Nearest County Finder

Given a dataset of US Counties and their coordinates, find nearest county using a linked list and kd-tree.

Class: BU ENG EC504 - Advanced Data Structures

Team: David Abadi

Build & Run

To run the linked list solution, download and extract the zip file on your local machine, then run the following commands:

```
 cd ./sol_1/
module load gcc
make
./state_finder NationalFile_StateProvinceDecimalLatLong.txt
Enter k (1-10): 6
Enter latitude and longitude: 40.16482 -130.4679
```

To run the kd-tree solution:

```
 cd ./sol_2/
module load gcc
make
./state_finder NationalFile_StateProvinceDecimalLatLong.txt
Enter k (1-10): 6
Enter latitude and longitude: 40.16482 -130.4679
```

**Disclaimer: Need at least 1.5 MB of memory in order to run.

Definitions

Adapted from Wikipedia:

List

> Implemented as a doubly link list, an std::list has no fast random access. Doubly linked list consist of nodes with data and pointers to the previous and next node, with the exeption of the first node which only has a pointer to the next node and the last node which only has a pointer to the previous node. You can access the beginning or the end of the list and from there move to another node.

KD-tree

> KD-trees are use to organize points in multiple dimentions. Each level of the KD-tree is a partition on a different dimention, for example, for three dimentional points level 1 is a partition in x, level 2 a partition in

> y and level 3 a partition in z. Ideally each node has the median of the points that it is partitioning in that level. That way searching would have an average case of o(log n) and a worst time o(n).

Algorithms

Pseudocode:

List

Searching

Complexity: o(n)

> When searching for k nearest neighbors, keep an array of k best nodes. For all nodes in my list, check if the distance to the target is less than the distance from the worst element in our best nodes list, then place the node in the correct position of the array and eliminate the worst node in the array.

KD-tree

Build

Complexity: o(n log^2 n)

> Having all the points in a list, sort the list with respect to one dimention, in my case latitude, and pick the median as the root. Divide the list into two list partitioning at the median and then sort them according to their longitude. Pick the median of both list as the left and right child and recursively perform the pastitioning of the lists, each time alternating between latitude and longitude.

Searching

Complexity:

- average - o(log n)
- worst - o(n)

> When searchin k nearest neighbors, keep a list of k best nodes. Starting at the root check if the distance to the searching node is better thant the worst node in our list of best nodes. Set the left and right child as a good and bad side depending on which side is the point we are searching for. Recursevely perform the same operation as the root for the good side and if there could exist a point in the bad side that is better than at least one of our point in the best nodes list then recursevely perform the same operation in the bad side as well.

Results

> Both data structures were able to find the correct answers in less than 1 second for our number of data points. If the number of query is not big enough, it would be better to use the list because it does not have a build penalty like the KD-tree.

References:

- https://en.wikipedia.org/wiki/Doubly_linked_list
- https://en.wikipedia.org/wiki/K-d_tree
- https://gopalcdas.com/2017/05/24/construction-of-k-d-tree-and-using-it-for-nearest-neighbour-search/