Universidad Católica San Pablo | Department of Computer Science

Spatial Data Structures

Erick Gomez Nieto, PhD
emgomez@ucsp.edu.pe

Outline

- ❏ Introduction
- ❏ Spatial Indexing
- ❏ A brief review of BTree
- ❏ Multidimensional variations of BTree
- ❏ QuadTrees and its variations
- ❏ OcTrees
- ❏ Kd-Tree

Introduction

Spatial data consists of spatial objects made up of points, lines, regions, rectangles, surfaces, volumes, and even data of higher dimension which includes time. Examples of spatial data include cities, rivers, roads, counties, states, crop coverages, mountain ranges, parts in a CAD system, etc. Examples of spatial properties include the extent of a given river, or the boundary of a given county, etc. Often it is also desirable to attach non-spatial attribute information such as elevation heights, city names, etc. to the spatial data.

Spatial Indexing

A spatial index is a data structure that allows for accessing a spatial object efficiently. It is a common technique used by spatial databases. Without indexing, any search for a feature would require a "sequential scan" of every record in the database, resulting in much longer processing time. In a spatial index construction process, the minimum bounding rectangle serves as an object approximation. Various types of spatial indices across commercial and open-source databases yield measurable performance differences. Spatial indexing techniques are playing a central role in time-critical applications and the manipulation of spatial big data.

Zhang, X and Du, Z. (2017). Spatial Indexing. *The Geographic Information Science & Technology Body of Knowledge* (4th Quarter 2017 Edition), John P. Wilson (ed).

A brief review of B-Tree

Motivation for B-Trees

Index structures for large datasets cannot be stored in main memory

Storing it on disk requires different approach to efficiency

Assuming that a disk spins at 3600 RPM, one revolution occurs in 1/60 of a second, or 16.7ms

Crudely speaking, one disk access takes about the same time as 200,000 instructions

Motivation (cont.)

Assume that we use an AVL tree to store about 20 million records

We end up with a **very** deep binary tree with lots of different disk accesses; $\log_2 20{,}000{,}000$ is about 24, so this takes about 0.2 seconds

We know we can't improve on the log *n* lower bound on search for a binary tree

But, the solution is to use more branches and thus reduce the height of the tree!

- As branching increases, depth decreases

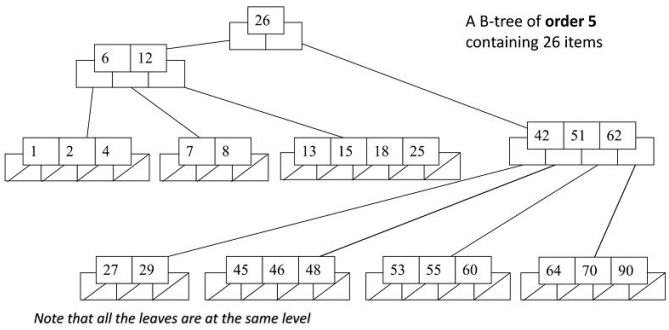
Definition of a B-tree

A B-tree of order *m* is an *m*-way tree (i.e., a tree where each node may have up to *m* children) in which:

1. the number of keys in each non-leaf node is one less than the number of its children and these keys partition the keys in the children in the fashion of a search tree
2. all leaves are on the same level
3. all non-leaf nodes except the root have at least $\lceil m / 2 \rceil$ children
4. the root is either a leaf node, or it has from two to *m* children
5. a leaf node contains no more than $m - 1$ keys

The number *m* should always be odd

An example B-Tree

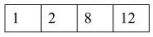


Constructing a B-tree

Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45

We want to construct a B-tree of order 5

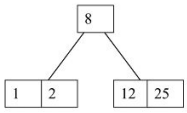
The first four items go into the root:



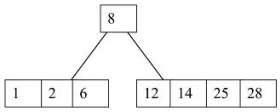
To put the fifth item in the root would violate condition 5

Therefore, when 25 arrives, pick the middle key to make a new root

Constructing a B-tree (contd.)

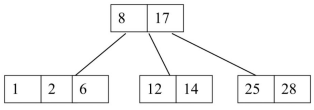


6, 14, 28 get added to the leaf nodes:

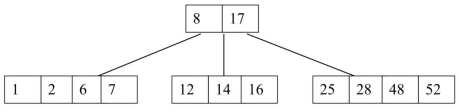


Constructing a B-tree (contd.)

Adding 17 to the right leaf node would over-fill it, so we take the middle key, promote it (to the root) and split the leaf

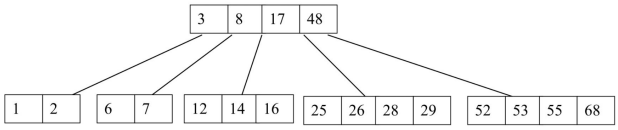


7, 52, 16, 48 get added to the leaf nodes



Constructing a B-tree (contd.)

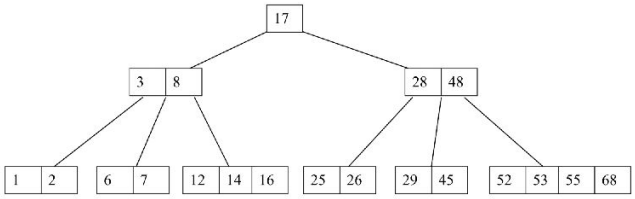
Adding 68 causes us to split the right most leaf, promoting 48 to the root, and adding 3 causes us to split the left most leaf, promoting 3 to the root; 26, 29, 53, 55 then go into the leaves



Adding 45 causes a split of 25 26 28 29

and promoting 28 to the root then causes the root to split

Constructing a B-tree (contd.)



What's wrong with B-Trees

B-Trees cannot store new types of data

Specifically people wanted to store geometrical data and multi-dimensional data

The R-Tree provided a way to do that (thanx to Guttman '84)

B-Tree multidimensional variations

K-D-B-tree (k-dimensional B-tree)

Robinson, John (1981). The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data. Sigmod '81. pp. 10–18.

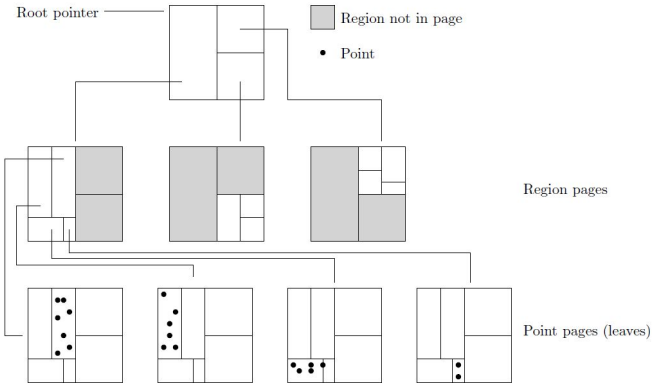
BKD tree
Procopiuc, Octavian; Agarwal, Pankaj; Arge, Lars; Vitter, Jeffrey Scott (2003). Bkd-Tree: A Dynamic Scalable kd-Tree. Advances in Spatial and Temporal Databases. Lecture Notes in Computer Science. 2750. pp. 46–65

K-D-B-tree is a tree data structure for subdividing a k-dimensional search space. The aim of the K-D-B-tree is to provide the search efficiency of a balanced k-d tree, while providing the block-oriented storage of a B-tree for optimizing external memory accesses.

The K-D-B-tree contains two types of pages:

Region pages: A collection of (region, child) pairs containing a description of the bounding region along with a pointer to the child page corresponding to that region.

Point pages: A collection of (point, location) pairs. In the case of databases, location may point to the index of the database record, while for points in k-dimensional space, it can be seen as the point's coordinates in that space.



Page overflows occur when inserting an element into a K-D-B-tree results in the size of a node exceeding its optimal size. Since the purpose of the K-D-B-tree is to optimize external memory accesses like those from a hard-disk, a page is considered to have overflowed or be overfilled if the size of the node exceeds the external memory page size.

Throughout insertion/deletion operations, the K-D-B-tree maintains a certain set of properties:

- The graph is a multi-way tree. Region pages always point to child pages, and can not be empty. Point pages are the leaf nodes of the tree.
- Like a B-tree, the path length to the leaves of the tree is the same for all queries.
- The regions that make up a region page are disjoint.
- If the root is a region page the union of its regions is the entire search space.
- When the child of a (region, child) pair in a region page is also a region page, the union of all the regions in the child is region.
- Conversely in the case above, if child is a point page, all points in child must be contained in region.

QuadTrees

Quad Trees

The term quadtree is often used in a more general senses to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. They can be differentiated on the following bases:

1. the type of data that they are used to represent,

2. the principle guiding the decomposition process, and

3. the resolution (variable or not).

Currently, they are used for points, rectangles, regions, curves, surfaces, and volumes. The decomposition may be into equal parts on each level (termed a regular decomposition), or it may be governed by the input. The resolution of the decomposition (i.e., the number of times that the decomposition process is applied) may be fixed beforehand or it may be governed by properties of the input data.

Region QuadTree

The most studied quadtree approach to region representation 1 termed a region quadtree, is based on the successive subdivision of the image array into four equal-size quadrants. If the array does not consist entirely of I's or entirely of O's (i.e., the region does not cover the entire array), it is then subdivided into quadrants, subquadrants, etc., until blocks are obtained (possibly single pixels) that consist entirely of I's or entirely of O's, i.e., each block is entirely contained in the region or entirely disjoint from it. Thus the region quadtree can be characterized as a variable resolution data structure.

Region Quadtree

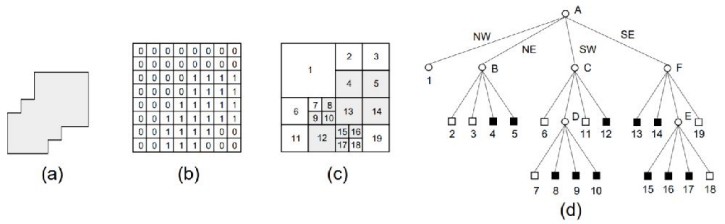
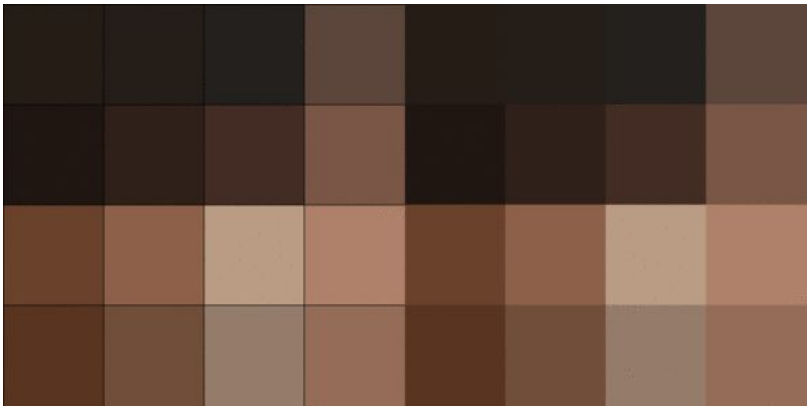


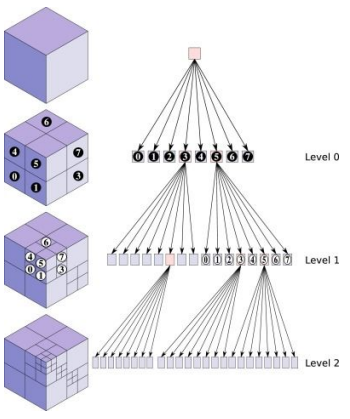
Figure 4: (a) Sample region, (b) its binary array representation, (c) its maximal blocks with the blocks in the region being shaded, and (d) the corresponding quadtree.



Region Quadtree

The region quadtree is easily extended to represent 3-dimensional data and the resulting data structure is termed an octree. It is constructed in the following manner.

We start with an image in the form of a cubical volume and recursively subdivide it into eight congruent disjoint cubes (called octants) until blocks of a uniform color are obtained, or a predetermined level of decomposition is reached.



Viljami Laurmaa, Marco Picasso, Gilles Steiner, An octree-based adaptive semi-Lagrangian VOF approach for simulating the displacement of free surfaces, Computers & Fluids, Volume 131, 2016, Pages 190-204.

The octree structure as a mesh (left) and as a graph (right) with ***Morton ordering***. Everytime a cell in the octree is split into 8 cells, the corresponding node in the graph gets 8 children nodes. The leaf nodes in the graph, shown in purple, are the computational cells in the mesh.

Morton ordering (***Z-order curve***) map multidimensional data to one dimension while preserving locality of the data points.

Point QuadTree

The point quadtree is an adaptation of a binary tree used to represent two-dimensional point data. It shares the features of all quadtrees but is a true tree as the center of a subdivision is always on a point. It is often very efficient in comparing two-dimensional, ordered data points, usually operating in ***O(log n)*** time. Point quadtrees are worth mentioning for completeness, but they have been surpassed by k-d trees as tools for generalized binary search.

Point quadtrees are constructed as follows:

Given the next point to insert, we find the cell in which it lies and add it to the tree.

The new point is added such that the cell that contains it is divided into quadrants by the vertical and horizontal lines that run through the point.

Consequently, cells are rectangular but not necessarily square. In these trees, each node contains one of the input points.

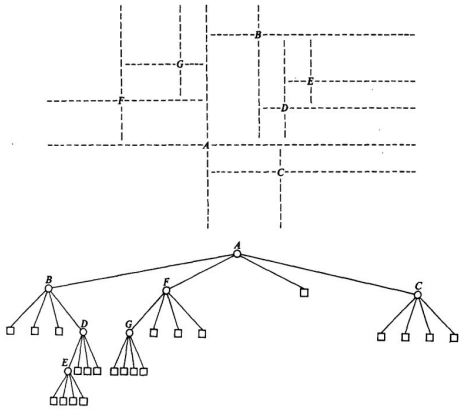


Fig. 1. Correspondence of a quad tree to the records it represents. RECORDS *A*, *B*, *C*, *D*, *E*, *F*, *G*. Null subtrees are indicated by boxes, but they do not appear explicitly in computer memory

Since the division of the plane is decided by the order of point-insertion, the tree's height is sensitive to and dependent on insertion order. Inserting in a **"bad" order** can lead to a tree of height linear in the number of input points (at which point it becomes a linked-list). If the point-set is static, pre-processing can be done to create a tree of balanced height.

Detailed in: *Finkel, R. A.; Bentley, J. L. (1974). "Quad Trees A Data Structure for Retrieval on Composite Keys". Acta Informatica. Springer-Verlag. 4: 1–9.*

PR Quadtree

It is an adaptation of the region quadtree to point data which associates data points with quadrants. The PR quad tree (see Figure 5) is organized in the same way as the region quadtree. The difference is that leaf nodes are either empty (i.e., WHITE) or contain a data point (i;e., BLACK) and its coordinates. A quadrant contains at most one data point.

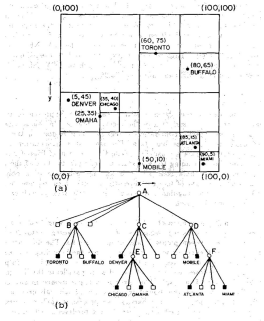


Figure 5. A PR quadtree (b) and the records it represents (a).

PR Quadtree

The shape of the pr quadtree is independent of the order in which data points are inserted into it.

The disadvantage of the pr quadtree is that the maximum level of decomposition depends on the minimum separation between two points. In particular, if two points are very close, then the decomposition can be very deep. This can be overcome by viewing the blocks or nodes as buckets with capacity c and only decomposing a block when it contains more than c points.

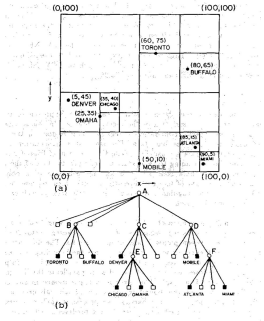


Figure 5. A PR quadtree (b) and the records it represents (a).

How it works?

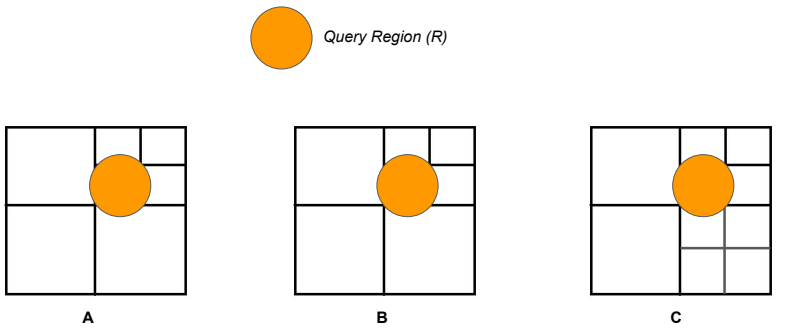
Let's see interactively at https://jimkang.com/quadtreevis/

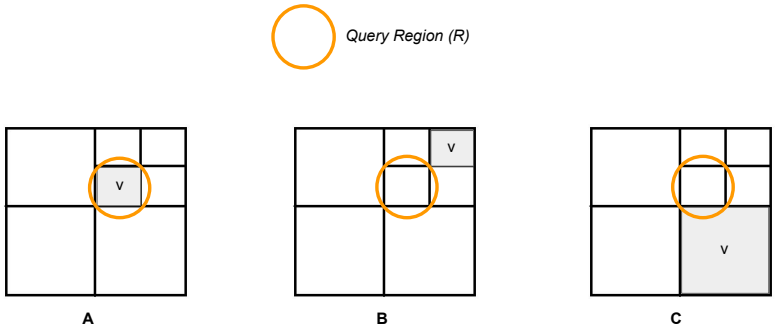
Thanks to Jim Kang

PR QuadTree | Range queries

Given a PR quadtree and a query region R (which is possibly open-ended), the result of a range query is all of the stored points in the PR quadtree that lie within R. This operation may correspond to finding records in a database that meet a certain criteria. To do the range query, first we find the smallest node/cell u in the tree that completely contains R. For each child cell v of u:

A. If v lies completely within R, add the points of v's subtree to the query result and continue the query.
B. If v and R do not intersect, discard v and continue with the other children.
C. If v and R intersect, process the children of v similarly in turn (brute-force check leaves).





Kd-Tree

Definition

Given: a set **X** of **n** points in R^d

Nearest neighbor: for any query point $q \epsilon R^d$ return the point $x \epsilon X$ minimizing **D(x,q)**

Intuition: Find the point in **X** that is the ***closest*** to **q**

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509-517.

Motivation

Learning: Nearest neighbor rule

Databases: Retrieval

Data mining: Clustering

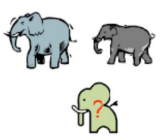
Donald Knuth in vol.3 of ***The Art of Computer Programming*** called it the post-office problem, referring to the application of assigning a resident to the ***nearest-post office***

Nearest-neighbor rule

Consider a test point x.

x' is the closest point to x out of the rest of the test points.

Nearest Neighbor Rule selects the class for x with the assumption that: $\omega(x) = \omega(x')$

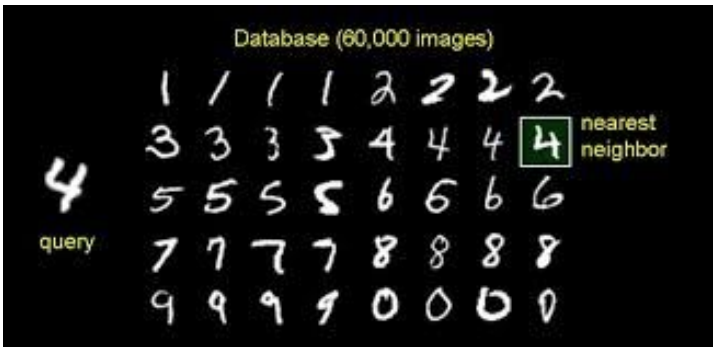


Is this reasonable?

Yes, if x' is sufficiently close to x.

If x' and x were overlapping (at the same point), they would share the same class.

As the number of test points increases, the closer x' will probably be to x.



The MNIST dataset (*http://yann.lecun.com/exdb/mnist/*)

Methods for computing NN

Linear scan: O(nd) time

This is pretty much all what is known for exact algorithms with theoretical guarantees

In practice:
- ***kd-trees*** work "well" in "low-medium" dimensions

2-dimensional kd-trees

A data structure to support range queries in R^2
- Not the most efficient solution in theory
- Everyone uses it in practice

Preprocessing time: **O(nlogn)**

Space complexity: **O(n)**

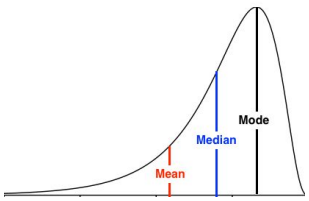
2-dimensional kd-trees

- Jon Bentley, 1975
- Tree used to store spatial data.
  - Nearest neighbor search.
  - Range queries.
  - Fast look-up
- k-d tree are guaranteed $\log_2 n$ depth where n is the number of points in the set.
  - Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

2-dimensional kd-trees

Algorithm:
- Choose **x** or **y** coordinate (alternate)
- Choose the median of the coordinate; this defines a horizontal or vertical line
- Recurse on both sides

We get a binary tree:
- Size **O(n)**
- Depth **O(logn)**
- Construction time **O(nlogn)**

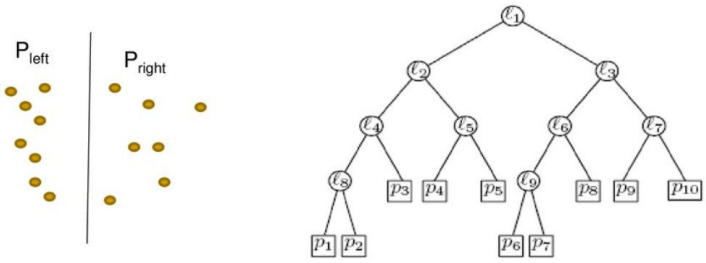


2-dimensional kd-trees

- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.
- Division strategies
  - divide points perpendicular to the axis with widest spread.
  - divide in a round-robin fashion (book does it this way)

A **round robin** is an arrangement of choosing all elements in a group equally in some rational order, usually from the top to the bottom of a list and then starting again at the top of the list and so on.

- Every node (except leaves) represents a hyperplane that divides the space into two parts.
- Points to the left (right) of this hyperplane represent the left (right) sub-tree of that node.

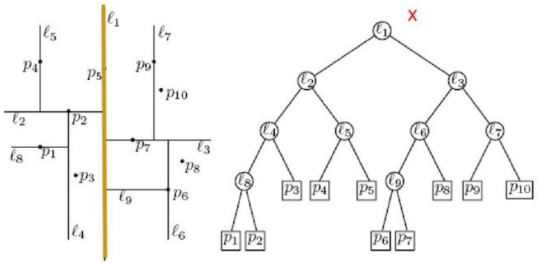


As we move down the tree, we divide the space along alternating (but not always) axis-aligned hyperplanes:

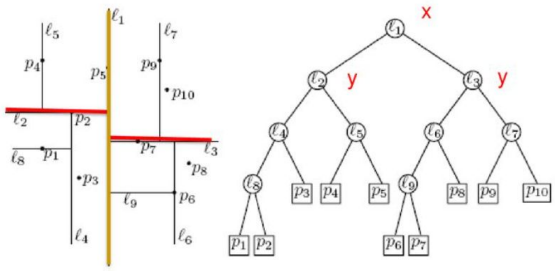
Split by x-coordinate: split by a vertical line that has (ideally) half the points left or on, and half right.

Split by y-coordinate: split by a horizontal line that has (ideally) half the points below or on and half above.

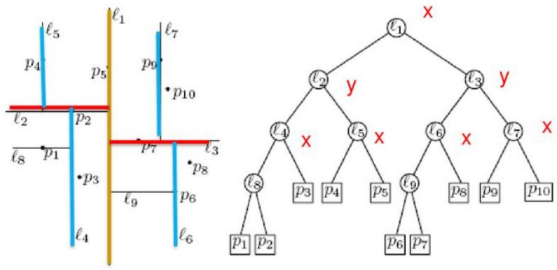
Split by x-coordinate: split by a vertical line that has approximately half the points left or on, and half right.



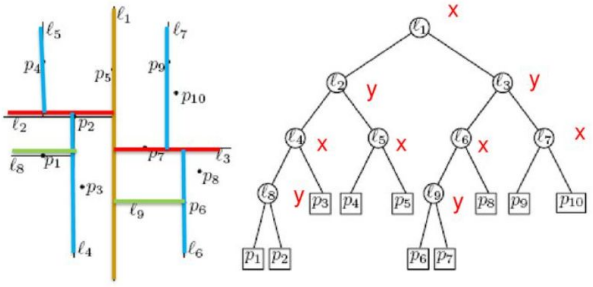
Split by y-coordinate: split by a horizontal line that has half the points below or on and half above.



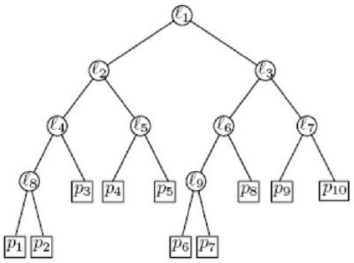
Split by x-coordinate: split by a vertical line that has half the points left or on, and half right.



Split by y-coordinate: split by a horizontal line that has half the points below or on and half above.



- A KD-tree node has 5 fields
  - Splitting axis
  - Splitting value
  - Data
  - Left pointer
  - Right pointer

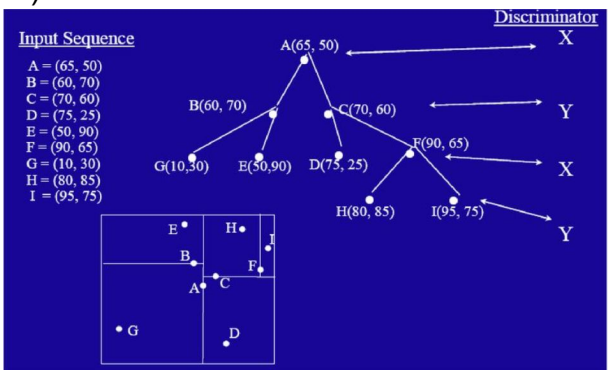


Splitting Strategies

- Divide based on order of point insertion
  - Assumes that points are given one at a time.
- Divide by finding median
  - Assumes all the points are available ahead of time.
- Divide perpendicular to the axis with widest spread
  - Split axes might not alternate

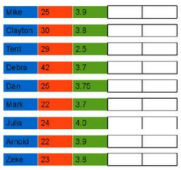
… and more!

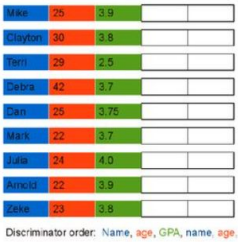
Example -- using order of point insertion (data stored at nodes)



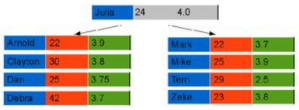
Example -- using median

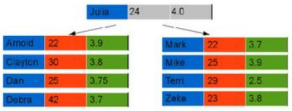
Students relation (name, age, GPA):
Mike, 25, 3.9
Clayton, 30, 3.8
Terri, 29, 2.5
Debra, 42, 3.7
Dan, 25, 3.75
Mark, 22, 3.7
Julia, 24, 4.0
Arnold, 22, 3.9
Zeke, 23, 3.8

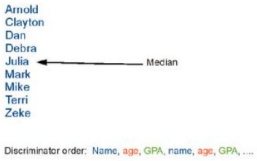




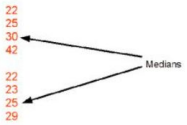
Arnold
Clayton
Dan
Debra
Julia ← Median
Mark
Mike
Terri
Zeke



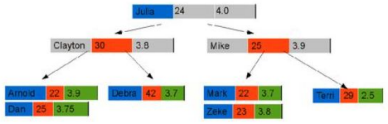


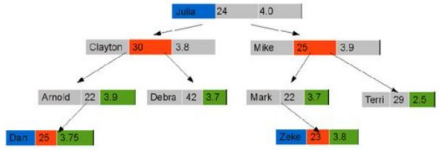


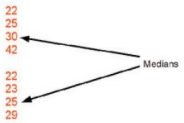
Discriminator order: Name, age, GPA, name, age, GPA, ...



Discriminator order: Name, age, GPA, name, age, GPA, ...



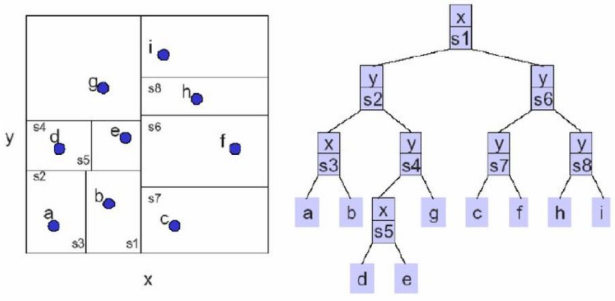




Discriminator order: Name, age, GPA, name, age, GPA, ...

Discriminator order: Name, age, GPA, name, age, GPA, ...

Example -- split perpendicular to the axis with widest spread



An illustrated example can be found at:

https://slideplayer.com/slide/14139272/

Algorithms using Quadtree and Octree

One of the advantages of using quadtrees for image manipulation is that the set operations of union and intersection can be done simply and quickly.

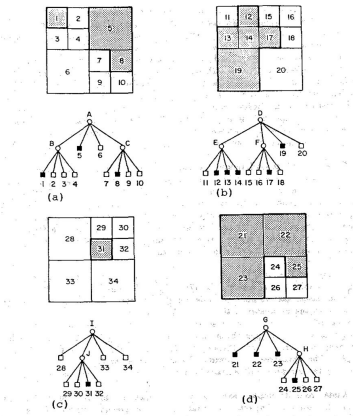


Figure 3. Example of set-theoretic operations. (a) sample image and its quadtree; (b) sample image and its quadtree; (c) intersection of the images in (a) and (b); union of the images in (a) and (b).