**AA Assignment 1 Report**

Jingxuan Feng (s3843790)

Varun Pereira (s3842244)

We certify that this is all our group’s original work. If we took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in our submission. We will show that we agree to this honour code by typing ``Yes": YES.

# **Experimental Setup**

The data is generated randomly, the RandomDataGeneration.java takes a number if parameters, which then outputs Datasets and a command file.

The dataset is generated by filling and array list with Points objects. These objects’ use a pre-determined longitude and latitude value, which then gets slowly incremented for every point that is generated, which prevents issues with repeated locations. These points are then stored in an array list, the array list is shuffled, then the points are printed out in an output file.

We tested using different sized datasets as well as different K values based off 1 randomly generated location. The parameters settings we decided to test on is the K-value and dataset size. We hypothesized that by increasing the K- value, the amount of searching by both algorithm increases which increases computation time. We also expect that increasing the dataset size has will have the same effect. Thus, we can then compare how different k values effect computational time vs how the same k-values perform on a larger dataset, then compare the naïve and KD tree implementations.

**Scenario 1**

We wanted to see the performance distance of finding nearest neighbours as the dataset increases in the two algorithms, thus choose to increase the dataset size to be multiple times larger than the sample data provided. The K-values used were kept constant across the datasets so that the dataset size is the independent variable. According to [7], the optimal k value depends on the dataset which could be large or small. Thus, we tested a range of k values from 2-20 to see how the increase effects the two algorithms.

**Scenario 2**

For this part, we use the same dataset size as scenario 1. The number of points to be removed then added is the square root of the size of the file, ie for 50000 points, we remove 223 points then add 223 points. Then we performed search again using the same spread of k values. By using the same dataset size, we can compare the knn searches in scenario 1 and see how removing then adding points effect the search time, compared to searching without tampering with the data set.

**Timing**

Our timing starts when as the first line inside the called method, it will then end and print out the time it took before the end of the method ie before the “return” statement or be the last 2 lines for a void method. When the command file is run, we will get a printout of how long each method took to run in the console.

When taking the results, we will run each test for each dataset in both scenarios 5 times, then average the results. The values used in the following graphs are an average of the 5 run times.

# **Evaluation**

## **Scenario 1 (k-nearest neighbour searches) (~1 page)**

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

From the results, we can observe similar trends across different data sizes. For the naïve implementation we generally see no change to time used to find the nearest neighbours represented by the k value, even as the k value increases from 2 incrementally by 2 to 20. This contrasts the Kd tree implementation, where across the three dataset sizes (5000, 10000 and 50000), as the k value increased, the amount of time increased exponentially.

## // Analyse, compare and discuss your results across different parameter settings, data structures/algorithms and scenarios. Provide your explanation on why you think the results are as you observed. You may consider using the known theoretical time complexities of the operations of each approach to help in your explanation.

## **Scenario 2 (Dynamic points set) (~1 page)**

//aa

# **Recommendation (and conclusion, ~0.5 page)**

For different scenarios, which data structures do you recommend to use?

//aa

## **References (~0.5 page)**

* [1] “k-d tree,” Wikipedia. Jul. 12, 2021. Accessed: Sep. 06, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=K-d\_tree&oldid=1033191291
* [2] “algorithm - How to efficiently find k nearest neighbours from a kd tree,” Stack Overflow. https://stackoverflow.com/questions/36603146/how-to-efficiently-find-k-nearest-neighbours-from-a-kd-tree (accessed Sep. 06, 2021).
* [3] “algorithm - How to implement nearest neighbor search using KDTrees?,” Stack Overflow. https://stackoverflow.com/questions/4093392/how-to-implement-nearest-neighbor-search-using-kdtrees (accessed Sep. 06, 2021).
* [4] “K Dimensional Tree | Set 1 (Search and Insert),” GeeksforGeeks, Oct. 31, 2014. https://www.geeksforgeeks.org/k-dimensional-tree/ (accessed Sep. 06, 2021).
* [5] “K Dimensional Tree | Set 3 (Delete),” GeeksforGeeks, Oct. 06, 2015. https://www.geeksforgeeks.org/k-dimensional-tree-set-3-delete/ (accessed Sep. 06, 2021).
* [6] “K Dimensional Tree | Set 3 (Delete),” GeeksforGeeks, Oct. 06, 2015. https://www.geeksforgeeks.org/k-dimensional-tree-set-3-delete/ (accessed Sep. 06, 2021).
* [7] S. Aggarwal, “K-Nearest Neighbors,” *Medium*, Jun. 08, 2020. <https://towardsdatascience.com/k-nearest-neighbors-94395f445221> (accessed Sep. 10, 2021).