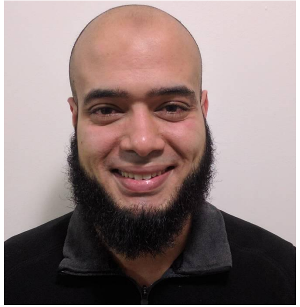
Algorithms

Graph Traversal Thoughts

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

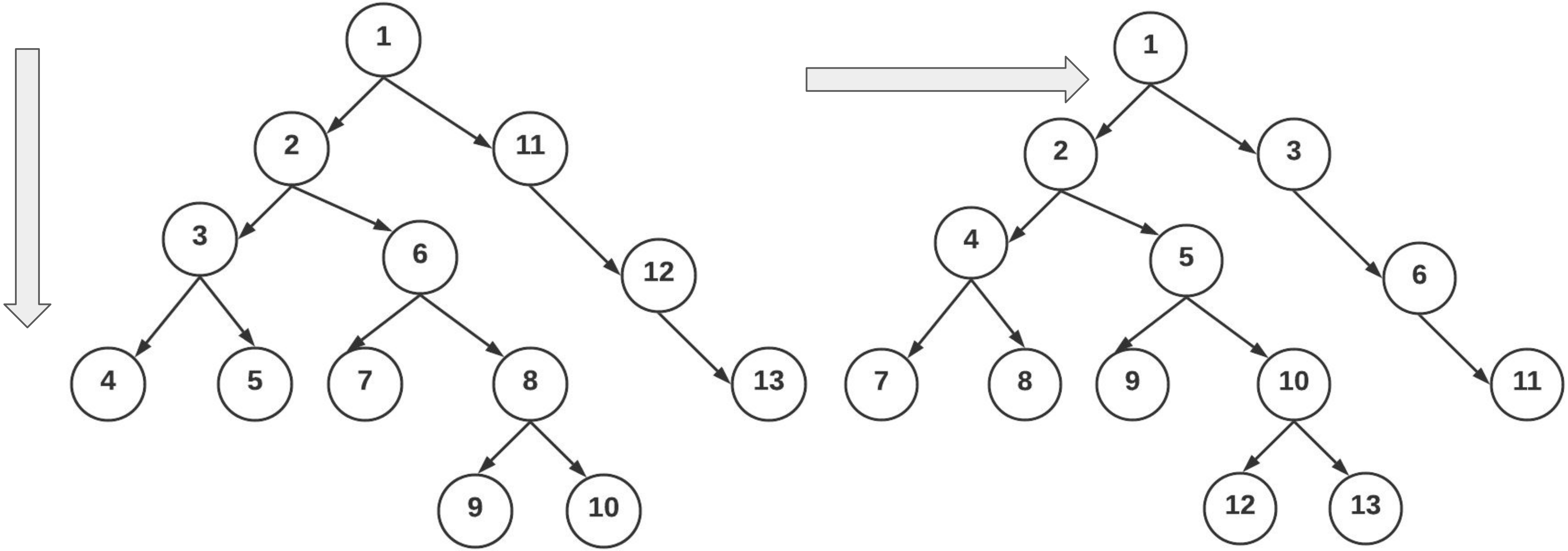


Graph Traversal

- Given a graph of N nodes, we want to visit the nodes
- 2 main traversal algorithms: DFS & BFS
- Graph theory applications:
    - Reachability, Find connected components, Detect cycles in undirected graph, Bipartite test
- Real life applications:
    - Anything that involves traversing the graph
    - Web Crawling, Garbage Collection, Social Networking, etc

DFS vs BFS: Flow

- DFS keeps going **deeper** and deeper
- BFS keeps going **level** by level



DFS vs BFS: Properties

- **Like** DFS, BFS is used to traverse a graph, and hence is applicable in many traversing problems like DFS (e.g. CC)
- **Contrary** to DFS, BFS is more efficient for **shortest paths** in unweighted graphs
- However, DFS has a lot of other **properties** that play a key role in other algorithms
  - Nodes time-stamp, Edge Classification, Topological Sort, Strongly Connected Components, Articulation edges, Euler tour
  - *Note: we did not cover these algorithms*

DFS vs BFS: Implementation

- DFS
  - A few lines of recursive code
  - Visited array to avoid cycles. But **no need for Trees**
    - Note: In DAG, we may revisit a node twice. 1-2-3, 1-4-3
  - Use the built-in stack, which is small. Be careful with dense graphs
    - We can write an iterative version using our library stack
- BFS
  - A bit longer iterative code using the queue data structure
  - Length array is used to avoid cycles & find the shortest path to a node
- Both have the same time complexity O(E+V) and space O(V)

DFS Notes

- Nature
  - The graph can be weighted. The usage depends on the problem
- Usage
  - Mainly exploring the graph (reachability, CC, finding a path)
- It has 1-M relationship (Single source to Multi-destination)
  - What if we need M-1? Reverse the graph and apply a single DFS
  - **Reverse** thinking in general
    - dfs(0) instead of DFS from each other node
      - This way we turn DFS into an **M-1 relationship**
    - Example: LeetCode 417 - Pacific Atlantic Water Flow: bfs(boundaries)
    - Make sure your code is correct after reversing the order
      - $A < B \quad \Rightarrow \quad A > B$

BFS Notes

- Nature
  - Unweighted Graph or weighted Graph with constant value (i.e. 1)
  - We learned how to code smartly to iterate level-by-level
  - Tip: Verify the starting node if you have to
- Usage
  - Mainly **shortest path** problems. Application: Tree Diameter
- 3 variants
  - 1-1, M-1, M-M relationships. All have **the same time** complicity
  - M-M is Multi-src Multi-destination
- 0-1 BFS variant (later)
  - The edge value is either 0 or 1
  - It is close to the **Dijkstra algorithm.**

The graph

- **Re**visiting a node
  - In undirected graphs, there are 2 cases
    - A real cycle 1-2-3-4-1
    - A fake cycle: going back to the parent like 1-2-1 (where no multiple edges/loops)
  - In directed graphs: No fake cycle
  - In DAGs, there are no cycles, but we still can revisit a node
    - 1-2, 2-3, 3-4, 1-5, 5-3. 3 will be visited twice!
  - In trees: there are only fake cycles in the flow, but no cycles in the graph itself
    - No need for a visited array (the only case)
- The graph can be given in a domain (employees management)
- The graph can be a 2D grid (implicit graph)
  - Use direction arrays or nested loops to generate neighbours
- Graph values are on edges, but a few times on nodes too

Reduction to graph

- The key is to figure out the nodes first, then the edges
- We met several problems where the numbers are the nodes
  - For example in fraction a/b: a and b are nodes, fraction value is the edge weight
  - In consecutive sequences: values are the nodes, edges are for val and val+1
- We met problems where strings are the nodes
  - The if operation(string1) = string2, then there is an edge between these 2 strings
  - If the operation has a cost, then this is the edge weight
- Generalization:
  - If we have a state (number, string, vector, object) and an **operation** that convert to another
  - Then states are the nodes and reductions create the edges

Reduction to graph: State Graph

- Like a state diagram, it is a graph where each node is a state
  - E.g. the node is bulb id and is it on or off **or** the node is 3 strings and integer
  - E.g. node is a vector of N values
- Your visited set will be based on the number of different states
  - E.g. If each state is 5 lowercase letters, then we have 26^5 differents states = ~11 Million
    - The good news? We usually traverse only a small subset of them (problem-based)
- Avoid converting a state graph of an explicit graph
- All that you need
  - Think how to generate edges for a given state (typically huge)
  - Write your code to be dynamic and iterate on states
- Problem example: LeetCode 1129 - Shortest Path with Alternating Colors
  - Backtracking problems are all based on states

DFS or BFS?

- This is a hard question and area of debate
- We need to know the graph properties: size, density, etc
- In large dense graphs, both may have big computational problems
- In BFS, if the target will exist at a reasonable depth, use it
- In a dense wide graph, BFS will keep adding a lot nodes: memory concern
- Be careful about DFS as it exhausts the built-in stack
- We may use IDDFS to limit the depth
- DFS might be more suitable for a general graph search
- DFS is more friendly to use in a distributed environment

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."