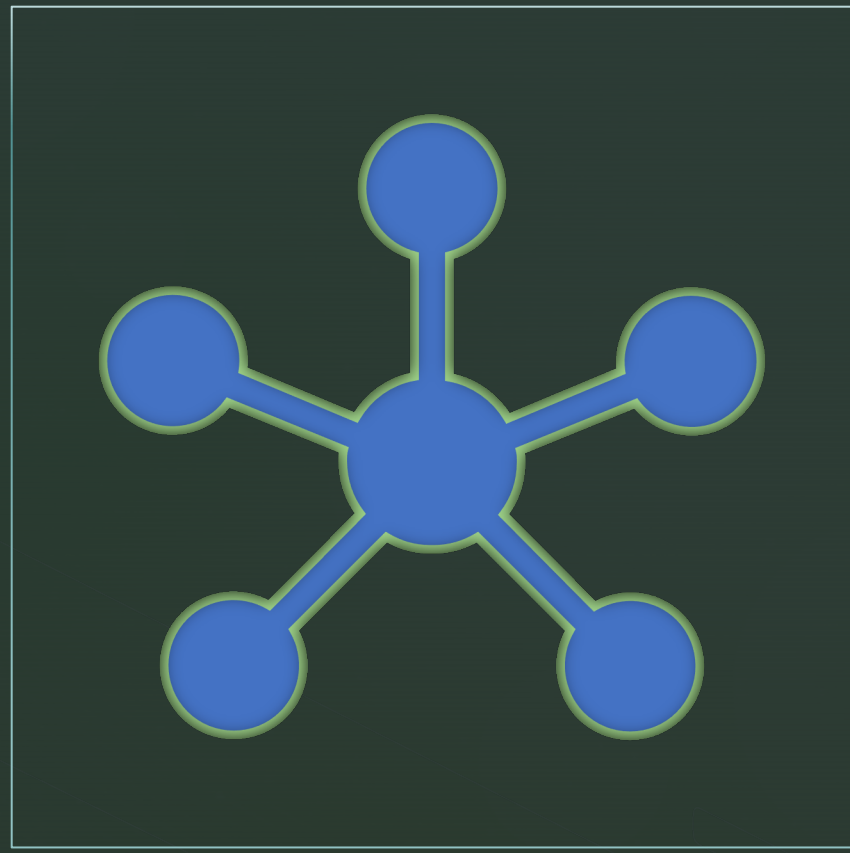
02-601 Recitation – Matthew Wolff

Network Searching

What is a Network?

- Here we'll just use "network" as a blanket term for various kinds of graphs, be they trees or otherwise
- A map of US cities and the highways that connect them
- We often want to explore a network to find something in particular



What is a search?

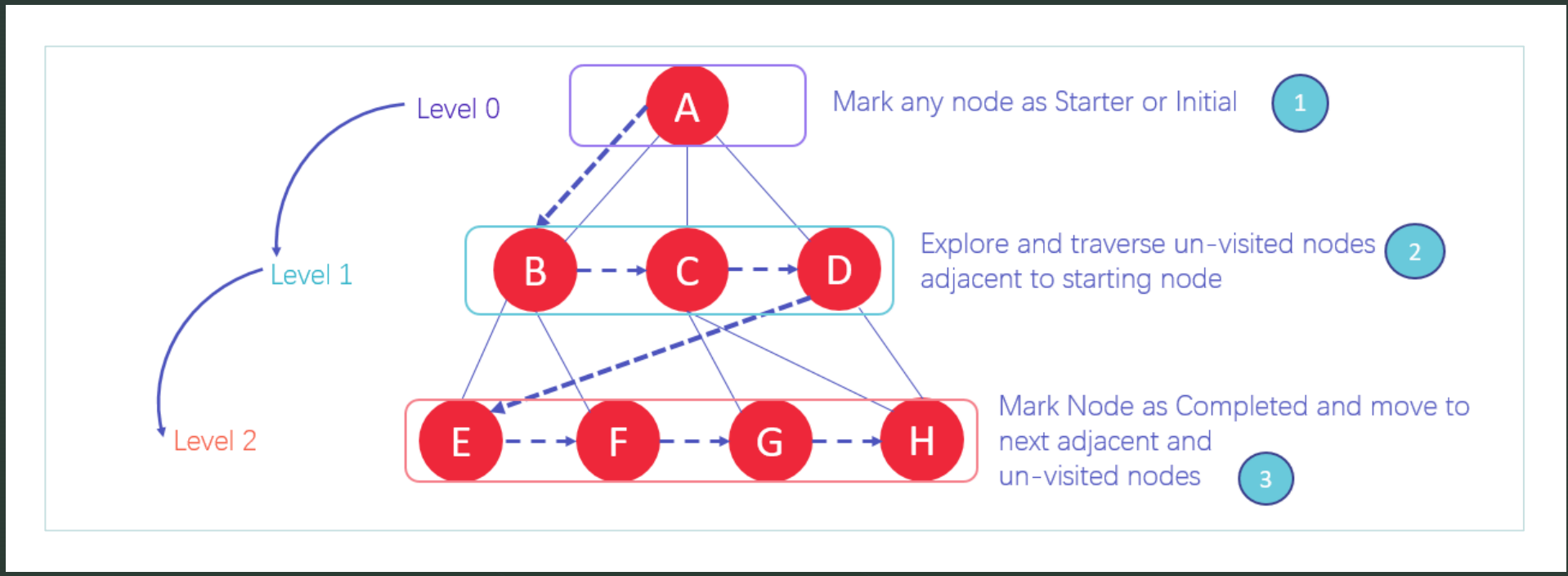
A search is an algorithm that attempts to find a particular item in the network

Imagine you're at City A, and you want to know if:

1. City B exists
2. A path to City B

How can we find a way to city B? a search!

A search will establish a path from node ***s*** to every other node it finds. This effectively creates a tree rooted at ***s***!



Search Algorithm: Breadth-First Search

Breadth First Search Qualities

We explore our neighbors first, then our neighbors' neighbors!

We implement it using a data structure called a Queue (first-in, first out!)

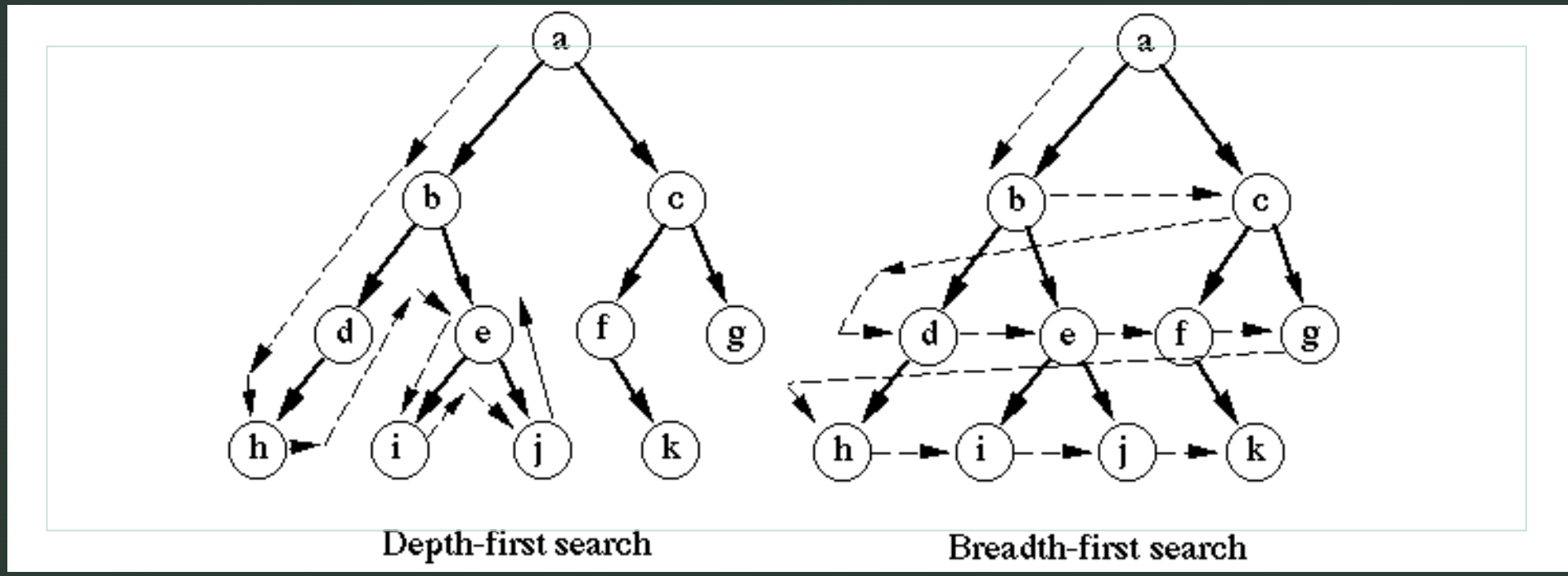
It will produce the same tree every time

It keeps a queue in memory, which can get expensive if you have a very wide graph

It prioritizes searching nodes that are nearby

Example

https://www.cs.usfca.edu/~galles/visualization/BFS.html



Search Algorithm: Depth-First Search

Depth-First Search Qualities

We explore one neighbor, then their neighbor, then their neighbor's neighbor, etc!

We implement this using a stack. A stack is Last-in, First out!

It can produce *different* trees (challenge: why?)

Depth-First search is easily implemented using recursion!

If you have a very, very deep graph, you might get lost down the wrong path

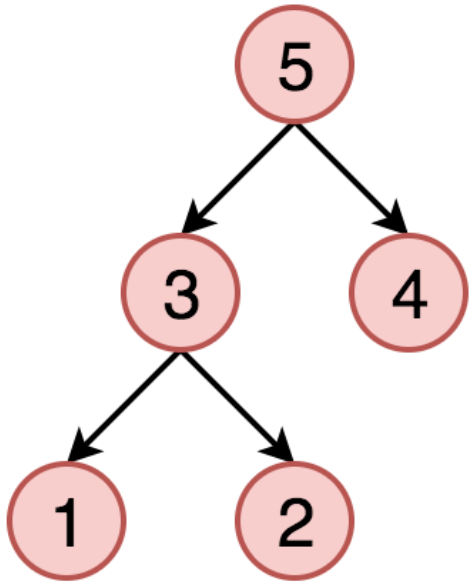
Example

https://www.cs.usfca.edu/~galles/visualization/DFS.html

Types of DFS Traversals

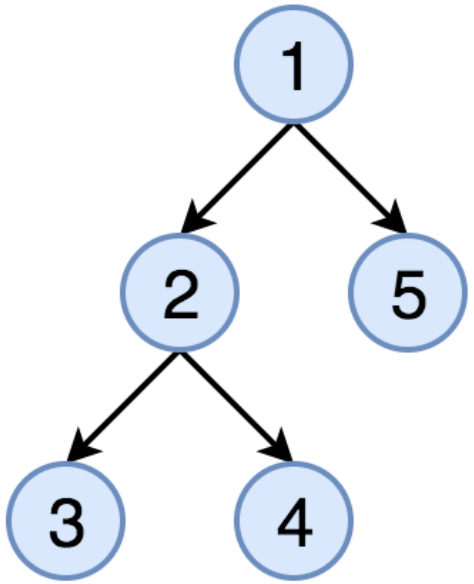
DFS Postorder

Bottom -> Top
Left -> Right



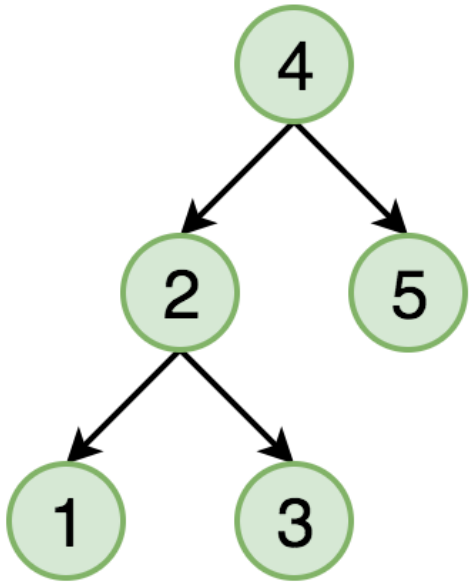
DFS Preorder

Top -> Bottom
Left -> Right



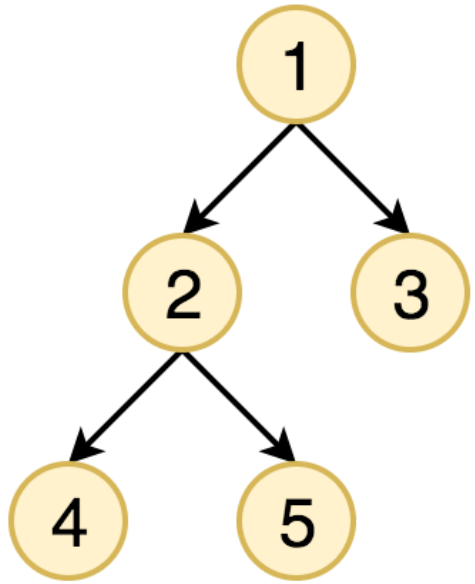
DFS Inorder

Left -> Node -> Right

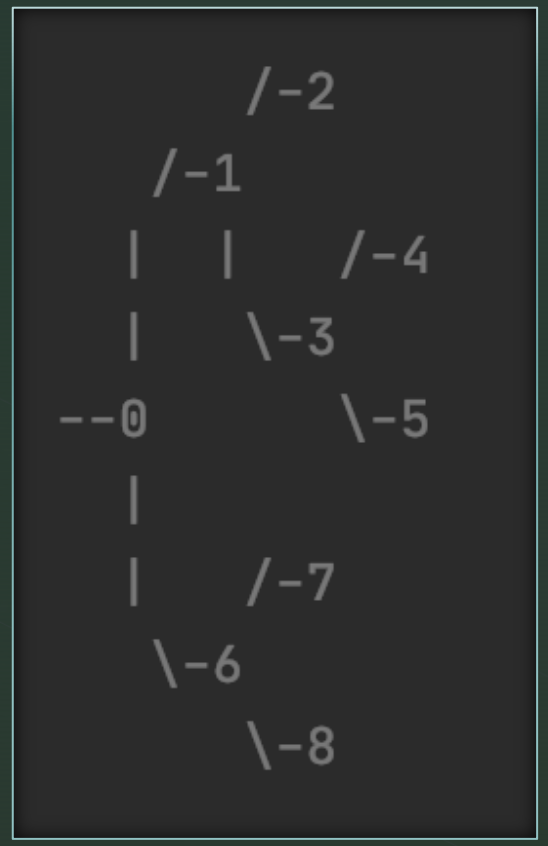


BFS

Left -> Right
Top -> Bottom



So what does it look like?




```
---- TRAVERSALS ----
---- PRE ----
0 1 2 3 4 5 6 7 8
---- IN ----
2 1 4 3 5 0 7 6 8
---- POST ----
2 4 5 3 1 7 8 6 0
```

```go
func (node *TreeNode) inOrderDFS() {
    if node == nil {
        return
    }
    if node.left != nil {
        node.left.inOrderDFS()
    }
    _, _ = fmt.Println(node.label)
    if node.right != nil {
        node.right.inOrderDFS()
    }
}
```

```go
func (node *TreeNode) postOrderDFS() {
    if node == nil {
        return
    }
    if node.left != nil {
        node.left.postOrderDFS()
    }
    if node.right != nil {
        node.right.postOrderDFS()
    }
    _, _ = fmt.Println(node.label)
}
```

```go
func (node *TreeNode) preOrderDFS() {
    if node == nil {
        return
    }
    _, _ = fmt.Println(node.label)
    if node.left != nil {
        node.left.preOrderDFS()
    }
    if node.right != nil {
        node.right.preOrderDFS()
    }
}
```

Implementing Various Traversals

BFS vs DFS: Key Differences

If you know a solution is not far from the root of the tree, a breadth first search might be better.

If the tree is very deep and solutions are rare, depth first search might take an extremely long time... so BFS could be faster.

If the tree is very wide, a BFS might need too much memory for its queue, so it might be impractical.

If solutions are frequent but located deep in the tree, BFS could be impractical.

If the search tree is very deep you will need to restrict the search depth for depth first search (DFS), anyway (for example with iterative deepening).

Code Time!

https://github.com/MatthewWolff/network_searching