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 1

Introduction

- Processing a spatial query leads to the execution of complex and costly geometric operations
  - An example: Point queries
    - Sequentially scanning and checking whether each object of a large collection contains a point involves a large # of disk accesses and the repeated expensive evaluation of geometric predicates
  - The time-consuming geometric algorithms and the large volume of spatial collections stored on secondary storage
    - Motivate the design of efficient spatial access methods (SAMs)
    - SAM reduce the set of objects to be processed
      - Expect a time that is logarithmic in the collection size, or even smaller
- SAM uses an explicit structure, called a *spatial index*

SAM

- SAMs for points, lines, and polygons
- Instead of indexing the object geometries themselves, index a simple approximation of the geometry
  - minimum bounding rectangle of the objects' geometry, or *minimum bounding box* (i.e., *mbb)*
  - By using the *mbb* as the geometric key for constructing spatial indices,
    - Can save the cost of evaluating expensive geometric predicates during the index traversal
    - Can use the constant-size entries

SAM

- A spatial index is built on a collection of entries
  - Entries are in the forms of pairs [*mbb, oid*] where *oid* identifies the object whose minimal bounding box is *mbb*
  - *oid* allows us to access directly the page containing the physical representation of the object
    - The values of the descriptive attributes and the value of the spatial component
  - An operation involving a spatial predicate on a collection of objects indexed on their *mbb* is performed in two steps
    - *filter step*
    - *refinement step*

SAM

- *filter step*
  - selects the objects whose *mbb* satisfies the spatial predicate
    - Traverse the index, applying the spatial test on the *mbb,* and output a set of *oids*

- *refinement step*
  - An *mbb* might satisfy a query predicate, whereas the exact geometry does not
    - The objects passing the filter step are a superset of the solution
  - The superset is sequentially scanned, and the spatial test is done on the actual geometries of objects
    - The objects passing the filter step but not the refinement step are called *false drops*

SAM

- In traditional DBMSs, the most common access methods use B^+-trees and hash tables
  - They guarantees the # of I/Os to access data is respectively logarithmic and constant in the collection size, for exact search queries
  - However, they cannot be used in the context of spatial data
    - B^+-tree relies on a *total order* on the key domain !!!
      - e.g., the order on natural numbers, the lexicographic order on strings of characters
- A convenient order for geometric objects in the plane is one preserving object proximity
  - Two objects close in the plane should be close in the index structure
    - An example: Objects inside a rectangle

SAM

- A large # of SAMs were designed to try as much as possible to preserve object proximity
- Two families of indices representative of two major trends
  - Grid
  - R-tree
- Their structures are simple, robust, and efficient
- For each index family, consider
  - Index construction
    - Insertion of one entry or of a collection of entries
  - Search operations
    - Point and window queries
- The design of efficient SAMs is crucial for optimizing point and window queries as well as spatial operations such as join

Issues in SAM Design

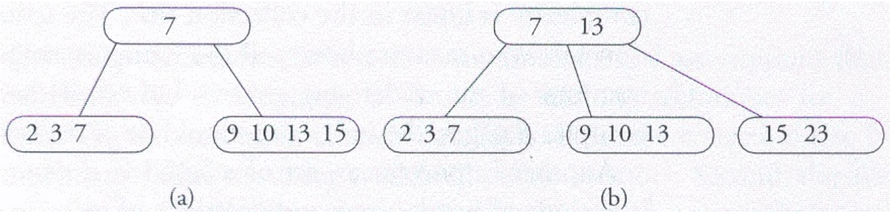
- Fundamental assumptions
  - The collection size is much larger than the space available in main memory
  - The access time to disks is long compared to that of random access main memory
  - Objects are grouped in pages, which constitute the unit of transfer between memory and disk
    - Typical page sizes range from 1K to 4K bytes
  - CPU time is assumed to be the negligible compared to I/O time
    - CPU time should be taken into account in some situations
  - Do not consider the caching of pages in main memory
    - Each time a page is accessed, it is not present in main memory and a page fault occurs

Issues in SAM Design

- What is expected of a SAM
  - A SAM should fulfill:
    - Time complexity
      - It should support exact (point) and range (window) search in sublinear time
    - Space complexity
      - Its size should be comparable to that of the indexed collection
        - If the indexed collection occupies n pages, the index size should be $O(n)$
    - Dynamicity
      - It must accept insertions of new objects and deletions of existing ones, and adapt to any growth or shrink of the indexed collection (or to a nonuniform distribution of objects) without performance loss

Issues in SAM Design

- Illustration with a B^+ Tree
  - B^+ tree index built on the set of keys {2, 3, 7, 9, 10, 13, 15}
  - It is balanced tree where each node is a disk page and stores entries [*key, ptr*]



    - Time complexity: logarithmic
    - Space complexity: at least half the space needed to store the index is actually used
    - Dynamic update: adapt to itself to any distribution

Issues in SAM Design

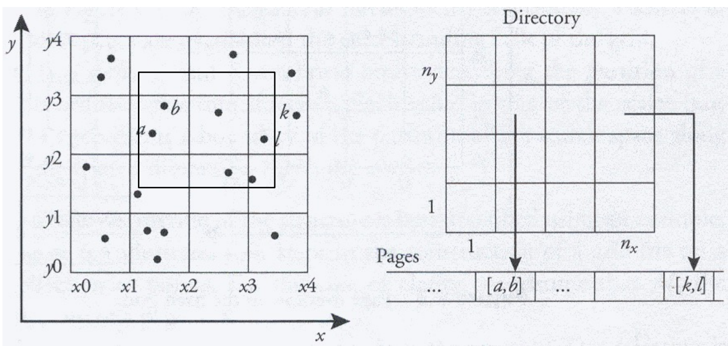
- Space-Driven Versus Data-Driven SAMs
  - For indexing to spatial data, we must use a different construction principle, not based on the ordering of objects on their coordinates
  - Space-driven structures: *Grid file and linear structures* methods
    - Based on partitioning of embedding 2D space into rectangular cells, independently of the distribution of objects (points or *mbb*) in the 2D plane
    - Objects are mapped to the cells according to some geometric criterion
  - Data-driven structures: R-tree, R^* tree, R^+ tree
    - Organized by partitioning the set of objects, as opposed the embedding space
    - The partitioning adapts to the objects' distribution in the embedding space

Issues in SAM Design

- Space-Driven Versus Data-Driven SAMs
  - The lack of robustness of SAMs w.r.t. the statistical properties of the objects in 2D plane is true for all SAMs whether they are space driven or data driven
  - While B^+ tree is a worst-case optimal structure, the common multidimensional access methods such as SAMs are not

Space-Driven Structure: The Grid File

- The fixed grid
  - The search space is decomposed into rectangular *cells*
    - The resulting regular grid is an $n_x \times n_y$ array of equal-size cells
    - Each cell c is associated with a disk page
    - Point P is assigned to cell c if the rectangle *c.rect* associated with cell c contains P
    - The objects mapped to a cell c are sequentially stored in the page associated with c



(a1, b1) → (i1, j1) -> (2, 2)
(a2, b2) -> (i2, j2) -> (4, 4)

2 <= i <= 4
2 <= j <= 4

(2, 2) (2, 3), (2,4)
(3, 2), (3,3), (3,4)
(4, 2), (4,3), (4,4)

Space-Driven Structure: The Grid File

- The fixed grid
  - The search space has for an origin the point with coordinates (x0,y0)
  - The index requires a 2D array $DIR[1{:}n_x,\ 1{:}n_y]$ as a directory
    - Each element $DIR[i,j]$ of the directory contains the address *PageID* of the page storing the points assigned to cell $c_{i,j}$
  - If $[S_x, S_y]$ is the 2D size of the search space,
    - Each cell's rectangle has size $[S_x/n_x, S_y/n_y]$

Space-Driven Structure: The Grid File

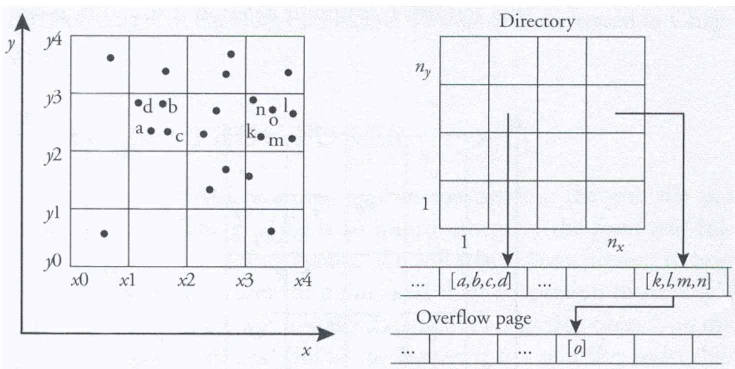
- The fixed grid
  - Inserting P(a, b):
    - Compute $i = (a\text{-}x_0)/(S_x/n_x) + 1$ and $j = (b\text{-}y_0)/(S_y/n_y) + 1$
    - Read page $DIR[i,j]$.*PageID* and insert *P*
  - Point query:
    - Given the point argument *P*(*a, b*), get the page as for insertion, read the page, scan the entries, and check whether one is *P*
  - Window query:
    - Compute the set *S* of cells c such that *c.rect* overlaps the query argument window *W*
    - Read, for each cell $c_{i,j}$ in *S*, page $DIR[i,j]$.*PageID* and return the points in the page that are contained in the argument window *W*

Space-Driven Structure: The Grid File

- The fixed grid
  - Point query is efficient
    - Assume the directory resides in central memory, a point query requires a single I/O
  - The # of I/Os for a window query depends on the # of cells the window intersects
    - Proportional to the area of the window
  - The grid resolution depends on the number *N* of points to be indexed
    - Given a page capacity of *M* points, can create a fixed grid with at least *N/M* cells
    - A cell to which more than M points are assigned overflows
      - Overflow pages are chained to the initial pages

Space-Driven Structure: The Grid File

- The fixed grid
  - Page overflow in the fixed grid



  - If the points are uniformly distributed in the search space, overflows seldom occur and the overflow chains are not long

Space-Driven Structure: The Grid File

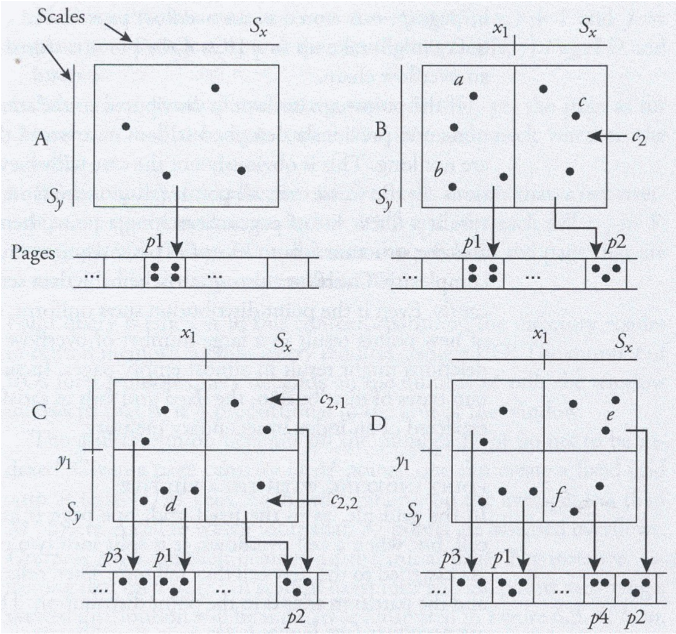
- Point indexing with the grid file
  - When a cell overflows, it is split into two cells and points are assigned to new cell they fall into
    - Cells are of different size and the partition adapts to the point distribution
    - Three data structures are necessary
      - The directory DIR
        - Being similar to that of fixed grid, the important difference is that two adjacent cells can reference the same page
      - Two scales S_x and S_y
        - Linear arrays describing the partition of a coordinate axis into intervals
        - Each value in one of scales (e.g., S_x) represents a boundary in the partition of the search space along the related dimension (here, the *x* axis)

Space-Driven Structure: The Grid File

- Point indexing with the grid file
  - Insertions into a grid file
    - No cell split
    - Cell split and no directory split
    - Cell split and directory split
  - The grid file overcomes major limits of the fixed grid
    - Point search can always be performed with two disk accesses
      - One for accessing the page ***p*** referenced in the directory and one for the data page associated with one entry in ***p***
    - The structure is dynamic and present a reasonable behavior w.r.t. space utilization
  - For large data sets, the directory size may not fit in main memory

Space-Driven Structure: The Grid File

- Insertions into a grid file



Sx : [x1, inf, ….]
Sy: [y1,inf, …]
DIR[1,1] : p3
DIR[1,2]: p1
DIR[2,1]: p2
DIR[2,2]: p4
f: (fx, fy) (2,2)

No cell split

Cell split and no directory split

Cell split and directory split

Example: Show a final grid file generated by inserting **(0,0), (0,1), (1,1), (1,0), (0,2), (1,2), (2,2), (2,1), (2,0)** points sequentially.
Assume that each disk page can hold at most three data objects.