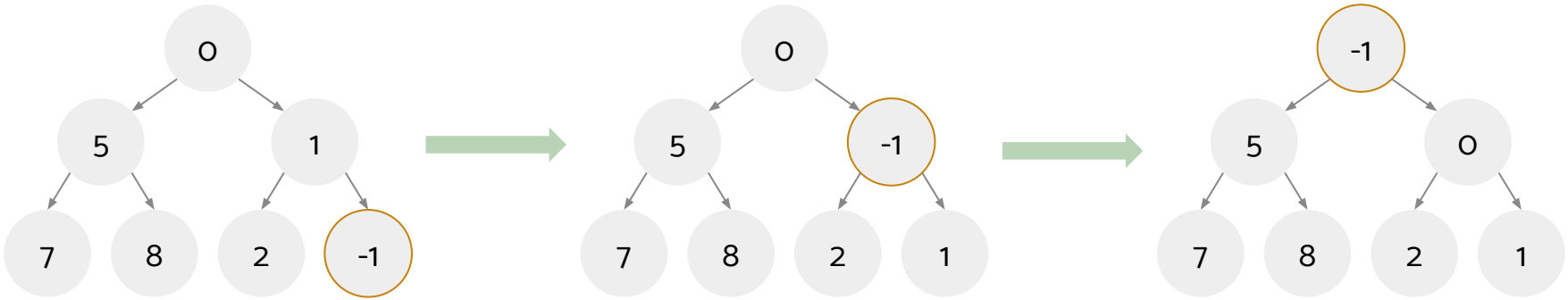
Heaps

Heaps are special trees that follow a few basic rules:

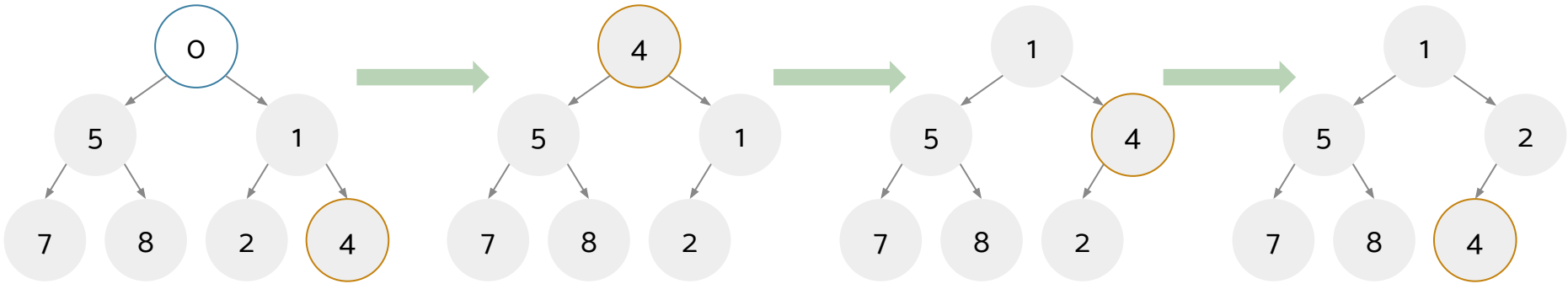
1. Heaps are complete - the only empty parts of a heap are in the bottom row, to the right
2. In a min-heap, each node must be *smaller* than all of its child nodes. The opposite is true for max-heaps.



Insertion into Heaps



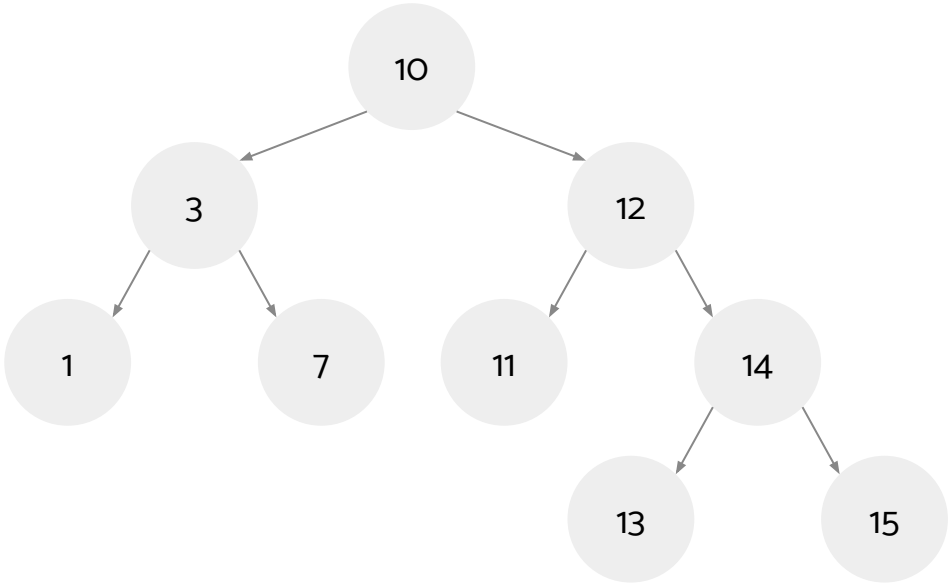
Deletion from Heaps



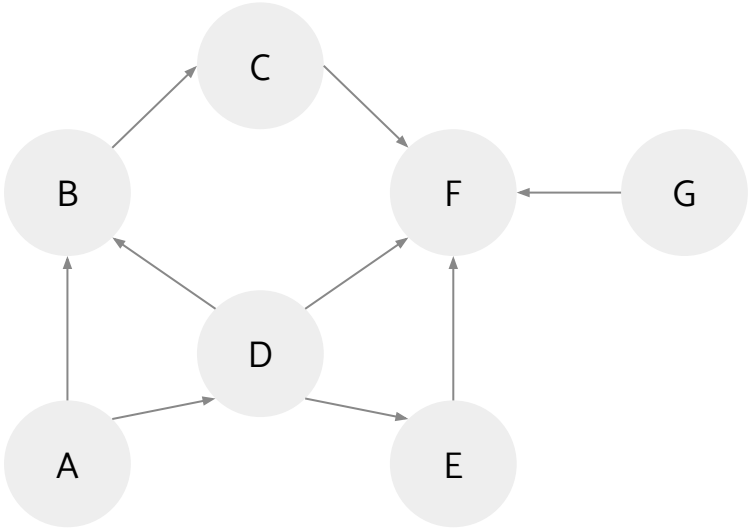
Worksheet

1 Tree Traversals

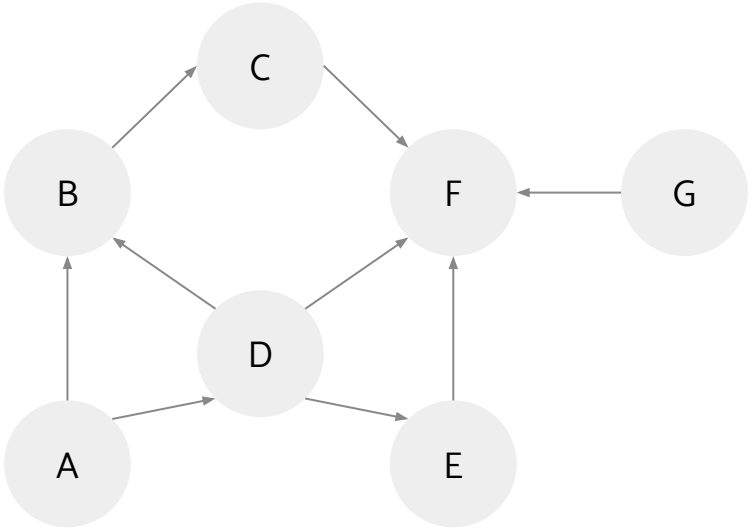
Write out the pre-order, in-order, post-order, and level-order traversals.



2a Graph Representations Write out the adjacency matrix and adjacency list.



2a Graph Representations Write out the adjacency matrix and adjacency list.

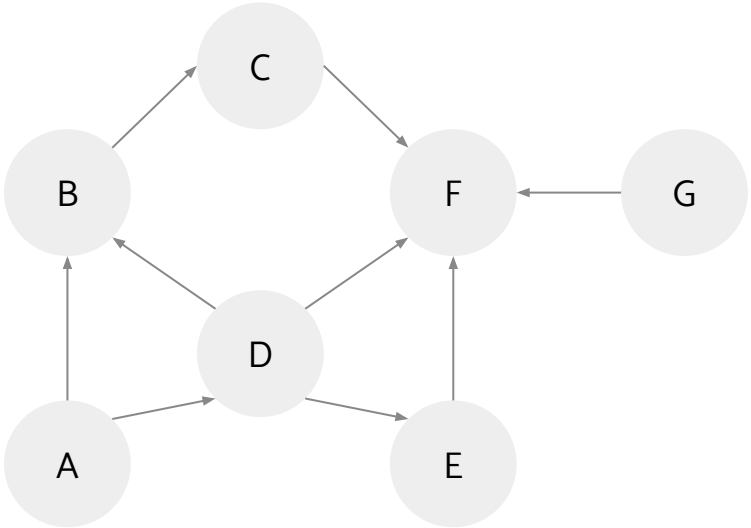


	A	B	C	D	E	F	G
A							
B							
C							
D							
E							
F							
G							

To

From

2a Graph Representations Write out the adjacency matrix and adjacency list.

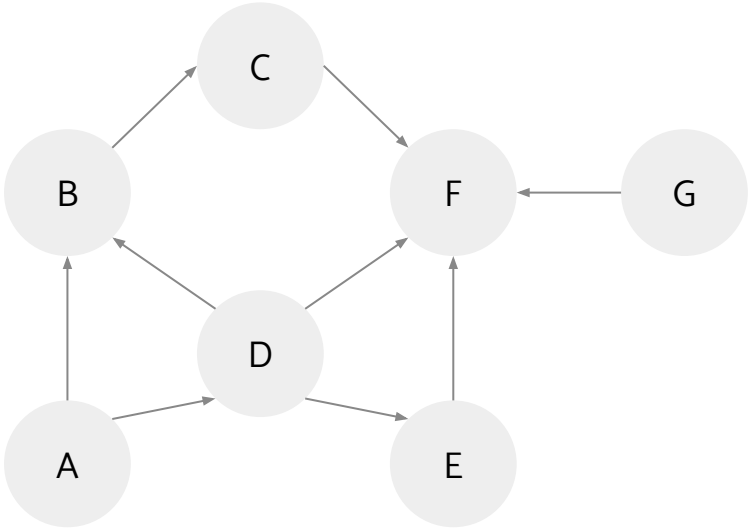


	A	B	C	D	E	F	G
A		✓		✓			
B							
C							
D							
E							
F							
G							

To

From

2a Graph Representations Write out the adjacency matrix and adjacency list.

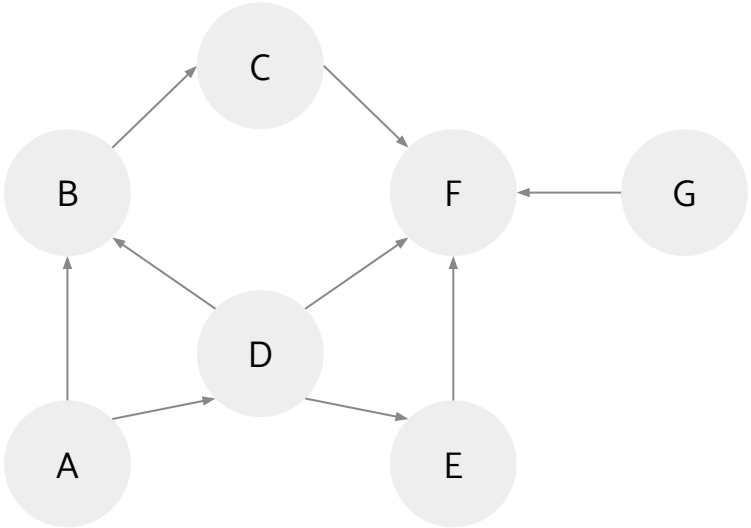


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C							
D							
E							
F							
G							

To

From

2a Graph Representations Write out the adjacency matrix and adjacency list.

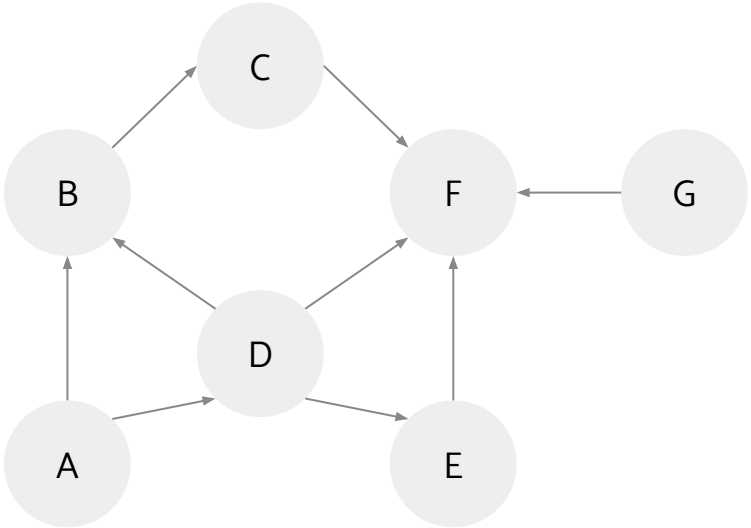


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C						✓	
D							
E							
F							
G							

To

From

2a Graph Representations Write out the adjacency matrix and adjacency list.

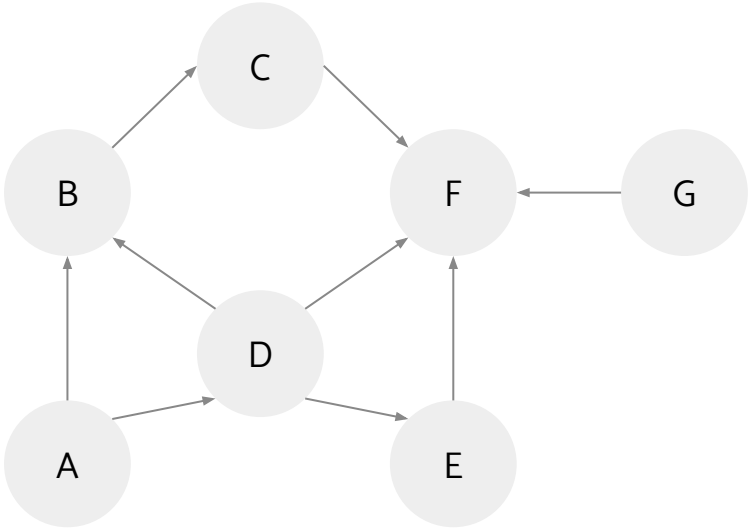


	A	B	C	D	E	F	G
A		✓		✓			
B			✓				
C						✓	
D		✓			✓	✓	
E						✓	
F							
G						✓	

To

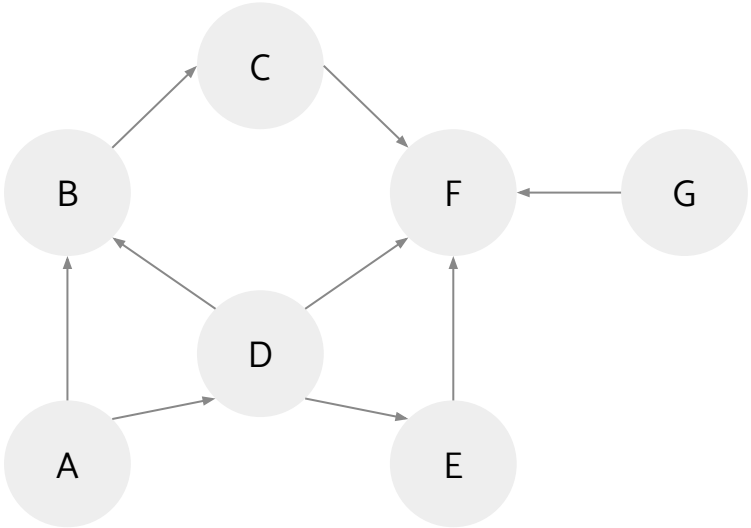
From

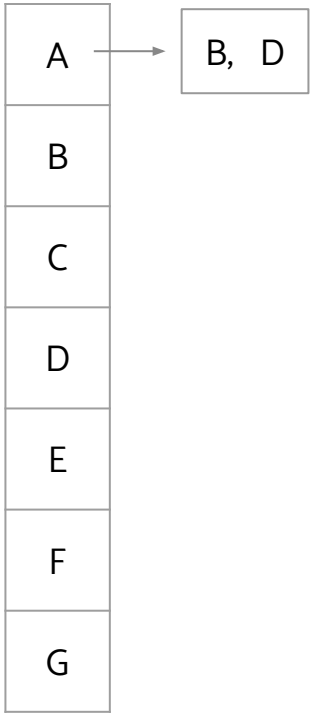
2a Graph Representations Write out the adjacency matrix and adjacency list.



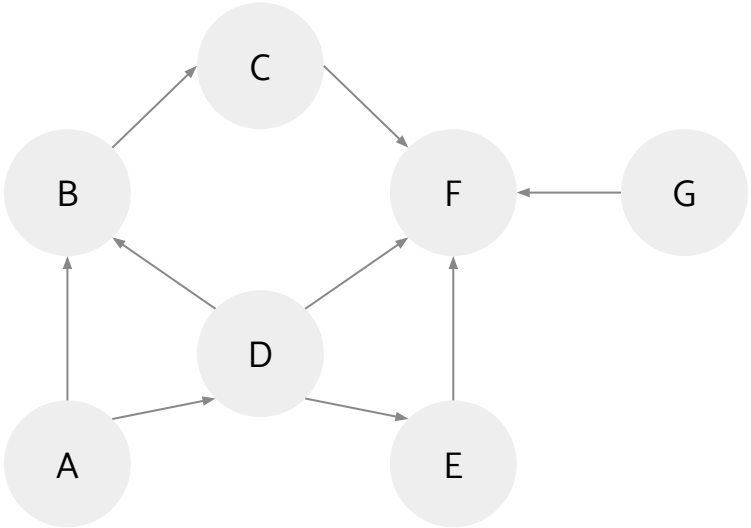
A
B
C
D
E
F
G

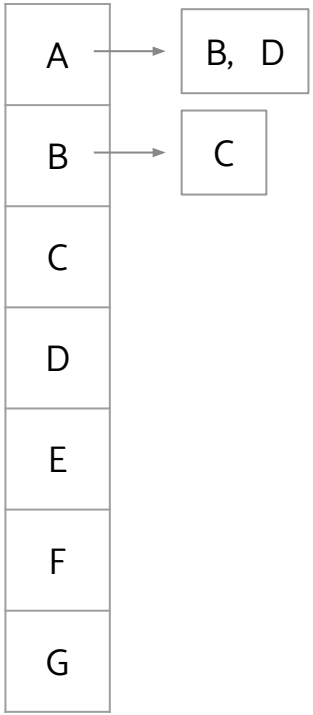
2a Graph Representations Write out the adjacency matrix and adjacency list.



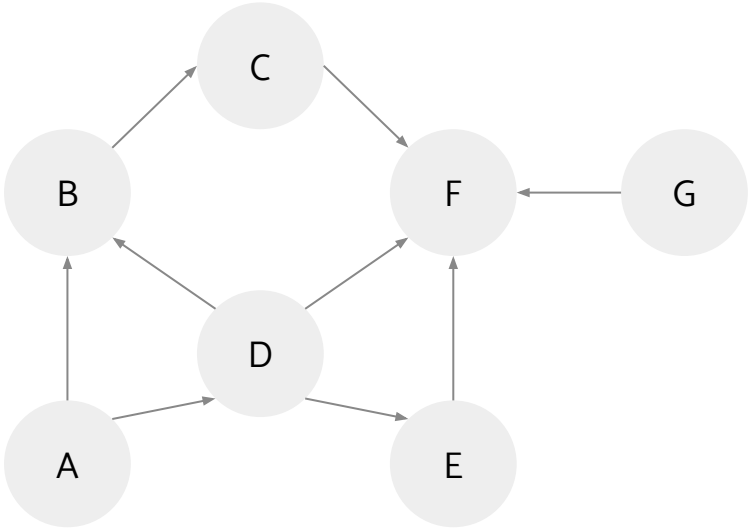


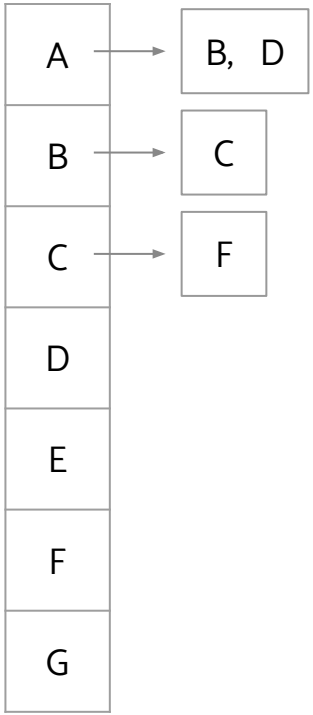
2a Graph Representations Write out the adjacency matrix and adjacency list.



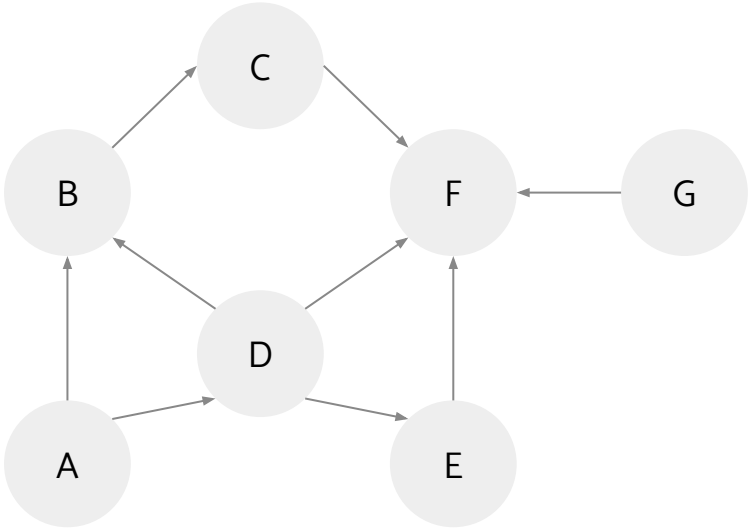


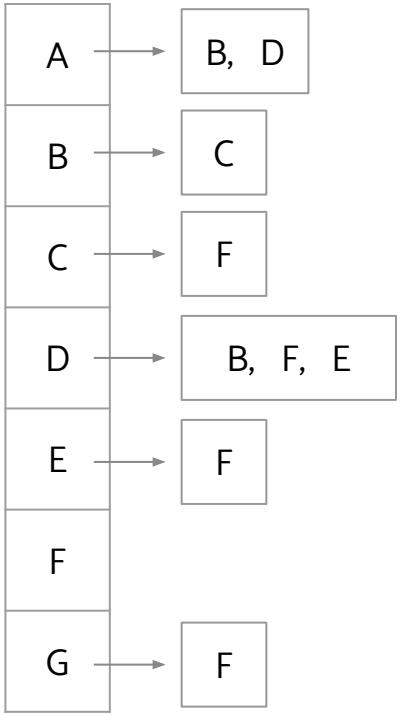
2a Graph Representations Write out the adjacency matrix and adjacency list.





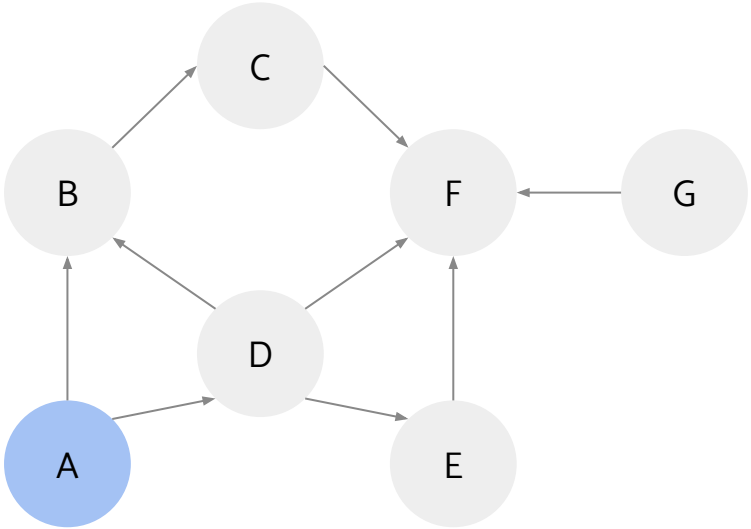
2a Graph Representations Write out the adjacency matrix and adjacency list.





2b Graph Representations

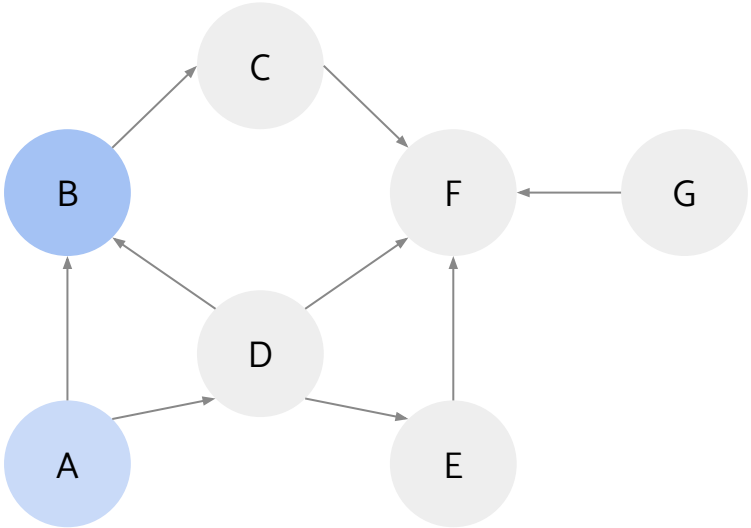
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



DFS Pre-Order:
A

2b Graph Representations

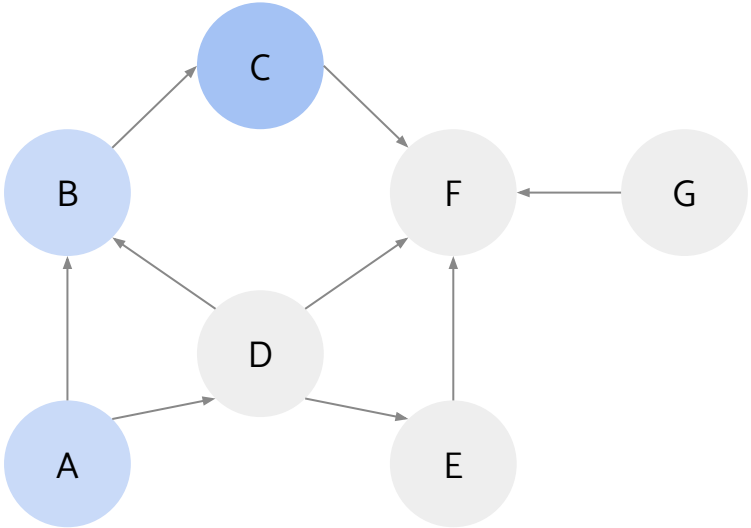
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



DFS Pre-Order:
A, B

2b Graph Representations

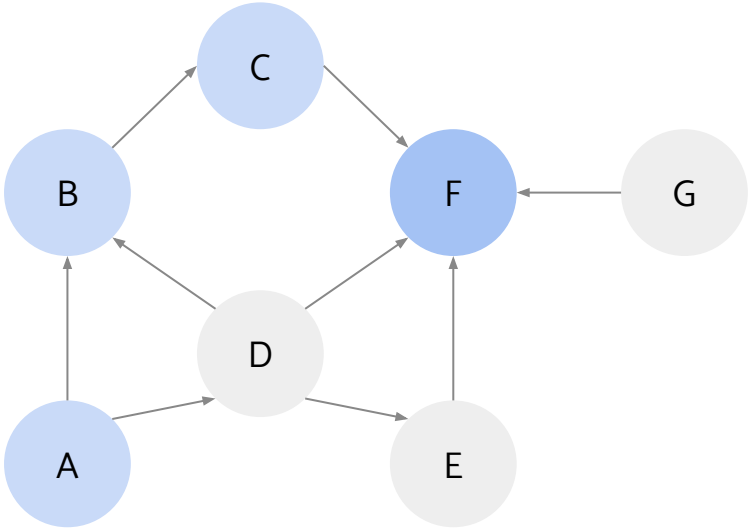
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



DFS Pre-Order:
A, B, C

2b Graph Representations

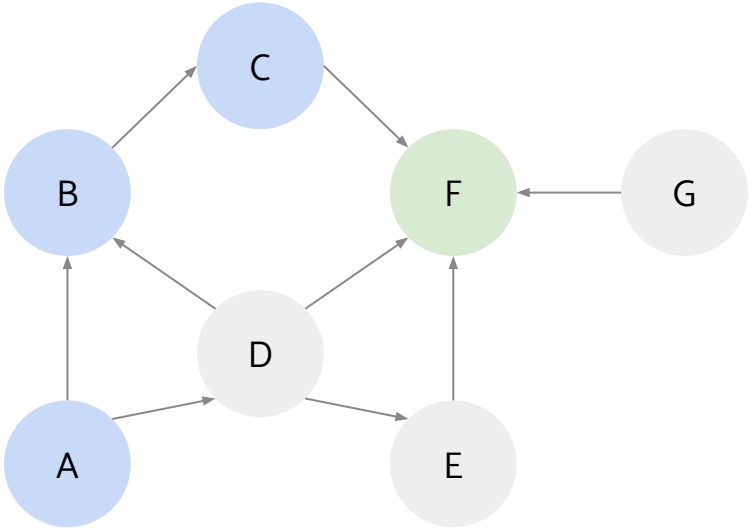
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

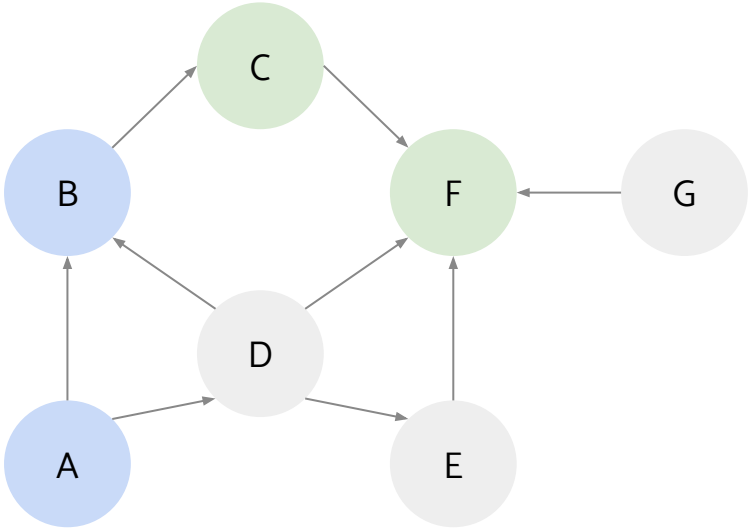
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

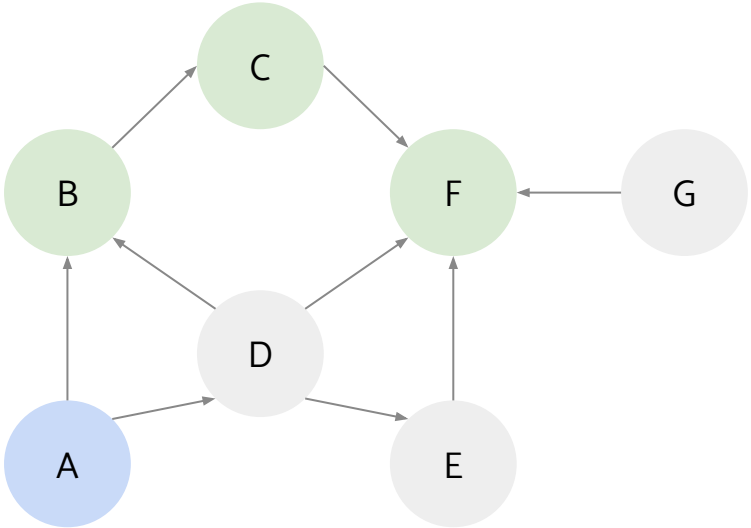
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

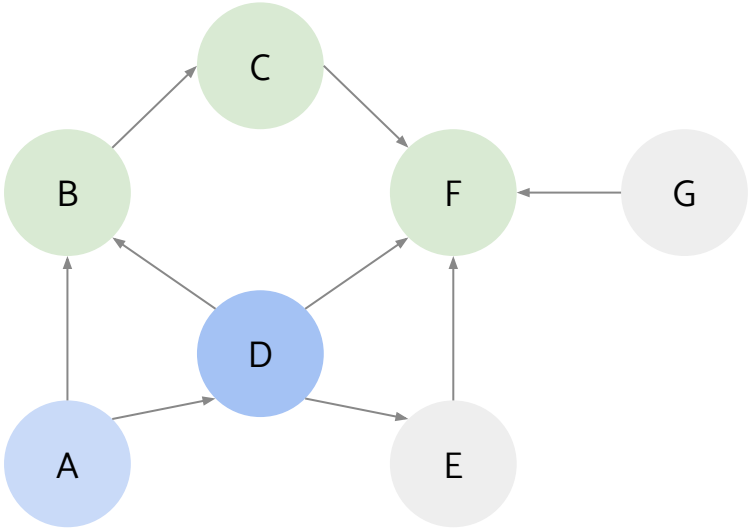
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

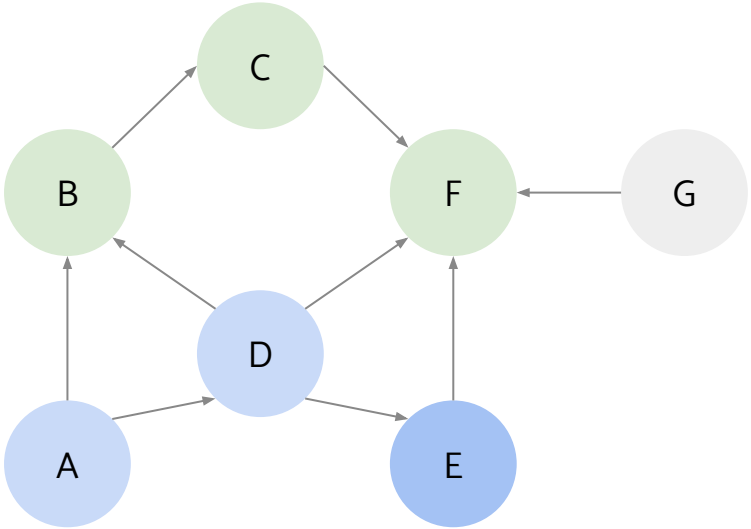
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

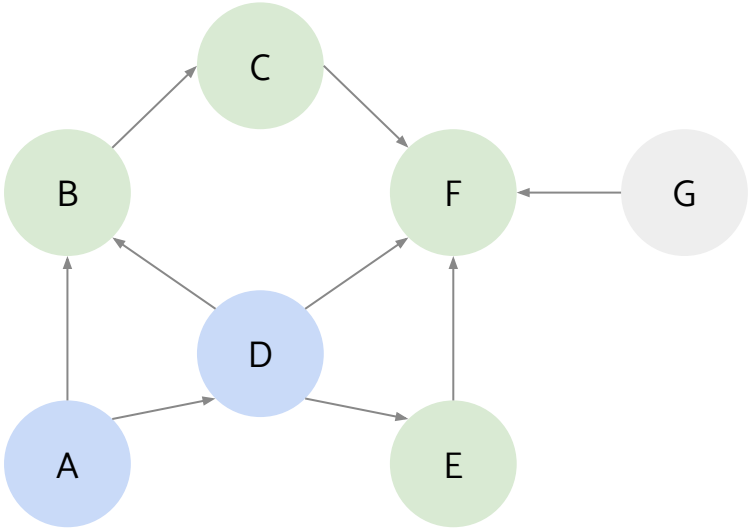
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

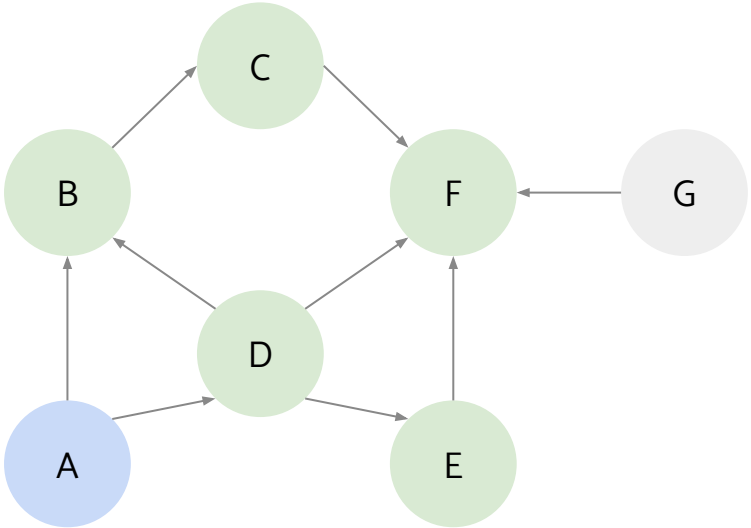
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

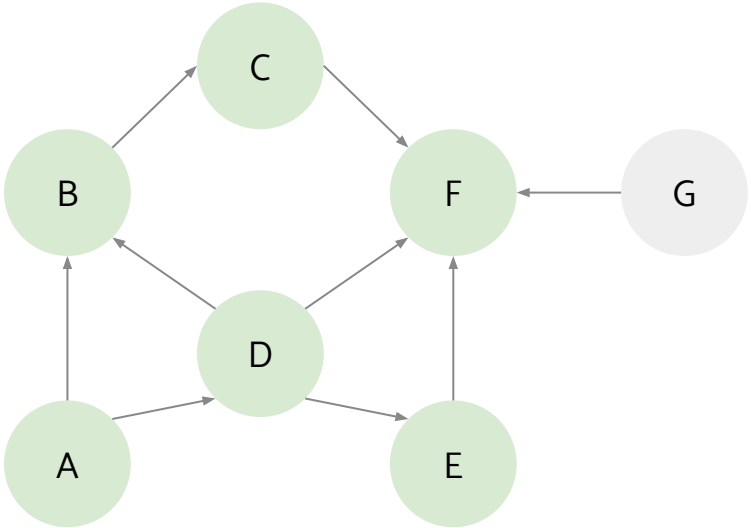
Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

2b Graph Representations

Write out the order in which nodes are visited by DFS pre-order. Extra: Do post-order and BFS.



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

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

```
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: [ ]

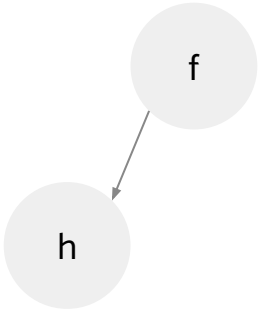
3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

```
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]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

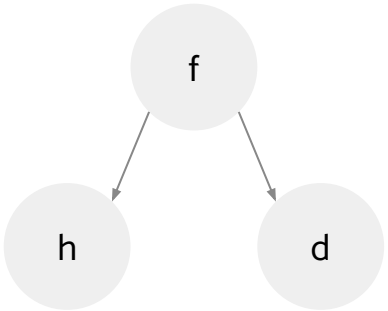
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [f h]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

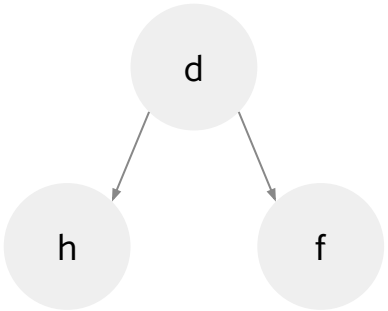
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [f h d]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

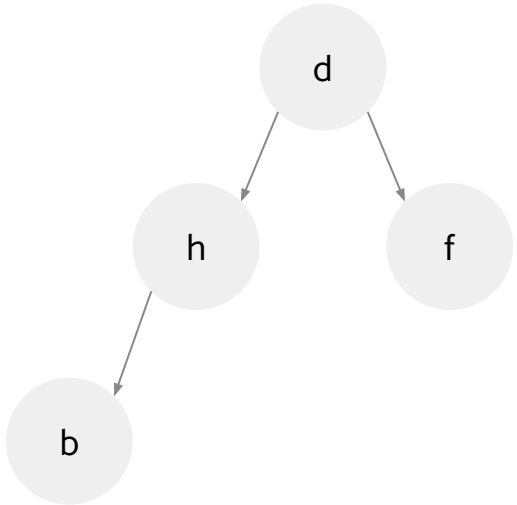
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [d h f]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

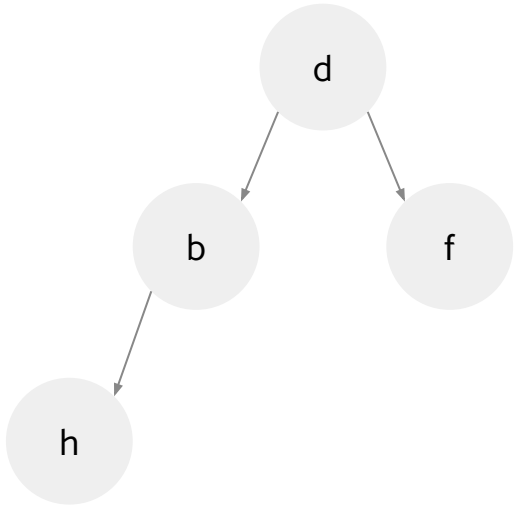
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [d h f b]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

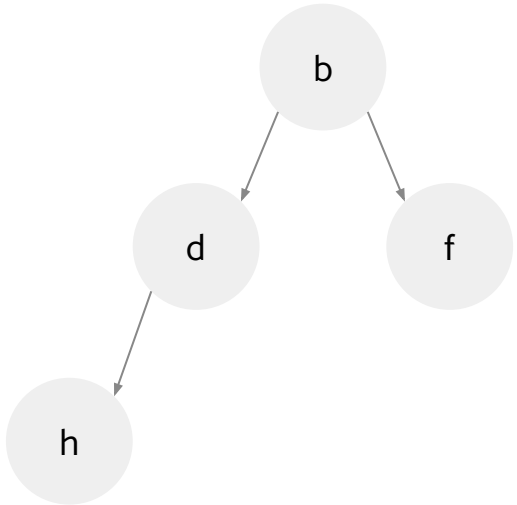
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [d b f h]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

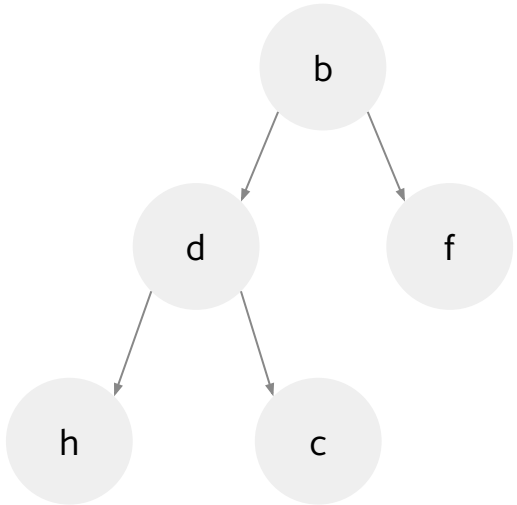
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [b d f h]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

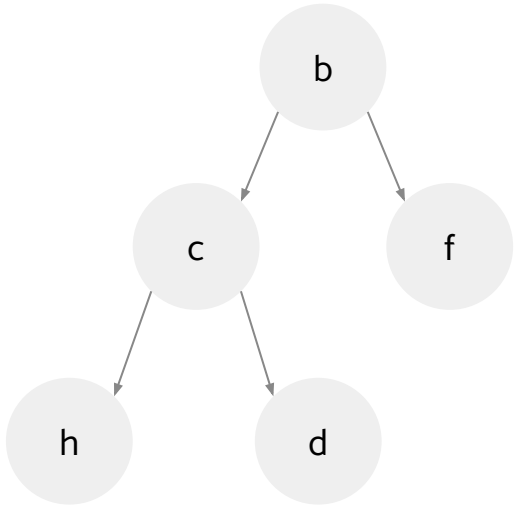
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [b d f h c]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

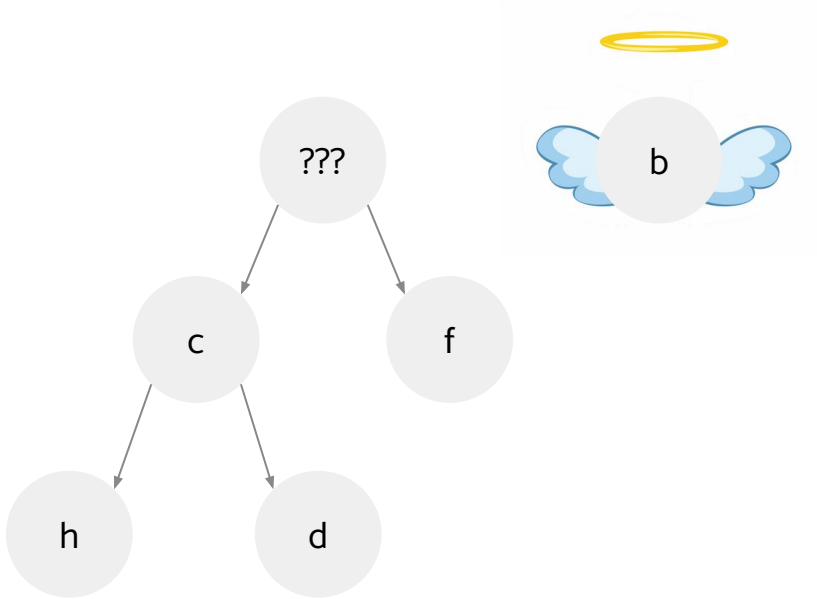
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [b c f h d]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

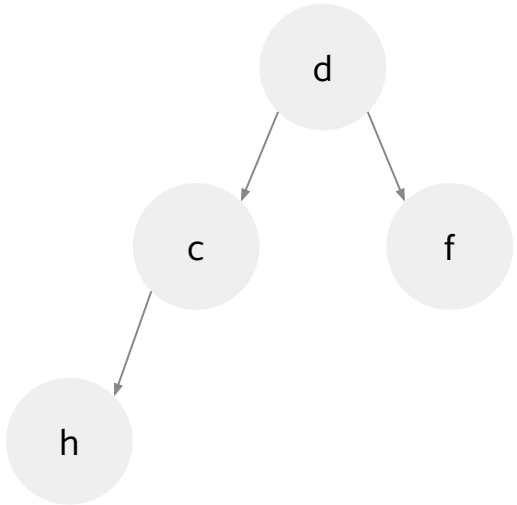
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [? c f h d]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

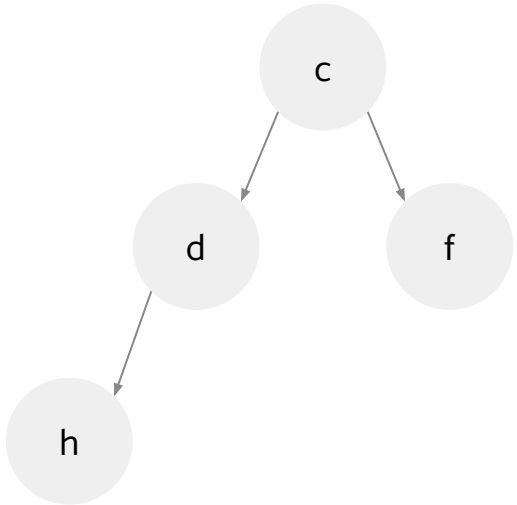
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [d c f h]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

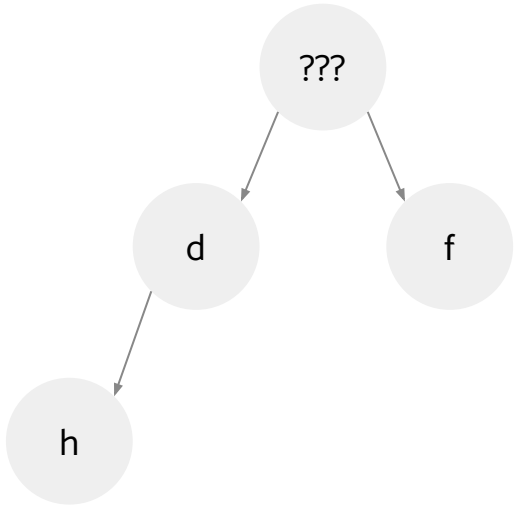
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [c d f h]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

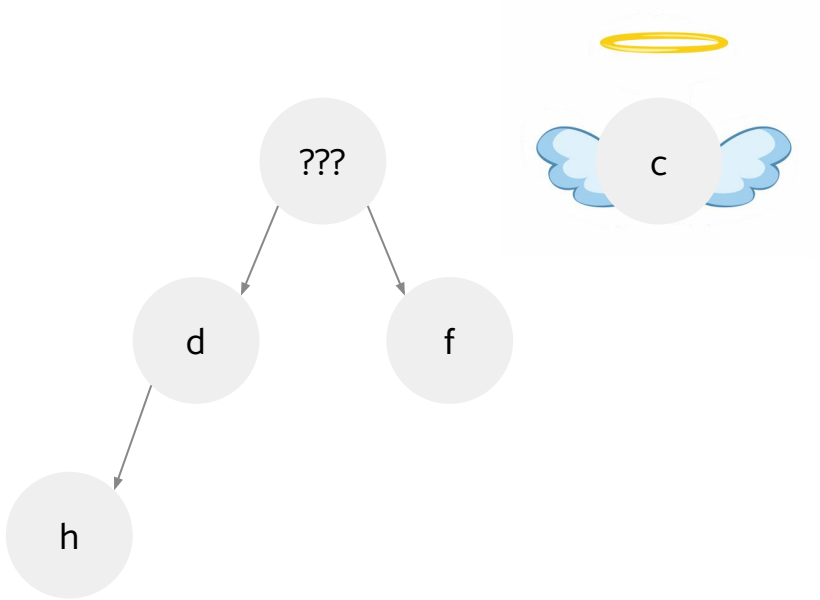
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [? d f h]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

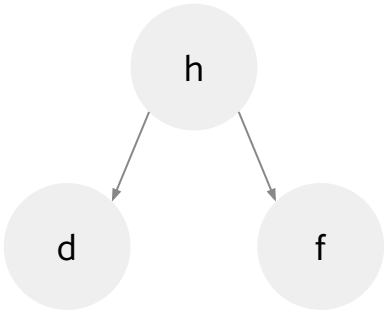
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [? d f h]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

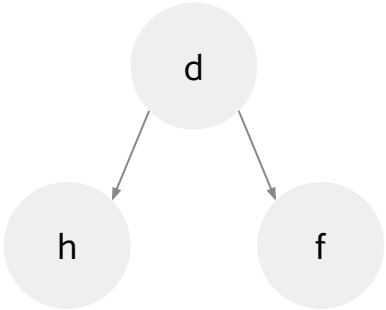
```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```



Underlying array: [h d f]

3a Heaps of Fun Draw the heap and its corresponding array after the operation allow.

```
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]

3b Heaps of Fun

Your friendly TA Tony 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.

Extra: Open midterm #2 Q&A