Binary Search Trees

Group 9

About binary search trees

A Binary Search Tree (BST) is a type of Binary Tree Data Structure where the following properties must be true for any node "X" in the tree:

- The X node's left child and all of its descendants (children, children's children, and so on) have lower values than X's value.
- The right child, and all its descendants have higher values than X's value.
- Left and right subtrees must also be Binary Search Trees.

These properties makes it faster to search, add and delete values than a regular binary tree.

Basic operations

Search operation

How it works:

1. Start at the root node.
2. If this is the value we are looking for, return.
3. If the value we are looking for is higher, continue searching in the right subtree.
4. If the value we are looking for is lower, continue searching in the left subtree.
5. If the subtree we want to search does not exist, depending on the programming language, return None, or NULL, or something similar, to indicate that the value is not inside the BST.

Insert operation

Inserting a value in the correct position is similar to searching because we try to maintain the rule that the left subtree is lesser than root and the right subtree is larger than root.
We keep going to either right subtree or left subtree depending on the value and when we reach a point left or right subtree is null, we put the new node there.

Delete operation

How it works:

1. If the node is a leaf node, remove it by removing the link to it.
2. If the node only has one child node, connect the parent node of the node you want to remove to that child node.
3. If the node has both right and left child nodes: Find the node's in-order successor, change values with that node, then delete it.

Manual run through

Let's try to do the searching manually, just to get an even better understanding of how Binary Search works before actually implementing it in a programming language.

We will search for value 11 in the array below

[2, 3, 7, 7, 11, 15, 25]

The value in the middle of the array at index 3, is it equal to 11?

7 is less than 11, so we must search for 11 to the right of index 3. The values to the right of index 3 are [11, 15, 25]. The next value to check is the middle value 15, at index 5.

15 is higher than 11, so we must search to the left of index 5. We have already checked index 0-3, so index 4 is only value left to check.

We have found it!

Value 11 is found at index 4.

Returning index position 4.

Binary Search is finished.

Binary search time complexity

Each time Binary Search checks a new value to see if it is the target value, the search area is halved.

This means that even in the worst case scenario where Binary Search cannot find the target value, it still only needs log2n comparisons to look through a sorted array of n values.

Time complexity for Binary Search is O(log2n)

Application domains

Binary search trees

1. Search Engines
   - Use Case: Quickly locating documents or web pages containing specific keywords in a pre-sorted index.
   - Benefit: Reduces search time significantly compared to linear search.

2. Databases
   - Use Case: Searching for records in a sorted dataset, such as retrieving user profiles by their ID.
   - Benefit: Enhances performance for large datasets, especially in primary key lookups.

3. E-commerce
   - Use Case: Finding products in a sorted catalog by price, rating, or other attributes.
   - Benefit: Improves user experience by providing faster search results.

4. Games
   - Use Case: Searching for a specific item or character in a sorted list of game assets.
   - Benefit: Increases efficiency in resource management and loading times.

5. Routing Algorithms
   - Use Case: Finding the shortest path or nearest location in a sorted list of waypoints.
   - Benefit: Optimizes navigation systems by quickly accessing relevant data.

6. Scheduling
   - Use Case: Finding available time slots in a sorted schedule (e.g., for appointments).
   - Benefit: Streamlines the scheduling process by quickly identifying gaps.

Implementing binary search tree

Basic routing application

Implementing a basic routing algorithm application using binary search can illustrate how to find the shortest path or nearest location among sorted waypoints. Below is a simple example in Python that demonstrates a routing scenario where we search for the nearest location based on a sorted list of distances.

Python code example: nearest location finder

```
class Location:
    def __init__(self, name, distance):
        self.name = name
        self.distance = distance

def binary_search(locations, target_distance):
    left, right = 0, len(locations) - 1

    while left <= right:
        mid = (left + right) // 2
        if locations[mid].distance == target_distance:
            return locations[mid]
        elif locations[mid].distance < target_distance:
            left = mid + 1
        else:
            right = mid - 1

    # If exact match not found, return the closest location
    if right < 0:
        return locations[0]  # Closest location if target is less than the first
    if left >= len(locations):
        return locations[-1]  # Closest location if target is greater than the last

    # Return the closest of the two surrounding distances
    if (locations[left].distance - target_distance) < (target_distance - locations[right].distance):
        return locations[left]
    else:
        return locations[right]

# Sample sorted locations (sorted by distance)
locations = [
    Location("Location A", 5),
    Location("Location B", 10),
    Location("Location C", 15),
    Location("Location D", 20),
    Location("Location E", 25)
]

# Example: Finding the nearest location to a given distance
target_distance = 12
nearest_location = binary_search(locations, target_distance)
```

Explanation

1. Location Class: Represents a location with a name and a distance.
2. Binary Search Function: Implements the binary search algorithm to find the nearest location based on the target distance.
    - It searches through a sorted list of locations.
    - If an exact match is found, it returns that location.
    - If not, it returns the closest location.
3. Sample Data: A list of Location objects is created, sorted by distance.
4. Usage: The application finds and prints the nearest location to a specified target distance.

Thank you!