Lecture 23: Tree and Graph Traversals

10/19/2020

Trees and Traversals

Tree Definition

- A tree consists of:
    - A set of nodes
    - A set of edges that connect those nodes
        - Constraint: There is exactly one path between any two nodes

Rooted Trees Definition

- A rooted tree is a tree where we've chosen one node as the "root"
    - Every node N except the root has exactly one parent, defined as the first node on the path from N to the root
    - A node with no child is called a leaf

Trees

- Trees are a more general concept
    - Organizational charts
    - Family lineages

Example: File Tree System

- Sometimes you want to iterate over a tree. For example, suppose you want to find the total size of all files in a folder called 61b
    - What one might call "tree iteration" is actually called "tree traversal"
    - Unlike lists, there are many orders in which we might visit the nodes
        - Each ordering is useful in different ways

Tree Traversal Orderings

- Level Order
    - Visit top-to-bottom, left-to-right (like reading in English)
- Depth First Traversals
    - 3 types: Preorder, Inorder, Postorder
    - Basic (rough) idea: Traverse "deep nodes" before shallow ones
    - Note: Traversing a node is different than "visiting" a node

Depth First Traversals

```
preOrder(BSTNode x) {
    if (x == null) return;
```

```
    print(x.key)
    preOrder(x.left)
    preOrder(x.right)
}
```

- Preorder traversal: "Visit" a node, then traverse its children

```
inOrder(BSTNode x) {
    if (x == null) return;
    inOrder(x.left)
    print(x.key)
    inOrder(x.right)
}
```

- Inorder traversal: Traverse left child, visit, then traverse right child

```
postOrder(BSTNode x) {
    if (x == null) return;
    postOrder(x.left)
    postOrder(x.right)
    print(x.key)
}
```

- Postorder traversal: Traverse left, traverse right, then visit

A Useful Visual Trick (for Humans, not algorithms)

- Preorder traversal: We trave a path around the graph, from the top going counter-clockwise. "Visit" every time we pass the LEFT of a node
- Inorder traversal: "Visit" when you cross the bottom of a node
- Postorder traversal: "Visit" when you cross the right of a node

What Good are all These Traversals

- Example: Preorder Traversal is natural for printing directory listings
- Example: Postorder Traversal for gathering file sizes

Graphs

Trees and Hierarchical Relationships

- Trees are fantastic for representing strict hierarchical relationships
  - But not every relationship is hierarchical
  - Example: Paris Metro map
- The Paris Metro map is not a tree: It contains cycles!

Graph Definition

- A graph consists of:
  - A set of nodes
  - A set of zero or more edges, each of which connects two nodes
  - Note, all trees are graphs
- A simple graph is a graph with:
  - No edges that connect a vertex to itself, i.e. no "loops"
  - No two edges that connect the same vertices, i.e. no "parallel edges"
- In 61B, "graph" refers to "simple graph" unless otherwise stated

Graph Type

- Directed Graph
  - Each edge has a "directionality"
- Undirected Graph
  - Each edge is undirected
- Acyclic Graph:
  - A tree. A graph with no cycles
- Cyclic:
  - A graph that contains a cycle
- With Edge Labels

Graph Terminology

- Graph
  - Set of **vertices**, aka **nodes**
  - Set of **edges**: Pair of vertices
  - Vertices with an edge between them are **adjacent**
  - Vertices or edges may have **labels** (or **weights**)
- A **path** is a sequence of vertices connected by edges
- A **cycle** is a path whose first and last vertices are the same
  - A graph with a cycle is "cyclic"
- Two vertices are **connected** if there is a path between them. If all vertices are connected, we say the graph is connected

Graph Problems

Graph Queries

- There are lots of interesting questions we can ask about a graph:
  - Shortest path between two nodes
  - Longest path between two nodes without cycles
  - Is there a tour that uses each node exactly once?
  - Is there a tour that uses each edge exactly once?

Graph Queries More Theoretically

- Some well known graph problems and their common names:
  - **s-t Path**. Is there a path between vertices s and t?

- **Connectivity**. Is the graph connected, i.e. is there a path between all vertices?
- **Biconnectivity**. Is there a vertex whose removal disconnects the graph?
- **Shortest s-t Path**. What is the shortest path between vertices s and t?
- **Cycle Detection**. Does the graph contain any cycles?
- **Euler Tour**. Is there a cycle that uses every edge exactly once?
- **Hamilton Tour**. Is there a cycle that uses every vertex exactly once?
- **Planarity**. Can you draw the graph on paper with no crossing edges?
- **Isomorphism**. Are two graphs isomorphic?

- Often can't tell how difficult a graph problem is without very deep consideration

Graph Problem Difficulty

- Some well known graph problems
  - **Euler Tour**
  - **Hamilton Tour**
- Difficulty can be deceiving
  - An efficient Euler tour algorithm O(# edges) was found as early as 1873
  - Despite decades of intense study, no efficient algorithm for a Hamilton tour exists. Best algorithms are exponential time

Depth-First Traversal

s-t Connectivity

- Let's solve a classic graph problem called the s-t connectivity problem
  - Given source vertex s and a target vertex t, is there a path between s and t?
- Requires us to traverse the graph somehow
- One possible recursive algorithm for connected(s, t)
  - Does s == t? If so, return true
  - Otherwise, if connected(v, t) for any neighbor v of s, return true
  - Return false
- What is wrong with it? Can get caught in an infinite loop
- How do we fix it?
  - Mark s
  - Does s == t? If so, return true
  - Otherwise, if connected(v, t) for any unmarked neighbor v of s, return true
  - Return false
- Basic idea is same as before, but visit each vertex at most once
  - Marking nodes prevents multiple visits

Depth First Traversal

- This idea of exploring a neighbor's entire subgraph before moving on to the next neighbor is known as Depth First Traversal
  - Called "depth first" because you go as deep as possible

The Power of Depth First Search

- DFS is a very powerful technique that can be used for many types of graph problems
- Another example:
  - Let's discuss an algorithm that computes a path to every vertex
  - Let's call this algorithm DepthFirstPaths

DepthFirstPaths

- Gaol: Find a path from s to every other reachable vertex, visiting each vertex at most once. dfs(v) is as follows:
  - Mark v.
  - For each unmarked adjacent vertex w:
    - set edgeTo[w] = v
    - dfs(w)
  - Now you're left with an artifact to compute paths from s to every other reachable vertex

Tree Vs. Graph Traversals

Tree Traversals

- There are many tree traversals
  - Preorder
  - Inorder
  - Postorder
  - Level order (non depth-first traversal)

Graph Traversal

- What we just did in DepthFirstPaths is called "**DFS Preorder**"
  - **DFS Preorder: Action** is **before DFS** calls to neighbors
    - Our action was setting edgeTo
    - Preorders are equivalent to the order of dfs calls
- Could also do actions in **DFS Postorder**
  - **DFS Postorder: Action** is **after DFS** calls to neighbors
    - Equivalent to the order of dfs returns
- There are also many graph traversals, given some source:
  - DFS Preorder
  - DFS PostOrder
  - BFS order: (Breadth first search) Act in order of distance from s
    - Analogous to "level order". Search is wide, not deep

Summary

Summary

- Graphs are a more general idea than a tree
  - A tree is a graph where there are no cycles and every vertex is connected
  - Key graph terms: Directed, Undirected, Cyclic, Acyclic, Path, Cycle
- Graph problems vary widely in difficulty
  - Common tool for solving almost all graph problems is traversal

- A traversal is an order in which you visit / act upon vertices
- Tree traversals:
  - Preorder, inorder, postorder, level order
- Graph traversals:
  - DFS preorder, DFS postorder, BFS
- By performing actions / setting instance variables during a graph (or tree) traversal, you can solve problems like s-t connectivity or path finding