COMP20003 Assignment 2: Experimentation

Kumi Sotomatsu 940966

**Introduction**

A variety of datasets with varying sizes and order of items (sorted or random) were created to empirically test the complexity of the algorithms in stage 1 and stage 2. In stage 1, we have an algorithm that finds businesses located in the point of the dataset that is nearest to the query point. In stage 2, the algorithm searches all the businesses in the dataset that are located within a certain radius from the query point. The algorithm for both stages were run using these datasets with keys in the query files, and the average number of comparisons between the node of the 2-dimensional tree, created from the datasets, and the query points were computed. The datasets and query files used in the experiment were created using UNIX commands as below:

|  |
| --- |
| tail -n +2 CLUEdata2018\_sortx.csv > sortx\_all.csv  for size in 3000 6000 9000 12000 15000 18000; do head -n $size sortx\_all.csv > sortx\_$size.csv; shuf sortx\_$size.csv > rand\_$size.csv; done  cat sortx\_all.csv | awk -F ',' '{if(($9~/^[0-9]+/) && ($10~/[0-9,.]+/) ) {print $9,$10;}}'|shuf> ./x  head -n 100 ./x | awk '{printf("%lf%lf\n", $1, $2)}' > q1.txt  head -n 100 ./x | awk '{printf("%lf%lf 0.0005\n", $1, $2)}' > q2.txt  for map in 1 2; do for size in 3000 6000 9000 12000 15000 18000; do for type in sortx rand; do ./map$map ${type}\_$size.csv out-$map-$type-$size.csv < q$map.txt > map$map-$type-$size.txt; done; done; done  for type in sortx rand; do for size in 3000 6000 9000 12000 15000 18000; do cat map1-$type-$size.txt | awk -v size=$size -v type=$type 'BEGIN{sum=0; n=0}{sum = sum+$4; n++}END{print ”map1", type, size, sum/n}' >> expts.txt ; done ; done  for type in sortx rand; do for size in 3000 6000 9000 12000 15000 18000; do cat map2-$type-$size.txt | awk -v size=$size -v type=$type 'BEGIN{sum=0; n=0}{sum = sum+$5; n++}END{print ”map2", type, size, sum/n}' >> expts.txt ; done ; done |

Query files containing 100 keys were used for each dataset. These were created by randomly shuffling the order of the sorted full-sized dataset and taking the x-coordinate and y-coordinate fields from the first 100 rows of data. The resulting number of comparisons for all the datasets with query files passed in were taken from expts.txt and analyzed.

**Method**

For stage 1 and stage 2

1. Remove the header file from the sorted file, CLUEdata2018\_sortx.csv, that was provided. Name this file sortx\_all.csv.
2. Create in total 12 sample datasets from sortx\_all.csv. Datasets of 2 kinds each of 6 different sizes being, 3000, 6000, 9000, 12000, 15000 and 18000 lines of data are created.
3. 6 datasets with different sizes in sorted order
4. 6 datasets with different sizes in random order
5. From sort\_all.csv, we create query files containing the queries to be used in the experiment. The x and y coordinate fields from sort\_all.csv are taken and the order of the key pair is shuffled. The first 100 keys are taken for the query files for both stage 1 and 2.
6. Query file containing 100 keys to be used for stage 1– named q1.txt
7. Query file containing 100 keys with input radius 0.0005 to be used for stage 2 – named q2.txt
8. Run the map1 program with the query file q1.txt and the map2 program with the query file q2.txt, both with the 6 different datasets to count the number of comparisons.
9. Get the average number of comparisons for each dataset by dividing by 100 (the number of keys in the query file) and store this output data for analysis into expts.txt.

**Data & Comparisons to Theory**

We expect in theory, a larger number of key comparisons as the size of the sorted dataset gets larger in both stage 1 and stage 2. This is because we have more rows of data that we have to compare to each key, and we expect an increase in linear time O(n). The data from the dataset is always inserted to the right node of the 2-dimensional tree and we have to in worst-case, look through all n items.

Figure 1

Theoretical Expectation of sorted dataset for both stage 1 and stage 2

Figure 2

Practical Outcome of sorted dataset for stage 1

Figure 3

Practical Outcome of sorted dataset for stage 2

However, this was not the case with the algorithm in stage 1, as it can be seen from figure 2. It does not follow the same trend and our expectation of O(n) in figure 1. This is interesting but can perhaps be explained by how larger datasets are more likely to contain the items in the query files. As the algorithm in stage 1 returns once an exact match in keys are found, the average number of comparisons might have been lower with larger data sets as they might have on average, not have to traverse down the 2-dimensional tree as much as smaller datasets.

Comparing figure 3 with figure 1, we can see that the time complexity of the search algorithm in stage 1 increased as the size of the dataset increased as expected. The complexity seemed to follow O(n), increasing in linear time. The algorithm in stage 2 continues to search for more possible points within the radius even if there is an exact match with the query point, giving the expected complexity.

Similar to sorted data sets, we also expect in theory, a larger number of key comparisons as the size of the random dataset gets larger in both stage 1 and stage 2. The reason is similar, because we have more rows of data that we have to compare to each key. However, this time we expect a time complexity resembling O(log2 n) so the increase would not be as great. This is because, with random datasets we can perform binary search.

Figure 4

Theoretical Expectation of random dataset for stage 1 and 2

Figure 5

Practical Outcome of random dataset for stage 1

Figure 6

Practical Outcome of random dataset for stage 2

However, both search algorithms in stage 1 and stage 2 do not seem to follow the expected

complexity. Stage 1 could be explained similar to the explanation in the sorted dataset. Stage 2 on the other hand could be explained by possibly how the dataset was produced randomly. Since it was random, it might have contained some data that were in ascending or descending order making the 2-dimensional tree not as balanced as we would have expected it to be.

As it has been slightly mentioned above, it can also be observed that the number of comparisons to find the query points in the sorted dataset to much greater than the random dataset for both algorithms. This is in line with what we expect. In theory, the 2-dimensional tree created from the sorted dataset has a depth equal to the number of rows in the dataset and hence O(n) complexity, while the 2-dimensional tree created from the random dataset to resemble that of a balanced tree and hence O(log n).

**Conclusion**

In conclusion, it was verified that there is a great amount of difference in the number of key comparisons for sorted datasets and random datasets in general, even if the random dataset is not a nice median dataset. It was also found that the algorithm plays a big role in determining complexity, even if the dataset is the same and both are conducting linear search.

**Recommendations**

It would be interesting in future experimentation to repeat the experiments using a median dataset to find out if the algorithm in stage 2 in particular follow the O(log2n) behavior. It would also be interesting to experiment using queries that are different to that of the datasets, or a mix of different and same data. This can allow us to see find out if stage 1 algorithms for both datasets follow the expected complexity.