Multidimensional Access Methods

Search operations in large data sets require special support.

In databases and other sets of structured data the support is usually organized by indices. To formalize and implement indices the corresponding data structures are required.

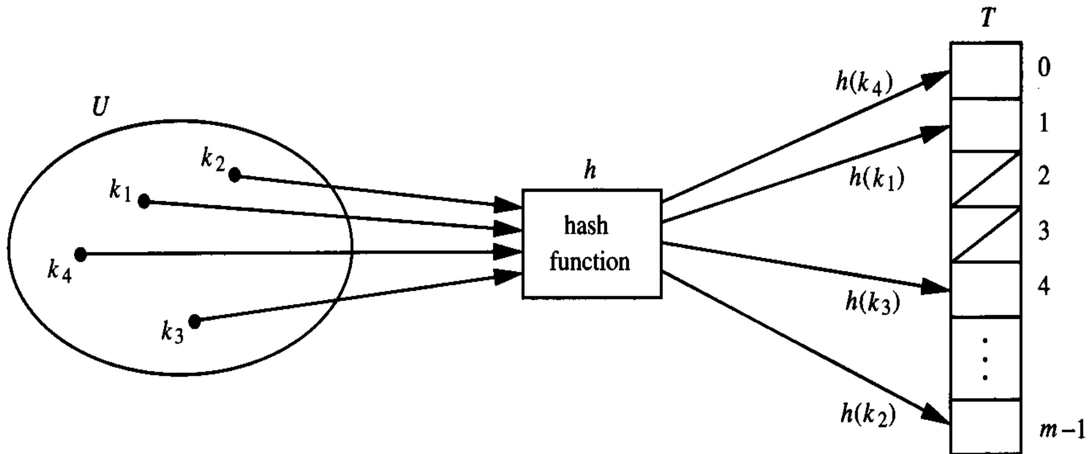
Index or data structure define an access method to data.

Data structures are grouped into classes:

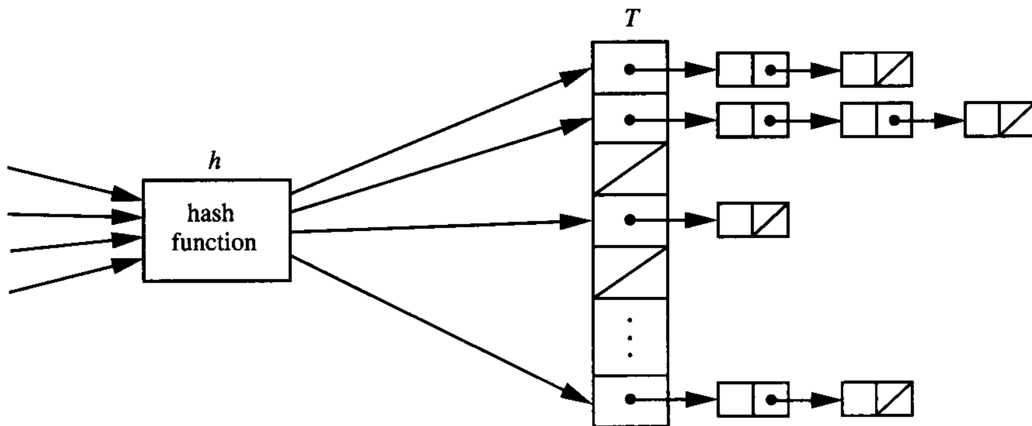
- One-dimensional data structures
- Main memory multidimensional data structures
- Point access methods (PAMs)
- Spatial access methods (SAMs)
- Etc.

One-dimensional data structures

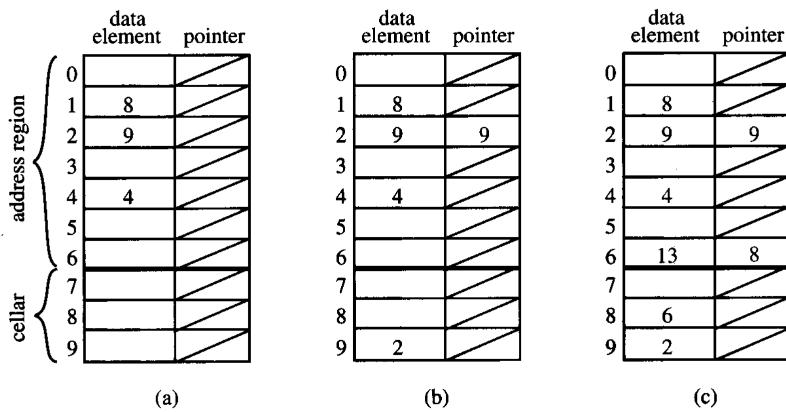
Hashing



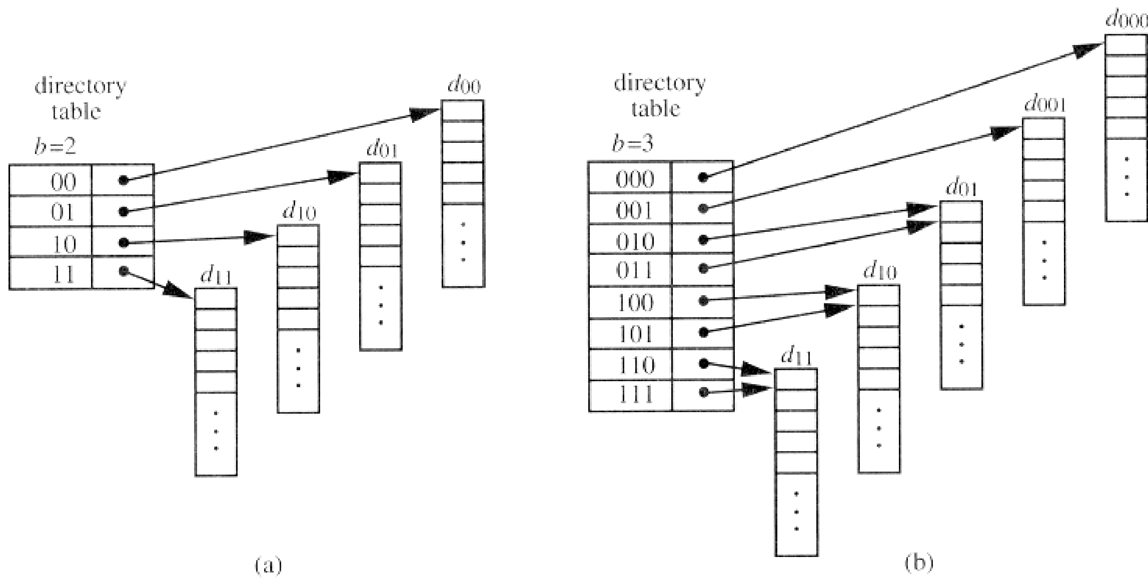
Separate chaining



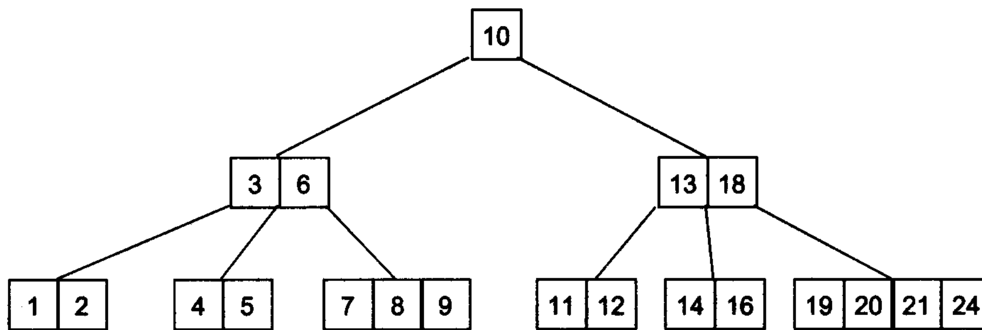
Coalesced Hashing



Extendible hashing



B-trees

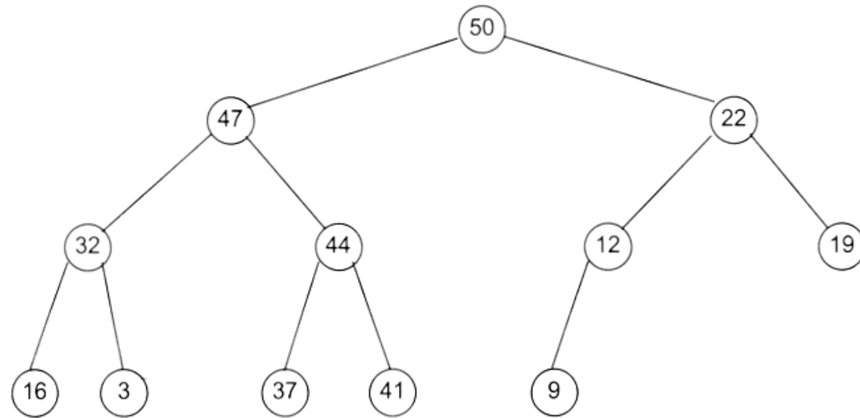


Heaps and Priority Queues

A heap is a special kind of binary tree that leaves no gaps in an array implementation:

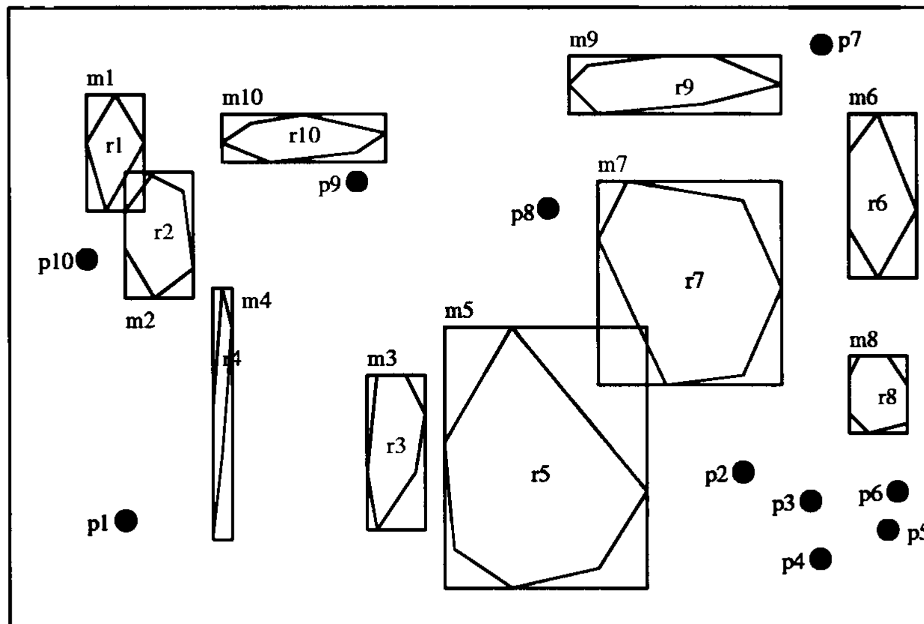
- All leaves are on two adjacent levels.
- All leaves on the lowest level occur at the left of the tree.
- All levels above the lowest are completely filled.
- Both children of any node are again heaps.
- The value stored at any node is at least as large as the values in its two children.

The first three conditions ensure that the array representation of the heap will have ***no gaps*** in it, the last two conditions give a heap ***a weak amount*** of order.

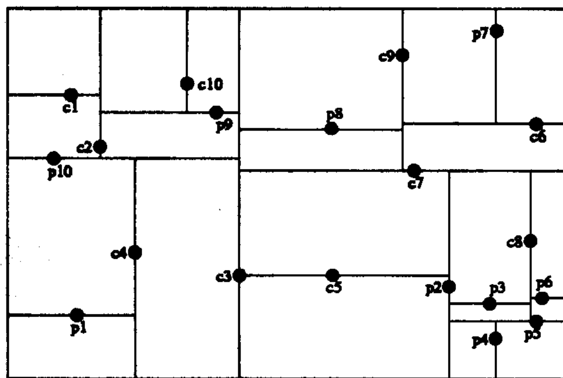


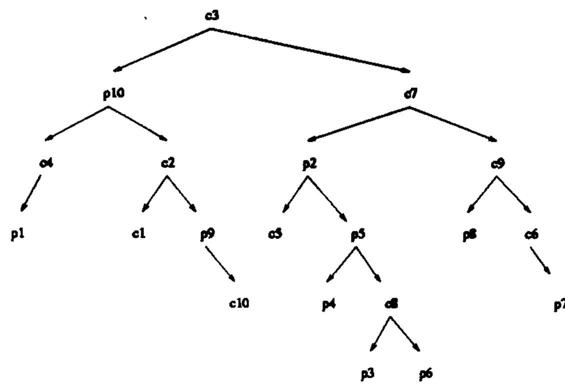
Main Memory Structures

Running example:



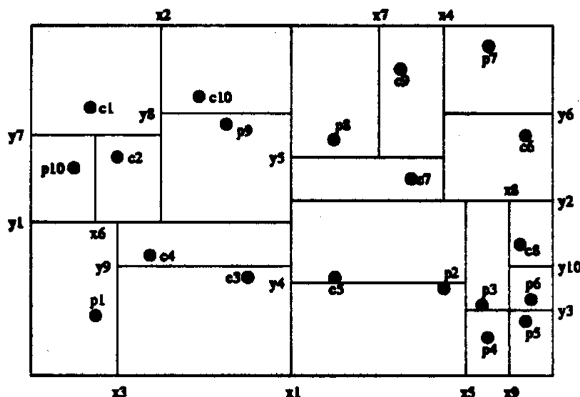
K – D – Tree

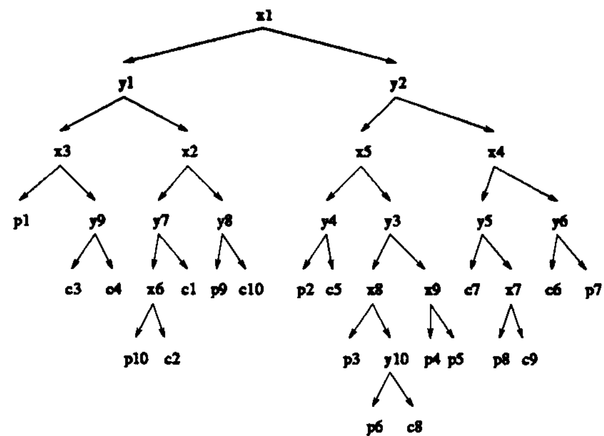




k-d-tree is a binary search tree that represents the recursive subdivision of the universe into subspaces by means of (d-1)-dimensional hyperplanes.

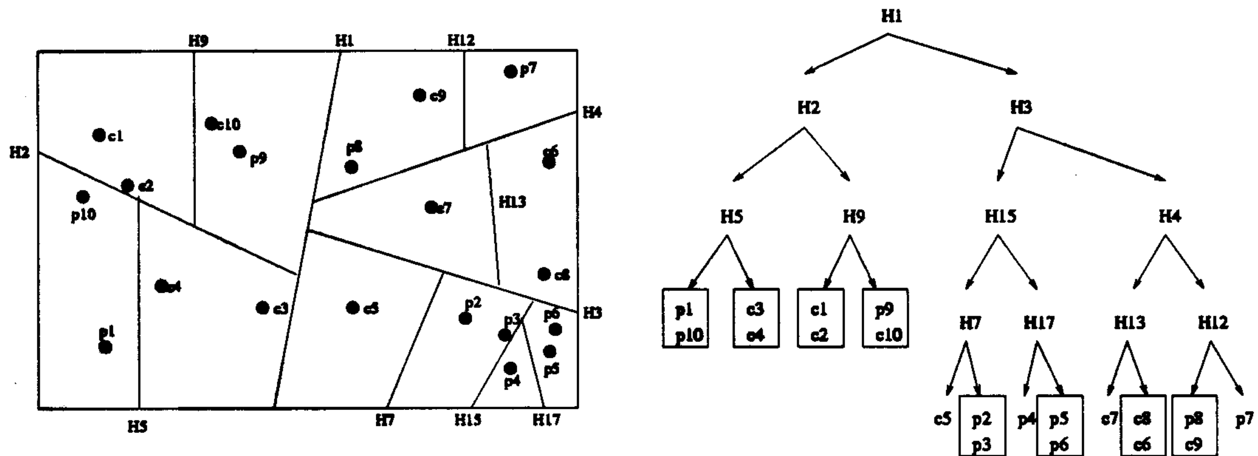
Adaptive k-d-tree





Adaptive k-d-tree choose a split such that one finds about the same number of elements on both sides. While the splitting hyperplanes are still parallel to the axes, they do not have to contain a data point and their directions do not have to be strictly alternating anymore. As a result, the split points are not part of the input data; all data points are stored in the leaves. Interior nodes contain the dimension (e.g. x or y) and the coordinate of the corresponding split. Splitting is continued recursively until each subspace contains only a single point. The adaptive k-d-tree is not a very dynamic structure; it is obviously difficult to keep the tree balanced in the presence of frequent insertions and deletions. The structure works best if all the data is known a priori and if updates are rare.

BSP Tree



Splitting the universe only along iso-oriented hyperplanes is a severe restriction. Allowing arbitrary orientations gives more flexibility to find a hyperplane that is well-suited for the split. The ***binary space partitioning*** *(BSP)* tree are binary trees that represent a recursive subdivision of the universe into subspaces by means of (d - 1)- dimensional hyperplanes.

Each subspace is subdivided independently of its history and of the other subspaces. The choice of the partitioning hyperplanes depends on the distribution of the data points in a given subspace. The decomposition usually continues until the number of points in each subspace is below a given threshold.

The resulting partition of the universe can be represented by a BSP tree, where each hyperplane corresponds to an interior node of the tree and each subspace corresponds to a leaf. Each leaf stores references to those data points that are contained in the corresponding subspace. Figure shows a BSP tree for the running example with no more than two data points per subspace.

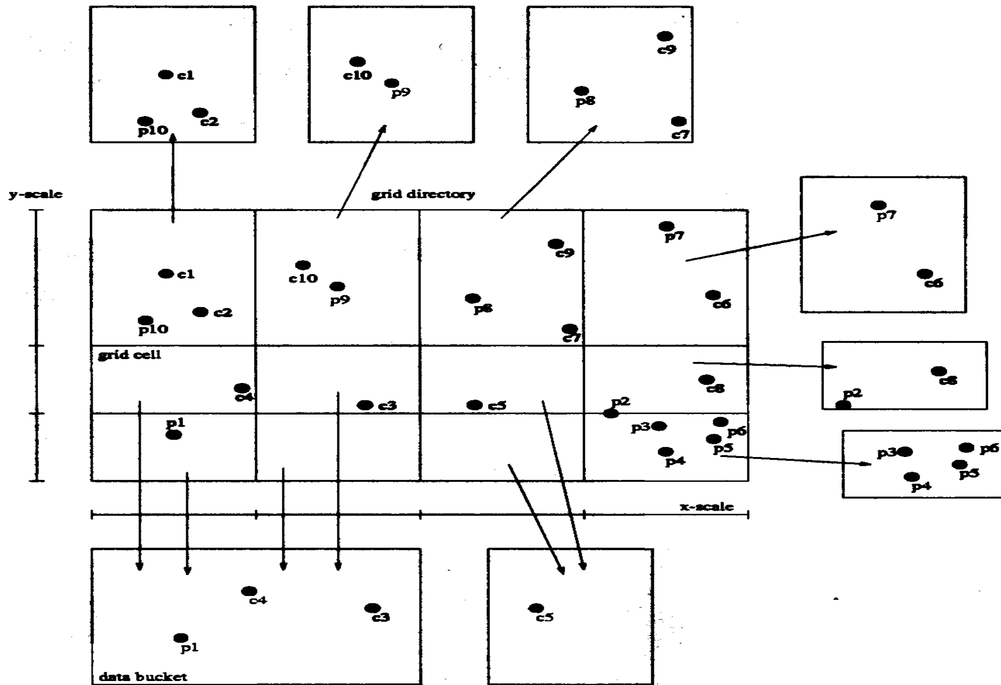
The Quadtree

Point quadtree

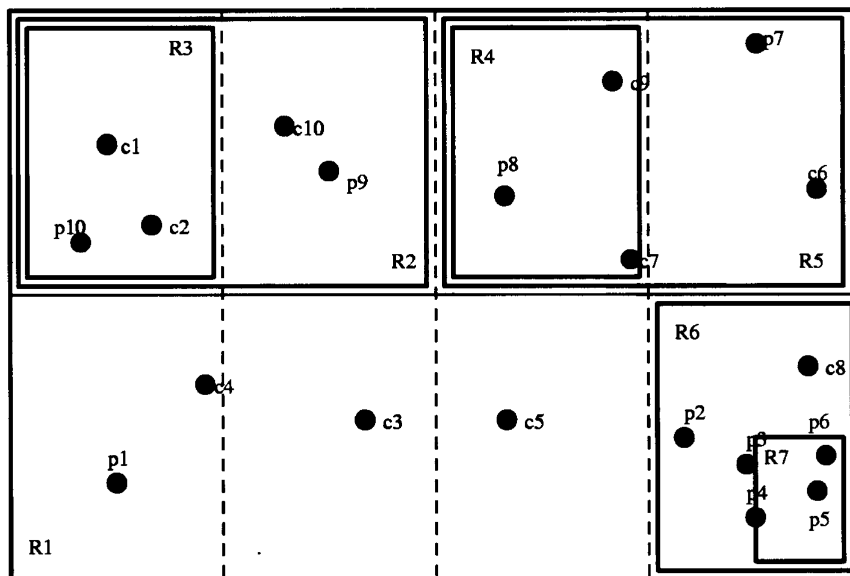
Region tree

Point Access Methods

Multidimensional Hashing
Grid File



BANG File



Two-Level Grid File

data bucket

c9
p8
c7
p7
c6

c1
c2
p10
c10
p9

subdirectory page

root directory

c10
c1
p9
c2
p10
p7
c9
p8
c6
c7
c4
c3
c5
p2
p3
c8
p6
p1
p4
p5

subscales

root scales

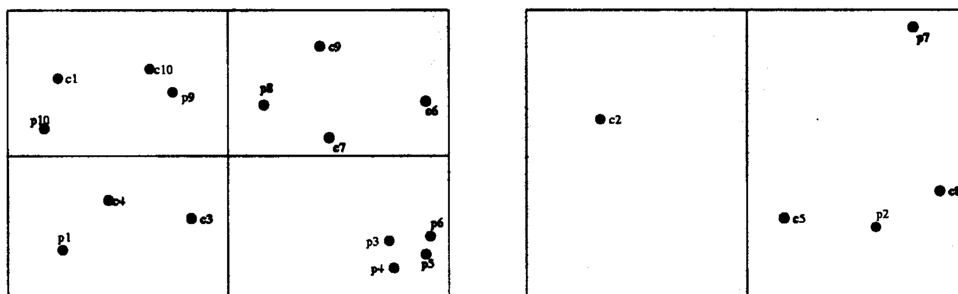
c4
c3
p1

c5

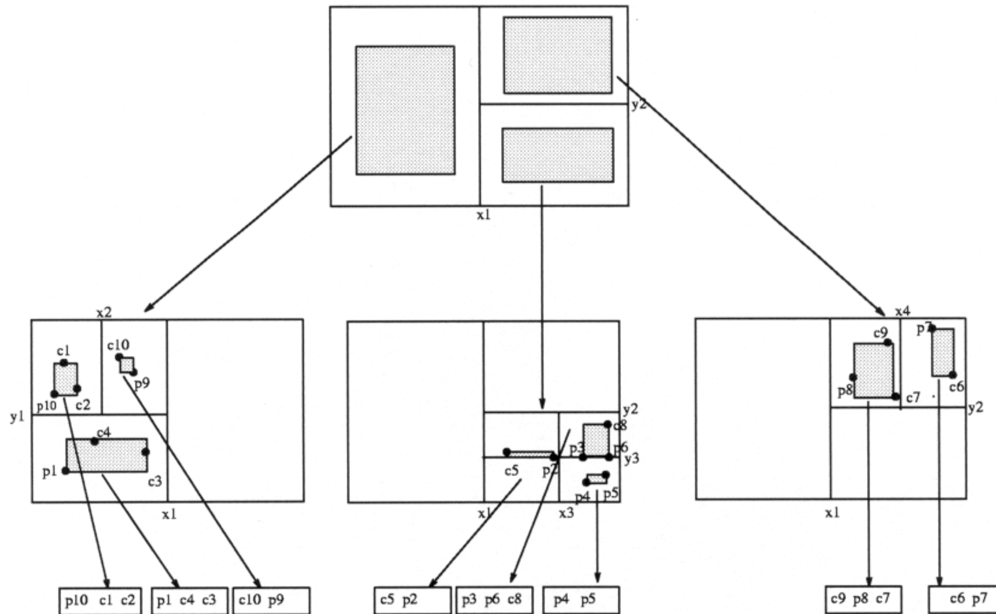
p2
c8

p3
p6
p4
p5

Twin Grid File



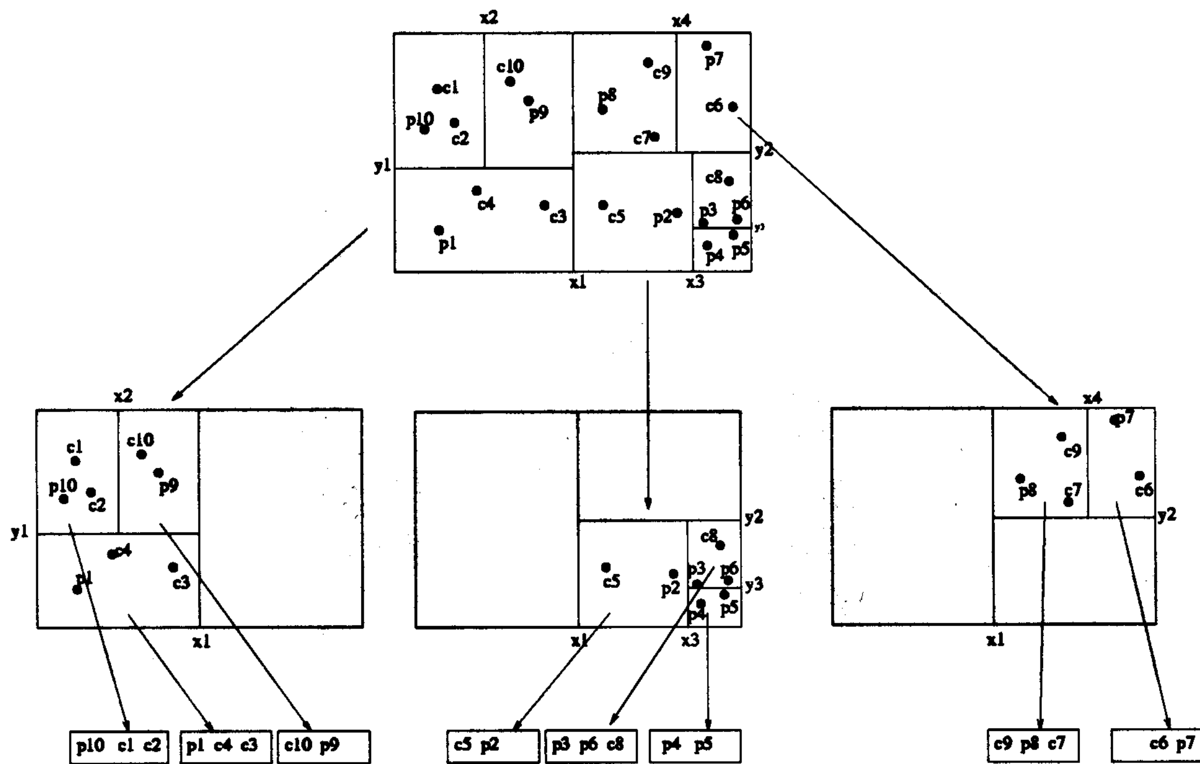
Buddy Tree



The buddy tree is a dynamic hashing scheme with a tree-like directory. The universe is cutted recursively into two parts of equal size with iso-oriented hyperplanes, and each interior node corresponds to a partition together with interval. The interval corresponds to MBB, covering points below of given node. Also:

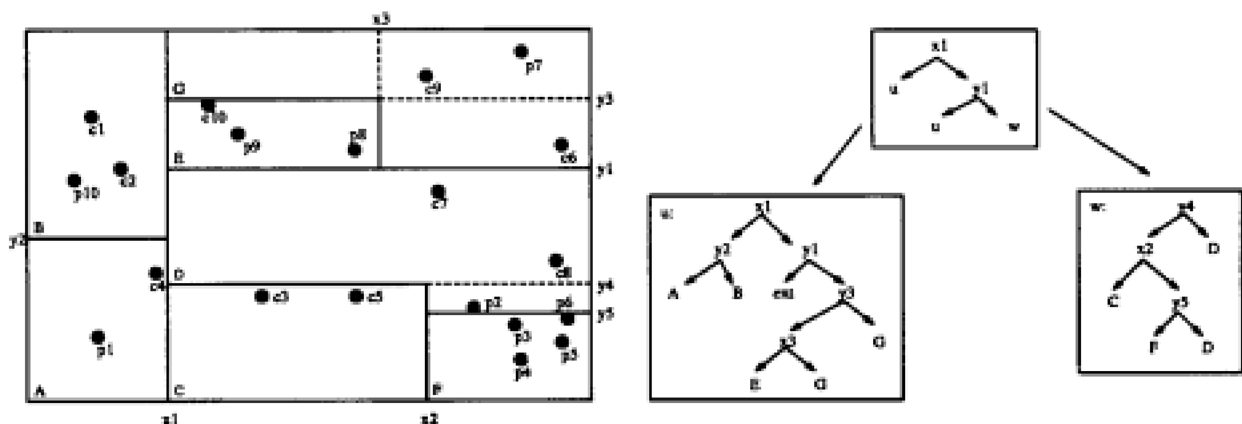
- Each directory node contains at least two entries;
- Whenever a node is split, the MBB and subnodes are recomputed, to fit situation;
- Except for the root of the directory, there is exactly one pointer referring to each directory page.

K-D-B-Tree



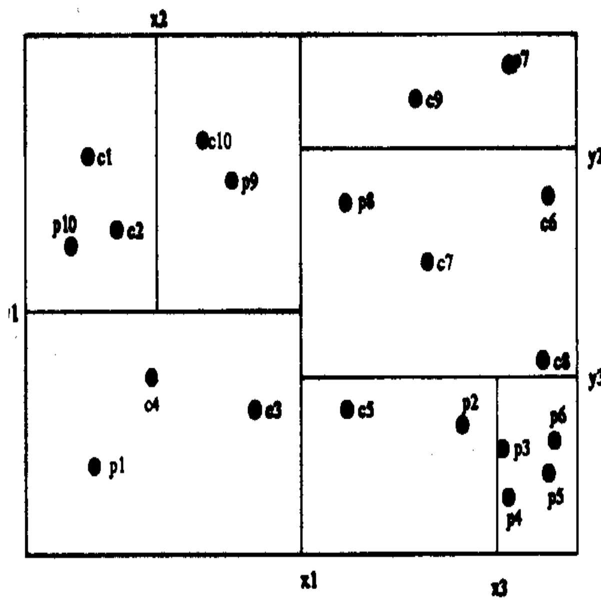
The k-d-B-tree combines properties of the adaptive k-d-tree and the B-tree.

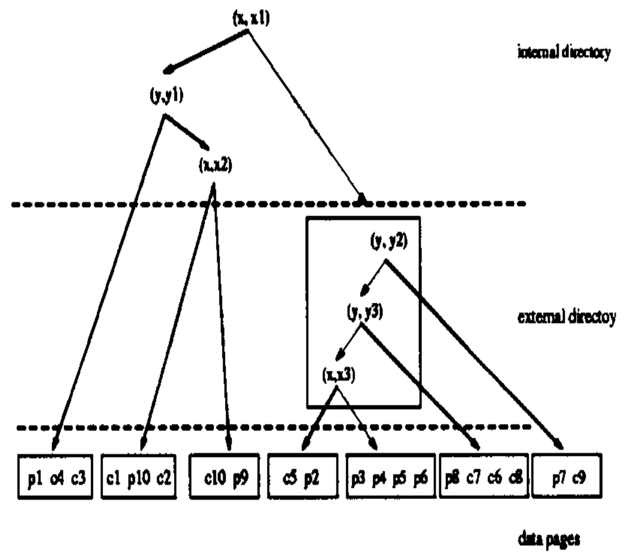
hB-tree



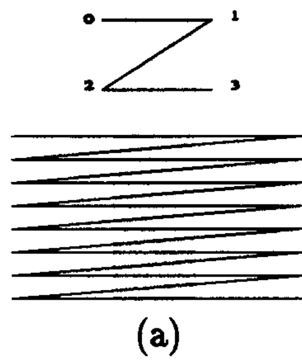
The hB-tree (holey brick tree) is similar to k-d-B-tree, except that splitting of the node is done based on ***multiple attributes***, the result is somewhat fractal structure, with external ***enclosing regions*** and several cavities called ***extracted regions***.

LSD Tree

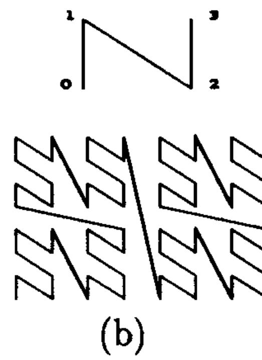




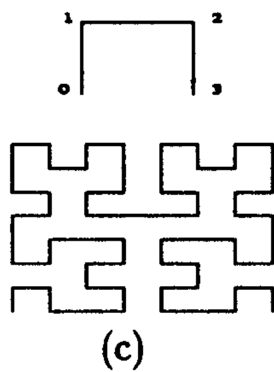
Space-filling curves



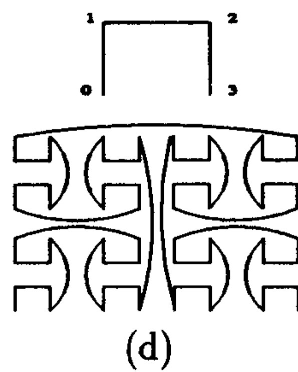
(a)



(b)

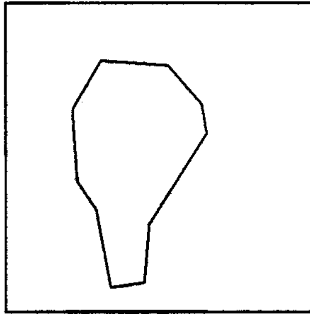


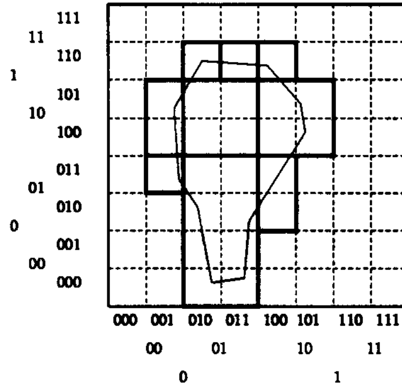
(c)

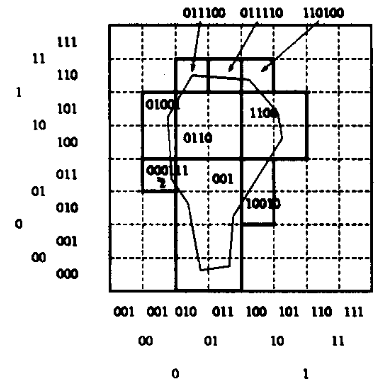


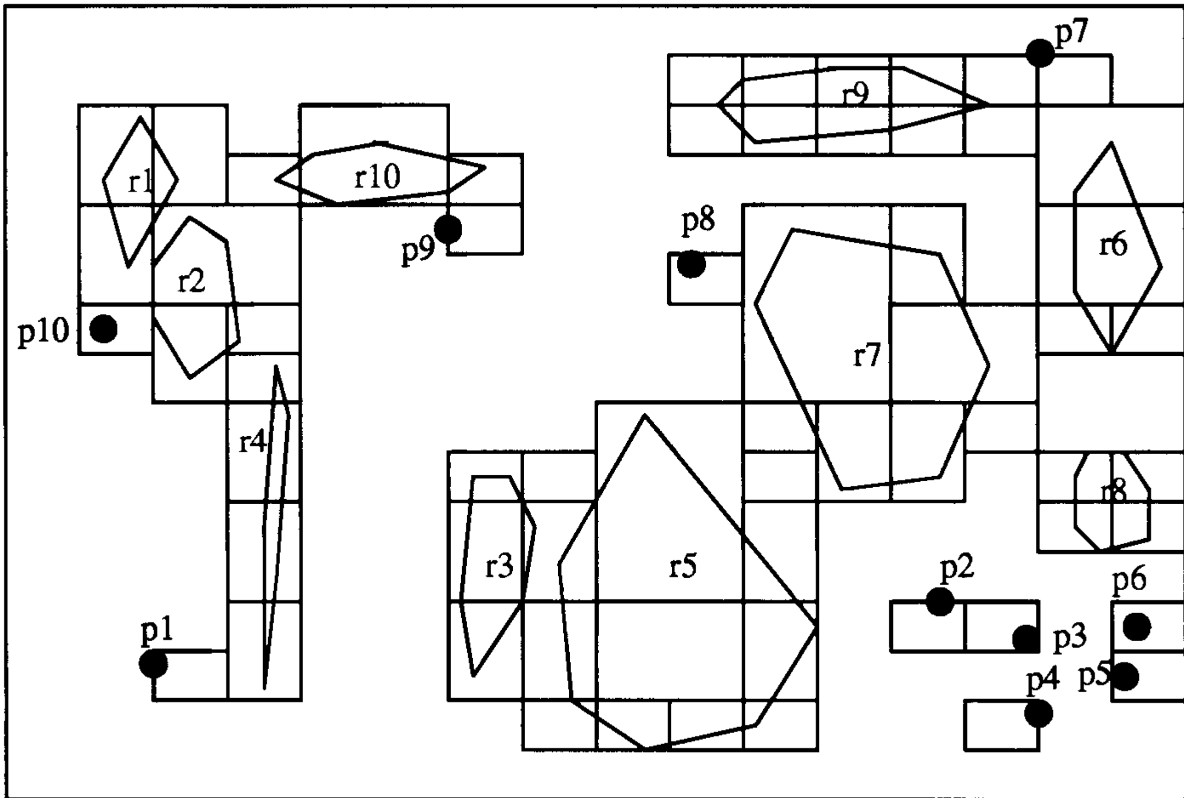
(d)

Z-ordering









Basic properties of spatial data:

1. spatial data has a complex structure (a spatial data object may be composed of a single point or several thousands of point sets, arbitrarily distributed across space. It is usually not possible to store collections of such objects in a single relational table with a fixed tuple size)

2. spatial data is often dynamic (insertions and deletions are interleaved with updates, and data structures used in this context have to support this dynamic behavior)

3. spatial databases tend to be large (the seamless integration of secondary and tertiary memory is therefore essential for efficient processing)

4. there is no standard algebra defined on spatial data (no standardized set of base operators. The set of operators heavily depends on the given application domain)

5. many spatial operators are not closed (the intersection of two polygons, for example, might return any number of single points, dangling edges, or disjoint polygons)

6. although the computational costs vary between operators, spatial database operators are generally more expensive than standard relational operators