An AVL tree is a special kind of binary search tree that stays balanced on its own. For any node in the tree, the difference in height between its left and right sides can’t be more than one. This difference is known as the balance factor. If one side gets too tall, the tree will make adjustments to even things out, which helps keep searching and other operations quick and efficient.

**Example of AVL Trees:**

A diagram of numbers and circles

Description automatically generated

### Operations on an AVL Tree:

### [Insertion](https://www.geeksforgeeks.org/insertion-in-an-avl-tree/)

### [Deletion](https://www.geeksforgeeks.org/deletion-in-an-avl-tree/)

### [Searching](https://www.geeksforgeeks.org/avl-trees-containing-a-parent-node-pointer/) [It is similar to performing a search in BST]

## **Advantages of AVL Trees / Binary Search Tree**

1. Time Complexity:
2. Balanced Structure:
3. **Self-Balancing**:
4. **Height-Balanced**:
5. **Fast Searches**:
6. **Not Skewed**:

**Time Complexity**:

In a library management system, searching for a specific book can be done quickly, even if the collection grows large. Whether there are 100 or 10,000 books, the time it takes to find one remains manageable. This efficiency allows users to locate books without long waits.

Operations like searching, inserting, and deleting take logarithmic time. This is true no matter how the data is arranged.

**Balanced Structure**:

Consider an online gaming platform that stores user scores. The balanced structure ensures that retrieving a user’s score is quick and consistent. If the data structure becomes unbalanced, some operations could take much longer, negatively impacting the user experience.

They stay balanced, which leads to consistent and reliable performance.

**Self-Balancing**:

In a social media app, when users add or remove friends, the system automatically adjusts itself. This feature means that no manual intervention is needed to maintain efficiency. As users interact with the platform, it stays optimized for performance.

They automatically adjust themselves after every change, keeping the tree balanced.

**Height-Balanced**:

Imagine a university course registration system that manages a large number of student enrollments. By keeping the tree's height balanced, the system ensures that searching for a specific course remains quick and efficient, even as the number of courses grows. The shorter height of the tree allows students to quickly find and register for courses, enhancing the overall user experience.

They maintain a height of O(log N), ensuring they don’t become too tall.

**Fast Searches**:

In a social media application, an AVL tree is used to manage user posts for quick access. The balanced structure enables fast searches, allowing users to retrieve posts based on timestamps or popularity efficiently. This speed enhances user experience, as posts load almost instantaneously. Additionally, as users interact with the feed—liking, commenting, or sharing—the AVL tree adjusts, maintaining its balance for continued performance. Overall, AVL trees facilitate quick data access, essential for keeping users engaged.

They allow for quicker searches compared to other tree types, which is great for applications that need quick data access.

**Not Skewed**:

In an airline reservation system, AVL trees are used to manage flight schedules efficiently. The tree remains balanced, preventing any part from becoming overly deep, which ensures fast searches for available flights. This balance allows for quick insertions and deletions when flights are added or removed. As a result, the system can respond swiftly to customer queries and booking requests. Overall, AVL trees provide consistent performance, crucial for optimizing airline operations.

AVL trees are not unbalanced

**Disadvantages of AVl Trees**

1. Difficult to Implement:
2. Less Common:
3. Slow Insertions and Deletions:
4. **Not Ideal for Real-Time Analytics**:

**Difficult to Implement**:

Imagine you’re building a system to manage a library’s books. You decide to use a special type of tree to keep track of everything. However, your team finds it tricky to make sure the tree stays balanced when adding or removing books. If someone makes a mistake in the balancing process, it could lead to problems, like not being able to find a book or even crashing the system. This complexity can make the project take longer and require more testing to get it right.

While easier than Red-Black trees, AVL trees are still harder to implement than regular binary search trees.

**Less Common**:

A new tech startup is looking to hire a developer who knows about advanced data structures. As they interview candidates, they notice that most people have experience with simpler structures, like lists or basic trees. Because the specific type of tree they want to use is less common, it takes longer to find the right person for the job. They might have to choose someone who isn’t as experienced or spend extra time training a new hire.

They are not used as frequently as Red-Black trees in practice.

**Slow Insertions and Deletions**:

Think about an online store during a big sale, like Black Friday, when many items are being added or sold out quickly. If the system uses a complicated tree structure to manage inventory, it could slow down the process of updating the stock. Each time a product is added or removed, the system has to do extra work to keep everything balanced. This could lead to delays, meaning customers might see outdated stock information, which could frustrate them and potentially lead to lost sales.

The strict balancing can make adding or removing data slower because they require more rotations to maintain balance.

**Not Ideal for Real-Time Analytics**:

In a stock trading app, speed is everything. Traders need to see the latest stock prices instantly to make quick decisions. If the app uses a complex data structure that takes time to balance, it could slow down how fast they can access that information. For example, if a trader tries to check the latest prices, they might experience delays, causing them to miss important trading opportunities. In fast-paced environments like this, systems need to be quick and efficient, so simpler structures are often a better choice.

They may not be the best choice for applications that require quick, concurrent access to data and approximate query results.

### Applications of AVL Trees

1. **Maps and Sets**:
2. **Priority Queues**:
3. **File Systems**:
4. **In-Memory Databases**:
5. **Graphics and Game Development**:

**Maps and Sets**:

Think of a contact list on your phone. Each contact is a key (like a name), and the details (like phone numbers or email addresses) are the values. AVL trees keep this list organized, so when you want to find a contact, add a new one, or remove an old one, it happens quickly without getting messy.

They help store key-value pairs and unique items, making it easy to find, add, or remove things quickly.

**Priority Queues**:

Imagine a hospital emergency room where patients are seen based on the severity of their condition. An AVL tree acts like a system that organizes patients by priority—those in critical condition are treated first. This ensures that the most urgent tasks are handled efficiently, allowing for quick access to the highest priority patients.

They can be used to manage items based on priority, like organizing tasks that need to be done first.

**File Systems**:

Consider your computer’s file explorer. When you save documents, pictures, and folders, the system needs to keep track of all these items. An AVL tree helps manage this organization, making it easy to find, add, or delete files without slowing down the entire system. You can quickly search for a specific file among thousands.

Help efficiently track files and folders.

**In-Memory Databases**:

Picture a digital library where you can quickly search for books, articles, or resources. An AVL tree helps this library stay organized in memory, allowing for rapid searches and updates. When you add a new book or check out a resource, it’s done swiftly, ensuring users have fast access to information.

Facilitate fast data storage and retrieval.

**Graphics and Game Development**:

Think about a video game where characters and objects need to interact with each other, like when a character jumps and collides with an enemy. An AVL tree can manage these interactions efficiently, keeping track of which objects are nearby and enabling fast calculations for collisions or movement. This helps ensure the game runs smoothly without lag.

Useful for handling tasks like detecting collisions between objects or finding paths in games.

A screenshot of a computer

Description automatically generated

**The advantages of an AVL tree over a binary search tree (BST) include:**

1. Self-Balancing:
2. Improved Search Efficiency:
3. Efficient Insertions and Deletions:
4. Consistent Performance:
5. Efficient Use of Memory:

**Self-Balancing**:

Think of a bookshelf that’s supposed to hold books in a specific order. If you keep adding books without organizing them, they can end up leaning to one side, making it hard to find a book. An AVL tree is like a bookshelf that automatically rearranges itself whenever you add or remove books, keeping them balanced. This means you can find any book quickly.

AVL trees automatically keep themselves balanced, leading to faster search, insertion, and deletion operations.

**Improved Search Efficiency**:

Imagine looking for a specific file in a huge filing cabinet. If the files are neatly organized (like an AVL tree), you can quickly find what you need by checking fewer drawers. But if the files are jumbled up (like an unbalanced BST), you might have to go through every drawer, which takes much longer. AVL trees keep everything sorted, so searching is faster.

**Because AVL trees** are balanced, searching for a value is quicker—about O(log n) time, whereas unbalanced BSTs can degrade to O(n) in the worst case.

**Efficient Insertions and Deletions**:

Picture a line of people at a coffee shop. If the line is organized and everyone steps forward efficiently (like adding/removing nodes in an AVL tree), things move smoothly. If people are cutting in and out randomly (like an unbalanced BST), the line can get chaotic and slow. AVL trees might need to rearrange a bit when someone joins or leaves, but they still keep the line moving quickly.

Adding or removing nodes in an AVL tree may require some extra steps (like rotating nodes), but these actions are still quick, around O(log n) time. This is much better for large trees compared to slow operations in unbalanced trees

**Consistent Performance:**

Think of a restaurant that always serves food at the same speed, regardless of how busy it gets. An AVL tree behaves similarly; it maintains a steady performance for adding, finding, or removing items, no matter how many books (or nodes) are on the shelf. So, you can always expect a good experience without delays.

**AVL** **trees** offer reliable speed for searching, adding, and removing nodes, so you can trust that they’ll perform well, no matter how many nodes are in the tree.

**Efficient Use of Memory**:

Consider a closet that’s organized by size and type of clothes. If your clothes are neatly folded and arranged, you can fit more in without wasting space. An AVL tree keeps data structured in a way that minimizes wasted space and allows for quick access. So, when you need something, you can grab it easily without digging through a mess.

**AVL trees** use memory better because their balanced shape organizes data efficiently. This can also make it faster to access the data when you traverse the tree.