COMP7106 – Big data management

Assignment 2 – Spatial Data

Deadline: March 25, 2023, 5pm

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

Part 1 (50%, index development)

You are asked to write a program, which constructs an R-tree for a set of polygons using **bulk-loading**. You are going to read the data from two files given to you. Your program should compute the MBRs of each polygon and then bulk-load the MBRs to an R-tree. You are going to use the z-order curve for sorting the MBRs. You can find information about z-order at the lecture notes and at the following links:

https://en.wikipedia.org/wiki/Space-filling_curve
https://en.wikipedia.org/wiki/Z-order_curve

You are given two input files: **coords.txt** and **offsets.txt**. The first file contains coordinates of points at the form <x>,<y>. The second file includes records of the form <id>,<startOffset>,<endOffset>, where id is a unique identifier of a polygon object and startOffset (respectively, endOffset) is the line number in the file coords.txt where the coordinates of the points that form the polygon begin (respectively, end). For example, the first line in file offset.txt is "0,0,11" which means that object with id=0 is a polygon with 12 vertices; the coordinates of the first vertex are in line 0 of file coords.txt (i.e., -82.010671,32.183124), the coordinates of the second vertex are in line 1 of file coords.txt, …, the coordinates of the 12th vertex are in line 11 of file coords.txt.

By reading the two files (concurrently), access the coordinates of each polygon object and then compute the MBR of each object. For each MBR, compute the coordinates of the geometric center (by taking the mean of each dimension) and then compute the z-order value of the center. Use this value as the sorting key and "pack" the MBRs to form the leaf nodes of the R-tree. Then, construct the upper levels of the tree recursively. Please note that you do not have to sort the MBRs of R-tree nodes to form the upper levels, but you should just place them to non-leaf nodes in the order they are generated. For the tree construction, assume that each R-tree node has maximum capacity **20** and minimum number of entries 0.4*20 = **8** (the root of the R-tree is an exception because it can have less than 8 entries). This means that if the last node at each level has less than 8 entries, you need to adjust the number of entries in that node and the previous one at the same level, such that the last node has exactly 8 entries and the previous sibling less than 20. The leaves of the R-tree have entries of the form [id, MBR], where MBR is the MBR of the object (in the form [x-low, x-high, y-low, y-high]) with identifier id. The non-leaf nodes contain entries of the form [id, MBR], where id is the identifier of the R-tree node pointed by the entry and MBR is the MBR of all the entries in the pointed node (again, the MBR is in the form [x-low, x-high, y-low, y-high]).

If you are using Python, you may use function interleave_latlng from the following module:

https://github.com/trevorprater/pymorton/blob/master/pymorton/pymorton.py

to convert a pair of (x,y) coordinates to a z-order code. In this function, note that lat=latitude=y-coordinate and lng=longitude= x-coordinate, so you should call interleave_latlng(y,x) to get the z-order value for an (x,y) point (note that the return value is a string). You can sort the produced strings.

Alternatively, you are free to write your own z-order value computation function, if you wish to do so.

Your program should take as command-line arguments the two files that are given to you. At the program output you should print the number of vertices per level, for example:

500 nodes at level 0
25 nodes at level 1
2 nodes at level 2
1 node at level 3

Your program should write to output text file **Rtree.txt** all nodes of the tree in order of their id. The format of each each line of Rtree.txt should be as follows:

[isnonleaf, node-id, [[id1, MBR1], [id2, MBR2], …, [idn, MBRn]]]

where isnonleaf=1 if the node is not a leaf (isnonleaf=0), node-id is the id of the node and then follow the entries of the node. For each entry, id is either a node-id, if the entry points to a node (isnonleaf=1), or an object-id if the entry points to an object (isnonleaf=0). The MBR is of form[x-low, x-high, y-low, y-high] and it is either the MBR of a node (if isnonleaf=1) or the MBR of an object (if isnonleaf=0). For example, the first line of Rtree.txt could be as follows:

[0, 0, [[5868, [-170.844179, -170.707084, -14.373776, -14.287277]], ..., [3060, [-157.850181, -157.848054, 21.301518, 21.303834]]]]

The vertex at the last line of Rtree.txt should be the root (since it is the last finalized node).

Part 2 (20%, range queries)

Implement range query evaluation using the R-tree that you constructed. The input of the query should be an R-tree and a rectangle W and the objective is to find the object MBRs that intersect W. The function that implements range search should take as input the node-id of the R-tree root, the query rectangle W in the form [x-low, x-high, y-low, y-high]. To verify the correctness of your function, you are given file **Rqueries.txt**, which includes range queries W of the form:

<x_low> <y_low> <x_high> <y_high>

where x-low is the lower bound of W and x-high is the upper bound of W on the x-dimension (the bounds for the y-dimension are defined correspondingly). For each query, output the ids of the objects whose MBR overlaps with the window W of the query (filter step).

Your program should take as command-line arguments the R-tree file that you have created in Part 1 and the queries file Rqueries.txt. It should first construct the R-tree, by reading the contents of the Rtree.txt file and then read and evaluate each query from the query file. For each query, it should print at the output the line of the query (starting from line 0), the number of query results in parenthesis, and the ids of the results, i.e., the identifiers of the objects whose MBR intersects the query range.

Example:

0 (7): 2527,2712,8371,5042,7080,7656,7944

...

Part 3 (30%, kNN queries)

Implement the best-first nearest neighbor search algorithm. The version that you should implement is the incremental NN algorithm, which uses a priority queue (heap) to organize the accessed R-tree entries based on their distance to the query point. The heap should include node MBRs and object MBRs. When an object MBR is de-heaped, it is the next nearest neighbor object to q.

Write a function that takes as input the root of the R-tree and a query point q, and computes and outputs the k nearest neighbor object-ids to q in the R-tree, based on the object MBRs.

Your program should take as command-line arguments the R-tree file you created in Part1, file **NNqueries.txt** and number k (number of nearest neighbors). It should first construct the R-tree, by reading the contents of the Rtree.txt file and then read and evaluate each query from the query file. For each query, it should print at the output the line of the query (starting from line 0), the number of query results in parenthesis, and the ids of the results, i.e., the identifiers of the k objects whose MBR is the nearest to the point (x,y) of the corresponding line in the query file.

Example (for k=10):

0 (10): 9311,7001,803,5361,6764,3905,1642,3260,4669,5762

…

Deliverables: You should submit your 3 programs and a single PDF file which documents the programs and includes any additional comments. You can use a programming language of your choice, but your program should be OS-independent. Please submit a single ZIP file with all requested programs and documents to Moodle on or before 5:00pm, March 25th, 2023. Make sure all contents are readable. Please do not submit any data files, your input files should be the same as given (please don't rename them or use any self-modified 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.