

CSE 332: Data Structures & Parallelism

Lecture 15: Graph Traversals

Ruth Anderson
Winter 2025

Graph Traversals

Next problem: For an arbitrary graph and a starting node v , find all nodes *reachable* (i.e., there exists a path) from v

- Possibly “do something” for each node (an iterator!)
 - E.g. Print to output, set some field, etc.

Related Questions:

- Is an undirected graph connected?
- Is a directed graph weakly / strongly connected?
 - For strongly, need a cycle back to starting node

Basic idea:

- Keep following nodes
- But “mark” nodes after visiting them, so the traversal terminates and processes each reachable node exactly once

Graph Traversal: Abstract Idea

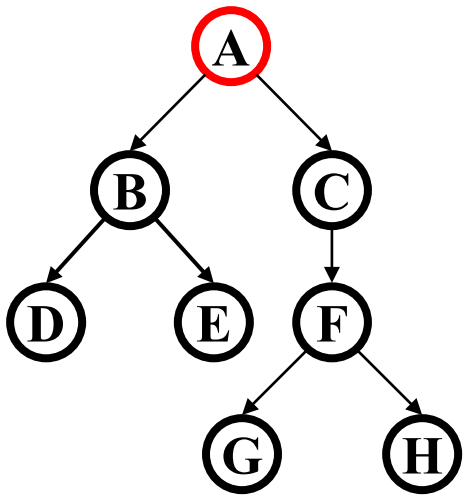
```
traverseGraph(Node start) {
    Set pending = emptySet();
    pending.add(start)
    mark start as "visited"
    while(pending is not empty) {
        next = pending.remove()
        for each node u adjacent to next
            if(u is not marked) {
                mark u
                pending.add(u)
            }
    }
}
```

Running time and options

- Assuming add and remove are $O(1)$, entire traversal is $O(|E|)$
 - Use an adjacency list representation
- The order we traverse depends entirely on how add and remove work/are implemented
 - Depth-first graph search (DFS): a stack
 - Breadth-first graph search (BFS): a queue
- DFS and BFS are “big ideas” in computer science
 - Depth: recursively explore one part before going back to the other parts not yet explored
 - Breadth: Explore areas closer to the start node first

Recursive DFS, Example : trees

- A tree is a graph and DFS and BFS are particularly easy to “see”

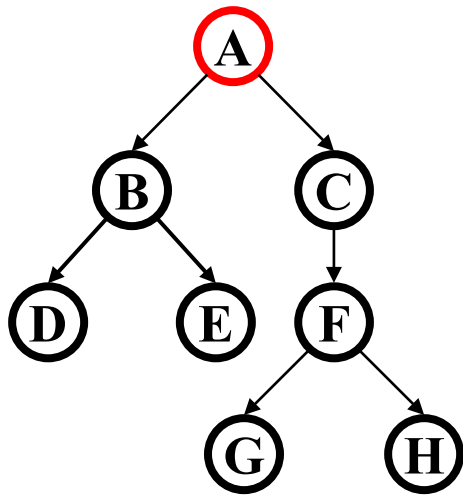


```
DFS(Node start) {  
    mark and “process” (e.g. print) start  
    for each node u adjacent to start  
        if u is not marked  
            DFS(u)  
}
```

Order processed: A, B, D, E, C, F, G, H

- Exactly what we called a “pre-order traversal” for trees
- The marking is not needed here, but we need it to support arbitrary graphs , we need a way to process each node exactly once

DFS with a stack, Example: trees

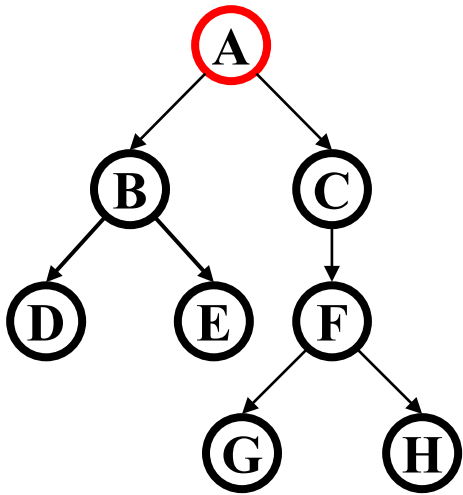


```
DFS2(Node start) {  
    initialize stack s to hold start  
    mark start as "visited"  
    while(s is not empty) {  
        next = s.pop() // and "process"  
        for each node u adjacent to next  
            if(u is not marked)  
                mark u and push onto s  
    }  
}
```

Order processed:

- A different but perfectly fine traversal

BFS with a queue, Example: trees



```
BFS(Node start) {  
    initialize queue q to hold start  
    mark start as "visited"  
    while(q is not empty) {  
        next = q.dequeue() // and "process"  
        for each node u adjacent to next  
            if(u is not marked)  
                mark u and enqueue onto q  
    }  
}
```

Order processed:

- A "level-order" traversal

DFS/BFS Comparison

Breadth-first search:

- Always finds shortest paths, i.e., “optimal solutions”
 - Better for “what is the shortest path from \mathbf{x} to \mathbf{y} ”
- Queue may hold $O(|V|)$ nodes (e.g. at the bottom level of binary tree of height h , 2^h nodes in queue)

Depth-first search:

- Can use less space in finding a path
 - If *longest path* in the graph is \mathbf{p} and highest out-degree is \mathbf{d} then DFS stack never has more than $\mathbf{d} \cdot \mathbf{p}$ elements

A third approach: *Iterative deepening (IDDFS)*:

- Try DFS but don’t allow recursion more than \mathbf{k} levels deep.
- If that fails, increment \mathbf{k} and start the entire search over
- Like BFS, finds shortest paths. Like DFS, less space.

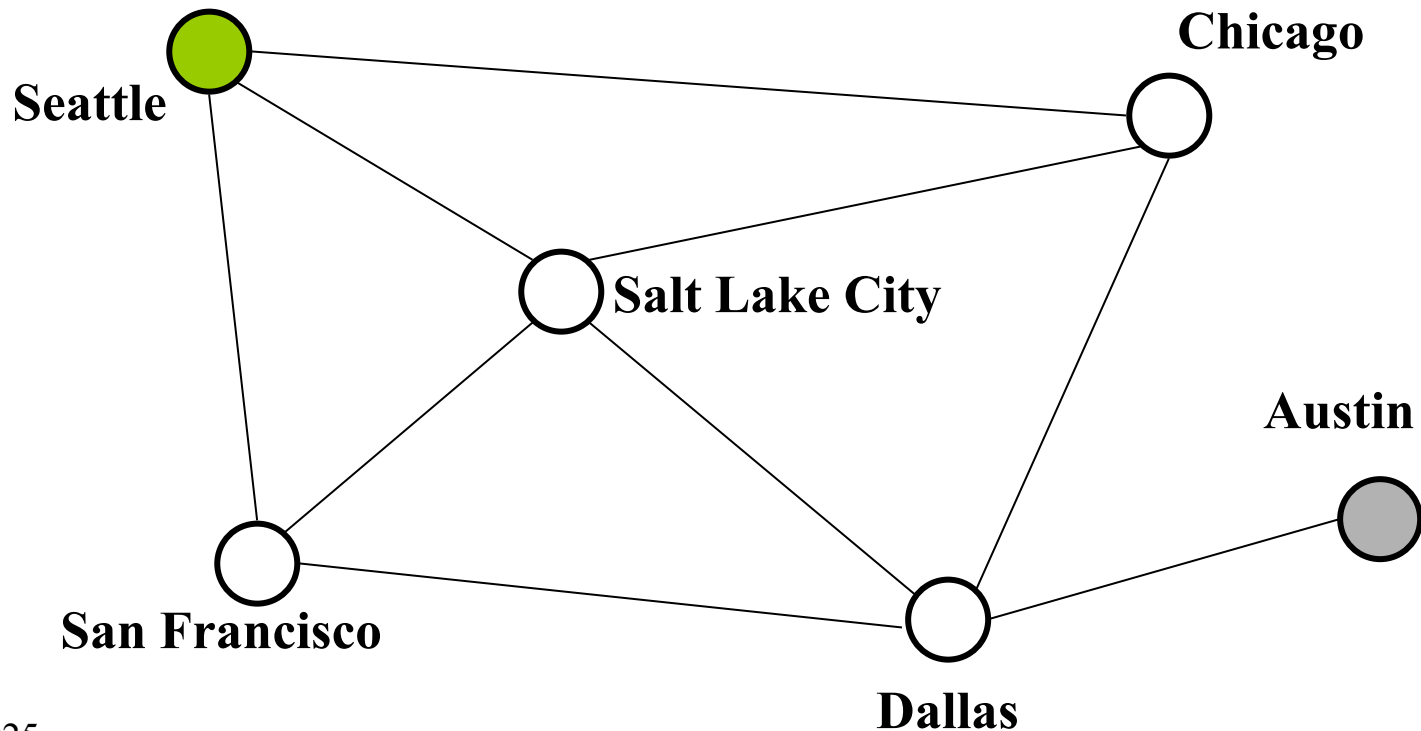
Saving the path

- Our graph traversals can answer the “reachability question”:
 - “**Is there** a path from node x to node y ?”
- Q: But what if we want to **output the actual path?**
 - Like getting driving directions rather than just knowing it’s possible to get there!
- A: Like this:
 - Instead of just “marking” a node, store the **previous node** along the path (when processing u causes us to add v to the search, set $v.pred$ field to be u)
 - When you reach the goal, follow **`pred`** fields backwards to where you started (and then reverse the answer)
 - If just wanted path *length*, could put the integer distance at each node instead

Example using BFS

What is a path from Seattle to Austin

- Remember marked nodes are not re-enqueued
- Note shortest paths may not be unique



Example using BFS

What is a path from Seattle to Austin

- Remember marked nodes are not re-enqueued
- Note shortest paths may not be unique

