Le Xuan Thuong
Professor Ted Pawlicki
CSC 172
April 13 2016

K-d Tree

What is a K-d Tree?

K-d tree is a space partitioning data structure represented by binary space partitioning tree. It is generally used for keeping k-dimensional spatial data as points. K is denoted by k∈ (1,2,...k) or k ∈(x,y,z,....k) and any other variations, where sets represent dimensions.Each coordinate in k□ dimension serves as a key for searching the tree.

Construction and Methods

Construction of K-d tree greatly varies depending on purposes and types of trees that you are trying to build. Generally the k-d trees are unbalanced, as it is difficult (impossible) to implement tree rotations in high dimensions. Some specific balancing techniques are used in k-d B tree, divided k-d tree, pseudo k-d tree and etc., many of which are adaptive k-d trees.

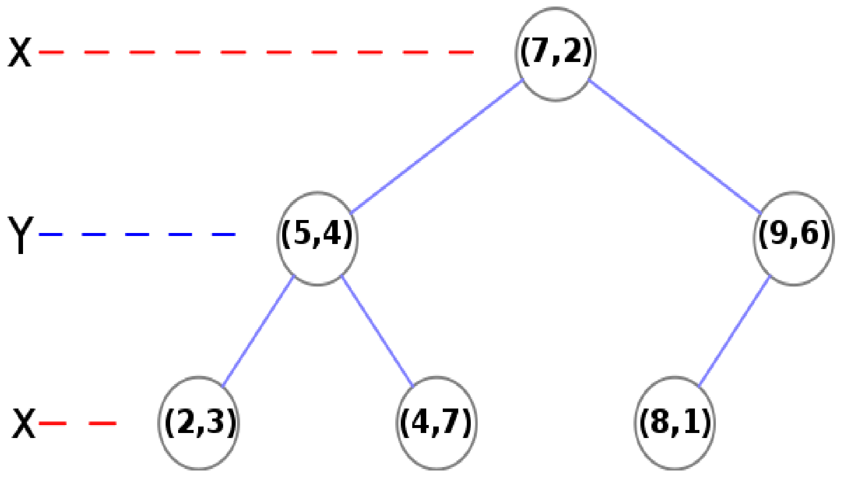
The k-d tree I am going to talk about is the one I constructed in my project: basic version of k-d tree that allows inputting points of any number of dimensions desired. It works similarly to Binary Search tree where insertion is based on series of comparisons and where all the nodes in left subtree contain data < data in nodes of right subtree.

Let's start from ***constructor***:

Constructor of the KDNode (which is a tree itself) is no different than that of the Binary Search tree, except the type of data stored at the node. In k-d tree data is an array of doubles, such as {1,2,0,9} where 1 is on x axis, 2 is on y axis and etc..

In ***insertion*** method we take as parameters double [] array and depth. The axis of comparison is defined by depth % array.length, as the length of array represents number of dimensions.

Traversing the tree recursively we keep cycling through axis of comparison. For example, at root node we compare data at axis x, at root's child we compare data at axis y, and if we only have two dimensions at root's grandchild we compare data at axis x again.



For instance imagine inserting point (3,6) in this k-d tree. We first compare point's x to root's x (3<7). Then we go to left subtree and compare y coordinates (6>4) and go to right subtree (3<4). Point (3,6) becomes the left child of node with (4,7).

Find minimum method is required for deletion.This method is not looking for the smallest point in the tree, it returns the node with smallest coordinate in given dimension.

```
public  KDNode findMin(int searchAxis, int depth, int k)
```

If dimension at the current level is the same as searchAxis, recursively call findMin() in left subtree (if left is not null)

If dimension at the current level is not the same, call finMin() on both left and right subtrees and on current node and return the minimum of all three.

In ***delete*** method there are 3 cases at every node:

1. If node to be deleted is a leaf -> simply set it to null (as in BStree)
2. If node's right child is not null -> find minimum in right subtree, replace node's data with min, and recursively delete min from right subtree
3. If node's left child is not null -> find min in left subtree, replace node's data with min, recursively delete from min from left subtree and set left subtree as right child of node.

One additional method I would like to mention is *Nearest Neighbor Search (),* and it is particularly useful in several applications to be discussed in the rest of the paper. It looks for the point in the tree closest to the given point.

Nearest neighbor search works as following:

1. Traverse the tree same way as in insert method
2. When the node is leaf, save the distance between the node's point and given point.
3. Unwind the tree by going back to parent's node. If distance between parent's node point and given point is smaller than the current best, update current best
4. If parent has two children, we find the distance between parent's coordinate at current axis (if parent's axis is x, take coordinate x) and given point's coordinate at current axis. If it's < current best distance, check the other child.
5. Return the point

How is K-d tree different from other Data Structures

As a space partitioning data structure, in terms of functionality k-d tree is usually compared to **Octree**, 3-d space partitioning tree. Whereas octree is advantageous in 3d space, k-d tree works better in high and low dimensions.

It mostly resembles BSTree, as insertion and partly deletion are practically same with the difference in comparisons, and types of data stored. Whereas for bstrees we can only store 1-2 dimensional data, k-d tree can store any dimensional data.

K-d tree's runtime analysis (average):

O(log n) for insertion

O(log n) for deletion

O(log n) for nearest neighbor search

Space wise it takes O(n) and for all of the above mentioned worst case is O(n).

Applications

K-d tree can seem a theoretical data structure, however due to easy implementation, abstraction and good run time, K-d tree holds multiple important applications, some of which are:

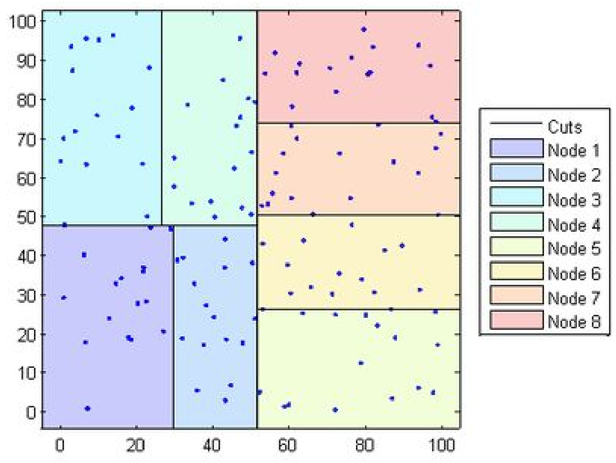
Nearest Neighbor search:

At first, nearest neighbor search can look trivial, however, at a deeper look, it is widely implemented in apps, maps, personal assistants and other data queries. For instance, if you are intending to build an app, that looks for nearest restaurant (such as GrubHub), k-d tree can quickly give you a coordinate and distance to the one. You might not think of this, but if you need medical help, k-d tree would provide you with nearest hospital you can go to and will save your life!

Range queries:

Databases

Suppose your company needs to send out emails to all the employees aged from 24-30 who earn from 45000-60000$. Exhaustive method might take a year to perform, when k-d tree would work instantaneously. By inserting x = age of the employee and y = salary into the tree, we can simply find which points (employees) are located inside requested regions.



Collision detection:
Computer Graphics
Computer graphics provide plenty of applications for k-d trees' performances, as they directly interact with points in 3d space. As we mentioned before, k-d tree is less used in 3 dimensional space, but still in such problems, as collision detection and optimization, k-d tree efficiently breaks space into cubes and bounds objects and characters in virtual reality. Sometimes, when only a certain part of the object is supposed to react to some action, game makers set some specific part of the tree (that bounds a certain part of object) rather than the whole tree to fire up at the action.
N-body problem:
Physics
How to predict the movements of collection of objects given mutual gravitational attraction? The first, and not the best solution would be to calculate the movement for each object individually, but instead by partitioning the space into subdivisions, we can find the total effect of gravitation on the rest of the space. It involves calculating the effect at the leaves and returning the value to its parent until recursively the method reaches the root.
Color Reduction:
What is an intelligent way to pick 256 colors to represent a full color image?
Naively, we would pick 256 most popular colors hoping to fill the image with them. Instead, k-d tree can partition the space until the number of leaves is equal to 256. Taking the average value of each leaf and returning it would give us a more precise color palette for the picture.

Overall

K-d tree is extremely easy to implement and use for such seemingly difficult problems as described above. If built carefully and if balanced, k-d tree doesn't take neither much space or time to perform and for these reasons I'm happy to present the k-d tree.

WebSources:

Wikipedia "k-d tree"

Quora “What is a k-d tree and what is it used for”
MIT Lecture “kd Trees”
GeeksForGeeks “K Dimensional Tree | Set 1 (Search and Insert)
| Set 2 (Find Minimum)
| Set 3 (Delete)