ΜΑΧΖΟΥΜΠ ΑΔΑΜ 1115201600099
ΠΑΛΛΙΓΚΙΝΗΣ ΔΗΜΗΤΡΙΟΣ 1115201600122
COMPILE: make
CLEAN : make clean
RUN:
$./lsh -d <input file> -q <query file> -k <int> -L <int> -o <output file>
$./cube -d <input file> -q <query file> -k <int> -M <int> -probes <int> -o <output file>
$./curve_grid_lsh –d <input file> –q <query file> –k_vec<int> -L_grid<int> -o<output file>
$./curve_grid_hypercube –d <input file> –q <query file> –k_hypercube <int> -M <int> -probes <int> -L_grid -o<output file>
$./curve_projection_lsh –d <input file> –q <query file> –k_vec<int>-L_vec <int>-e <double> -o<output file> -size <arraySize>
$./curve_projection_hypercube –d <input file> –q <query file> -k_hypercube<int> -M <int> -probes <int>-e <double>-o<output file> -size <arraySize>
,where -size <arraySize> defines the size of the array for the traversals.
$./brute_force_curve –d <input file> –q <query file> -o<output_file>
$./ brute_force_points -d <input file> -q <query file> -o <output file>

inputA.txt,queryA.txt are used as in put for the first part of the exercise and PointsLSH.txt,PointsCube.txt as output.BruteForcePoints.txt serves both purposes.
BruteForceGrid.txt,QueryBi.txt,queryBii.txt, inputB.txt,inputBii.txt are used for the second part of the exercise as input,while ProjectionLSH.txt,ProjectionCube.txt,GridLSH.txt,GridCube.txt are an example of output.BruteForceProjection.txt serves both purposes.

!!! -probes in cube must be a number 5-10 and in curve_projection_cube, curve_grid_cube must be between 150-300,-probes must be <log(count of input) !!!

LSH :
Program reads the arguments. Then it fills every a hashtable with every point of input file, using the G hash function. The G function is created by the construction of K*h_functions. Each h function is created by using the values ai, where ai = florr((xi-si)/w) where si is a random value between 0 and w.When we fill our hash table, we hash the queries points and search the bucket until we exceed the threshold or we have no more points to compare with.

Hypercube :
We follow the same strategy as in LSH. This time though we add our points into a map using previous G functions. Any G funtions value gives us onebinary value (1,0). By using L G functions we get a string of length L from 0 and 1. We use this string as a binary key to our map. When we are looking for NN for our query we follow the same strategy searching the list of inputs with the same binary key, or list with at most 1 different key value f.e. if our query's binary string is "000" and our forbes are 2, we are going to check on lists with key : "000" and "001".
Grid:
Program reads the arguments and begins to build the grid curves,by initializing t[2] for every L,and find every point on the input files for every curve,then we calculate ai*delta+t for x,y,if the result was

not previously made then we add it to the vector(first all the x,then all the y),finally for vector that its less than the maximum size of the vectors we pad them with the greatest x,y.We then hash the inputs' vectors(LSH or Cube),hash the queries' vectors and search the bucket(or list for Cube) until we exceed the threshold or we have no more curves to compare with.

Projection :

Program reads the arguments and begins to build the grid curves,by initializing G(K*L uniformly canonical random numbers),calculate all traversals within the MXM array,where |i-j|<4,and find every point of the input files,just like in the Grid program.Vectors are for every x in the traversal(where I,for input, or j,for query, is equal to a traversal for an I,j array.We then multiply with the G array and concatanate the result so we have a size of K.Then we hash and search for the aNN just like in the Grid program.

In both programs we read brute force results to compare with aNN.