WEEK 4: BINARY TREES

Introduction:

This week, we concentrated on Binary Trees, a data structure of great importance to computer science. I read up on the structural characteristics of binary trees (such as the Full Binary Tree Theorem which says that a binary tree with n internal nodes has 2n + 1 total nodes). I studied tree traversals—preorder, postorder, and inorder—and why each traversal has a different algorithmic function. I also studied two ways to implement binary trees: using pointer-based nodes and arrays. Each approach has its trade-offs regarding memory and efficiency. For instance, pointer-based implementations allow dynamic memory allocation, while array-based ones work better for complete trees due to their predictable structure. I was introduced to specialized binary trees such as search trees, heaps, and priority queues. Huffman coding trees particularly intrigued me as they demonstrate how binary trees can be applied in real-world problems like data compression.

Difficulties Faced:

I'll say the hardest part of this week was wrapping my head around the different methods of traversing and which to use when. As I write recursive functions to traverse the tree, I had a hard time understanding when each node was being visited. Another challenge was identifying types of binary trees -- specifically, binary search trees, heaps, and Huffman trees. Although the programming assignment clarified some of these differences, implementing them in code without mixing up their logic required close attention. Additionally, the space complexity aspects of different implementations were a bit confusing, particularly how array-based trees manage indices for children and parents.

Activities Performed:

I participated in a discussion post where I explained the fundamentals of binary trees, such as their hierarchical nature, the roles of root, parent, and child nodes, and the concepts of depth and height. I also emphasized the utility of binary trees in organizing hierarchical data and maintaining order through structures like binary search trees (Goodrich et al., 2014). The assignment required implementing binary trees in Python, where I created a basic Node class and developed functions to insert elements and traverse the tree in preorder and postorder sequences. I also completed a self-quiz, which tested my understanding of key concepts like the Full Binary Tree Theorem, traversal orders, and the advantages of array-based implementations. These activities helped reinforce theoretical knowledge through practical application and critical thinking.

Conclusion:

This week deepened my understanding of binary trees and their broad applications. From traversals to implementations and real-world examples like Huffman coding, the topic highlighted the elegance and versatility of binary trees. Despite initial challenges, I gained confidence through coding and active participation. The structure and efficiency of binary trees make them a vital component in algorithms and data systems.

Reference:

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Python*. Wiley. https://www.wiley.com/en-us/Data+Structures+and+Algorithms+in+Python%2C+1st+Edition-p-9781118290279

Word count: 441