COMP7107B Management of Complex Data Types
Assignment 2
Point data
Due Date: March 25, 2025, 5:00pm

The goal of this assignment is to develop techniques for indexing and searching spatial data.

Part 1 (30%, index development)

Download file Beijing_restaurants.txt from Moodle. The file includes the coordinates of 51970 **points** which are locations of restaurants in Beijing. The first line has the number of points. Starting from the second line, each line includes the coordinates (x and y) of a restaurant. You are asked to write a program that constructs a simple spatial grid index. The grid divides the area covered by the points into 10*10 = 100 rectangles of equal size (**cells**).

To construct the grid, you should first read the points from the file to a data structure in memory and determine the smallest and the largest value at each dimension. To each of the points you should give an identifier (a unique integer) which should be the same as the line number in the file which includes the point. For example, the identifier of the point in the first line (39.856138,116.42394) should be 1, point (39.813336,116.486149) should have identifier 2, etc. Then, divide the range of values in each dimension into 10 equal-sized intervals. The next step is to sort the points according to the cell that contains them. This means that all points in cell (0,0) should be before all points in cell (0,1) etc. Write the sorted points to a text file grid.grd, wherein each line has the form <identifier x-coordinate y-coordinate>. In addition, create another file grid.dir (directory), with the following content. The first line of grid.dir should have the smallest and the largest value per dimension. Then, there should be one line for each non-empty cell, which includes the cell's coordinates in the grid (i.e., (0,0), (0,1) etc.), the position at the file grid.grd (in terms of characters) where the first line that includes the first point in that cell begins and the number of points in the cell. This way, if we have the coordinates of a cell, we can use the data in file grid.dir to locate and load all the points of the cell.

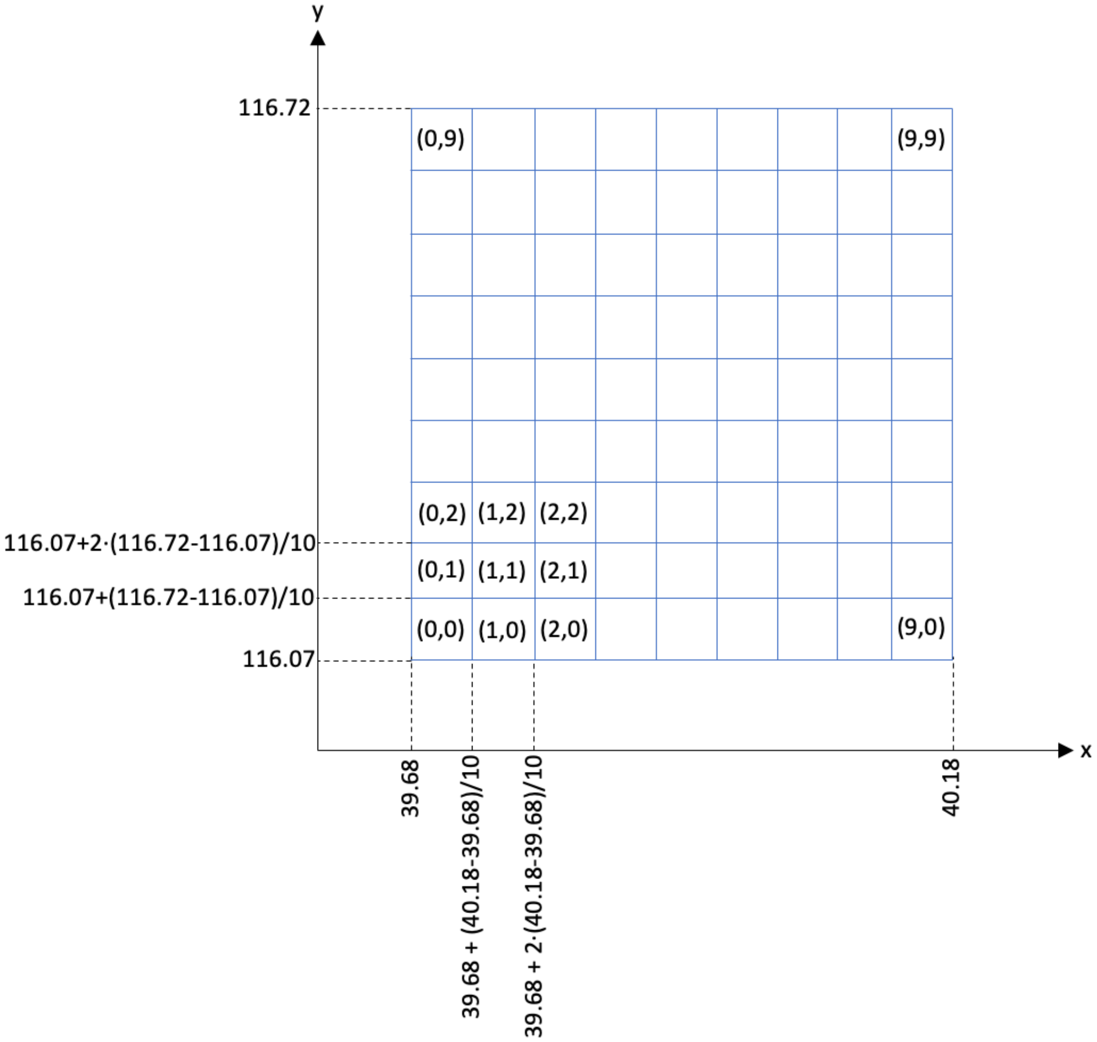
In the figures that follow, you can see a snapshot of the first 11 lines of the two files and a pictorial example of the grid. In the pictorial example, observe how the values that divide the range of each axis into 10 intervals are determined. Each point is assigned to the cell which contains it. If a value falls exactly on a line of the grid, it is assigned to the cell that follows this line. For example, a point with x-coordinate equal to 39.68 + (40.18-39.68)/10 and y value 116.072 falls in cell (1,0).

First lines of file grid.dir

```
39.680090 40.179911 116.070466 116.719976
0 0 0  108
0 1 2894 179
0 2 7688 20
0 3 8226 117
0 4 11368 356
0 5 20915 91
0 6 23352 58
0 7 24907 18
1 0 25393 83
```

First lines of file grid.grd

```
56 39.729270 116.119278
573 39.729398 116.128704
1253 39.723127 116.121828
1372 39.729585 116.127883
1395 39.729571 116.128738
2692 39.706018 116.123828
2846 39.727021 116.121720
3427 39.726623 116.120578
3804 39.723701 116.123683
4146 39.728675 116.135041
```

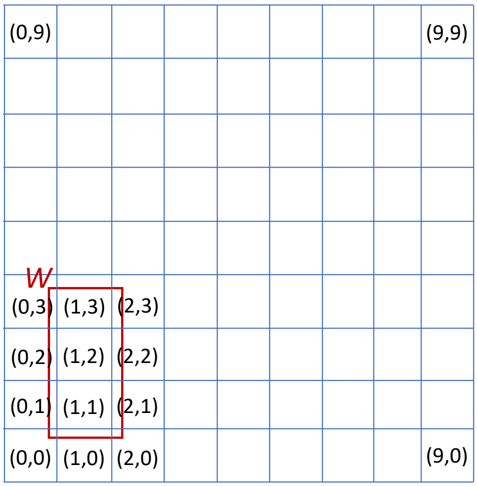


Verify the correctness of your program, by cross-examining the contents of the three files: grid.grd, grid.dir and Beijing_restaurants.txt.

Part 2 (30%, range selection queries)

You are asked to write a program which will use the grid that you constructed in Part 1, to evaluate window selection queries. Your program should take as command-line arguments the lower and the upper bound of the window at each dimension, i.e., four values <x_low> <x_high> <y_low> <y_high>. It should compute and print the points which are included in the window.

At first, the program should read the entire contents of file grid.dir and should store them in a data structure. This information will be used in the query evaluation. Then it should read all data for each cell which intersects the window. If the cell is entirely covered by the window, then all points in the cell should be reported without any comparisons. If the cell is only partially covered by the window, then we need to verify for each point in the cell if it is included in the window. In the pictorial example that follows, for the window query *W*, we report all points in cells (1,2) and (1,1) without examining their coordinates, while for cells (0,3),(1,3),(2,3),(0,2),(2,2),(0,1),(2,1),(0,0),(1,0) and (2,0) we have to compare the coordinates of the points to those of the window to confirm whether they points are inside *W*.



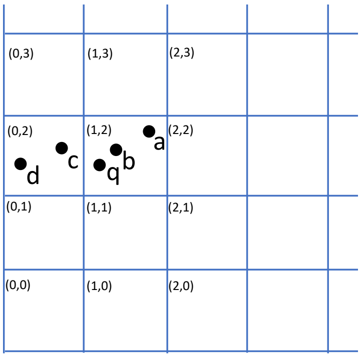
Check the correctness by comparing the result with what you take if you evaluate the window query directly on all the points of file Beijing_restaurants.txt. For checking, you should count and return the number of cells which intersect *W* for each query.

Part 3 (40%, nearest neighbor queries)

Develop a program which uses the grid index that you constructed in Part 1 to evaluate nearest neighbor queries incrementally. Your program should take as command-line arguments a number k and the coordinates of a query point q. It should output the k nearest restaurants. Initially, the program reads the entire file grid.dir and stores in a main-memory data structure its contents, which should be used in query evaluation. The nearest neighbor search function should be implemented as an iterator (e.g. as a generator function), such that in every function call the next nearest neighbor is returned. The program should call the function k times in order to find and print the k nearest neighbors of q in increasing order of distance.

Your function should use a priority queue which manages cells and points according to their distance to q. Initially the function should add to the queue the nearest cell to q (e.g., the cell that contains q). Then, the function should de-heap the element which is the nearest one to q (i.e., the heap's top element). If this element is a cell, all points inside the cell should be read from file grid.grd (you can use the data from grid.dir to locate them) and all these points are added to the heap. In addition, the neighboring cells of the de-heaped cell, **which have never been added to the heap before**, should be added to the heap. If the currently de-heaped element is a point, then it is returned as the next nearest neighbor of q.

Consider the example that follows, which depicts some cells, some points in them (a,b,c,d) and a query point q. Assume that we are running the function that you have to implement. Since q is inside cell (1,2), this cell is first added on the heap as the only heap content. Then, cell (1,2) is de-heaped and we add on the heap (i) points a and b which are contained in cell (1,2) and (ii) cells (0,3), (1,3), (2,3), (0,2), (2,2), (0,1), (1,1), (2,1) which are neighbors to (1,2). The next nearest heap element to q is cell (0,2), which is de-heaped and we add on the heap points c and d which are inside cell (0,2). However, we do not add any neighboring cells of cell (0,2) to the heap, because all of them have been added to the heap before. The next element which is deheaped is point b. Since b is a point, it is reported as the next nearest neighbor. When the function is called again, it continues from the point where it stopped, by de-heaping the next nearest element to q, which is cell (1,1). All contents of cell (1,1) and its neighboring cells (0,0), (1,0), (2,0) are added to the heap and the algorithm continues to de-heap cell (0,1), whose contents are read from file grid.grd and added to the heap.



You will have to write a function mindist, which computes the nearest Euclidean distance between a point (q) and a cell. This distance can be composed by the distances at each axis, as we have explained in class: it is the square root of the sum of the squared distances at each dimension.

Besides the k nearest points to q, your program should also print the cells whose contents were read by your function.

Deliverables: Submit your code for all three parts of the assignment using Moodle. You should also submit a report in a PDF file, which explains your code at a high level and includes any instructions for running your programs. You can use a programming language of your choice, but your program should be OS-independent. Please submit a single ZIP file with all programs and documents to Moodle on or before 5:00pm, March 25th, 2025. Make sure all contents are readable. Please do not submit any data files. Please feel free to post your questions on Moodle forum or contact the TA of the course if you encounter any difficulty in this assignment. We would be happy to help.