IUST CS Batchlor Winter 2024

Mysteries Family Tree

Data Structure Project

Teacher: Dr. Vahidi

Contributors: Ali Zargar, Aminreza Nademi

Topics discussed in the project

Secure Hashing Algorithm 2 (SHA-256) and Optimized Tree Algorithms with Different Practical Methods

Choosing Hashing Algorithm

Hashing algorithms play a crucial role in securing sensitive information in this hypothetical state by converting data into fixed-size hash values. we compared the characteristics and security considerations of LAN Manager (LM), NTLM, SHA-1, SHA-2 (specifically SHA-256), and MD5 hashing algorithms which were mentioned in the text of the project.

After evaluating the various algorithms, we choose SHA-2, specifically SHA-256, for the following reasons:

Security: SHA-256 offers a higher level of security compared to the deprecated SHA-1 and MD5. It is currently considered secure for most cryptographic applications.

Resistance to Attacks: SHA-2 is resistant to collision attacks, providing a robust defense against unauthorized access and data manipulation.

Contemporary Standards: SHA-2 is widely accepted and used in modern security protocols, making it a standard choice for contemporary projects.

Versatility: SHA-2 is versatile and suitable for a wide range of applications, including password storage, data integrity verification, and cryptographic operations.

By selecting SHA-2 (SHA-256) for our project, we prioritize security and align with current best practices in cryptography.

Tree Algorithms Implantations

Overview

In this part we provide an overview of the algorithms and data structures used in the implementation of the project. It aims to explain the reasoning behind the selection of each algorithm and their typical advantages over alternative approaches.

Tree Structure

Choice of Tree Structure

The project utilizes a tree data structure to represent hierarchical relationships among entities. The decision to use a tree structure is driven by the following considerations:

Hierarchy Representation: Trees naturally represent hierarchical relationships, making them suitable for modeling structures like genealogies or organizational hierarchies.

Ease of Traversal: Tree structures facilitate efficient traversal and exploration of hierarchical relationships, providing a convenient way to navigate from ancestors to descendants.

Tree Node Implementation

The “TreeNode” class is chosen to represent each node in the tree. This decision is based on:

Node Characteristics: The “TreeNode” class encapsulates essential information about each node, such as its data, parent, and children, making it a comprehensive representation.

Customization: The class allows for easy customization and extension as per the specific requirements of the project.

Tree Operations

Addition and Deletion of Nodes

The “addNode” method is employed for adding new nodes to the tree. This method is chosen due to its simplicity and efficiency. It ensures that duplicate nodes are not added and maintains the hierarchical structure.

The “delNode” method is selected for removing nodes from the tree. It efficiently removes the specified node and the vector of its children, ensures that the tree structure is properly updated.

Traversal and Exploration

The “findNode” method is utilized for locating a specific node within the tree by a depth-first search (DFS). Its recursive nature enables efficient traversal and exploration of the tree structure.

The “furthestNode” method is also utilized for locating a specific node within the tree but it works with BFS. It locates the furthest node from another node.

Tree Analysis

Size and Height

The “getSize” and “findHeight” methods are employed for analyzing the total number of nodes and the height of length of the longest path from the root to a leaf node, respectively. These methods provide valuable insights into the overall structure of the tree.

Both methods utilize a recursive approach; “getSize” visiting each node and incrementing a counter, ensures that every node is counted exactly once and exploring each branch and determining the height of the subtrees.

It should be noted that “findHeight” method is used indirectly to find the distance of the farthest offspring from a particular node, considering that node as the root.

Relationship Analysis

The “isAncestor”, “areSiblings”, “areCousins” and “lowestCommonAncestor” methods are used for analyzing relationships within the tree. These methods offer functionality for determining ancestor-descendant connections and common relationships in a real family.

1. The “isAncestor” method uses a recursive approach to traverse the tree, efficiently checking for ancestor relationships. This recursive nature aligns with the hierarchical structure of the tree, making it a suitable choice.

2. The “areSiblings” method assumes constant-time access to the parent's children, making it suitable for scenarios where the average number of children per node is relatively small and constant.

3. The “areCousins” method efficiently checks if two nodes have the same depth, indicating a cousin relationship. It avoids redundant traversal of the entire tree by checking the depths of the specified nodes directly. This enhances the efficiency of the algorithm.

4. The “lowestCommonAncestor” method employs an iterative approach to efficiently traverse the tree. By moving upwards from the two specified nodes simultaneously, it identifies the point where their paths diverge, marking the lowest common ancestor.

The method minimizes the search space by comparing the depths of the two nodes and adjusting the traversal accordingly. This optimization reduces unnecessary traversal.

Also, it adapts to scenarios where the two specified nodes may be at different depths within the tree. It adjusts the traversal paths to align with the hierarchical structure, making it versatile for various tree configurations.

Family Analysis

The “diameter” method employs an approach to traverse the tree and find the two family members that are have furthest relation with each other. By finding the furthest person from the great ancestor and then finding the furthest person from that individual again. It can be proved that these two people are furthest individuals from each other.

Visualizing the tree

A python script is also defined to use “matplotlib” library to plot the family tree hash values. Since the used hash function creates very different hash values, the first 10 character of hash values, in our work scale, will be all different with no collisions, so plotting the first 10 character of each individual will be sufficient to our work.

Conclusion

The choice of algorithms and data structures in the project is driven by their suitability for representing and analyzing hierarchical relationships within a tree structure. The selected methods prioritize simplicity, efficiency, and ease of customization, ensuring the project's overall effectiveness.

For the latest updates and access to the source code, please refer to our GitHub repository: [https://github.com/aminnademi/family-tree-project]."