Nearest Neighbor Search · k-Nearest Neighbors (algorithm) · Data Structures · Machine Learning · +1

How does a k-d tree find the K nearest neighbors?

2 Answers

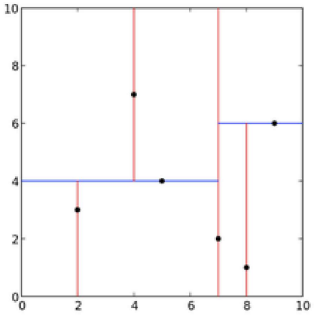
Steven Schmatz, BSE/MSE Computer Science, University of Michigan College of Engineering (1900)

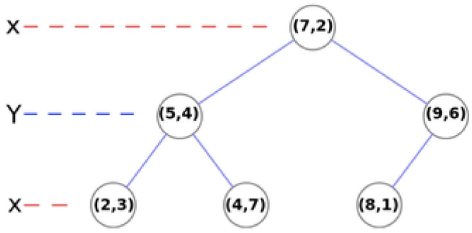
Answered May 29, 2018 · Upvoted by Kaan Yılmaz, M.S. Machine Learning & Complex Adaptive Systems, Chalmers University of Technology (2020) and Christian Howard, PhD Computer Science, University of Illinois at Urbana-Champaign (2024)

Originally Answered: How does a k-d tree work to find the K nearest neighbors?

K-d trees are a wonderful invention that enable $O(k \log n)$ **(expected) lookup times** for the k nearest points to some point x. This is extremely useful, especially in cases where an $O(n)$ lookup time is intractable already.

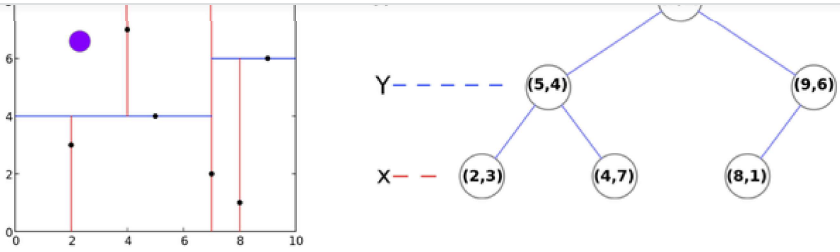
First, you need to understand what a k-d tree is. A k-d tree is a data structure that partitions space by repeatedly choosing a point in the data set to split the current partition in half. It accomplishes this by alternating the dimension which performs the split.



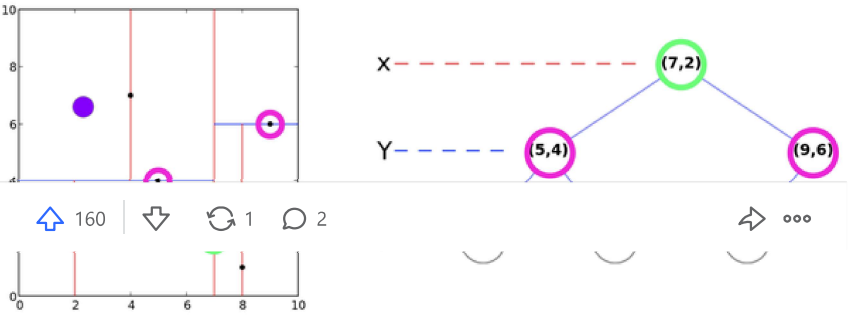


So here, we first start by splitting at the line $x = 7$, then we split by $y = 4$ on the left and $y = 6$ on the right. We do so recursively until there are no points left in each region.

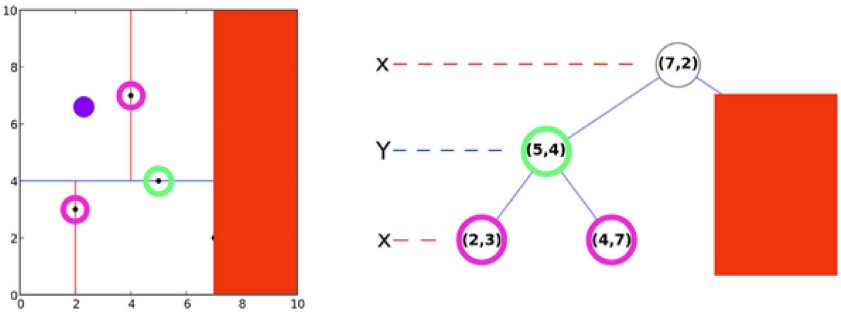
Now, it's simple to understand how a nearest neighbor can be found in $O(\log n)$ time. Say that we have the following purple query point.



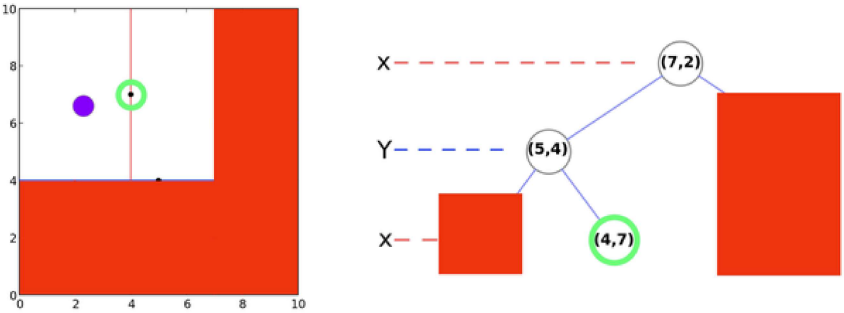
Now, we begin at the root of the tree. Now notice that our point **must** lie either on the left or right partition, but not both! That is, **we can eliminate an *entire* part of the k-d tree** by determining which of $(5, 4)$ or $(9, 6)$ are closer.



We can evaluate the Euclidean distance to find that $(5, 4)$ is closer. Hence, we can eliminate the right partition and continue recursively until the bottom.



And finally, we eliminate the bottom region, and we get:



And this is the basic gist of it! Note that it's possible that you could finish earlier, if the current root is closer to the query point than either of the children. If so, save that as the current best and continue down the tree.

If you're finding the *k* nearest neighbors, simply repeat this operation *k* times! However, rather than just the single nearest, then the process is very similar. You just ignore the nodes which you have already determined to be a nearest neighbor in a previous query.

I strongly believe that visual descriptions and color can be awesome tools to help teach computer science, and I think k-d trees are a great example of this.