OBJECTIVE :-

To build a KD (K-Dimensional) Tree and to perform basic operations such as insert and basic search.

Apart from that, we use this tree for finding the nearest neighbour search in a logarithmic time and also prove that how it is better than the normal binary search tree.

Research paper used : https://www.researchgate.net/publication/331700383_The_k-d_tree_data_structure_and_a_proof_for_neighborhood_computation_in_expected_logarithmic_time

Description of Research Paper :-

The k-d tree data structure and a proof for neighborhood computation in expected logarithmic time - Martin Skrodzki , March 13, 2019

MOTIVATION :-

An efficient algorithm to find the nearest neighbour in logarithmic time compared to the polynomial time taken by the basic binary search tree. Also, a lot of industry applications like, for Eg: Connecting our call to nearest police station for reporting crime, in hospitals, keeping track of patients with similar medical condition, etc…

Other applications are : Efficient storage of spatial data, range search etc… We will try to incorporate these as well.

APPROACH :-

The method is to build the tree by insertion of values in nlog(n) complexity using linked lists or arrays and make it ready for finding the nearest neighbour. Let a point set P and a corresponding k-d tree built. A neighbourhood query for any point p ∈ Rd is then performed by traversing the tree to the leaf representing the box which contains the query point. From there, the query goes back to the root, investigating subtrees along the path where the splitting hyperplane is closer to p than the currently found nearest neighbours. When reaching a node, all elements in it are investigated as to whether they are closer to p than the current closest points found. The algorithm is fast, as it can be expected that several subtrees do not have to be investigated.

ALGORITHM :-

Algorithm for Nearest Neighbour Search in k-d trees. (Much improvements are done in the coding part)

```
1: procedure NNk-dTree(point p, k-d tree T, distance ε, number k)
```

```
2: L ← empty list
3: return NNk-dTreeRec(p,root,$\varepsilon$, k,L)
4: end procedure
5: procedure NNk-dTreeRec(point p, k-d tree T, distance $\varepsilon$, number k, list L)
6:     if T = $\emptyset$ then
7:         return L
8:     end if
9:     Extract point $p_j$ from T.root and store it in L if $\|p_j - p\| \le \varepsilon$.
10:    if L is larger than k then
11:        Delete the point with largest distance to p from L.
12:    end if
13:    if T is just a leaf then
14:        return L
15:    end if
16:    if T.root.leftSubtree contains p then
17:        $T_1$ = T.root.leftSubtree, $T_2$ = T.root.rightSubtree
18:    else
19:        $T_2$ = T.root.leftSubtree, $T_1$ = T.root.rightSubtree
20:    end if
21:    NNk-dTreeRec(p, $T_1$, $\varepsilon$, k,L)
22:    if $|L|$ < k and $\|p -$ T.root.hyperplane$\| < \varepsilon$ then
23:        NNk-dTreeRec(p, $T_2$, $\varepsilon$, k,L)
24:    else if $\|$L.farthest $- p\| > \|p -$ T.root.hyperplane$\|$ and
       $\|p -$ T.root.hyperplane$\| \le \varepsilon$ then
25:        NNk-dTreeRec(p, $T_2$, $\varepsilon$, k,L)
26:    end if
27:    return L
28: end procedure
```

EXPECTED RESULTS :-

Return of the nearest neighbour and comparison with the regular BST by showing the wall clock time too. Successful insertion and search are also displayed. Also additional search predicate is also shown.

FUTURE WORKS :-

The use of KD tree in

1. Efficient storage of spatial data, and
2. Database queries involving a multidimensional search key

Will be worked upon.