Due: December 22nd, 2015

Submitted to:

NAWWAF KHARMA

NB-TREE – Report

ENCS | Concordia University

COEN 352 Final PROJECT

Michael Gravino ID: 27088140 Buddy: McQuenton bonimy id: 26259812

Project 2

# Abstract

Data structures and algorithms are essential and fundamental aspects of every programming language. Most real world programs require large amounts of information to be sorted, stored, traversed, and recovered in short periods of time. For this purpose, there exist many indexing storage and retrieval algorithms. In this report we examine one algorithm, the NB-Tree, which is used to map multi-dimensional data points to a 1-D line easy and fast storage and retrieval. We discuss aspects of its operation, our implementation, and the results of that implementation.

# Objectives

The objective of question two was: Given a large number of highly-dimensional vectors of integers; design, implement, and test a method that is able to output all existing vectors within a given Euclidean distance of an input vector. Each input data set represented a grey-scale image of a person’s face. The program’s main objective is given an input image and a decided Euclidean distance; the program will collect all images in that range, sum them together, normalize the output, and then output a sum image.

# Method

The NB-Tree is a combination of an indexing technique and the B+-Tree data structure. The overall purpose of the NB-Tree is mapping data from a multi-dimensional data points to a 1D line through computing their Euclidean Norm [1]. This effectively reduces the dimensions of the data point. We then sort and store these 1D points into a B+-Tree for further operations. Each multi-dimensional data point which is entered into the NB-tree is stored using its Euclidean norm as the key [1].

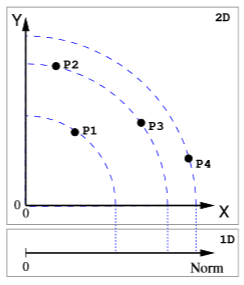


Figure : 2-Dimensional reduction to 1D [1]

Next, for the searching process, we use a single input as a query point. We calculate the norm of this query point then use an entered distance to determine the range of the search in 1D, using a higher and lower limit [1]. This is called a Range Query. Each point that is found in this range is added to a list of points to be returned, these points are sorted by their distance to the query point [1].

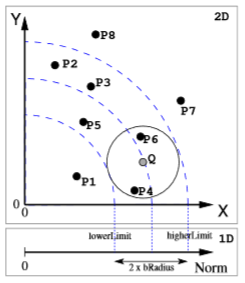


Figure : Range Query Visual [1]

For our project, our input data were grey-scale pictures in a multi-dimensional integer format from 0 -255, which represented the pixels in each photo. Each picture vector was normalized then stored in the NB-Tree, then input vectors along with a distance was entered to perform the range query. The range query returned a single output file which was the summation of all vectors within the determined range and normalized back into 0-255 so that it can be viewed as an image. Below is a flow chart that describes the top-level operations of our program when it is executed.

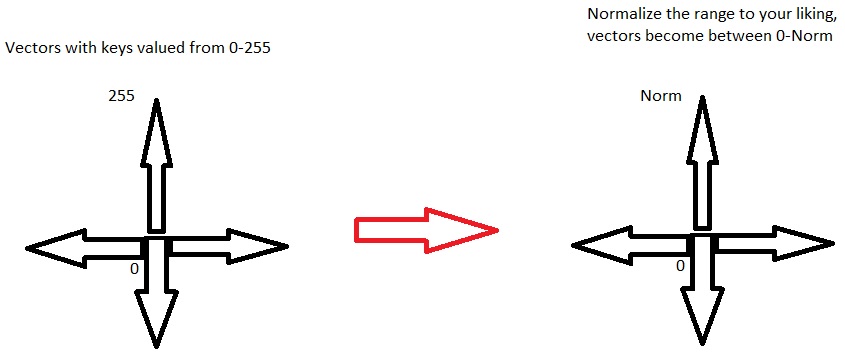


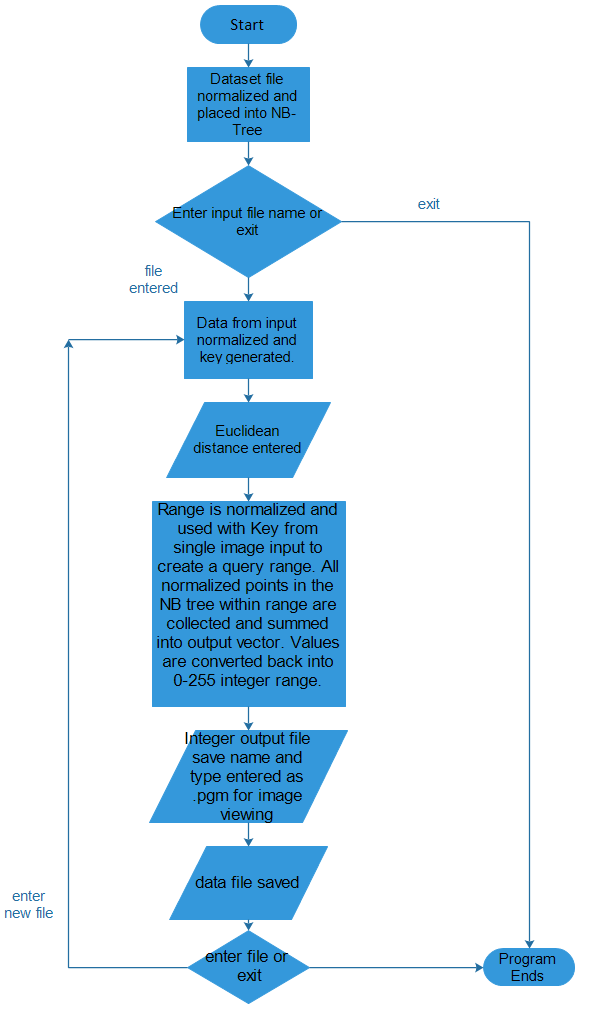
Figure : Conceptional figure of the normalization of the vectors

This is a figure representation of the vectors that are comprised of keys that are ranged between 0 and 255; we then normalize it using a b-tree to obtain a smaller range of the same values to make our program more efficient and clean. It will also help by minimizing the size of Euclidian distance used during each test. The time complexity of this program will be demonstrated in the next paragraph.

**Time Complexity:**

In theory, this program is basically inserting values into a b-tree and computating the norm, the computation of the norm should be constant time O(1) and the insertion into the b-tree is O(logN), all in all, everything should be done is O(logN) time.

Experimentally, we used 5 values to plot where N represents the number of vectors to be loaded into the program from the dataset. We inserted 10, 50, 200, 500 and 1000 vectors into the program and obtain these results.



exit

Figure : Flow-Chart for top-level description of our program

# Results & Discussion

|  |  |  |  |
| --- | --- | --- | --- |
| Input Order | Input Image | Euclidean Distance | Output Image |
| Input 1 |  | 250 |  |
| Input 2 |  | 45 |  |
| Input3 |  | 20 |  |
| Input 4 |  | 3 |  |
| Input 5 |  | 100 |  |
| Input 6 |  | 175 |  |
| Input 7 |  | 90 |  |
| Input 8 |  | 150 |  |
| Input 9 |  | 200 |  |
| Input 10 |  | 15 |  |

Table 1: Range, input images, and output images of the program.

As we see in the displayed results, the program successfully generated pictures that are sums of the photos that were in the range of the input. Some impacts that I noticed about the code and the dataset had on the output, is that the pictures that were used for the dataset had large sections where each picture was the same person making different expressions in various intensity and angles of light. I believe that according to the generated key of the input and the entered range, there may be many pictures in that range of another person with very different features, so the output photo would almost be unrecognizable from the input photo. My partner and I also fixed the problem in the code where the key was re-dimensioned using the sum of all the square roots of the keys of the vector instead of just normalizing it normally. Now as you can see through the results that the images do get more and more out of focus when the Euclidian distance is augmented, thus fixing the problem. In reality we used the square root function as a dimension redactor for all keys.

# Conclusion

# References

[1] Fonseca, M., & Jorge, J. (n.d.). Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases. Retrieved December 20, 2015, from http://virtual.inesc-id.pt/tr/mjf-jaj-TR-01-03.pdf

[2] "B-tree | set 2 (insert)," in *Advanced Data Structure*, GeeksforGeeks, 2013. [Online]. Available: http://www.geeksforgeeks.org/b-tree-set-1-insert-2/. Accessed on: Dec. 22, 2015.

[3] Lafore, R., & Stephens, M. (2003). *Data Structures & Algorithms in Java* (Second ed.). Retrieved December 20, 2015. Indianapolis, Indiana: Sams Publishing.