K-DIMENSIONAL TREE

Introduction:

A KD-tree (short for k-dimensional tree) is a data structure used for organizing points in a k-dimensional space. It's particularly useful for multidimensional search operations, such as nearest neighbor search. Here are some key features and characteristics of KD-trees:

Space Partitioning: KD-trees recursively partition the space along different dimensions (axes) at each level of the tree. This allows for efficient organization of points in multidimensional space.

Balanced Tree Structure: Ideally, KD-trees aim to maintain a balanced structure, where each level of the tree divides the space evenly. This balance helps in achieving efficient search and insertion operations.

Nearest Neighbor Search: One of the primary applications of KD-trees is nearest neighbor search. Given a query point, KD-trees allow for quickly finding the closest point(s) in the space.

Node structure:

A node in K-tree stores an array of points corresponding to the dimension k and two pointers to left and right

Operations and Algorithms

Insert algorithm:

Input: array of integers , dimension of the space , depth at which function is performing and a pointer to the root node or subtree

Output: Pointer to the root node of the modified tree after insertion

Algorithm:

1. Check if root is NULL, if yes create a node and make the root pointing to the newnode and return

2.If not, then determine the dimension to split (split = depth%dimension) based on the split value compare the node to be inserted and the current node, if greater, recursively call the function to move right

Else recursively call the function to move left and increase the depth by 1

3.Return the root for the modified tree

Time complexity analysis:

Best case scenario:

The time complexity is O(Log n) when the tree is balanced

Worst case scenario:

Even during worst case the time complexity is close to O(logn ) might degrade to O(n) when the dimensions increase

Search Algorithm:

Input: Pointer to the root node, array of coordinates to search , depth of tree where the function is performing and the dimension of space

Output: bool (true if found,else false)

Algorithm:

1.check is root is NULL,if yes return False

2. If not NULL, compute the current dimension on which it should be split and compare the current key in thenode based on that dimension and the input coordinates, if input coordinates is greater

(a)then move to right part of the subtree and

Perform the same operation and also increase the depth by 1

* + - * 1. Else, move to the left subtree and perform the above operation until you reach NULL

Time Complexity Analysis:

Kd-tree is a roughly balanced tree so the average time complexity is O(logn) for thr search operation

Delete Algorithm:

Input: array of integers , dimension of the space , depth at which function is performing and a pointer to the root node or subtree

Output: Pointer to the root node of the modified tree after insertion

Algorithm:

1.check if the coordinate provided by user in present in tree,if yes

1. if the node in which it is found is a leaf node,
   1. Dynamically delete the node directly
      1. If it has one child.
         * 1. If the node has left child but not right child, then store the left child in a separate pointer and delete the node and return separate pointer
           2. Follow the same step for right child also
      2. If it has two children
         * 1. Then find the node with minimum value in right subtree recursively
           2. Copy the contents of the minimum node to the node to be deleted
           3. Then recursively delete the minimum node from the right sub tree

Time Complexity analysis:

The average case time complexity for kdtree is O(logn) because it tends to balanced or nearly balanced  
If It is skewed then worst case Time complexity may tend to O(n)

Applications of K-D tree

1.K-d tree is used in Machine learning and in K-NN classification algorithms to search the nearest neighbour based on partition of space

2. KD-trees support range search operations, where all points within a specified distance or range of a given query point are retrieved. This is valuable in spatial databases, GIS (Geographic Information Systems), and location-based services

3. In computer graphics, KD-trees are employed in ray tracing algorithms to accelerate the intersection tests between rays and scene geometry. They are also used for collision detection in simulations and video games to efficiently detect collisions between objects.

Fast and Efficient search,insert,delete and offering multidimensionality makes kd-tree suitable for the following applications mentioned above