K Nearest State/County Finder Using kd-tree

Yifei Wang

U54574131

1.Description

I am given a huge number of points as reference (static dataset). Each time when a user inputs a coordinate (latitude and longitude), the program shall return the K nearest reference points.

Note that, the distance will be computed using this equation[1]:

$$x = (\lambda 2-\lambda 1) * \cos((\varphi 1+\varphi 2)/2);$$
$$y = (\varphi 2-\varphi 1);$$
$$Distance = \sqrt{x*x + y*y} * R;$$

where φ is latitude, λ is longitude, R is earth's radius (mean radius = 6371km).

2.Design

2.1 Kd-tree

The k-d tree is a binary tree in which every leaf node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces[2].

Below is an example of a kd-tree. As we can see from the images, each node is a pivot which divides the space into 2 half spaces of exact same size, and the division is according to either x or y.

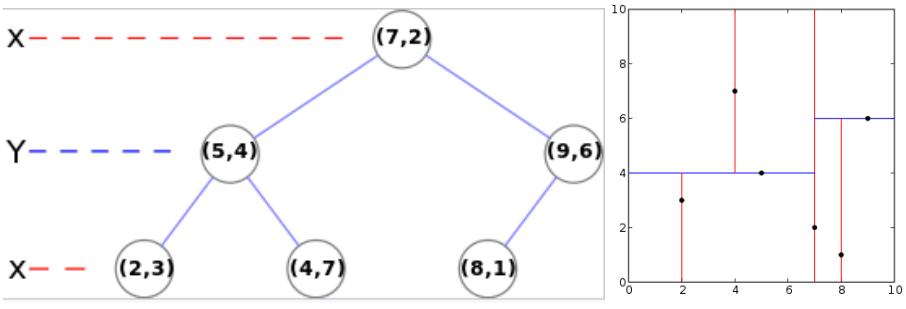


Figure 1 example of kd-tree[3]

As a result, we successfully divide the whole original space into several subspaces, and we hope to eliminate some subspaces when we search the tree (think of binary search tree), so that we can improve the efficiency of search.

When it comes to this project, I decide to use kd-tree to solve this problem. In general, I divide this problem into 2 subproblems: construction of kd-tree and query. In the end, I successfully solve the problem using this idea and apply 2 different construction methods, which will be compared with each other.

2.2 Construction

In the construction of kd-tree using our dataset, we keep dividing the space by the pivot, which is defined as the median of x or y. Once we find the median, we put all the smaller points into median's left child and larger ones into right child. Thus, the problem goes to median finding algorithm, which are discussed as follows.

2.2.1 Quicksort method

First, I apply some sort algorithm to find the median. The idea is that when all the points are sorted, the median would be easy to find. Specifically, I choose quicksort as the sort algorithm. A brief introduction of quicksort is that each time I choose a pivot and partition all the points recursively until all the points are sorted.

The complexity of quicksort is O(nlog(n)).

2.2.2 Random sample method

As we can see in quicksort, the process of sorting may be not time-effective. So in practice, instead of finding the exact median, I apply the random sample method to approximate the median.

The idea of random method is that, I only choose a sample of the dataset to sort, so that I get the median of the sample but not the dataset. However, in practice this method can be very efficient [3], because the complexity would be O(mlog(m)) if the sample size is m. Furthermore, if m is a small number such as 5 or 10, the complexity reduces to a small constant, which is O(1).

Note that if we want to separate all the n points into a binary tree, the number of division will be log(n). Thus, the total complexity of these 2 construction methods will be:

methods	quicksort	Random sample
Complexity of construction	$O(n(\log(n))^2)$	O(log(n))

2.3 Query

In a kd-tree search, we follow several principles:

(1) From root to leaf, we prefer to search the closer child first. (similar to BST).

(2) Each time we visit a node, update the best distance, and decide whether it is possible that the best node is in the other side of the tree. (check the brother of the current node)

As we can see from the steps, query of kd-tree is similar to that of binary search tree. In general, the complexity of searching one point will be O(log(n)).

2.4 Other requirements

In the previous parts I have established the method of construction kd-tree and how to search a input coordinate. Here I will explain how to solve other requirements in this project.

- Find k nearest points instead of the nearest one.

  The idea is to set up a priority queue with length k, and:

  (1) add a point if the queue is not full.

  (2) if it is full, compare new point with the worst point in the queue to decide which to stay in the queue.

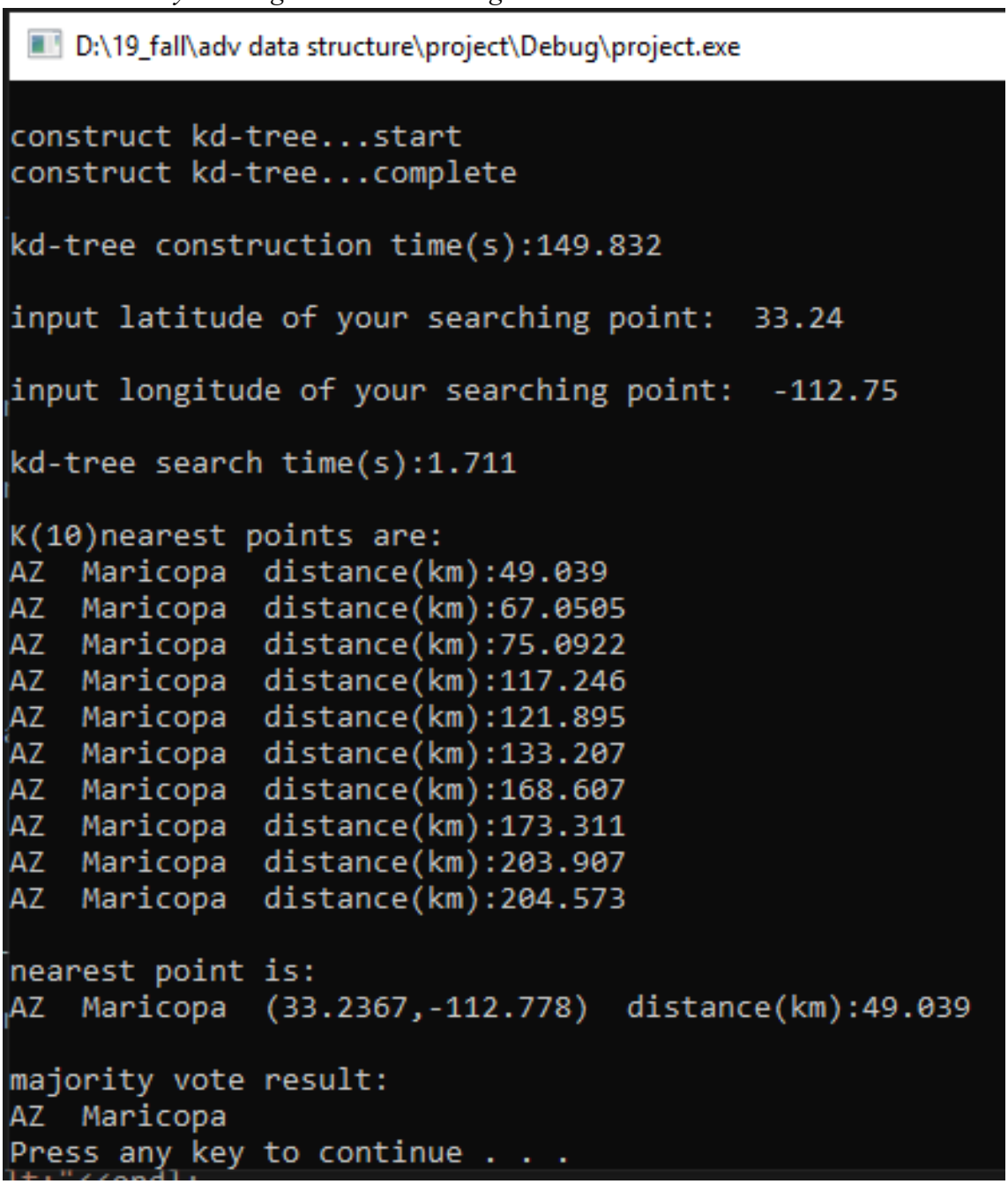
- Majority voting

  Count the frequency of state/county among the first 5 nodes in the queue. If there is a tie, choose the state/county with higher priority in the queue.

3. Implementation and comparison

I implemented this problem using C++, and the code can be found in the github link. If you follow the readme document, it can successfully run on SCC. In the following I will show a list of the features from the project proposal that have been implemented.

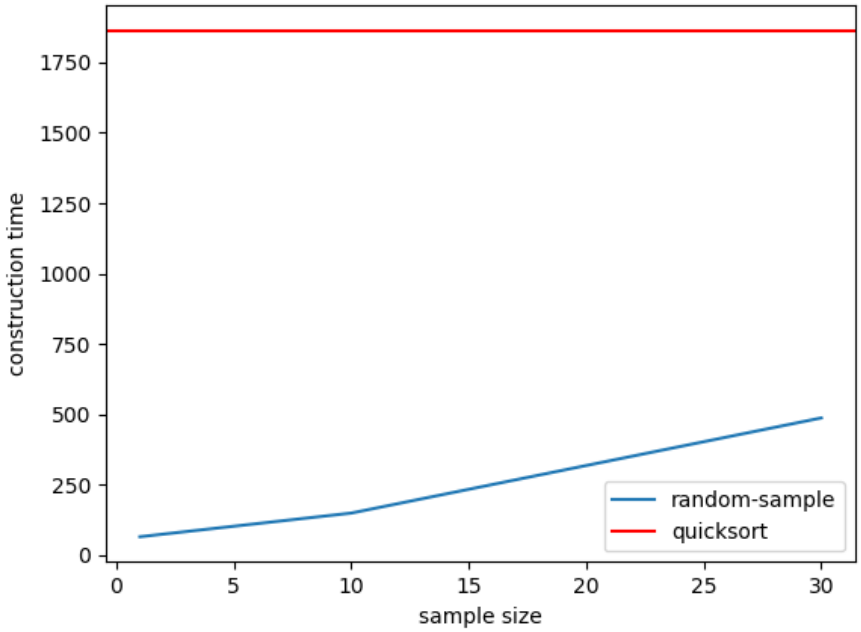
Minimum Requirements [60%]
• Command line user interface that lets loading of reference points, and querying of coordinates for
state and county with a given K nearest neighbors.



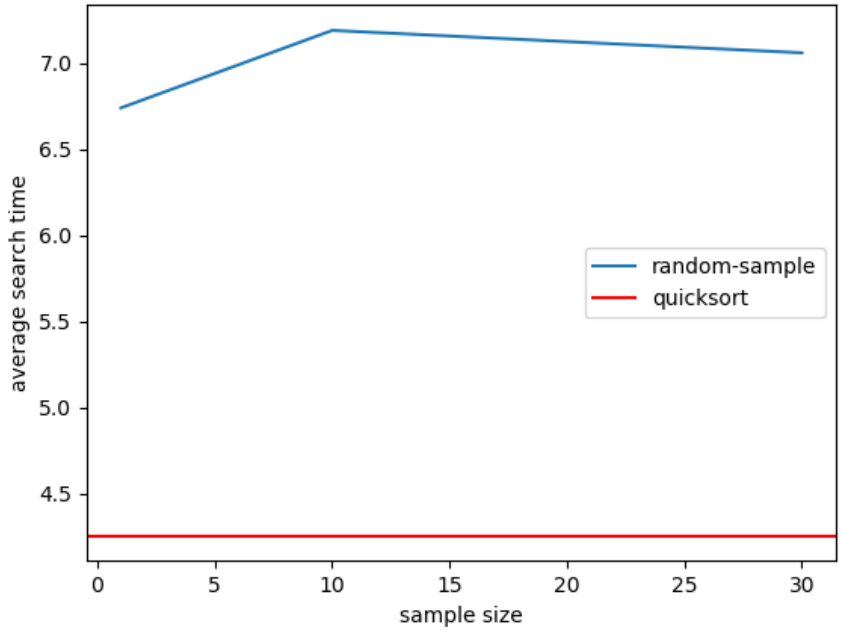
Possible Extensions [15%] Implement alternative approaches and compare speed of response for different data structures.

As shown before, I apply 2 different methods to construct the kd-tree, and the comparison is as follows.

Construction time(seconds)



Search time (seconds)



In conclusion, if we apply exact median finding algorithm, kd-tree does well in searching, but it is not very time-effective to construct the tree. On the other hand, random sample method can save much time in constructing, but on average it will

take more time to search, compared with finding an exact median. (so in practice we may need to make a balance)

4.Reference

[1]https://github.com/brower/EC504/blob/master/PROJECT_INFORMATION/EC504_Project_instructions.pdf

[2] Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching". Communications of the ACM. 18 (9): 509–517.

[3] https://en.wikipedia.org/wiki/K-d_tree