PROJECT-1

Gkika Violetta 1115201600222

Compile and run

Code can be compiled with the make command and can be runned as described in the project presentation. We have used the -03 flag for optimization.

Modules

Item: It represents an item of dataset and basically it just sotres a vector as in the dataset file storing in the first index the id and then the coordinates of the vector. It also has some functions to access that information.
Utils: Implements some basic helping function and more importantly the function that calculates the euclidean distance of two vectors.

Method: Is a mother class for LSH,CUBE AND exhaustive and is created with the purpose to accomplish run time polymorphism.

Exhaustive: Implements the exhaustive search which basically calculates the exact distance of a query from the data and returns the exact N nearest neighbors and all the neighbors found in the given radius(for range search).

Hash: Implements a hashtable to be used on LSH and Hypercube. There can be found all the functions that calculate h, g(for LSH) and f(for Hypercube) and all the functions to “handle” the hashtable.
*To store the information of the hashtable we have used unorder_multimaps where the key is either g or f depending on the used method and the value is the item that corresponds to that key.

LSH: Implements LSH algorithm.

CUBE: Implements Hypercube algorithm.

Clustering: Implements the B part of the project.

Input: This is the general module that handles all the reading of the input files,builds the appropriate structures to store that information, handles the printing of the output and also determines which function of which method should be called every time. The last one is achieved using compile-time conditions and passing the appropriate flags when giving the orders to compile modules in the Makefile.

Results

A.

Choosing the best value for w: To determine which value works best for w , we analysed the algorithm and specifically the function h. We know that w is used on h function and it determines the number of buckets and we know that to have good results for LSH , two points(vectors) that are close to each other should collide ,in some point, into the same bucket. So, to achieve that we need a w value that will guarantee that for each point p, q given the fact that p and q are neighbor vectors, so they have a small distance between them, that those points will hash in the same function.
h=floor[((p*v)+t)/w]
H value depends on random values p and t and the static value of w. So we can distinguish two different cases. Let's call p*v for point x, r(p) and for point q, r(q).

1. r(p)/w > r(q)/w => |r(p)-r(q)| > w, the two points will fall into different buckets for sure.
2. r(p)/w < r(q)/w => |r(p)-r(q)| < w, now it depends on the random value of t to determine if the buckets will fall in the same bucket or not.

Since t value is random we can not be completely sure that we can guarantee the property expressed on point 2.

So, generally we need a w value that is big enough to guarantee |r(p)-r(q)| = dist(p,q) < w.
We have implemented some functions to calculate and print the distances between all the points in the dataset and with raw testing of different values we observed that the values that worked best are values around 1000.
Functions for w are in the comment section and are currently not running. They can be found on Exhaustive.cpp and Input.cpp.

Hypercube:
We observed that for a small number of probes the hypercube method fails to find all N nearest neighbors, and this happens due to the fact that neighbors are spreaded in different vertices, and so the number probes*number of items checked < N.
For larger numbers of probes the hypercube method works better, giving a better time than LSH but less accurate results compared to LSH.

B.
-

Observations:
For small input files and small query files the project works well and fast. Also for big input files and small query files even though the program runs slower it returns satisfactory results.
For big input and big output the exhaustive method fails to return results so we made an “optimization” by limiting the number of queries tested. This part is currently commented out but you can uncomment to try it. The max number of queries is 5000.