HW 2

Name: `Yuan Fangxu`

Stu No.: `20799126`

Task1

(1)

In this assignment, I referenced the open source R-Tree library implement by STRtree *(org.locationtech.jts:jts-core 1.18.0 API)*.

(2)

The algorithm is based on the following idea:

- Load all data and store them in a list.
- Specify d (the fan-out) and n (the size of the bucket).
- Build the tree from bottom to top.

Each node has three important characteristics:

1. An MBR with two points indicating its position
2. A list containing all the children of a node
3. An integer level indicating the level of the node,level = 0 means it is a leaf node, which will be used in DFS algorithm.

In fact, this algorithm does not involve any R-tree insertion operations. It simply builds the tree recursively from bottom to top. Once built, the tree cannot be modified. This construction algorithm is effective when the number of points is large.

The practical disadvantage, however, is that this algorithm does not support any R-tree insertion operations. It simply builds the tree recursively from bottom to top. Once built, the tree cannot be modified. However, when the number of points is large, this construction algorithm will be very effective.

I have designed several functions to meet the requirements of the task as laid out. **The main change** I made to the source code is that it does not support specifying the size of n buckets since it only supports specifying d (fan-out). I have modified it so that when it creates parents for leaf nodes, the number of parents is obtained from the number of points/n; when it creates parents for internal nodes, the number of parents is obtained from the number of MBRs/d at the current level.

(3)

1. For the first half of D:
    - N = 64, d = 8: height = 6, non-leaf = 497, leaf = 2161
    - N = 64, d = 32: height = 4 , non-leaf = 114, leaf = 2118
    - N = 256, d = 8: height = 5, non-leaf = 127, leaf = 537

    - N = 256, d = 32: height = 3, non-leaf = 29, leaf = 535
2. For the entire D:
    1. N = 64, d = 8: height = 6, non-leaf = 991, leaf = 4308
    2. N = 64, d = 32: height = 4, non-leaf = 216, leaf = 4264
    3. N = 256, d = 8: height = 6, non-leaf = 252, leaf = 1064
    4. N = 256, d = 32: height = 6, non-leaf = 57, leaf = 1056

Discussion

A particular feature is that d (fan-out) can greatly affect the height of the tree. The height of the tree decreases when more branches are possible in the non-leaf nodes. This design idea is widely used in real-world database software. For example, in the B+ tree index structure of MySQL's InnoDB engine, a single page (non-leaf index node) can contain over 1000 branches, which makes most B+ trees less than or equal to 3. When the tree height is small, the number of I/O operations is reduced.

Task 2

(1)

Design ideas:

I use depth-first search to implement nearest neighbours using R-trees. Also, I use a variable to store the nearest distance to date to the corresponding points in the query points Q and D. During DFS, I update this variable if a point is closer to Q.

- Start the DFS at the root.
- If a leaf node is encountered, fetch all points within that node. Calculate the actual Q between Q and each point inside and the actual distance between each point inside. If a closer point is found, update the closest distance so far if a closer point is found.
- If an interior node, i.e. a non-leaf node, is encountered, all MBR index nodes belonging to that node are obtained. Pruning rule 1 and rule 3 are applied to these MBRs. next, they are sorted in mental ascending order. The DFS of these MBRs is continued according to the aforementioned order.

The reason I do not mention pruning rule 2 is that rule 2 is designed to remove a point that is not necessarily the closest; it is not used for pruning. In DFS, when a leaf node is encountered, a point that is not necessarily the closest is automatically replaced, the actual distance is calculated and a point closer than the previous candidate is found.

(3)

	Random point	Nearest Neighbor	L2 distance	N nodes visited	N points calculated	Rule 1 (MBR)	Rule 2 (Object)	Rule 3 (MBR)	min_time(ms)	max_time(ms)	avg_time(ms)
0	(115.5741, 40.6977)	(115.7757, 40.5198)	0.268942	6	150	19	22500	0	2	26	7.6
1	(116.9935, 39.9337)	(116.9172, 39.8187)	0.137974	9	213	33	45369	0	36	234	98.6
2	(117.1504, 40.8416)	(117.1561, 40.6932)	0.148511	6	195	22	37359	0	3	9	5
3	(116.8586, 39.6477)	(116.8336, 39.6622)	0.028864	6	163	19	26331	1	18	28	21.8

4	(115.8082, 40.7236)	(115.8890, 40.5444)	0.196633	6	150	19	22480	0	0	1	0.4
5	(116.2519, 40.0754)	(116.2437, 40.0647)	0.013455	6	168	17	25694	1	17	30	20.8
6	(116.8776, 39.5007)	(116.7779, 39.5949)	0.13715	6	189	20	35721	0	0	1	0.4
7	(116.8906, 40.4121)	(116.4541, 40.1169)	0.526886	6	223	15	49729	0	18	58	32
8	(116.1946, 39.9648)	(116.2796, 39.9631)	0.085088	7	227	19	51302	1	22	73	42.8
9	(117.2482, 40.8875)	(117.1561, 40.6932)	0.215058	6	195	22	38025	0	0	1	0.2
10	(115.8902, 40.8190)	(116.0541, 40.5770)	0.292285	6	239	19	57121	0	17	28	20.4
11	(116.2312, 40.4864)	(116.2621, 40.1331)	0.354653	9	297	22	44525	0	0	1	0.2
12	(116.9310, 40.0847)	(116.4541, 40.1169)	0.477929	6	223	14	49683	1	16	18	17.2
13	(117.1983, 40.9272)	(117.1561, 40.6932)	0.237808	6	195	22	38025	0	0	1	0.2
14	(116.7615, 40.9457)	(116.7034, 40.8784)	0.089004	6	183	20	31517	0	17	71	36.4
15	(116.1770, 39.9726)	(116.1914, 39.9434)	0.032583	8	169	25	28561	0	0	1	0.4
16	(117.2436, 39.4912)	(116.9012, 39.6844)	0.393121	6	110	20	12100	0	16	23	18.2
17	(116.4040, 39.5386)	(116.4487, 39.5566)	0.048214	6	137	20	18769	0	0	1	0.2
18	(115.9765, 40.6393)	(116.0333, 40.5618)	0.09607	6	239	19	57121	0	16	18	17.2
19	(117.1500, 39.8919)	(117.0901, 40.0711)	0.188965	6	121	16	14641	0	0	1	0.4
20	(116.5356, 40.5525)	(116.4436, 40.1310)	0.431485	6	223	15	49729	0	17	106	38.8
21	(116.1111, 40.2665)	(116.1335, 40.1325)	0.135852	6	163	16	26569	0	0	1	0.2
22	(117.1880, 39.5348)	(116.9012, 39.6844)	0.323482	6	110	20	12100	0	16	21	17.8
23	(116.8028, 39.8324)	(116.7479, 39.9526)	0.132144	6	223	19	49729	0	1	1	1
24	(115.8186, 39.9924)	(115.8024, 39.9870)	0.017041	7	209	14	29883	4	17	23	18.6
25	(117.1485, 40.4169)	(116.9148, 40.0871)	0.404218	6	147	18	21609	0	0	1	0
26	(116.2201, 40.8210)	(116.2638, 40.7039)	0.124973	6	188	18	27358	3	18	95	38.6
27	(116.0402, 40.1053)	(116.0987, 40.0643)	0.071421	6	189	15	35721	0	0	1	0.8
28	(117.1442, 40.1338)	(117.1159, 40.1361)	0.028436	8	393	20	77985	0	0	1	0.2
29	(115.6341, 40.9369)	(115.7757, 40.5198)	0.440486	6	150	19	22500	0	0	1	0.8

Discussion

From the result we can see that the R-tree NN algorithm typically takes less than 1 ms for a given dataset, d and n parameters, whereas an exhaustive search typically takes a lot of time.

In R-tree NN, the number of points for calculating their actual distance from Q is greatly reduced, usually to around 1000. This is a very small number compared to the total number of 182,323 points.