Department of Computer Science
The University of Hong Kong
COMP3323: Advanced Database Systems
Assignment 3 (10% of total marks)
Due date: Thursday, December 3, 2020, 11:59pm

In this assignment, you are going to do a "mini-research" on *k*-nearest neighbor search algorithms, for data points stored in a structure called the *grid index*. A grid index partitions the 2D space into equal-sized squares called *grids*, and data points are stored in the grid. Given a query point *q*(*x, y*), our goal is to find the k-nearest neighbors of *q* by using the grid index.

Part 1: Grid index construction

(1) *Data pre-processing* [http://snap.stanford.edu/data/loc-gowalla.html]

a. Prepare a dataset with attributes ([*latitude*], [*longitude*], [*location id*]), which contains spatial points called [*location id*], with 2D coordinates ([*latitude*], [*longitude*]). To obtain this dataset, please download ***loc-gowalla_totalCheckins.txt.gz*** from the above URL to your local folder, and decompress it. There is a total of 6,442,892 tuples, and we only need the 3rd, the 4th, and the 5th columns (i.e., [*latitude*], [*longitude*], [*location id*]).

b. Please check that the minimum and maximum longitude and latitude values are as follows:

x_min=-90.0, *x_max*=90.0, *y_min*=-176.3, *y_max*=177.5.

(Let us call this *maxBox* for convenience).

c. *Data deduplication:* There can be points in the dataset with the same locations but with different location ids. Here we only keep one such id. Hence, please delete the lines whose locations are same (longitude and latitude value), and only keep the one with the smallest location id. Note that the nodes with error coordinates should also be deleted, e.g., the nodes whose latitude is bigger than 400.

2) Next, divide the space defined by *maxBox* into ***n*** * ***n*** equal-sized rectangles called *cells*. For each data point *p*, assign *p* to the corresponding cell such that *p*'s coordinates are located within it. For each cell, you can access all the points and their associated information (e.g., their coordinates) associated with that cell. Notice that if *p* is located on boundaries but not on *maxBox*, then it should belong to the cell on its left or its top. An example of a 10*10 grid is given below, with red points as data points, and blue point as the query point *q*.

3) Save the index to a file named "index_path" as illustrated below.

```
************************************
Cell 0, 0: a.id_a.x_a.y b.id_b.x_b.y ...
Cell 0, 1: ...
...
Cell 0,99: ...
Cell 1,0: ...
...
Cell 99,99: ...
************************************
```

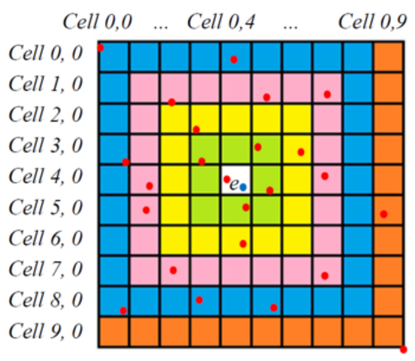

Fig. 1 An example of 10*10 grid index

[Example: in cell (0,0), two points with IDs *a.id* and *b.id* have coordinates (*a.x, a.y*) and (*b.x, b.y*)]

Part 2: k-nearest neighbors search (*knn_grid*)

1) Design a main-memory-based data structure to store the data points in Part 1. Specifically, load the disk file "index_path" (Part 1 (3)) to this data structure. Your data structure should be able to contain all the data in "index_path".

2) Given a query point *q,* please implement procedure *knn_grid* to find the *k nearest neighbors* of *q* from the data structure in Part 2 (1) efficiently. Please use the following steps for an efficient implementation:

a. Find the cell *e* that contains *q*. For each point *p* in *e* (if any), find the *k*-nearest neighbors of *q* by computing the Euclidean distance of *q* and *p*. Keep track of the *k*-NN set, and let *t* be the *k*-th largest distance in the *k*-NN set.

b. Progressively access the cells **layered** around *e*[1], and for each such cell *c*,
   i) compute *dlow*(*c*), which is the smallest distance from *q*;
   ii) if *dlow*(*c*)>*t*, skip the cell (*explain why is this possible in the report*);
   iii) otherwise access all points in *c*, compute their distances to *q*, and update *k*-NN.

c. Once the algorithm prunes an entire layer of cells, or finishes examining all the cells, report the query results. (*Explain why pruning an entire layer of cells is sufficient to terminate the algorithm*).

Part 3: Performance evaluation and report

1) Implement *knn_linear_scan*: this is a simple approach that solves the *k*-NN problem based on a linear scan (called *knn_linear_scan* in the code templates). It scans the points one by one in the original order from the dataset, and report the *k* nearest neighbors, *without* the help of the grid index.

2) Perform the following experiments: (*Hint: you can combine part c and d into one graph.*)
a. Generate 100 random query points within *maxBox*.
b. Plot a line graph to show the time of loading "index_path" to the main memory, the following value of *n:* 10, 50, 100, 150, 200.
c. Plot two line graphs by reporting the *average* execution times of *knn_grid* and *knn_linear_scan* for the query points generated in (a), against the following value of *n:* 10, 50, 100, 150, 200. For *knn_grid*, DO NOT consider the time for loading "index_path" to the main memory.
d. Plot two line graphs by reporting the *average* execution times of *knn_grid* and *knn_linear_scan* for the query points generated in (a), against the following value of *k=1, 2,.., 10*. For *knn_grid*, DO NOT consider the time for loading "index_path" to the main memory.

Please write down your experiment environments (e.g., machine used, CPU speed, disk speed, main memory sizes). Then, for the graphs in 2(b)-(d) above, write down a paragraph to explain in detail

[1] *Here 'layered' means accessing the cells around e in ascending order by the distance to e. For example, the cells in green will be accessed first, then those in yellow, pink, blue and finally those in orange.*

about your observations about the trends (e.g., why does the running time of *knn_grid* increases with *n*? How many times is *knn_grid* faster than *knn_linear_scan*?

How we evaluate your codes: Given a random *q* contained in *maxBox*, you should run *knn_linear_scan* and *knn_grid* and compare them. Your program will output the ids of *k*-NN, with input *q.x (double), q.y (double)*, *k (integer)* and *n(integer)*. To make it simple, we will use the following threshold to evaluate your program. $q.x \in (-90.0, 90)$, $q.y \in (-176.3, 177.5)$, $k \in [1, 1000]$, $n \in [10, 100]$. This part is fixed in the code templates given to you.

Part 4: Bonus (Up to 20%)

[Note: This part is optional, and we will award points based on the amount and quality of the work done, and clarity of the report. If you want to take up this challenge, try one or more of the following!]

- *knn_grid_fast:* Is it possible to improve the performance of *knn_grid*? For example, we can implement a priority queue to keep track of the top-*k* candidates in the *knn_grid* algorithm. Explain which part of *knn_grid* needs to be changed. Compare your results with *knn_linear_scan* and *knn_grid* based on the experiments in Part 3. Do you have any even brighter idea?
- *knn_grid_compress:* Is it possible to reduce the space needed by *knn_grid*? Propose a scheme that is more space-economical than *knn_grid*, with a data structure smaller than the one used by *knn_grid*, and write down your new scheme in the report. Show why it is smaller (e.g., by analyzing its complexity and in the experiments). Also, compare *knn_grid_compress* with *knn_linear_scan* and *knn_grid* according to Part 3.
- *knn_grid_disk:* Given a limited main memory buffer, implement a disk-based grid index scheme (called *knn_grid_disk*). The grid index file is organized in disk blocks, which are only fetched to the main memory if they are needed. Also, if the buffer is full, develop a page replacement policy to swap the pages between the main memory and the disk. Write down your scheme in the report in detail. Also, perform comparison of *knn_grid_disk* with *knn_linear_scan* and *knn_grid* according to Part 3.
- *knn_R_tree:* Implement *k*-NN search with an R-tree, and compare its performance with *knn_linear_scan* and *knn_grid*, according to Part 3.

Submission

Please submit two files for this assignment to Moodle:

1) *A PDF file, containing your report for assignment 3.*
2) *A zip file, containing a folder of C++ or Java codes.*
3) *A readme file to tell us how to compile and run your program.*

Note: each of your *k*-NN subroutines should input *q.x (double), q.y (double)*, *k (integer)* and *n(integer)*, and output the ids of the *k*-NN points.

Please DO NOT submit any other materials. No need to submit the index or dataset generated. No marks will be given for external libraries used. Late submission will NOT be accepted!