Lab 5: Tree

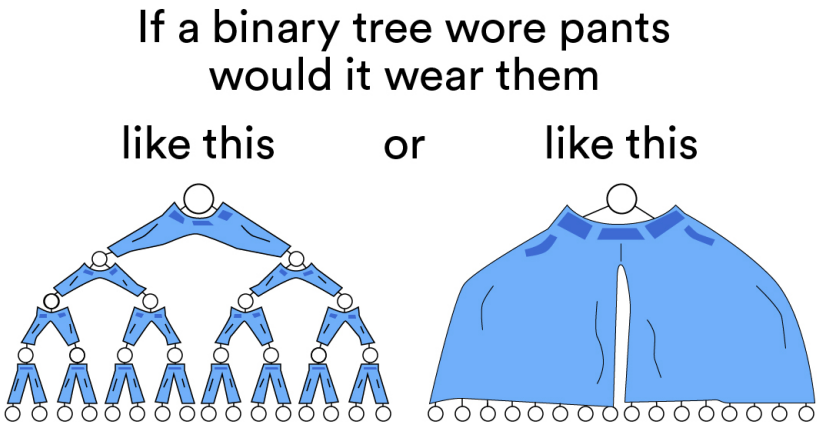

Figure 1: Good luck!

1 Exercise

1. Implement a binary tree class with methods for insertion, deletion, and searching for values.
2. Write functions to perform pre-order, in-order, post-order, and level-order traversals of a binary tree.
3. Extend the binary tree class to implement a binary search tree with methods for finding minimum and maximum values, checking if the tree is balanced, and calculating its height.
4. Write a function to calculate the diameter (the longest path between any two nodes) of a binary tree.
5. Write a function to calculate the height of a binary tree.
6. Implement a function to determine if a binary tree is height-balanced (where the height of the two subtrees of any node differs by no more than one).
7. Write a function to find the lowest common ancestor of two given nodes in a binary search tree.
8. Implement level-order traversal of a binary tree using a queue and return the values in each level as separate lists.
9. Write a function to find the maximum path sum in a binary tree, where the path may start and end at any node.
10. Given arrays representing the inorder and postorder traversals of a binary tree, reconstruct the original tree.
11. Implement vertical order traversal of a binary tree and return the values in vertical slices.

PREPARED BY Tran Vinh Khiem

12. Write an algorithm to find the lowest common ancestor (LCA) of two nodes in a binary tree.
13. Given two binary trees, merge them into one by adding overlapping nodes together.

Note

- Develop a visualization tool that graphically represents different searching algorithms (e.g., binary search, linear search) in action. Use animations to show how elements are compared and swapped during the sorting process. This can help in understanding the mechanics behind each algorithm. Allow users to input data and visualize the steps taken during the search process.
- Each sorting algorithm including buble sort, selection sort, insertion sort, binary insertion sort, quick sort, merge sort, heap sort

2 Homework

1. Implement a tree structure to represent a file system, where each node represents a file or directory. Allow users to navigate through the file system, create new files/directories, and delete existing ones.
2. Create a parser that reads an XML or HTML document and constructs a tree structure representing its elements. Implement methods to query specific elements or attributes from the tree.
3. Implement Huffman coding using binary trees to compress text data. Create functions for building the Huffman tree and encoding/decoding messages.
4. Develop a simple two-player game (like Tic-Tac-Toe) using a minimax algorithm represented as a game tree. Allow players to make moves and compute the best possible move for the AI.
5. Create an expression parser that converts infix expressions into syntax trees (abstract syntax trees) and evaluates them. Support basic arithmetic operations.
6. Build a Merkle tree to verify the integrity of data blocks in a distributed system. Implement functions to generate hashes and verify data authenticity using the root hash.
7. Design an organizational chart using a tree structure where each node represents an employee and its children represent subordinates. Implement methods for adding/removing employees and displaying hierarchy.
8. Merkle trees are used in blockchain to verify the integrity of data blocks efficiently by storing hashes of transactions in a hierarchical structure. Implement this tree in blockchain.
9. Phylogenetic trees represent evolutionary relationships between species or genes based on genetic data. They help researchers understand biodiversity and evolutionary history. Implement this tree.
10. Tries are used for storing a dynamic set of strings where common prefixes are shared. They enable fast retrieval operations such as autocomplete and spell checking. Implement this tree.

3 Additional homework for review test

1. Implement a graph using both an adjacency list and an adjacency matrix. Include methods for adding vertices and edges.
2. Write a function to perform depth-first search on a graph. Implement it using both recursive and iterative approaches.
3. Implement breadth-first search on a graph and return the order of traversal. Use a queue to manage the nodes.
4. Write a function to detect cycles in a directed graph using DFS. Extend this to check for cycles in an undirected graph.
5. mplement Dijkstra's algorithm to find the shortest path from a source vertex to all other vertices in a weighted graph.
6. Write a function to find all connected components in an undirected graph using DFS or BFS.
7. Implement an algorithm to find all bridges (edges whose removal increases the number of connected components) in an undirected graph.
8. Implement a community detection algorithm (e.g., Louvain method) to identify clusters within a social network graph.
9. Implement the PageRank algorithm to rank web pages based on their importance in a directed graph.
10. Implement Dijkstra's algorithm using a priority queue for efficient shortest path calculation in a weighted graph.
11. Implement a route planning algorithm for a delivery service to minimize travel time or distance between multiple delivery points using Dijkstra's or A* algorithm.
12. Create a model of a social network (e.g., Facebook or Twitter) where users are nodes and friendships are edges. Analyze the network to find influential users or communities using centrality measures or community detection algorithms.
13. Model a city's road network as a graph and use graph algorithms to identify traffic bottlenecks and suggest optimal traffic light timings or alternative routes.
14. Build a recommendation system for an e-commerce platform using collaborative filtering based on user-item interaction graphs to suggest products to users.
15. Model a computer network as a graph where nodes represent devices and edges represent connections. Use algorithms to optimize the network topology for efficiency and reliability.
16. Implement pathfinding algorithms (like A*) for NPCs (non-player characters) in a game environment represented as a grid graph to navigate obstacles efficiently.

Notice

- Use C++ for practice.
- In the programming file, the student should include the following complete information:

```
//STT: 39 (Example)
//Full Name: X, With X is you, don't need to find X anywhere else.
//Session 01 - Exercise 01
//Notes or Remarks: ......
```

- Develop a visualization tool that graphically represents different exercises in action. Allow users to input data and visualize the steps taken during the search process.
- Students can use Google colab for homework.

References :

[1]. Skiena, S. S. (1998). The algorithm design manual (Vol. 2). New York: springer.
[2]. Pai, G. V. (2023). A Textbook of Data Structures and Algorithms, Volume 3: Mastering Advanced Data Structures and Algorithm Design Strategies. John Wiley & Sons.
[3]. Anggoro, W. (2018). C++ Data Structures and Algorithms: Learn how to write efficient code to build scalable and robust applications in C++. Packt Publishing Ltd.
[3].Leetcode
[4].Codeforce
[5]. https://www.youtube.com/watch?v=RfXt_qHDEPw