CSEG601 & CSE5601:

Spatial Data Management & Applications

Sungwon Jung

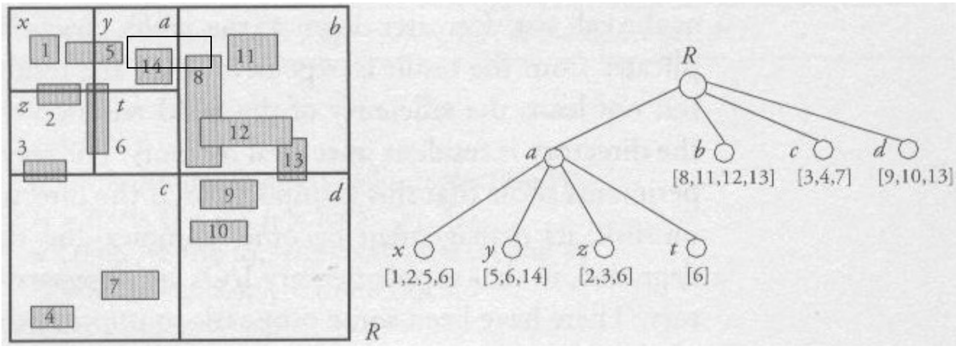
Big Data Processing & DB Lab
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

Spatial Access methods 3

The Linear Quadtree

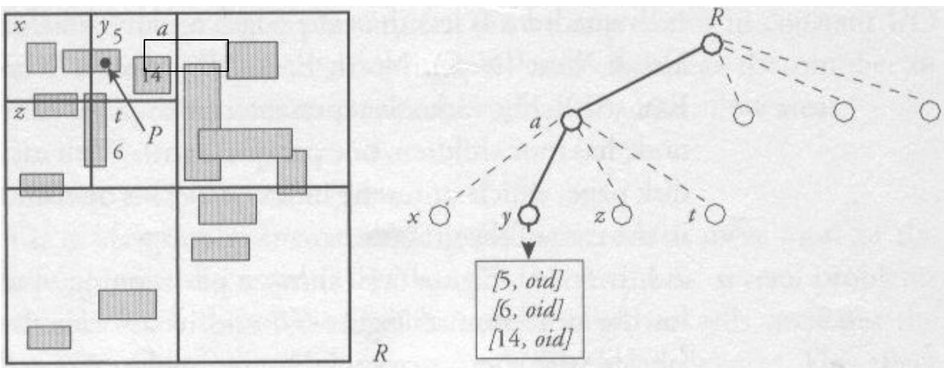
The Linear Quadtree

- The quadtree
  - The search space is recursively decomposed into quadrants until the number of rectangles overlapping each quadrant is less than the page capacity
  - Quadrant's name : NW, NE, SW and SE
  - The index is represented as a quaternary tree
    - Each leaf is associated a disk page



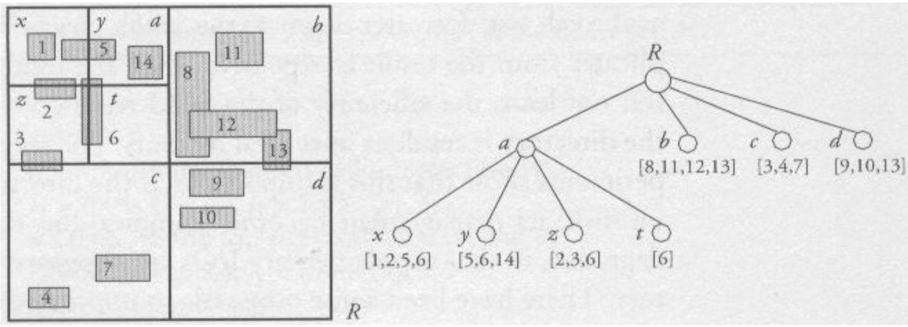
The Linear Quadtree

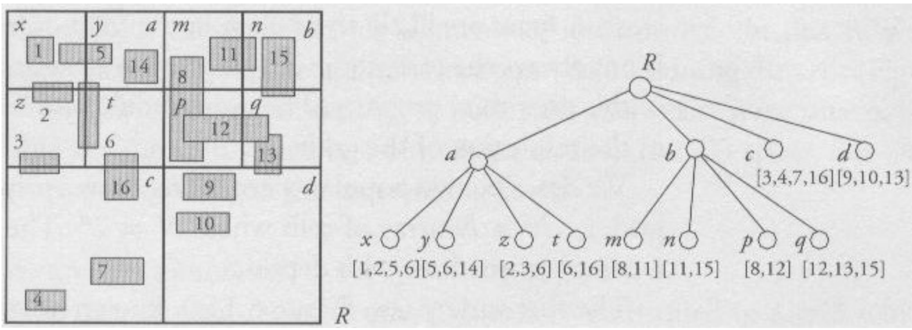
- The quadtree
  - Point query
    - A single path is followed from the tree root to a leaf
    - At each level, one chooses among the four quadrants the one that contains the point argument
    - The leaf is read and scanned



The Linear Quadtree

- Insertion of rectangles 15 and 16



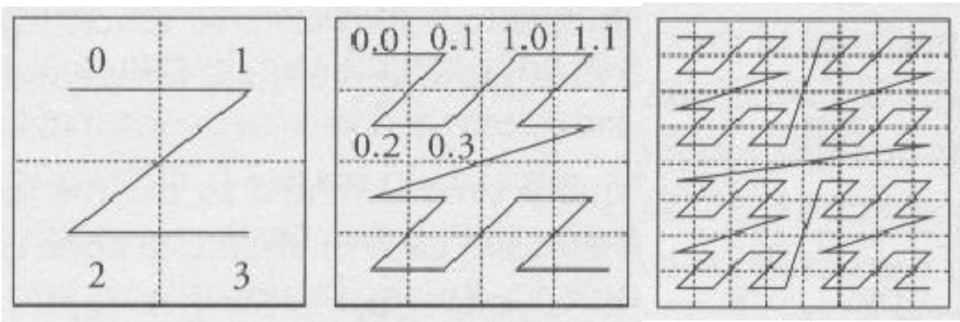


The Linear Quadtree

- The quadtree
  - Drawbacks
    - The small number of children is fixed to 4 occupying only a small part of a page
    - To map a quadtree to disk pages is not easy
      - Tree structure with large node fan-out (such as B-tree or R-tree) allow one to efficiently map a node to a disk page and thus more appropriate for secondary memory access methods
    - The quadtree query time is related to the tree depth, which might be large
    - High duplication rate

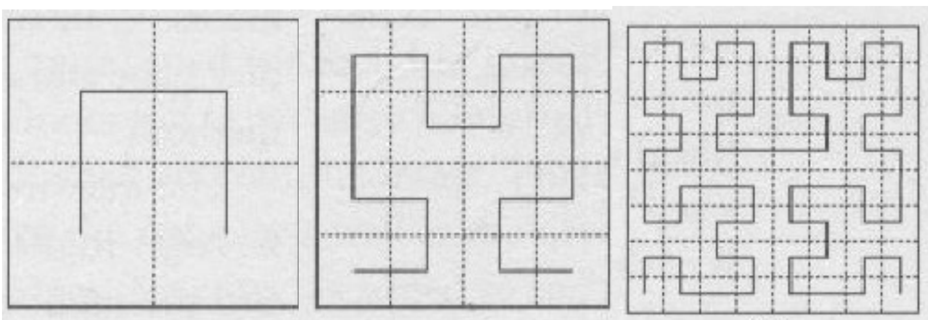
The Linear Quadtree

- Space-filling curves
  - Define a total order on the cells of a 2D grid
  - Z-order(z-ordering)
    - Can sort the cells according to their labels (lexicographical order)

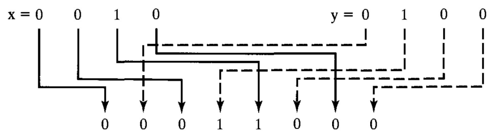


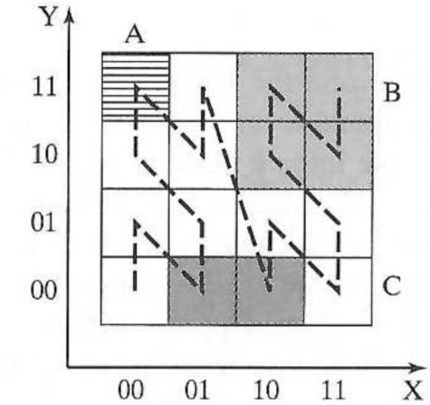
The Linear Quadtree

- Space-filling curves
  - Hilbert curve
    - Consist of segments of uniform length
  - In both cases (z-ordering and hilbert curve), there exist some unavoidable situations in which two objects are close in the 2D space, but far from one another on the space-filling curve

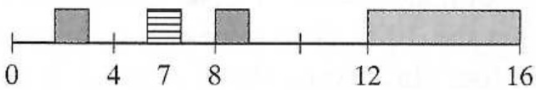


Example of Finding z-values





Object	Points	x	y	interleave	z-value
A	1	00	11	0101	5
B	1	10	10	1100	12
	2	10	11	1101	13
	3	11	10	1110	14
	4	11	11	1111	15
C	1	01	00	0010	2
	2	10	00	1000	8



Example of Finding Hilbert-values

Hilbert Curve

1. Read in the n-bit binary representation of the x and y coordinates.
2. Interleave bits of the two binary numbers into one string.
3. Divide the string from left to right into 2-bit strings, s_i, for $i = 1, \ldots, n$.
4. Give a decimal value, d_i, for each 2-bit string, as follows: "00" equals 0, "01" equals 1; "10" equals 3; "11" equals 2.
5. For each number j in the array, if

   $j = 0$ then switch every following occurrence of 1 in the array to 3 and every following occurrence of 3 in the array to 1;

   $j = 3$ then switch every following occurrence of 0 in the array to 2 and every following occurrence of 2 in the array to 0;

6. Convert each number in the array to its binary representation (2-bit strings), concatenate all the strings in order from left to right, and calculate the decimal value.

y \ x	00	01	10	11
00	0000	0010	1000	1010
01	0001	0011	1001	1011
10	0100	0110	1100	1110
11	0101	0111	1101	1111

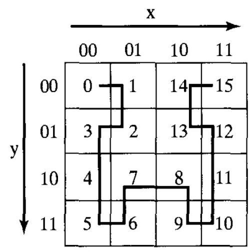
(a)

y \ x	00	01	10	11
00	00	03	30	33
01	01	02	31	32
10	10	13	20	23
11	11	12	21	22

(b)

y \ x	00	01	10	11
00	00	01	32	33
01	03	02	31	30
10	10	13	20	23
11	11	12	21	22

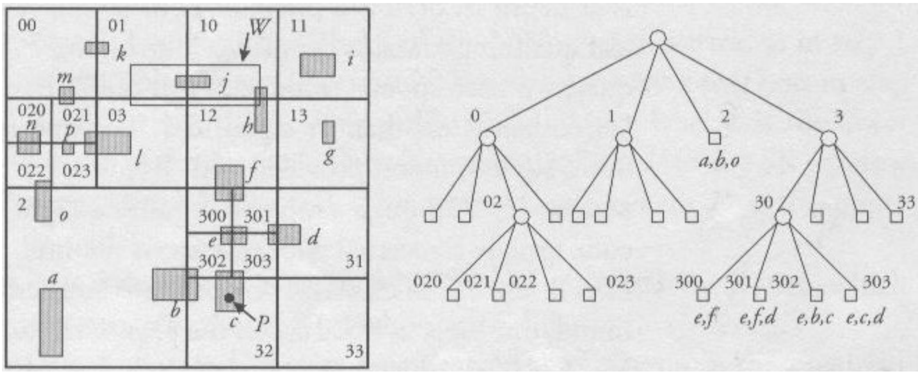
(c)



(d)

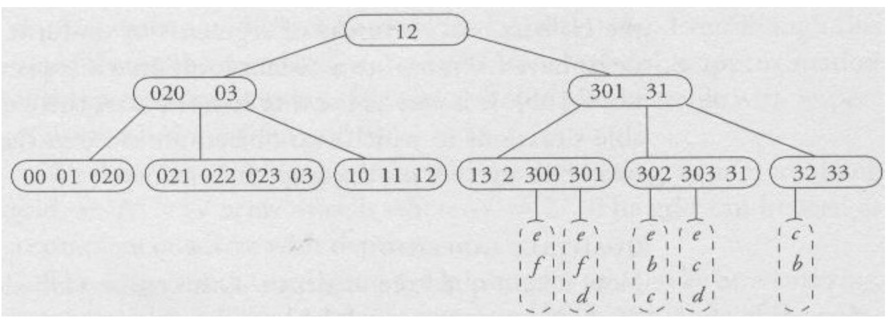
The Linear Quadtree

- Quadtree labeling
  - *d* : depth of quadtree
  - It can be embedded in a $N*N$ grid with $N=2^d$
  - The order of the leaves corresponds to a left-to-right scan of the leaves
  - The labels are not of the same size. The size of the label is the depth of the leaf in the tree.
  - The label of a leaf can also be seen as the label of a path from the root to the leaf



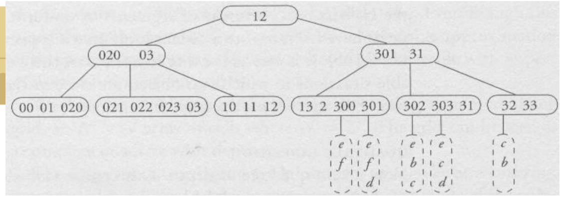
The Linear Quadtree

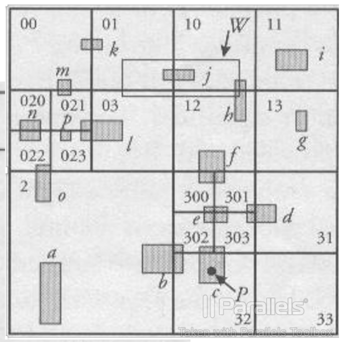
- Linear quadtree
  - Once the entries [*mbb*, *oid*] have been assigned to a quadtree leaf with label ***l***, and stored in a page with address ***p***, then we index in a B+tree the collection of pairs (***l***, ***p***) keyed on the leaf label ***l***



  - Redundancy problem
    - *mbb*s that overlap several quadtree leaves are duplicated in pages associated with these leaves

The Linear Quadtree



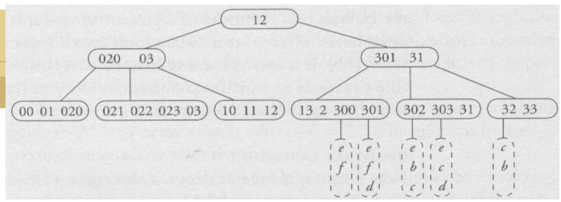


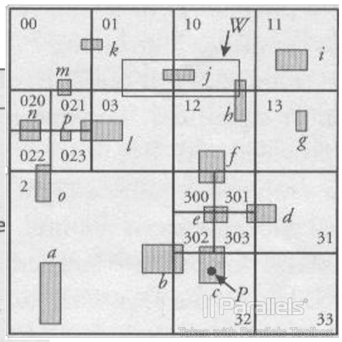
- Point query algorithm with a linear quadtree

```
LQ-POINTQUERY (P: point): set(oid)

begin
  result = Ø
  // Step 1: compute the label of the point
  l = POINTLABEL(P)
  // Step 2: the entry [L, p] is obtained by traversing the B+tree with key l.
  [L, p] = MAXINF (l)
  // Step 3: get the page and retrieve the objects
  page = READPAGE (p)
  for each e in page do
    if (e.mbb contains P) then result += {e.oid}
  end for
  return result
end
```


The Linear Quadtree





- Window query algorithm with a linear quadtree

```
LQ-WINDOWQUERY (W: rectangle): set(oid)

begin
  result = Ø
  // Step 1: From the vertices W.nw and W.se of the window, compute
  // the interval [L, L']. This necessitates two searches through the B+tre
  l = POINTLABEL (W.nw); [L, p] = MAXINF (l)
  l' = POINTLABEL (W.se); [L', p'] =MAXINF(l')
  // Step 2: The set Q of B+tree entries [l, p] with l ∈ [L, L'] is computed.
  Q = RANGEQUERY ([L, L'])
  // Step 3: For each entry in Q whose quadrant overlaps W, access
  // the page
  for each q in Q do
    if (QUADRANT (q.l) overlaps W) then
      page = READPAGE (q.p)
      // Scan the quadtree page
      for each e in page do
        if (e.mbb overlaps W) then result += {e.oid}
      end for
    end if
  end for
  // Sort the result, and remove duplicates
  SORT (result); REMOVEDUPL (result);
  return result
end
```

NW = 012 = l
L = 01
SE = 121 = l'
L' = 13

Q= {01,020, 021, 022, 023,03, 10, 11, 12}

Result = {j,h}

The Linear Quadtree

- Insertion of rectangles
  - Two cases must be considered:
    - There is no split of the embedded quadtree
      - The quadrant page is accessed and updated
    - There is a split of the embedded quadtree
      - One entry of B+tree must be deleted and replaced by four new entries(one per new quadtree page)

- Analysis
  - The number of I/O for a point query
    - $d+1$, where d is the depth of the B+tree
  - The number of I/O for a window query
    - For step 1, $2d$ I/Os
    - For step 2, $d+k$ I/Os where k is as many I/Os as there are chained B-tree leaves to be scanned
      - The number of I/O at step 2 is dependent on the size of interval