Understanding Tree Data Structures: Concepts, Types, Operations, and Applications

1. Introduction to Tree Data Structures

In the realm of computer science, a tree data structure stands out as a fundamental and versatile method for organizing information. Unlike linear data structures such as arrays or linked lists, a tree adopts a non-linear, hierarchical structure to represent the relationships between data elements.[1] At its core, a tree comprises a collection of interconnected nodes, linked by pathways known as edges, all originating from a singular point referred to as the root.[3] This hierarchical arrangement effectively mirrors many real-world scenarios, from organizational charts to file systems.[1]

The very definition of a tree often takes a recursive form, emphasizing its inherent hierarchical nature. A tree is considered either an empty entity or a structure consisting of a root node accompanied by zero or more disjoint substructures, each of which is also a tree in its own right.[3] This recursive perspective is not merely a theoretical construct; it profoundly influences how many algorithms for processing trees are designed and analyzed.

The significance of tree structures within computer science is multifaceted. Their hierarchical organization allows for efficient data storage and, more importantly, rapid retrieval.[1] By structuring data in a tree-like manner, searches can often be conducted much faster than in linear arrangements, where every element might need to be examined.[1] Furthermore, trees are exceptionally adept at portraying inherent hierarchical relationships that exist between different pieces of data.[1]

This report will delve into various types of tree data structures, each tailored for specific applications and offering unique characteristics. We will begin by examining the foundational concepts of binary trees and their common variations, including threaded binary trees, binary search trees, and AVL trees.[1] Following this, we will explore the specialized B Trees and B+ Trees, which are particularly useful in scenarios involving large datasets and disk-based storage. Finally, the report will provide a comparative analysis of these different tree types, highlighting their respective advantages, disadvantages, and the typical contexts in which they are employed.

The recursive definition of a tree is a cornerstone concept that permeates many aspects of tree manipulation.[3] This definition, where a tree is either nothing or a root with subtrees that are also trees, suggests a natural approach to solving problems related to trees: recursion. Many algorithms for traversing, searching, or modifying trees are most elegantly and efficiently expressed through recursive functions. The recursive calls mirror the hierarchical structure of the tree itself, processing a node and then recursively processing its children. This approach also has implications for the analysis of an algorithm's performance, particularly in terms of time and space complexity, as the depth of recursion can affect the call stack and the number of operations performed.

Furthermore, the fundamental difference between linear and non-linear data structures underscores the unique role of trees.[1] Linear structures arrange data in a sequential manner, making them suitable for representing lists or sequences. However, they often fall short when it comes to depicting inherent hierarchical relationships that exist in many datasets. Trees, with their branching structure and parent-child relationships, are specifically designed to capture these hierarchies. This makes them indispensable in applications where such relationships are central, such as in organizing files within a file system, representing the structure of documents like XML or HTML, or modeling decision-making processes.

2. Fundamental Tree Terminologies

To effectively discuss and understand tree data structures, it is essential to establish a common vocabulary. Several fundamental terminologies are used to describe the components and relationships within a tree.[1]

The **root** of a tree is the distinguished topmost node. It serves as the origin of the entire structure and is unique in that it does not have a parent node.[12] A **node** is a basic building block of a tree. It can hold data and may have connections, or edges, to other nodes in the tree.[11] An **edge** represents a connection or a link between two nodes, signifying a relationship between them.[11]

The hierarchical nature of a tree gives rise to the concepts of **parent** and **child** nodes. A **parent** node is one that has one or more nodes directly below it in the hierarchy; these are its children.[18] Conversely, a **child** node is directly connected to and situated below its parent.[18] A **leaf** node, also known as an external or terminal node, is a node that does not have any children.[1] A **subtree** is a self-contained portion of a tree, consisting of a node and all of its descendants.[2] Every node in a tree can be considered the root of its own subtree.

The **depth** of a node refers to the number of edges along the path from the root node to that specific node.[319] The root node, by definition, has a depth of 0. The **height** of a node, on the other hand, is the number of edges on the longest path from that node down to a leaf node in its subtree.[119] The height of a tree is typically defined as the height of its root node. Leaf nodes have a height of 0.

Beyond these fundamental terms, several other terminologies are commonly used. **Siblings** are nodes that share the same parent.[1] An **ancestor** of a node is any node that lies on the path from the root to that node.[2] Conversely, a **descendant** of a node is any node that can be reached by following a path from that node downwards in the tree.[1] An **internal node** is any node that has at least one child [2], while an **external node** is a node with no children, synonymous with a leaf node.[2] The **degree** of a node is the number of children it has, and the degree of a tree is the maximum degree of any node within the tree.[3] The **level** of a node indicates its generation in the tree, with the root at level 0, its children at level 1, and so on.[1] A **path** in a tree is a sequence of nodes and edges that connects a node to one of its descendants.[1] Finally, a **forest** is a collection of one or more disjoint trees.[6]

The consistency in the definitions of these terminologies across various sources underscores the standardized vocabulary used in the field of data structures.[1] This uniformity is crucial for clear communication and a shared understanding of tree concepts. The redundancy in these fundamental definitions across multiple snippets highlights their importance and foundational nature to the study of tree data structures.

3. Binary Trees

3.1 Definition and Properties

A binary tree is a specialized type of tree in which each node can have at most two children. These children are typically designated as the left child and the right child.[1] From a recursive standpoint, a binary tree can be defined as either an empty tree or a tree consisting of a root node that has a left binary subtree and a right binary subtree, which are themselves disjoint from each other and from the root.[18] This recursive definition highlights the fundamental structure of binary trees and is key to understanding many of their properties and associated algorithms.

Binary trees possess several important properties. At any given level 'l' in a binary tree, the maximum number of nodes that can exist is 2<sup>l</sup>.[33] Furthermore, in a binary tree of height 'h' (where the height of the root is considered 0), the maximum total number of nodes is 2<sup>h+1</sup> - 1.[33] These properties are particularly relevant when analyzing the efficiency and capacity of binary trees in various applications.

Within the broader category of binary trees, several distinct types exist, each with its own specific characteristics. A **full binary tree** is one where every node has either zero or exactly two children.[1] This implies that there are no nodes with only one child in a full binary tree. In contrast, a **complete binary tree** has all levels completely filled, with the possible exception of the last level, which is filled from left to right.[1] A **perfect binary tree** is a special case of a complete binary tree where all internal nodes have exactly two children, and all leaf nodes are at the same level.[1]

The balance of a binary tree is also an important consideration. A **balanced binary tree** is one where the height difference between the left and right subtrees of any node does not exceed a certain limit. AVL trees, which will be discussed later, are an example of a strictly balanced binary tree.[1] On the other hand, a **skewed binary tree** is one where each internal node has only one child, resulting in a structure that resembles a linked list (either left-skewed or right-skewed).[5] Similarly, a **degenerate** or **pathological binary tree** is also characterized by each parent node having only one child, leading to a linear structure.[2]

3.2 Binary Tree Operations and Algorithms

Binary trees support several fundamental operations that allow for the manipulation and retrieval of data stored within them. These operations include insertion, deletion, searching, and traversal.[2]

Insertion into a general binary tree typically involves finding an appropriate position to add a new node. This position might be determined by criteria such as maintaining a level order in a complete binary tree or adhering to specific ordering rules in specialized binary trees like binary search trees.[2] The complexity of insertion can vary. If the desired position for insertion is already known, the operation can be performed in constant time, O(1). However, in the worst-case scenario, such as inserting into a skewed tree, it might require traversing all n nodes, resulting in a time complexity of O(n).[41] For balanced binary search trees, the average time complexity for insertion is typically O(log n).[41]

Deletion of a node from a binary tree is more complex and depends on the characteristics of the node being deleted.[2] If the node to be deleted is a leaf node (has no children), it can simply be removed. If the node has one child, it can be removed, and its child takes its place in the tree. The most intricate case arises when the node to be deleted has two children. In this scenario, the node is typically replaced by either its in-order predecessor (the largest value in its left subtree) or its in-order successor (the smallest value in its right subtree), and then the predecessor or successor is deleted from its original position. Similar to insertion, the time complexity for deletion in a binary tree can be O(n) in the worst case, but averages to O(log n) for balanced binary search trees.[41]

Searching for a specific value within a binary tree involves starting from the root node and traversing down the tree. At each node, the value being searched for is compared with the value of the current node. Based on this comparison (and the specific rules of the binary tree type, if any), the search proceeds to either the left or the right subtree.[2] This process continues recursively until the target value is found or a leaf node is reached, indicating that the value is not present in the tree. The time complexity for searching in a binary tree can be O(n) in the worst case (e.g., if the tree is skewed or the target is the last node visited), but for a balanced binary search tree, the average and worst-case time complexity is O(log n).[41]

Traversal refers to the process of visiting each node in the tree exactly once in a systematic manner. There are several standard traversal methods for binary trees:

- **In-order Traversal (LNR):** This method visits the left subtree first, then the root node, and finally the right subtree. For a binary search tree, in-order traversal visits the nodes in ascending order of their values.[34]
- **Pre-order Traversal (NLR):** In this traversal, the root node is visited first, followed by the left subtree, and then the right subtree. Pre-order traversal is often used for creating a prefix expression from an expression tree or for making a copy of the tree.[34]
- **Post-order Traversal (LRN):** This method visits the left subtree first, then the right subtree, and finally the root node. Post-order traversal is useful in scenarios such as evaluating postfix expressions or deleting an entire tree.[34]
- **Level-order Traversal (Breadth-First):** This traversal visits nodes level by level, starting from the root and moving from left to right at each level. Level-order traversal typically utilizes a queue data structure to keep track of the nodes to be visited.[34]

The time complexity for all these traversal methods is O(n), where n is the number of nodes in the tree, as each node must be visited exactly once.[53] The space complexity is typically O(h), where h is the height of the tree, due to the recursive calls or the use of a queue (for level-order traversal).[43] In the worst case (e.g., a skewed tree), the height can be n, leading to a space complexity of O(n). For a balanced tree, the height is O(log n), resulting in a space complexity of O(log n).

4. Threaded Binary Trees

4.1 Definition and Properties

A threaded binary tree is a variation of a binary tree that aims to enhance the efficiency of in-order traversal by utilizing the null pointers that often exist in standard binary trees.[54] In a typical binary tree, leaf nodes have both left and right child pointers as null, and even internal nodes might have one or both child pointers as null. A threaded binary tree repurposes these null pointers to store addresses of other nodes in the tree, specifically the in-order predecessor or the in-order successor.[54] These special pointers are called threads.

There are two main types of threaded binary trees:

- **Single Threaded:** In this type, either the left null pointer of a node is made to point to its in-order predecessor (if it exists, forming a right-threaded tree), or the right null pointer is made to point to its in-order successor (if it exists, forming a left-threaded tree).[55]
- **Double Threaded:** In a double-threaded binary tree, both the left null pointer points to the in-order predecessor and the right null pointer points to the in-order successor (if they exist).[55] This allows for bidirectional traversal without the need for a stack or recursion.

To distinguish between a regular child pointer and a thread, an additional boolean flag is often included in each node.[55] This flag indicates whether the left and right pointers are pointing to actual children or to their in-order predecessor or successor, respectively.

The primary advantage of using a threaded binary tree is the ability to perform in-order traversal more efficiently, specifically in O(n) time, without the need for an auxiliary stack or recursion.[55] This can be particularly beneficial in memory-constrained environments where the overhead of stack space for recursive calls is undesirable.[55] Additionally, threaded binary trees enable finding the in-order predecessor and successor of a node more readily.[55]

However, threaded binary trees also come with certain drawbacks. They require extra space in each node to store the thread flags.[55] Moreover, the algorithms for insertion and deletion become more complex as they need to handle the maintenance of both the regular tree structure and the threads.[55]

4.2 Threaded Binary Tree Operations and Algorithms

The operations on a threaded binary tree differ from those on a standard binary tree due to the presence of threads. The most significant impact is on the traversal operation.

Traversal: In-order traversal of a threaded binary tree can be done iteratively. Starting from the leftmost node of the tree, the in-order successor can be found by following the right child pointer. If the right child pointer is a thread, it directly points to the successor. If it is a regular child pointer, then the successor is the leftmost node in the right subtree.[55] This process continues until all nodes have been visited. The time complexity for this traversal is O(n), and the space complexity is O(1) as no extra stack is required.[55]

Insertion: Inserting a new node into a threaded binary tree requires careful consideration of the threads. The process involves first finding the correct position for the new node, similar to insertion in a binary search tree (if the threaded tree maintains a sorted order). Once the position is found, the new node is inserted, and the threads of the new node and its parent need to be appropriately set to maintain the in-order sequence.[55] The complexity of insertion is generally O(h), where h is the height of the tree, similar to standard binary search tree insertion.[63]

Deletion: Deleting a node from a threaded binary tree is also more involved than in a regular binary tree due to the need to update the threads of the predecessor and successor of the deleted node.[55] The specific steps depend on whether the node to be deleted has zero, one, or two children, and how the threads are configured. The time complexity for deletion is typically O(h).[63]

Searching: Searching for a node in a threaded binary tree can be similar to searching in a standard binary tree, especially if the tree is also a binary search tree.[55] The threads themselves do not directly aid in the search process based on value, but they can be useful for navigating the tree once a node is found. The time complexity for searching is O(h) in the worst case.[63]

5. Binary Search Trees (BSTs)

5.1 Definition and Properties

A binary search tree (BST) is a specific type of binary tree that adheres to a particular ordering property, making it efficient for searching, insertion, and deletion operations.[41] This property states that for every node in the tree:

- The key (or value) of the node is greater than all the keys in its left subtree.
- The key of the node is less than all the keys in its right subtree.
- Both the left and right subtrees must also be binary search trees.[41]

This ordering ensures that an in-order traversal of a BST will yield the nodes in sorted order of their keys.[41]

Key properties of a binary search tree include the efficient average-case time complexity of O(log n) for search, insertion, and deletion operations, assuming the tree remains relatively balanced.[41] However, if the BST becomes skewed (e.g., due to insertions in sorted order), the performance can degrade to O(n) in the worst case, resembling a linked list.[41]

5.2 Binary Search Tree Operations and Algorithms

The ordering property of a BST dictates the algorithms for its fundamental operations.

Insertion: To insert a new key into a BST, we start at the root and compare the new key with the root's key.[41] If the new key is less than the current node's key, we move to the left child; if it's greater, we move to the right child. We continue this process until we reach a null pointer, which indicates the position where the new node should be inserted as a leaf.[41] The time complexity for insertion is O(h), where h is the height of the tree, averaging O(log n) for a balanced BST and O(n) for a skewed BST.[41]

Deletion: Deleting a node from a BST involves three main cases [41]:

- **Case 1: Deleting a leaf node:** If the node to be deleted has no children, it can be simply removed by setting the corresponding child pointer of its parent to null.[41]
- **Case 2: Deleting a node with one child:** If the node has only one child, it is removed, and its child takes its place by being directly linked to the deleted node's parent.[41]
- **Case 3: Deleting a node with two children:** If the node has two children, it is typically replaced by its in-order successor (the smallest node in its right subtree) or its in-order predecessor (the largest node in its left subtree). The successor or predecessor is then deleted from its original position, which falls under either Case 1 or Case 2.[41]

The time complexity for deletion is also O(h), averaging O(log n) for balanced BSTs and O(n) for skewed BSTs.[41]

Searching: Searching for a key in a BST starts at the root and proceeds by comparing the target key with the key of the current node.[40] If the target key is equal to the current key, the search is successful. If the target key is less than the current key, the search continues in the left subtree. If the target key is greater, the search continues in the right subtree. This process repeats until the key is found or a null pointer is encountered, indicating that the key is not in the tree. The time complexity is O(h), averaging O(log n) for balanced BSTs and O(n) for skewed BSTs.[41]

Traversal: BSTs can be traversed using the standard binary tree traversal methods (in-order, pre-order, post-order, and level-order), each with a time complexity of O(n).[40] As mentioned earlier, in-order traversal of a BST yields the nodes in sorted order.

6. AVL Trees

6.1 Definition and Properties

An AVL tree, named after its inventors Adelson-Velsky and Landis, is a self-balancing binary search tree.[14] The key characteristic of an AVL tree is that the heights of the two child subtrees of any node differ by at most one.[70] This strict balancing condition ensures that the height of the AVL tree remains logarithmic with respect to the number of nodes (O(log n)), which in turn guarantees efficient performance for search, insertion, and deletion operations in both average and worst-case scenarios.[45]

To maintain this balance, each node in an AVL tree stores a balance factor, which is the difference between the height of its left subtree and the height of its right subtree.[8] The balance factor of any node in an AVL tree must be either -1, 0, or 1.[73] If the balance factor of a node becomes +2 or -2 after an insertion or deletion, the tree performs rotations to restore the balance.[70]

6.2 AVL Tree Operations and Algorithms

The operations on an AVL tree are similar to those of a binary search tree, but with the added step of maintaining the balance of the tree after each insertion or deletion.

Insertion: Insertion in an AVL tree starts with a standard BST insertion.[70] After the new node is inserted, the balance factor of each ancestor node is checked, starting from the newly inserted node and moving up to the root.[70] If any node's balance factor becomes +2 or -2, rotations are performed to rebalance the subtree rooted at that node.[70] There are four types of rotations: left rotation, right rotation, left-right rotation, and right-left rotation, which are applied based on the specific imbalance.[70] The time complexity for insertion in an AVL tree is O(log n).[45]

Deletion: Deletion in an AVL tree also begins with a standard BST deletion.[70] After a node is deleted, the balance factors of the ancestor nodes are checked and updated, and rotations are performed if necessary to maintain the AVL property.[70] Similar to insertion, the rebalancing might involve one or more rotations. The time complexity for deletion in an AVL tree is also O(log n).[45]

Searching: Searching in an AVL tree is identical to searching in a binary search tree, as an AVL tree is a type of BST.[73] Due to the balanced nature of the AVL tree, the search operation has a time complexity of O(log n) in all cases.[45]

Traversal: AVL trees can be traversed using the same in-order, pre-order, post-order, and level-order traversal methods as standard binary trees, each with a time complexity of O(n).[73] The balancing property of AVL trees does not affect the traversal algorithms or their complexity.

7. Applications of Binary Trees

Binary trees, in their various forms, find extensive applications in computer science and other domains due to their hierarchical structure and efficiency in certain operations.[36]

One of the primary applications is in **efficient data organization and retrieval**, particularly through the use of binary search trees.[85] BSTs allow for quick searching, insertion, and deletion of data while maintaining a sorted order, making them suitable for implementing associative arrays, maps, and sets.[86] Balanced BSTs like AVL trees and red-black trees further enhance this efficiency by guaranteeing logarithmic time complexity for these operations.[86]

Binary trees are also used in **representing and evaluating expressions**, known as expression trees.[9] In such trees, internal nodes represent operators, and leaf nodes represent operands. This structure allows for the recursive evaluation of complex arithmetic expressions.[85]

Another significant application is in **data compression**, specifically with Huffman coding.[36] Huffman trees are binary trees constructed based on the frequency of characters in a text, allowing for the creation of efficient variable-length codes for data compression.[36]

Decision trees, used in machine learning and artificial intelligence, are also a form of binary tree.[1] These trees model decision-making processes by posing yes/no questions at each internal node, with the leaves representing the outcomes or classifications.[1]

Binary trees play a crucial role in the implementation of **heaps**, which are specialized tree-based data structures that satisfy the heap property.[9] Heaps are often used to implement priority queues, which are essential in various applications such as operating system scheduling and graph algorithms.[10]

Other applications of binary trees include their use in **representing hierarchical data** such as file systems and organizational charts [1], in **parsing XML and HTML documents** via the Document Object Model (DOM) [86], in **routing algorithms** in computer networks [86], and in specialized data structures like **binary tries** used in high-bandwidth routers.[87]

8. B Trees

8.1 Definition and Properties

A B-tree is a self-balancing tree data structure that is designed to efficiently handle large amounts of data, particularly in disk-based storage systems.[4] Unlike binary trees where each node has at most two children, a B-tree can have multiple children per node, ranging between a defined minimum and maximum number.[45] This property, known as the order or branching factor of the tree, allows B-trees to maintain a shallow height even when storing a vast number of keys, thus minimizing the number of disk accesses required for operations.[99]

According to a common definition, a B-tree of order *m* satisfies the following properties [99]:

- Every node has at most *m* children.
- Every non-leaf node (except the root) has at least $\lceil m/2 \rceil$ children.
- The root node has at least two children unless it is a leaf node.
- All leaf nodes appear on the same level.
- A non-leaf node with *k* children contains *k*-1 keys.

B-trees keep keys in sorted order within each node, facilitating efficient searching.[99] They use a hierarchical index to minimize disk reads and employ partially full blocks to speed up insertions and deletions.[99] The tree's balance is maintained through recursive algorithms that adjust the structure as needed.[99]

8.2 B Tree Algorithms and Analysis

B-trees support the fundamental operations of searching, insertion, and deletion, all with a time complexity of O(log n), where n is the number of keys in the tree.[45]

Searching: The search operation in a B-tree starts at the root and proceeds downwards. At each node, the target key is compared with the keys present in the node. If the key is found, the search is successful. If not, the search continues in the appropriate child node based on the range of keys in the current node.[99] Within each node, a binary search can be performed on the sorted keys to efficiently determine the next child to follow.[99]

Insertion: To insert a new key, the B-tree is traversed to find the leaf node where the key should reside.[99] If the leaf node has space, the new key is inserted in sorted order. If the leaf node is full, it is split into two nodes, and the median key is promoted to the parent node. This process might propagate up the tree, potentially leading to the splitting of internal nodes and even the root.[99]

Deletion: Deleting a key involves first searching for the key to be deleted.[99] If the key is in a leaf node, it is simply removed. If the key is in an internal node, it is replaced by its predecessor or successor from the subtrees, and that predecessor or successor is then deleted. If deleting a key causes a node to have fewer than the minimum number of keys, keys might be borrowed from sibling nodes, or the node might be merged with a sibling to maintain the B-tree properties.[99]

9. B+ Trees

9.1 Definition and Properties

A B+ tree is an extension of the B-tree data structure that is also self-balancing and widely used in database management systems and file systems for efficient indexing and data retrieval.[4] The key difference between a B-tree and a B+ tree is that in a B+ tree, all the actual data records (or pointers to them) are stored only in the leaf nodes.[109] The internal nodes of a B+ tree contain only keys and pointers to child nodes; they do not store any data records.[109]

Another significant feature of a B+ tree is that the leaf nodes are typically linked together in a sorted linked list.[117] This linking allows for efficient sequential access to the data records, which is particularly useful for range queries.[117]

Similar to B-trees, B+ trees have an order *m* that defines the maximum number of children an internal node can have.[116] For a B+ tree of order *m*:

- Each internal node can have at most *m* children.
- Each internal node (except the root) must have at least $\lceil m/2 \rceil$ children.
- Each internal node can have at most m-1 keys and at least $\lceil m/2 \rceil$-1 keys.
- Each leaf node can have between $\lceil m/2 \rceil$ and *m* values (data records or pointers).

All leaf nodes are at the same depth, ensuring the tree remains balanced.[116]

9.2 B+ Tree Algorithms and Analysis

B+ trees support search, insertion, and deletion operations with an average and worst-case time complexity of O(log n).[107]

Searching: Searching in a B+ tree starts at the root and traverses down to the appropriate leaf node by comparing the target key with the keys in the internal nodes.[116] Once the leaf node is reached, a search is performed within the leaf node to find the desired data record.[116]

Insertion: Inserting a new record in a B+ tree involves finding the correct leaf node based on the key.[116] The new record is inserted into the leaf node while maintaining the sorted order of keys. If the leaf node becomes full, it is split into two, and the middle key is promoted to the parent internal node. This process might propagate up the tree, potentially causing splits in internal nodes and the root.[116] The linked list of leaf nodes is also updated to reflect the split.

Deletion: Deleting a record from a B+ tree requires finding the leaf node containing the key and removing the corresponding record.[116] If the deletion causes the leaf node to have fewer than the minimum number of records, nodes might be merged or keys redistributed from sibling leaf nodes to maintain the B+ tree properties.[116] If a merge or redistribution occurs, the keys in the parent internal nodes might also need to be updated.

10. Comparison of Different Types of Trees

Feature	Binary Tree	Threaded Binary Tree	Binary Search Tree	AVL Tree	B Tree	B+ Tree
Structure	Each node at most 2 children	Uses null pointers for in-order traversal	Ordered binary tree	Self-balancing BST	Multi-way tree	Multi-way tree, data in leaves only
Ordering	No inherent order	No inherent order	Left < Root < Right	Left < Root < Right	Keys sorted within nodes	Keys sorted within nodes, data in leaves

Balance	Can be unbalanced	Can be unbalanced	Can be unbalanced	Strictly balanced (height difference <= 1)	Self-balancing, multi-way	Self-balancing, multi-way
Search Complexity	O(n) worst-case	O(n) worst-case	O(log n) avg, O(n) worst	O(log n)	O(log n)	O(log n)
Insert Complexity	O(1) to O(n)	O(h)	O(log n) avg, O(n) worst	O(log n)	O(log n)	O(log n)
Delete Complexity	O(n) worst-case	O(h)	O(log n) avg, O(n) worst	O(log n)	O(log n)	O(log n)
Traversal	O(n) for all types	O(n) in-order without stack/recursion	O(n) for all types	O(n) for all types	O(n)	O(n)
Space Complexity	O(n) worst-case (height)	O(n) + thread flags	O(n) worst-case (height)	O(n) + balance factor	O(n)	O(n)
Advantages	Simple structure, good for hierarchical data	Efficient in-order traversal without recursion	Efficient search, insertion, deletion (avg)	Guaranteed O(log n) operations	Efficient for large datasets, disk-based storage	Efficient for large datasets, range queries
Disadvantages	Can be inefficient if unbalanced	More complex insertion/deletion	Can be inefficient if unbalanced	More complex implementation, more rotations	Higher disk usage for small datasets	More memory for internal nodes
Use Cases	General hierarchic	Memory-constrained	Dictionaries,	Databases, frequent	Databases, file	Databases, file

	al data, expressio n trees	in-order traversal	indexing	lookups	systems	systems, range queries

11. Conclusions

Tree data structures provide a powerful and flexible means of organizing and managing data, particularly when hierarchical relationships are involved or when efficient search and retrieval are required. The choice of which type of tree to use depends heavily on the specific application requirements and the trade-offs between factors such as structural complexity, operational efficiency, and memory usage. Binary trees, with their fundamental structure, serve as the basis for many other specialized tree types. Threaded binary trees offer an optimization for in-order traversal in memory-constrained scenarios. Binary search trees provide efficient average-case performance for search, insertion, and deletion when the data maintains a degree of randomness. AVL trees guarantee logarithmic time complexity for these operations through strict balancing, making them suitable for applications with frequent lookups. Finally, B Trees and B+ Trees are designed to handle very large datasets efficiently, especially in disk-based systems, with B+ Trees offering additional advantages for range queries due to their linked leaf nodes. Understanding the nuances of each tree type is crucial for designing effective and efficient data management solutions.

Works cited

1. Trees in Data Structrure | What is Trees in Data Structure?, accessed April 20, 2025, https://www.mygreatlearning.com/blog/understanding-trees-in-data-structures/
2. Introduction to Tree Data Structure | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/introduction-to-tree-data-structure/
3. General Tree Definitions and Terminology, accessed April 20, 2025, https://www.cs.kent.edu/~durand/CS2/Notes/10_Binary_Trees/ds_treesA.html
4. Tree Data Structure | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/tree-data-structure/
5. MODULE 4: TREES DEFINITION TERMINOLOGY - Deepak D., accessed April 20, 2025, https://deepakdvallur.weebly.com/uploads/8/9/7/5/89758787/module-4-1.pdf
6. Tree (abstract data type) - Wikipedia, accessed April 20, 2025, https://en.wikipedia.org/wiki/Tree_(abstract_data_type)
7. 7.3. Vocabulary and Definitions — Problem Solving with Algorithms and Data Structures, accessed April 20, 2025,

https://runestone.academy/ns/books/published/pythonds/Trees/VocabularyandDefinitions.html
8. An Introduction to Tree in Data Structure - Simplilearn.com, accessed April 20, 2025, https://www.simplilearn.com/tutorials/data-structure-tutorial/trees-in-data-structure
9. Basic Tree Terminologies, their Representation and Applications - INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES, accessed April 20, 2025, http://www.ijcsit.com/docs/Volume%206/vol6issue01/ijcsit2015060184.pdf
10. root parent child leaf node edge - Engineering People Site, accessed April 20, 2025, https://people.engr.tamu.edu/j-welch/teaching/211.s03/lnotes3.pdf
11. Tree Terminology, accessed April 20, 2025, https://www.nku.edu/~longa/classes/mat385_resources/docs/trees.html
12. What is a Tree? — Practices in Data Structures and Algorithms with Java, accessed April 20, 2025, https://sites.ualberta.ca/~rosanna/books/PDSA/chapter8/tree.html
13. Trees and Binary Trees - Intro to Data Structures - Codology, accessed April 20, 2025, https://www.codology.org/intro-to-data-structures/trees-and-binary-trees
14. UNIT – III TREES 3.1 Basic Terminologies Terminologies used in Trees • Root – The top node in a tree. • Child, accessed April 20, 2025, https://aits-tpt.edu.in/wp-content/uploads/2018/08/DS-Unit-3.pdf
15. Data Structures Tutorials - Tree Terminology with examples - BTech Smart Class, accessed April 20, 2025, http://www.btechsmartclass.com/data_structures/tree-terminology.html
16. Terminology of tree | PPT - SlideShare, accessed April 20, 2025, https://www.slideshare.net/slideshow/terminology-of-tree/250594345
17. CS 2112/ENGRD 2112 Fall 2021 - Computer Science Cornell, accessed April 20, 2025, https://www.cs.cornell.edu/courses/cs2112/2022fa/lectures/lecture.html?id=trees
18. 7.2. Binary Trees — CS3 Data Structures & Algorithms - OpenDSA, accessed April 20, 2025, https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/BinaryTree.html
19. Binary Trees - andrew.cmu.ed, accessed April 20, 2025, https://www.andrew.cmu.edu/course/15-121/lectures/Trees/trees.html
20. What is node in a tree? - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/28762037/what-is-node-in-a-tree
21. What are the different types of Nodes in a Tree | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/what-are-the-different-types-of-nodes-in-a-tree/
22. www.geeksforgeeks.org, accessed April 20, 2025, https://www.geeksforgeeks.org/introduction-to-tree-data-structure/#:~:text=Basic%20Terminologies%20In%20Tree%20Data%20Structure%3A&text=Child%20Node%3A%20The%20node%20which,is%20called%20the%20root%20node.

23. www.vaia.com, accessed April 20, 2025, https://www.vaia.com/en-us/textbooks/computer-science/starting-out-with-c-from-control-structures-through-objects-8-edition/chapter-20/problem-3-what-is-a-leaf-node/#:~:text=A%20leaf%20node%20or%20an,to%20process%20or%20store%20information.
24. Data Structures: Lecture 8, accessed April 20, 2025, https://people.engr.tamu.edu/djimenez/ut/utsa/cs1723/lecture8.html
25. Leaf node - (Data Structures) - Vocab, Definition, Explanations | Fiveable, accessed April 20, 2025, https://library.fiveable.me/key-terms/data-structures/leaf-node
26. library.fiveable.me, accessed April 20, 2025, https://library.fiveable.me/key-terms/data-structures/subtree#:~:text=A%20subtree%20is%20a%20section,the%20hierarchical%20organization%20of%20data.
27. Subtree - (Data Structures) - Vocab, Definition, Explanations | Fiveable, accessed April 20, 2025, https://library.fiveable.me/key-terms/data-structures/subtree
28. Define the subtree in the context of binary trees. - TutorChase, accessed April 20, 2025, https://www.tutorchase.com/answers/ib/computer-science/define-the-subtree-in-the-context-of-binary-trees
29. Subtree of all nodes in a tree using DFS - GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/sub-tree-nodes-tree-using-dfs/
30. Trees - CS 2112/ENGRD 2112 Fall 2019 - Cornell University, accessed April 20, 2025, https://www.cs.cornell.edu/courses/cs2112/2019fa/lectures/lecture.html?id=trees
31. opendsa-server.cs.vt.edu, accessed April 20, 2025, https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/BinaryTree.html#:~:text=Definitions%20and%20Properties,other%20and%20from%20the%20root.
32. 10.2 Binary Trees Definitions and Properties - Runestone Academy, accessed April 20, 2025, https://runestone.academy/ns/books/published/bridgesds/sec-BTDef.html
33. Binary Tree in Data Structure: Types, Implementation & More - AnalytixLabs, accessed April 20, 2025, https://www.analytixlabs.co.in/blog/binary-tree-in-data-structure/
34. Binary Tree Data Structure | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/binary-tree-data-structure/
35. Properties of Binary Tree | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/properties-of-binary-tree/
36. Binary tree - Wikipedia, accessed April 20, 2025, https://en.wikipedia.org/wiki/Binary_tree
37. Introduction to the Binary Tree Data Structure | Baeldung on Computer Science, accessed April 20, 2025, https://www.baeldung.com/cs/binary-tree-intro
38. Introduction to Binary Tree: Properties, Types, Representation and Application, accessed April 20, 2025, https://www.enjoyalgorithms.com/blog/introduction-to-binary-tree/
39. Full Binary Tree: Properties, Operations, Implementation - WsCube Tech,

accessed April 20, 2025,
https://www.wscubetech.com/resources/dsa/full-binary-tree
40. Binary Tree in Data Structure (Examples, Types, Traversal, Operations) - WsCube Tech, accessed April 20, 2025,
https://www.wscubetech.com/resources/dsa/binary-tree
41. Comprehensive Guide to Binary Tree: Types, Structures, Applications - AK Coding, accessed April 20, 2025,
https://akcoding.com/dsa/non-linear-data-structures/tree-data-structure/binary-tree/
42. Binary Search Tree (BST) in Data Structure: Full Guide - WsCube Tech, accessed April 20, 2025, https://www.wscubetech.com/resources/dsa/binary-search-tree
43. Understanding Binary Search Trees (BST) - Launch School, accessed April 20, 2025, https://launchschool.com/books/advanced_dsa/read/binary_search_tree
44. Why is the worst case time complexity for a Binary Tree Insert/Delete operation different than a Linked List? - Stack Overflow, accessed April 20, 2025,
https://stackoverflow.com/questions/76176615/why-is-the-worst-case-time-complexity-for-a-binary-tree-insert-delete-operation
45. Complexity of different operations in Binary tree, Binary Search Tree and AVL tree | GeeksforGeeks, accessed April 20, 2025,
https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/
46. CS 367-3 - Binary Search Trees - cs.wisc.edu, accessed April 20, 2025,
https://pages.cs.wisc.edu/~siff/CS367/Notes/bsts.html
47. CSC378: Binary Search Trees, accessed April 20, 2025,
https://www.dgp.toronto.edu/~jstewart/378notes/15bst/
48. CS 225 | Binary Search Trees, accessed April 20, 2025,
https://courses.grainger.illinois.edu/cs225/sp2025/resources/bst/
49. Binary Search Tree - GeeksforGeeks, accessed April 20, 2025,
https://www.geeksforgeeks.org/binary-search-tree-data-structure/
50. Binary Search Tree(BST) - Programiz, accessed April 20, 2025,
https://www.programiz.com/dsa/binary-search-tree
51. Binary search tree - Wikipedia, accessed April 20, 2025,
https://en.wikipedia.org/wiki/Binary_search_tree
52. Introduction to Binary Search Tree | GeeksforGeeks, accessed April 20, 2025,
https://www.geeksforgeeks.org/introduction-to-binary-search-tree/
53. Complexities of binary tree traversals - Stack Overflow, accessed April 20, 2025,
https://stackoverflow.com/questions/4547012/complexities-of-binary-tree-traversals
54. en.wikipedia.org, accessed April 20, 2025,
https://en.wikipedia.org/wiki/Threaded_binary_tree#:~:text=%22A%20binary%20tree%20is%20threaded,order%20predecessor%20of%20the%20node.%22
55. Threaded Binary Tree | GeeksforGeeks, accessed April 20, 2025,
https://www.geeksforgeeks.org/threaded-binary-tree/
56. Threaded binary tree - Wikipedia, accessed April 20, 2025,
https://en.wikipedia.org/wiki/Threaded_binary_tree

57. Threaded Binary Tree In Data Structure, accessed April 20, 2025, https://blog.heycoach.in/threaded-binary-tree-in-data-structure/
58. Threaded Binary Tree in Data Structure with Examples | Hero Vired, accessed April 20, 2025, https://herovired.com/learning-hub/blogs/threaded-binary-tree/
59. Threaded Binary Tree线索二叉树 · Swift Algorithm - victorqi, accessed April 20, 2025, https://victorqi.gitbooks.io/swift-algorithm/content/threaded_binary_tree.html
60. unit – ii threaded binary tree - Sathyabama, accessed April 20, 2025, https://www.sathyabama.ac.in/sites/default/files/course-material/2020-10/unit2_5.pdf
61. Understanding Threaded Binary Trees | Baeldung on Computer Science, accessed April 20, 2025, https://www.baeldung.com/cs/threaded-binary-trees
62. Threaded Binary Tree (TBT) IN Data Structure Explaination - YouTube, accessed April 20, 2025, https://www.youtube.com/watch?v=ffgg_zmbaxw
63. Threaded Binary Tree - Scaler Blog, accessed April 20, 2025, https://www.scaler.in/threaded-binary-tree/
64. Threaded Binary Tree | Insertion - GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/threaded-binary-tree-insertion/
65. Algorithm to implement threaded binary tree - Punjabi University, Patiala, accessed April 20, 2025, http://www.punjabiuniversity.ac.in/pages/Images/elearn/24.pdf
66. courses.grainger.illinois.edu, accessed April 20, 2025, https://courses.grainger.illinois.edu/cs225/sp2025/resources/bst/#:~:text=A%20binary%20search%20tree%20(BST,these%20properties%20will%20remain%20true.
67. Introduction to Binary Search Tree (BST) in Data Structure - EnjoyAlgorithms, accessed April 20, 2025, https://www.enjoyalgorithms.com/blog/introduction-to-binary-search-tree/
68. 1 Binary search trees, accessed April 20, 2025, https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture8.pdf
69. Binary Search Tree and its operations with algorithm and Examples - Tech Skill Guru, accessed April 20, 2025, https://techskillguru.com/ds/binary-search-tree
70. AVL Tree Data Structure - EnjoyAlgorithms, accessed April 20, 2025, https://www.enjoyalgorithms.com/blog/avl-tree-data-structure/
71. fiveable.me, accessed April 20, 2025, https://fiveable.me/key-terms/data-structures/avl-tree#:~:text=An%20AVL%20tree%20is%20a,of%20O(log%20n).
72. AVL Tree - (Data Structures) - Vocab, Definition, Explanations | Fiveable, accessed April 20, 2025, https://fiveable.me/key-terms/data-structures/avl-tree
73. AVL Tree Data Structure: Rotations, Examples, Implementation - WsCube Tech, accessed April 20, 2025, https://www.wscubetech.com/resources/dsa/avl-tree
74. AVL Tree in Data Structure: Properties, Examples, Operations - Hero Vired, accessed April 20, 2025, https://herovired.com/learning-hub/blogs/avl-tree/
75. AVL Tree Data Structure | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/introduction-to-avl-tree/
76. Deep Dive into Data structures using Javascript - AVL Tree, accessed April 20,

2025, https://www.sahinarslan.tech/posts/deep-dive-into-data-structures-using-javascript-avl-tree/
77. What is AVL Tree | AVL Tree meaning - GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/what-is-avl-tree-avl-tree-meaning/
78. AVL tree - Wikipedia, accessed April 20, 2025, https://en.wikipedia.org/wiki/AVL_tree
79. Properties Of AVL Trees, accessed April 20, 2025, https://blog.heycoach.in/properties-of-avl-trees/
80. AVL Trees Simply Explained - YouTube, accessed April 20, 2025, https://m.youtube.com/watch?v=zP2xbKerIds&pp=ygUFI2F2bHI%3D
81. Data Structures and Algorithms: AVL Trees - Interview kickstart, accessed April 20, 2025, https://interviewkickstart.com/blogs/learn/data-structures-and-algorithms-avl-trees
82. Time Complexity Of AVL Tree Operations - HeyCoach | Blogs, accessed April 20, 2025, https://blog.heycoach.in/time-complexity-of-avl-tree-operations/
83. AVL Tree insert time complexity : r/learnprogramming - Reddit, accessed April 20, 2025, https://www.reddit.com/r/learnprogramming/comments/q17c54/avl_tree_insert_time_complexity/
84. How can an AVL tree have a complexity of O(log n) when the in-order traversal calls itself recursively twice? Doesn't that make it O(2^n)? - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/64409947/how-can-an-avl-tree-have-a-complexity-of-olog-n-when-the-in-order-traversal-ca
85. Applications and Use Cases of Binary Trees - Youcademy, accessed April 20, 2025, https://youcademy.org/use-cases-of-binary-trees/
86. Applications, Advantages and Disadvantages of Binary Tree - GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-tree/
87. What are the applications of binary trees? - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/2130416/what-are-the-applications-of-binary-trees
88. Applications of Binary Trees | Baeldung on Computer Science, accessed April 20, 2025, https://www.baeldung.com/cs/applications-of-binary-trees
89. What are some real life practical applications for binary trees? : r/learnprogramming - Reddit, accessed April 20, 2025, https://www.reddit.com/r/learnprogramming/comments/60fvps/what_are_some_real_life_practical_applications/
90. Applications, Advantages and Disadvantages of Binary Search Tree | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-

binary-search-tree/
91. APPLICATION OF BINARY TREES - Stony Brook CS, accessed April 20, 2025, https://www3.cs.stonybrook.edu/~rezaul/Spring-2015/CSE548/Yonghui-Wu/lecture-3.pdf
92. When are binary trees used in programming? : r/learnprogramming - Reddit, accessed April 20, 2025, https://www.reddit.com/r/learnprogramming/comments/ykpcbt/when_are_binary_trees_used_in_programming/
93. Applications of Complete Binary Tree? - Computer Science Stack Exchange, accessed April 20, 2025, https://cs.stackexchange.com/questions/142934/applications-of-complete-binary-tree
94. Binary Trees Explained: Traversal Techniques and Applications - Codedamn, accessed April 20, 2025, https://codedamn.com/news/programming/binary-trees-explained-traversal-techniques-applications
95. Binary Trees in Data Structures - Types, Implementation, Applications - ScholarHat, accessed April 20, 2025, https://www.scholarhat.com/tutorial/datastructures/binary-tree-in-data-structures
96. binary search tree use in real world programs? [duplicate] - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/13900481/binary-search-tree-use-in-real-world-programs
97. When are binary trees better than hashtables in real world applications?, accessed April 20, 2025, https://cs.stackexchange.com/questions/30347/when-are-binary-trees-better-than-hashtables-in-real-world-applications
98. en.wikipedia.org, accessed April 20, 2025, https://en.wikipedia.org/wiki/B-tree#:~:text=In%20computer%20science%2C%20a%20B,with%20more%20than%20two%20children.
99. B-tree - Wikipedia, accessed April 20, 2025, https://en.wikipedia.org/wiki/B-tree
100. Understanding the Basics of B-Tree Data Structures - TiDB, accessed April 20, 2025, https://www.pingcap.com/article/understanding-basics-b-tree-data-structures/
101. B Tree vs B+ Tree: Which to Use - Hypermode, accessed April 20, 2025, https://hypermode.com/blog/b-tree-vs-b-plus-tree
102. B-tree - CelerData, accessed April 20, 2025, https://celerdata.com/glossary/b-tree
103. CS 225 | B-Trees - Course Websites, accessed April 20, 2025, https://courses.grainger.illinois.edu/cs225/sp2023/resources/btrees/
104. B Tree in Data Structure: Properties, Examples, Implementation, Full Guide - WsCube Tech, accessed April 20, 2025, https://www.wscubetech.com/resources/dsa/b-tree
105. B-tree Data Structure | Baeldung on Computer Science, accessed April 20,

2025, https://www.baeldung.com/cs/b-tree-data-structure
106. B-trees and database indexes - PlanetScale, accessed April 20, 2025, https://planetscale.com/blog/btrees-and-database-indexes
107. Introduction of B-Tree | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/introduction-of-b-tree-2/
108. Understanding B-Trees: The Data Structure Behind Modern Databases - YouTube, accessed April 20, 2025, https://www.youtube.com/watch?v=K1a2Bk8NrYQ&pp=0gcJCdgAo7VqN5tD
109. Difference between B tree and B+ tree - GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/difference-between-b-tree-and-b-tree/
110. B -trees, accessed April 20, 2025, https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf
111. How is the insert operation in a B-Tree O(log n), should it not be O(log 2d -1) where d is the degree of the B-Tree - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/78980404/how-is-the-insert-operation-in-a-b-tree-olog-n-should-it-not-be-o-log-2d-1
112. Why B Tree complexity is O(log n), it is not a binary tree - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/57498018/why-b-tree-complexity-is-olog-n-it-is-not-a-binary-tree
113. B-tree - Programiz, accessed April 20, 2025, https://www.programiz.com/dsa/b-tree
114. Why is b-tree search O(log n)? - Computer Science Stack Exchange, accessed April 20, 2025, https://cs.stackexchange.com/questions/59453/why-is-b-tree-search-olog-n
115. en.wikipedia.org, accessed April 20, 2025, https://en.wikipedia.org/wiki/B%2B_tree#:~:text=A%20B%2B%20tree%20is%20an,with%20two%20or%20more%20children.
116. B+ tree - Wikipedia, accessed April 20, 2025, https://en.wikipedia.org/wiki/B%2B_tree
117. B+ Tree in Data Structure (Explained With Examples) - WsCube Tech, accessed April 20, 2025, https://www.wscubetech.com/resources/dsa/b-plus-tree
118. B+ Tree in DBMS | GATE Notes - BYJU'S, accessed April 20, 2025, https://byjus.com/gate/b-plus-tree-in-dbms-notes/
119. B+ Tree in Data Structure (DBMS) Properties & Operation - Simplilearn.com, accessed April 20, 2025, https://www.simplilearn.com/tutorials/data-structure-tutorial/b-plus-tree-in-data-structure
120. What is B+ Tree | B+ Tree meaning - GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/what-is-b-plus-tree-b-plus-tree-meaning/
121. Introduction of B+ Tree | GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/introduction-of-b-tree/
122. B+ Tree - Programiz, accessed April 20, 2025, https://www.programiz.com/dsa/b-plus-tree
123. B+ Trees Overview - Tutorialspoint, accessed April 20, 2025,

https://www.tutorialspoint.com/data_structures_algorithms/b_plus_trees.htm
124. B+ Trees - NITCbase, accessed April 20, 2025, https://nitcbase.github.io/docs/Misc/B+%20Trees
125. 15445.courses.cs.cmu.edu, accessed April 20, 2025, https://15445.courses.cs.cmu.edu/fall2021/notes/07-trees.pdf
126. What are the differences between B trees and B+ trees? - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/870218/what-are-the-differences-between-b-trees-and-b-trees
127. B-Tree & B+ Tree - Blocks and Files, accessed April 20, 2025, https://blocksandfiles.com/2022/04/19/b-tree/
128. When do you use b-tree and b+tree data structure? : r/compsci - Reddit, accessed April 20, 2025, https://www.reddit.com/r/compsci/comments/1dji7up/when_do_you_use_btree_and_btree_data_structure/
129. L73 : Difference between B and B+ Trees | Complete DBMS Course | Jobs - YouTube, accessed April 20, 2025, https://www.youtube.com/watch?v=xmCqrt4zLTk
130. [Bitesized] B-Trees vs B+Trees - the most important differences : r/SQLServer - Reddit, accessed April 20, 2025, https://www.reddit.com/r/SQLServer/comments/xv8k0a/bitesized_btrees_vs_btrees_the_most_important/
131. Implementation of B+ Tree in C - GeeksforGeeks, accessed April 20, 2025, https://www.geeksforgeeks.org/implementation-of-b-plus-tree-in-c/
132. vismaywalde/B-plus-tree: Implemented the B+ tree data structure in C, covering insertion, deletion, and search operations. Conducted performance analysis, comparing search time complexity with B-trees. - GitHub, accessed April 20, 2025, https://github.com/vismaywalde/B-plus-tree
133. Time complexity for insertion and search in B+ tree - Stack Overflow, accessed April 20, 2025, https://stackoverflow.com/questions/65672533/time-complexity-for-insertion-and-search-in-b-tree
134. How difficult is programming a B+ Tree? : r/learnprogramming - Reddit, accessed April 20, 2025, https://www.reddit.com/r/learnprogramming/comments/1250wmm/how_difficult_is_programming_a_b_tree/