CMSC 420: Coding Project 4
Extended KD Trees and KNN Queries

1 Due Date and Time

Due to Gradescope by Sunday 7 April at 11:59pm. You can submit as many times as you wish before that.

2 Get Your Hands Dirty!

This document is intentionally brief and much of what is written here will be more clear once you start looking at the provided files and submitting.

3 Assignment

We have provided the template `kd.py` which you will need to complete. More specifically you will fill in the code details to manage various aspects of extended KD trees. More details are given below.

4 Details

The class methods should do the following:

- `def insert(self,point:tuple[int],code:str):`

  Insert the `point`, `code` pair into the tree. Split according to either spread split or by alternating split with ties broken by first maximum spread coordinate of the point (for example if x and z have equal maximum spread, use x). The split method is set in the tracefile (see below). The point is guaranteed not to be in the tree.

- `def delete(self,point:tuple[int]):`

  Delete the `point` from the tree. The point is guaranteed to be in the tree.

- `def knn(self,k:int,point:tuple[int]) -> str:`

  Find the `k` nearest neighbors to `point`. This should use the method outlined in class and briefly explained here:

  - The list of points should always be sorted by distance (closest to furthest) to the target point with ties broken by code.
  - If we are at a leaf node: only update the list if the list is not full or if the points in the leaf are better (closer or the same distance but with better code) than the points in the list.
  - If we are at an internal node: If the subtree with the closest bounding box (with preference to the left one) might improve the list (meaning the list is not full or that bounding box is closer than the list's furthest element) then visit that subtree.

    After that is done if the other subtree might improve the list (meaning the list is not full or that bounding box is closer than the list's furthest element) then visit that subtree.

5 Additional Functions

You will probably want some additional functions as well as helper functions to handle the necessary operations.

6 What to Submit

You should only submit your completed `kd.py` code to Gradescope for grading. We suggest that you begin by uploading it as-is (it will run!), before you make any changes, just to see how the autograder works and what the tests look like. Please submit this file as soon as possible.

7 Testing

This is tested via the construction and processing of tracefiles.

- The first line in the tracefile is `initialize,splitmethod,k,m` which should initialize an instance of the `KDtree` class using splitmethod `splitmethod` (which is either `spread` or `alternating`), with dimension `k`, maximum leaf size `m`, and with root node `None`.
- Each remaining non-final line in a tracefile is either `insert,code,coord1,coord2,...` or `delete,coord1,coord2,...`. All together these lines result in the creation of a KD-tree.
- The final line is either `dump`, which dumps the tree, or `knn,k,point`, which finds the `k` nearest neighbors to the `point`.

You can see some examples by submitting the `kd.py` file as-is.

8 Local Testing

We have provided the testing file `test_kd.py` which you can use to test your code locally. Simply put the lines from a tracefile (either from the autograder or just make one up) into a file `whatever` and then run:

```
python3 test_kd.py -tf whatever
```