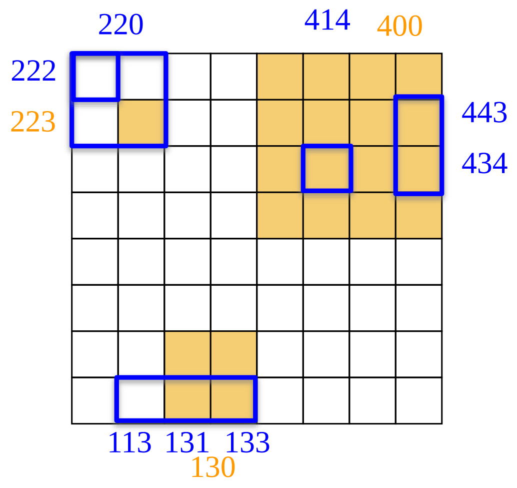
INFS4205/7205 Advanced Techniques for High Dimensional Data

Tutorial 4: Spatial Query Processing 1&2

Semester 1, 2021

Question 1: Sort-Merge Spatial Join with Z-Values. Suppose we have two sets of objects approximated by Z-values. The first set S is labelled in blue, and the second set R is labelled in yellow. Each object's Z-values are labelled correspondingly. Now describe the procedure to use these Z-values to do the sort-merge join.



Example Solution:
Firstly, we sort the Z-values of the two sets:

- {113, 131, 133, 220, 222, 414, 434,443}
- {130, 223, 400}

Then we visit both these two sorted lists from the smallest values at the same time:

1. 113 and 130, because 13 and 30 do not intersect, we skip this pair. Because 113 is smaller than 130, and 130 is at a higher level, we visit the next value in S.
2. 131 and 130, because 30 covers 31, we add (131, 130) into the candidate list. Because 131 is larger than 130, we visit the next value in R. Meanwhile, because 130 is at a higher level, we have to keep 130 in stack (do not remove it from the comparison list).
3. 131 and 223, they do not intersect, and 223 is larger than 131, we visit the next value in S.

4. 133 and 130, because 30 covers 33, we add (133, 130) into the candidate list.
5. 133 and 223, they do not intersect. Because 133 is smaller than 223, we visit the next value in S.
6. 220 and 130, they are totally different, and 130 is smaller than 220, they are at the same level, so it is guaranteed that no value in S will intersect with 130. Therefore, we remove 130 out of stack.
7. 220 and 223, because 220 covers 223, we add (220, 223) into the candidate set. Because 220 is at a higher level than 223, we put 220 into stack, and we visit next value in S.
8. 222 and 223, they do not overlap, and we visit the next value in S.
9. 414 and 223, they do not intersect, and 414 is larger than 223, we visit the next value in R.
10. 220 and 400, they do not intersect, we move 220 out of stack.
11. 414 and 400, 414 is covered by 400, so we put (414, 400) into the candidate list. 400 is the last value of R, and it is at a higher level, so we put 400 into the stack, and visit the next value in S.
12. 434 is covered by 400, so we add (434, 400) into the candidate list, and we visit the next value in S.
13. 443 is covered by 400, so we add (443, 400) into the candidate list, and the merge finishes.

After the procedure of the sort-merge, we have obtained the candidate list {(131,130), (133,130), (220,223), (414,400), (434,400), (443,400)}. We might do some housekeep check because some of the testing pairs are duplicated, like (131,130) and (133,130). Both 131 and 133 points to the same object, so we only need to retrieve and test the actual intersection once.

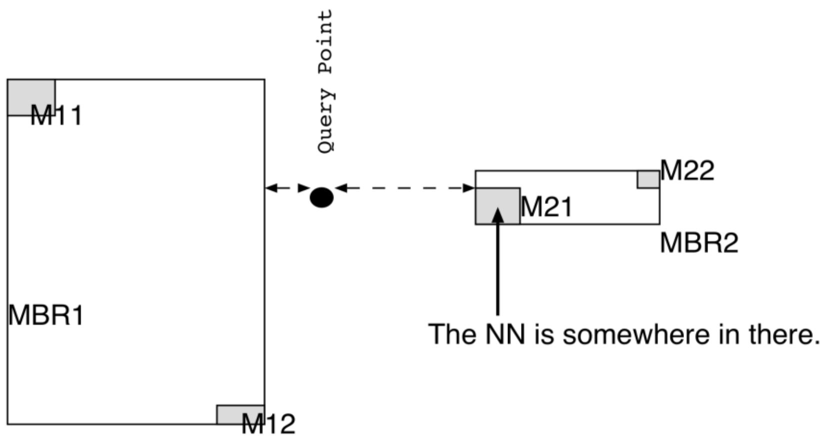
Question 2: When we do the hash based spatial join, we have two choices of hash functions: *single assignment* and *multiple assignment*, and two join procedures: *single join* and *multiple join*. Different index would require different join procedure. Please discuss which kind of join procedure should be used by R-Tree, Z-order, and R+-Tree, and explain why.

Example Solution:
For R-Tree, each object is enclosed by only one MBR, so it belongs to the single assignment. When we use the single assignment hash function, we have to run the multiple join procedure the guarantee the correctness. Therefore, to get the result, we have to join all the intersected objects together.

For Z-order and R+-Tree, one polygon can exist in several cells or MBRs, so it belongs to the multiple assignment. When we use the multiple assignment hash function, we only need to join once for each object to obtain the result. Otherwise, we will produce duplicate results.

Question 3: When using the R-tree to search for the nearest neighbor, we have two metrics to determine the search order: MINDIST and MINMAXDIST. The MINDIST is the optimistic choice, while the MINMAXDIST is the pessimistic one. Is the MINDIST always a better ordering than using the MINMAXDIST?

Example Solution:



MINDIST ordering is not always better than MINMAXDIST ordering.

As shown in the above example, the nearest neighbor is somewhere inside M21. If we use the MINDIST ordering, because MBR1 is nearer than MBR2, we will visit MBR1 first and also visit M11 and M12 in it. If we use the MINMAXDIST ordering, because MBR2 has smaller MINMAXDIST than MBR1, we will visit MBR2 first and then M21. When we visit MBR1, we can prune M11 and M12.

Question 4: The two metrics in the Question 1 are also used in pruning the search space. For example, when we have a query point p and two MBRs M_1 and M_2. If MINDIST $(p, M_1) >$ MINMAXDIST (p, M_2), we can M_1 discard directly. However, sometimes this condition is too loose and have a weak pruning power. When is this condition necessary and when can we use a tighter pruning condition?

Example Solution:
When there are many dead spaces in the nodes of the R-tree, we have to use MINMAXDIST as the pruning threshold because we cannot afford losing the correct results.

When there is no or very little dead space in the nodes of R-tree, using MINDIST is a better choice as it will prune more candidate MBRs and we can find the result earlier.

Question 5: Why the skyline result will appear in all monotonically increasing function?

Example Solution:
The property of a k-dimensional monotonically function f is that for any two values $f(x_1, x_2, \ldots, x_k)$ and $f(x'_1, x'_2, \ldots, x'_k)$

- If $\forall 1 \leq i \leq k, x_i \leq x_i'$
- And $\exists 1 \leq i \leq k, x_i < x_i'$
- $\Rightarrow f(x_1, x_2, \ldots, x_k) \leq f(x'_1, x'_2, \ldots, x'_k)$

No matter which dimension, as long as its value increases, the function value increases.

The requirement of skyline result is for any two k-dimensional points $p(x_1, x_2, \ldots, x_k)$ and $p'(x'_1, x'_2, \ldots, x'_k)$

- $\forall 1 \leq i \leq k, x_i \leq x_i'$
- $\exists 1 \leq i \leq k, x_i < x_i'$
- $\Leftrightarrow p$ dominates p'

The two conditions are exactly the same as the increasing condition of the monotonically function. Therefore, we can get the following equation:

- p dominates $p' \Rightarrow f(p) \leq f(p')$

Because the points in the skyline result set P dominate all the other points P', no matter how we change the actual function f', a point $p' \in P'$still can find some skyline points $p \in P$ that has smaller function value $f'(p) < f'(p')$. Otherwise, the skyline result is not complete.