Spatial Indexing

- Point Access Methods (PAMs) vs Spatial Access Methods (SAMs)
- PAMs index only point data objects
  - Space filling curve
  - Multidimensional Hashing
  - Hierarchical (tree-based) structures
- SAMs store point and non-point data objects (R-tree)
  - Overlapping Regions (R-tree, R*-tree)
  - Clipping methods (R+-tree)
  - Transformations

Space filling curves

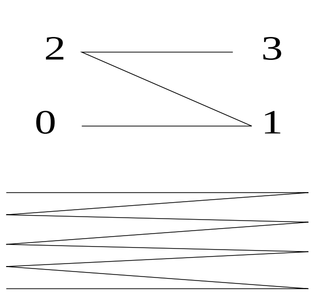
- Map two dimensional points to one dimensional ones
- Try to keep the proximity in the original space.
  Points that are close in two dimensions are also close in one dimension

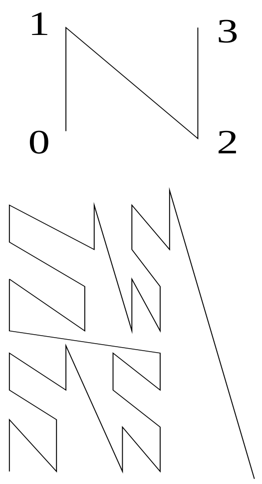
Space filling curves

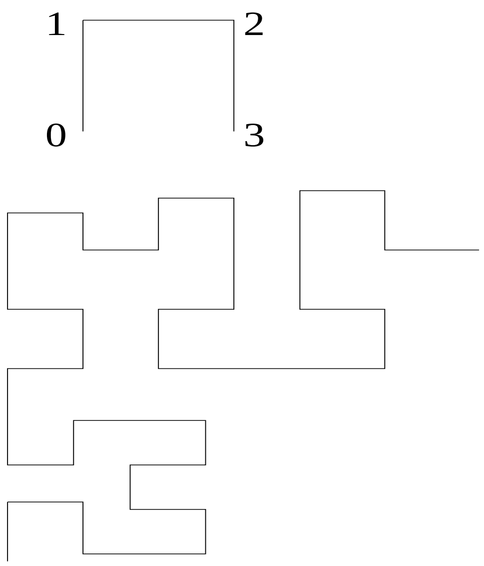
- First step: partition the domain with a grid
- Second step: Each point in the grid is labeled with a unique number using a space-filling curve
- Use a B+-tree to index the 1-dimensional points

Space filling curves

- Examples of SPC: (row-wise curve, Peano curve and Hilbert curve)

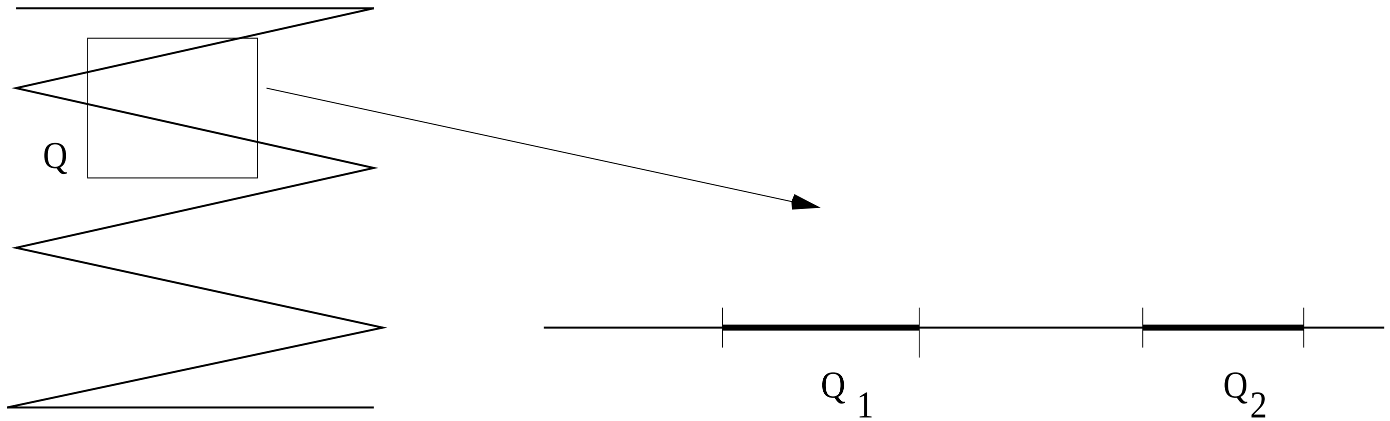






Space filling curves

- Answering a range query in 2-d: The query is mapped to a set of range queries in 1-d.

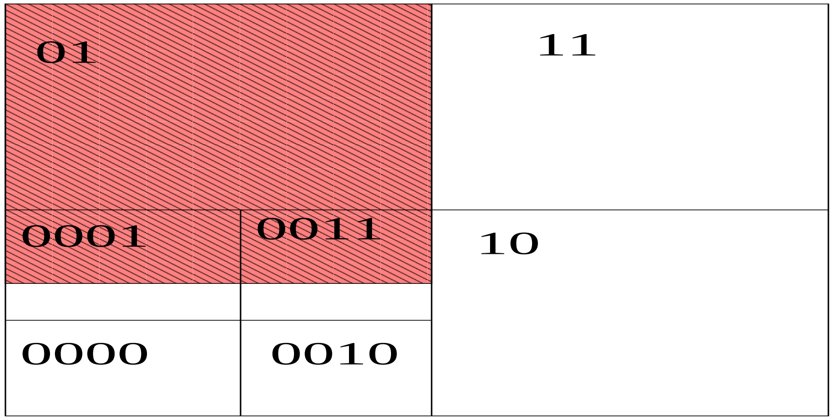


Space filling curves - Region data

- How to index regions data?
- Break regions in smaller parts using a grid with different sizes
- Give for each part a z-value
- store each value in a B-tree

Space filling curves - Region data

- Example: The object is represented by the (binary) numbers: 0001 0011, 01



- Finite precision in representing each coordinate

Hilbert R-Tree

- Another version of the R-tree
- Idea: Use a space filling curve to impose a linear ordering on the data
- Use the ordering when you insert a new object or split a node

Hilbert R-Tree

- For each rectangle store also the H-value of its center
- Thus, each entry in a node has the form: (MBR, LHV, ptr)
- LHV, largest Hilbert value of the data rectangles that belong to the subtree pointed by ptr
- s-to-(s+1) splitting policy. ex. 2-to-3 splitting policy: wait until 2 nodes will split into 3.

Hilbert R-Tree

Insert(R, IR)

I1: Invoke **ChooseLeaf(R, IR)**
I2: If node L has an empty slot add IR, else invoke **HandleOverflow(L, IR)**
I3: Invoke **AdjustTree**

Hilbert R-Tree

ChooseLeaf(R, IR)

C1: Set N to be the root of R
C2: If N a leaf, return N
C3: If N not a leaf, choose the entry (r, ptr, LHV) with the minimum LHV value greater than IR.h
C3: Set N the node pointed by ptr and go to C2

Hilbert R-Tree

Delete(R, IR)

D1: Find the leaf node L that contains IR
D2: remove IR from L
D3: If L underflows, borrow some entries from s cooperating siblings. If all siblings are ready underflow, merge s+1 to s nodes
D4: Invoke **AdjustTree** to adjust MBRs and LHVs in parent nodes

Hilbert R-Tree

HandleOverflow(R, IR)

H1: Let E be the set of all entries from R and its s-1 siblings
H2: Add IR to E
H3: If at least one of the s-1 siblings is not full, distribute E to the s nodes using H-values
H4: If all nodes full, create a new node and distribute E to s+1 nodes

Hilbert R-Tree

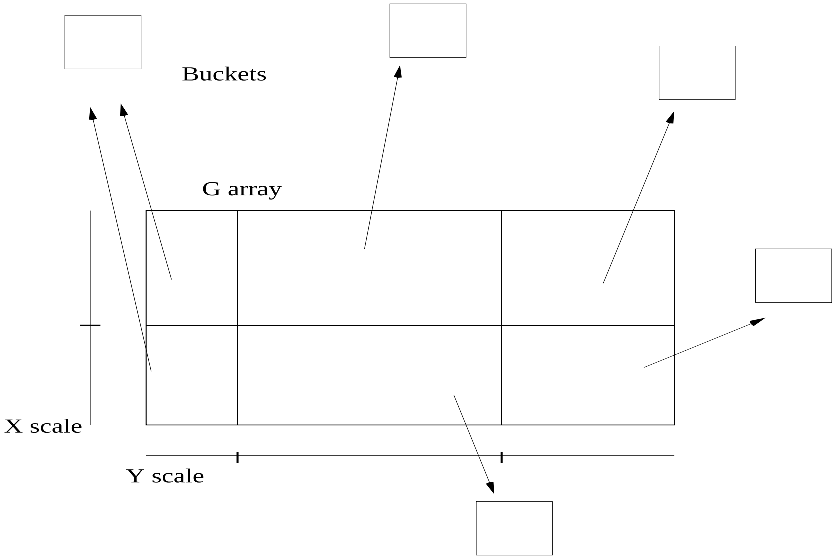
- Experiments show up to 28% improvement in search overhead over R*-tree
- Also 2-to-3 splitting policy is the best (no reinsertion)

Grid File

- Hashing methods for multidimensional points: Grid File
- Idea: Use a grid to partition the space (not necessarily regular, cells may be of different sizes).
- Each cell is associated with one bucket (one page)
- Two disk access principle (for exact match)

Grid File

- Example of a grid file partitioning:



Grid File

- Partition space using dividers along each dimension
- Many cells can point to the same page
- Cell directory potentially exponential to the number of dimensions

Grid File Implementation

- Dynamic structure using a grid directory:
  - Grid array: a 2 dimensional array with pointers to buckets (this array can be large, disk resident) G(0,..., nx-1, 0,...,ny-1).
  - Linear scales: Two 1 dimensional arrays that used to access the grid array (main memory) X(0,...,nx-1), Y(0,...,ny-1). Contain split positions for the dimensions.

Grid File

- Exact match queries:
  - Use linear scales to find the page where the corresponding grid cell is stored (one page)
  - Read the bucket (one more page)
- Range queries: similar

Grid File Insert

- Determine the bucket into which insertion must occur
- If bucket not full, insert
- Else, split bucket (splitting policies)
- If bucket split causes a cell directory to split do so and adjust linear scales
- Note: sometime we need to complete reorganize the cell directory (expensive)

Grid File

- Splitting a bucket not always need to split a cell
- Need to choose the dimension and the location
- Try to find the location that balances the split

Grid File

- Deletions may decrease the space utilization. Merge.
- We need to decide which cells to merge and a merging threshold
- Buddy system and Neighbor system.

Grid File

- Neighbor system: merge two adjacent regions if the result is rectangular region
- Buddy system: a bucket can merge with only one adjacent buddy in each dimension

Grid File

- To implement the grid array we can use linked lists
- Also we can use a small grid array for the G array
- Dynamic weighting of attributes: change the parameters for merging and splitting
- Extreme case: “merge-only” dimension

Grid File

- Advantages: Simple structure, easy to implement
- Disadvantages: Splitting is not a local operation $\rightarrow$ the size of the grid directory can become very large

hB-Tree

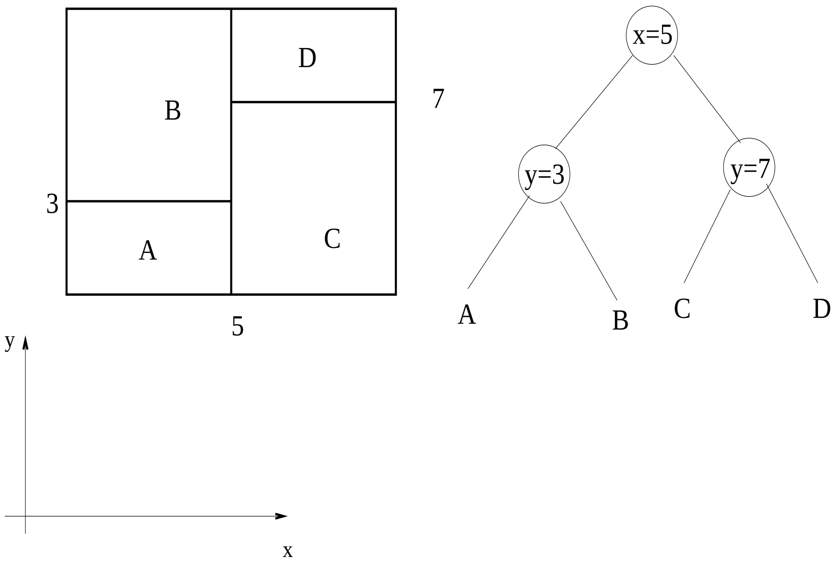
- HB-Tree (holey brick B-tree) is another PAM
- Basic idea: use k-d trees to organize internal (index) nodes
- Holey: Each node is a rectangle (brick) from which one or more smaller bricks may have been extracted.

k-d trees

- A k-d tree is a main memory binary tree to index multidimensional points
- Idea: Partition the space based on a value for each dimension
- Each node contains a dimension id and a value (did, value) and two pointers

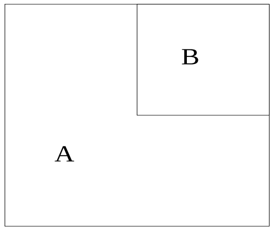
k-d trees

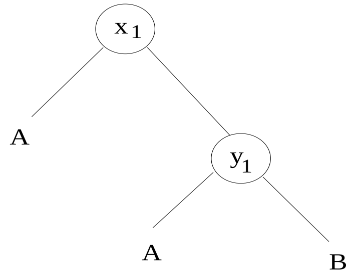
- At each level we use a different dimension to partition the (sub) space
- Not a balanced tree
- An example is the following:



hB-trees

- A node (page) in a hB-tree uses a kd-tree to partition the space
- A node uses more than one dimension to split.



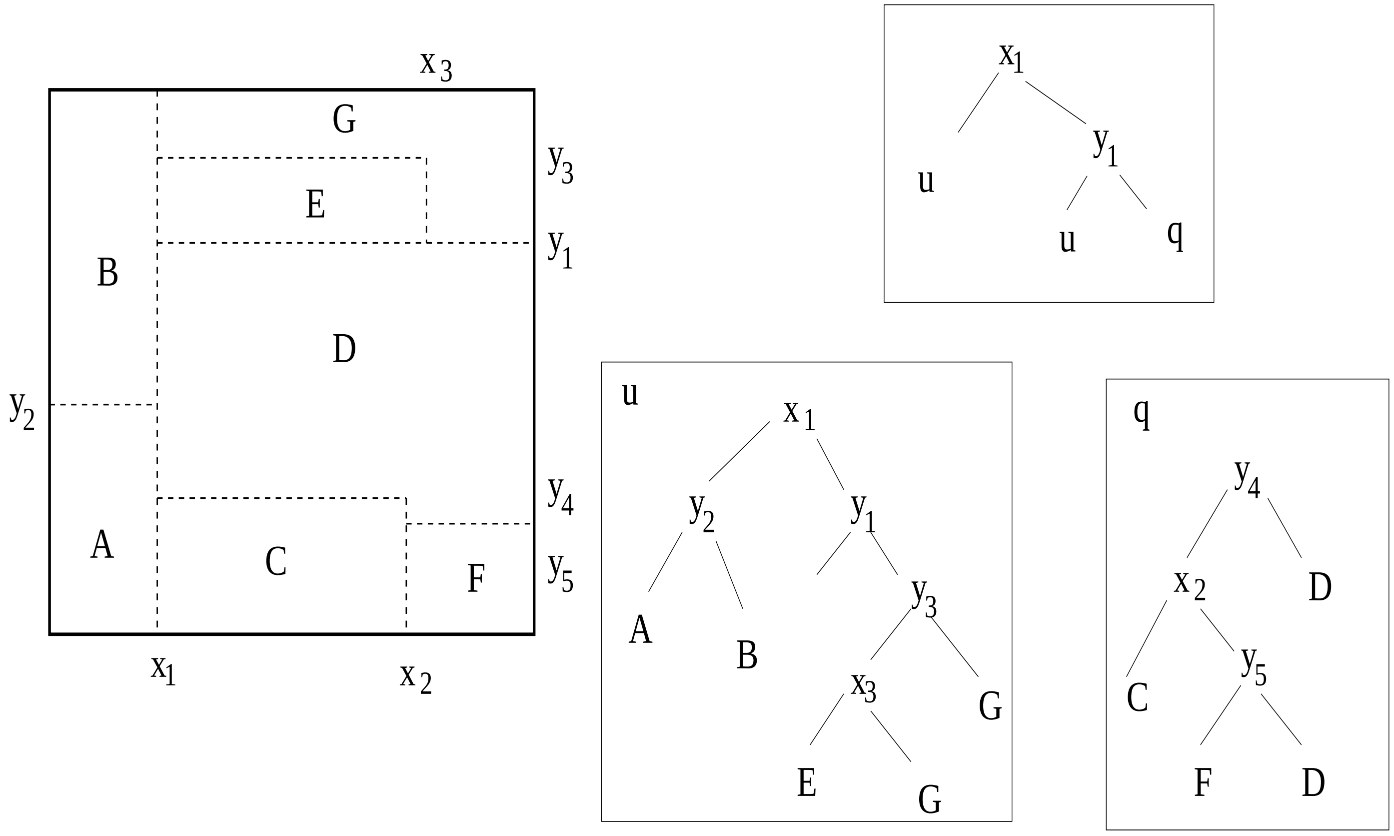


hB-tree node

- An index node contains a kd-tree and the boundaries for each dimension.
- To point to the next level (children hB-nodes) we use the leaf nodes of the kd-tree.
- A data node also contains a kd-tree and list of points for each subspace.

hB-tree example

- An example of an hB-tree



hB-tree

- Searching is straightforward
- Also insertions and deletions similar to B+-tree
- Node splitting however is different...

hB-tree: Node splitting

- Data nodes: Using two attributes we can always guarantee a 2:1 split in the data nodes
- Index nodes: We can split the kd-tree of the index node at the root. Problem: not always balanced split. Solution: Split at lower levels of the kd-tree. Use external marker for the the extracted region.

hB-tree

- We can prove that the node utilization is $\sim 63\%$.
- Also the fan out of the node is:

$$\frac{F = U \times ns}{t}$$

  where, U is node utilization, ns is node size and t size of index terms.

- For 4KB page the fan out is ~ 172.

hB-tree

Nod Splitting Algorithm

1. Start at the f-root. Traverse the tree until to find a subtree with size between $\frac{1}{3}$ and $\frac{2}{3}$.
2. If an f-leaf is reached, extent the f-tree to partition the points in the list (the node is a data node). One leaf of the extended f-tree refers to between $\frac{1}{3}$ and $\frac{2}{3}$ of the contents (subtree A).
3. Replace the subtree A with an "external" marker. Remove nodes from the f-tree, until the smallest subtree that contains the data or the lower level hB-nodes is found.
4. The subtree A is the basis for the new node. Found the smallest kd-tree for this node and the boundaries.

NN Search

- Given a location in space Q, find the object that is the closest to the query point.
- Use a tree based spatial index to answer the query

NN Search

- Start for the root and compute the distance of the query point to each subspace
- Traverse nodes in the tree in the order of the MINDIST
- MINDIST(Q, N) = dist(Q,N) if N is an object.
  else minimum distance between Q and any object in N, if N is an index node