Advanced Data Structures

Amalia Duch

CS-UPC

February 16, 2020

- Points, lines,
- rivers,
- maps, cities,
- roads,
- cubes,
- hyperplanes,
- hypercubes,
- mp3, mp4 and mp5 files,
- jpeg files,
- pixels,
- . . . ,

among many others are examples of multidimensional data to deal with in several nowadays computer-based applications.

Multidimensional Data

Multidimensional data can be divided into:

- Point multidimensional data.
- Spatial multidimensional data: lines, bounds, regions, . . .

Some applications are:

- database design,
- computer graphics,
- computer vision,
- computational geometry,
- image processing,
- geographic information systems (GIS),
- pattern recognition,
- very large scale integration (VLSI) design,
- ...

Multidimensional Data Structures

Multidimensional data structures are data management systems that support search and update operations in multidimensional data.

Point Multidimensional Data Structures

Point multidimensional data structures are:

- Multidimensional data structures that store multidimensional points.
- Frequently present in applications.
- Useful to store surrogates.

Surrogates

- Surrogates are K-dimensional point representations of non-point objects.
- For instance, an image can be represented as a vector with a fixed number of characteristics such as brightness, contrast, size, number of colors...
- Typically non-point objects are complex, occupy a lot of memory space, and testing properties on them is expensive (for instance, to decide whether two objects are similar or not).
- It is usual to use surrogates for filtering: a first search is performed in the space of surrogates in order to eliminate objects that do not satisfy associative queries.

Some Assumptions and Nomenclature

For simplicity in what follows,

- We will use the term multidimensional data structures to refer to point multidimensional data structures.
- Without loss of generality:
    - We will identify points in a K-dimensional space with their key $x = (x_0, x_1, \ldots, x_{K-1})$.
    - We will assume that each x_i belongs to $D_i = [0, 1]$ and hence the universe D is the hypercube $[0, 1]^K$.

Associative Retrieval

Given a file $\mathcal{F}$, that is a collection of n records, each *record* of $\mathcal{F}$ is an ordered K-tuple of values (the attributes or coordinates of the record) drawn from a totally ordered domain.
A retrieval of records from $\mathcal{F}$ is called a *query* of the file and it specifies certain conditions to be satisfied by the attributes of the records to be retrieved from $\mathcal{F}$.
The query is considered *associative* only if it specifies conditions dealing with more than one of the attributes.

Associative Retrieval

Multidimensional data structures must support:

- Usual insertions, deletions, (exact) queries
- Associative queries such as:

  Partial Match Queries: Find the data points that match some specified coordinates of a given query point q.

  Orthogonal Range Queries: Find the data points that fall within a given hyper rectangle Q (specified by K ranges).

  Nearest Neighbor Queries: Find the closest data point to some given query point q (under a predefined distance).

Partial Match Queries

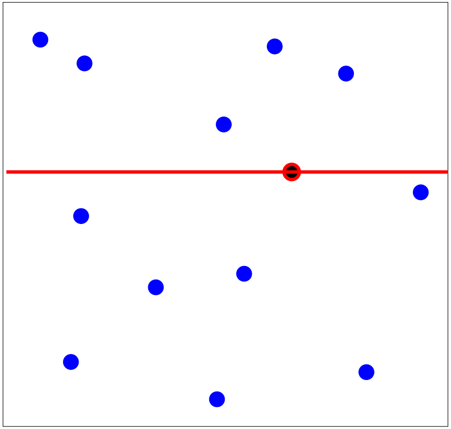
Definition

Given a file F of n K-dimensional records and a query $q = (q_0, q_1, \ldots, q_{K-1})$ where each q_i is either a value in D_i (it is specified) or $*$ (it is unspecified), a *partial match query* returns the subset of records x in F whose attributes coincide with the specified attributes of q. This is,

$$\{x \in F \mid q_i = * \text{ or } q_i = x_i, \forall i \in \{0, \ldots, K-1\}\}.$$

Example of Partial Match Queries

Query: $q = (*, q_2)$ or $q = (q_1, q_2)$ with specification pattern: 01



Orthogonal Range Queries

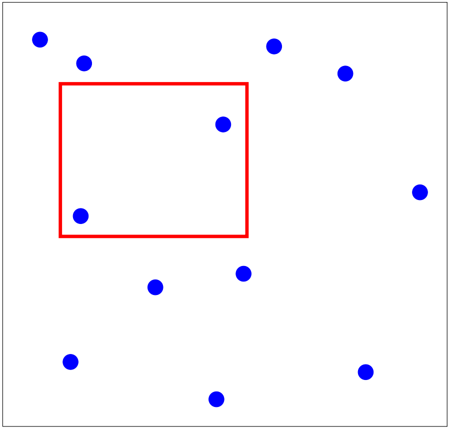
Definition

Given a file F of K-dimensional records and a hyper-rectangle $Q = [l_0, u_0] \times [l_1, u_1] \times \cdots \times [l_{K-1}, u_{K-1}]$, an *orthogonal range query* returns the subset of records in F which fall inside Q. This is,

$$\{x \in F \mid l_i \leq x_i \leq u_i, \forall i \in \{0, \ldots, K-1\}\}.$$

Example of Orthogonal Range Queries

Query: $Q = [\ell_1, u_1] \times [\ell_2, u_2]$



Nearest Neighbor Queries

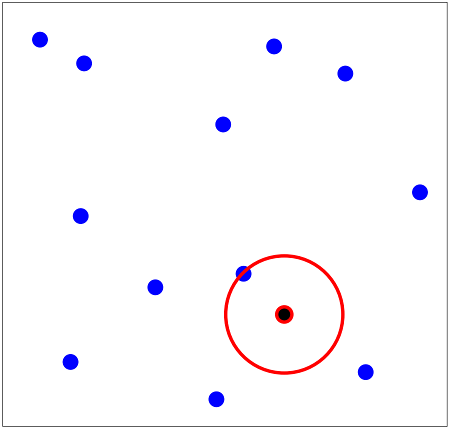
Definition

Given a file F of K-dimensional records and a query point q, a *nearest neighbor query* consists of finding the key in the file closest to q according to some predefined *distance* measure d. This is,

$$\{x \in F \mid d(q,x) \leq d(q,y), \forall y \in F\}.$$

Example of Nearest Neighbor Queries

Query: $q = (q_1, q_2)$



What’s the goal?

- Linear scanning of the collection is not efficient: we need to examine all or a substantial fraction of the n points to yield the answer.
- We would like to support insertion of new points and deletion of existing ones from the collection
- There exist specialized solutions for each type of associative query; however we would like also to have data structures that support *all* operations with good expected performance (less than linear) and using $\Theta(nK)$ memory space.

Multidimensional Data Structures

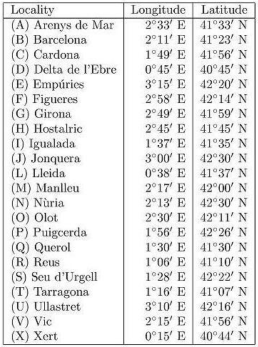
Multidimensional data structures can be:

- Static / Dynamic
- Data driven / Space driven
- General purpose / Ad-hoc
- Hierarchical (tree-like) / Non-hierarchical (hash-like)
- …

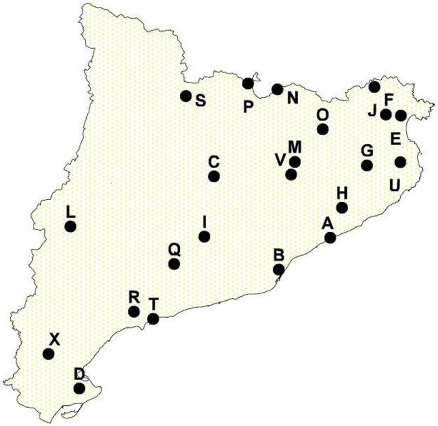
Inverted Files

The *projection* technique (also called *inverted files* [11]) consists of keeping, for each attribute, a sorted sequence of the records in the file.
Geometrically, this corresponds to projections of the points on each coordinate. The K lists representing the K projections can be obtained by using K times some standard sorting algorithm.

Two dimensional file and plot of localities of Catalonia



Locality	Longitude	Latitude
(A) Arenys de Mar	2°33′ E	41°33′ N
(B) Barcelona	2°11′ E	41°23′ N
(C) Cardona	1°49′ E	41°56′ N
(D) Delta de l'Ebre	0°45′ E	40°45′ N
(E) Empúries	3°15′ E	42°20′ N
(F) Figueres	2°58′ E	42°14′ N
(G) Girona	2°49′ E	41°59′ N
(H) Hostalric	2°45′ E	41°45′ N
(I) Igualada	1°37′ E	41°35′ N
(J) Jonquera	3°00′ E	42°30′ N
(L) Lleida	0°38′ E	41°37′ N
(M) Manlleu	2°17′ E	42°00′ N
(N) Nùria	2°13′ E	42°30′ N
(O) Olot	2°30′ E	42°11′ N
(P) Puigcerda	1°56′ E	42°26′ N
(Q) Querol	1°30′ E	41°30′ N
(R) Reus	1°06′ E	41°10′ N
(S) Seu d'Urgell	1°28′ E	42°22′ N
(T) Tarragona	1°16′ E	41°07′ N
(U) Ullastret	3°10′ E	42°16′ N
(V) Vic	2°15′ E	41°56′ N
(X) Xert	0°15′ E	40°44′ N



Inverted Files

Inverted file of localities in Catalonia

Long.	$X \to L \to D \to R \to T \to S \to Q \to I \to C \to P \to B \to N \to V \to M \to O \to A \to H \to G \to F \to J \to U \to E$
Lat.	$X \to D \to T \to R \to B \to Q \to A \to I \to L \to H \to V \to C \to G \to M \to O \to F \to U \to E \to S \to P \to N \to J$

Inverted Files

- To pre-process a file of n K-dimensional records we must perform K sorts of n elements, which takes $\Theta(Kn \log n)$ time.
- To store such a file require $\Theta(Kn)$ space.
- An exact match query can be answered by searching (using binary search) any of the K sorted lists in $\Theta(\log n)$ worst-case time.
- The worst-case cost for any kind of associative queries is $\Theta(Kn)$.

Inverted Files

Range queries can be answered by the following procedure:

- Choose one of the attributes, say the i-th.
- Find the two limits of the range query in the appropriate sorted sequence (using binary search).
- All the records satisfying the query will be in the list between these two positions. This (usually smaller) list is then searched by brute force.

For almost cubical range queries that have a small number of records satisfying them (and are therefore similar to nearest-neighbor searches), the range query time of projection is given by $\Theta(n^{1-\frac{1}{K}})$ in the average case when the point set is drawn from a smooth underlying distribution [2]. This technique has been applied by Friedman, Baskett, and Shustek [10] in their algorithm for nearest-neighbor queries.

Grid Files

- The *cell* or *fixed grid* method, first introduced by Nievergelt, Hinterberger and Sevcik [12] and based in extendible hashing, divides the space into equal sized cells or buckets (squares and cubes for two and three dimensional data respectively).
- The data structure is a K-dimensional array with one entry per cell.
- Each cell is implemented as a linked list storing the points within it.

Grid Files

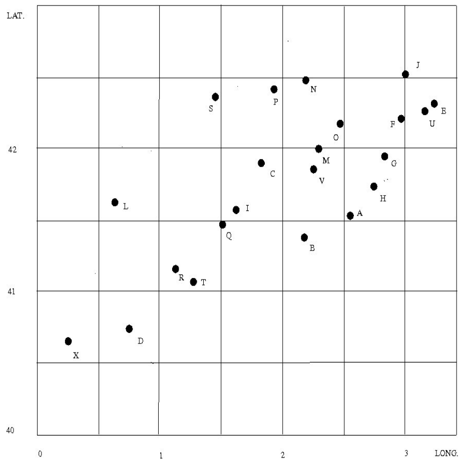


Figure: Illustration of a grid for the file of localities in Catalonia.

Grid Files

- The space required for this data structure can be super-linear in the number n of records, even if the data is uniformly distributed.
- In particular, Regnier [13] showed that the average size of the grid file while storing n K-dimensional records is $\Theta\left(n^{1+(K-1)/(Kb+1)}\right)$, where b is the bucket size, and that the average occupancy of the data buckets is approximately 69%.

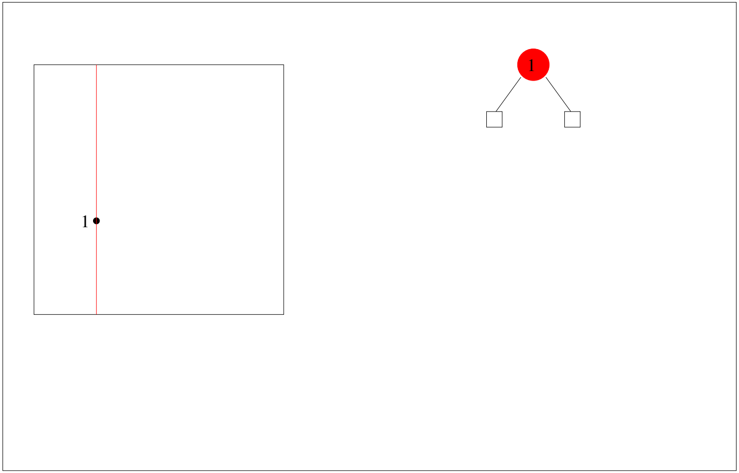
Grid Files

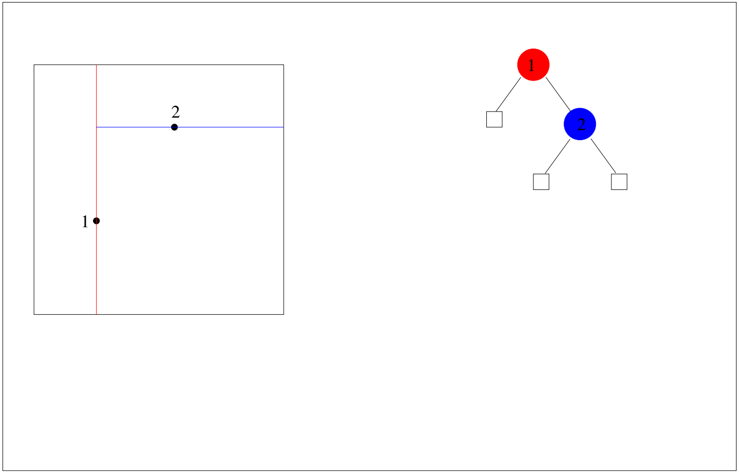
- Range queries with constant size can be answered (in a file organized by cells having the same size than the queries) with approximately 2^K cell accesses [3]. The expected search time in this case is proportional to 2^K times the average number of points per cell. In such files nearest neighbor queries have similar performance.
- In most applications, however, the queries will vary in size and shape, and there is little information available for making a good choice of cell size (and possibly shape).

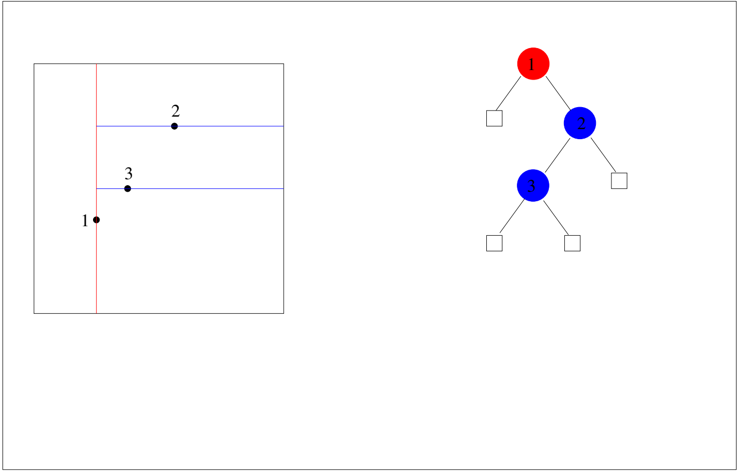
Relaxed K-d Trees

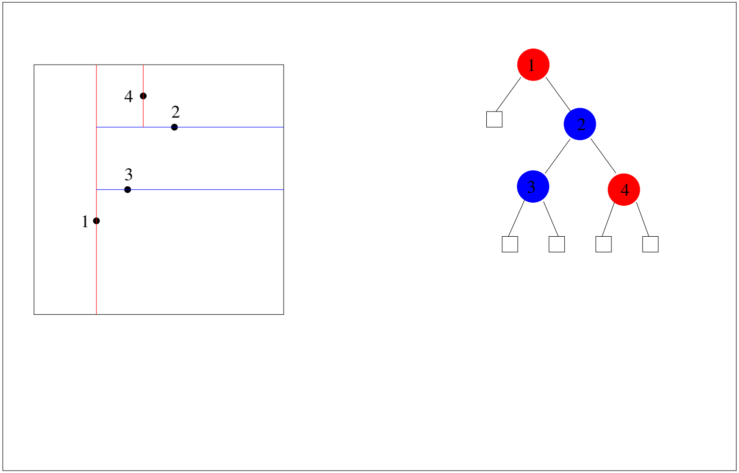
A relaxed K-d tree for a set of K-dimensional keys is a binary tree in which:

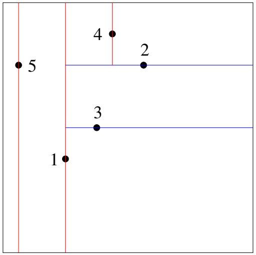
1. Each node contains a K-dimensional record and has associated an arbitrary discriminant $j \in \{0, 1, \ldots, K-1\}$.
2. For every node with key x and discriminant j, the following invariant is true: any record in the right subtree with key y satisfies $y_j < x_j$ and any record in the left subtree with key y satisfies $y_j \geq x_j$.

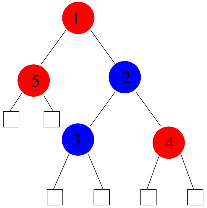

1
1


2
1
1
2


1
2
3
1
2
3


4
2
3
1
1
2
3
4


4
2
5
3
1


1
5
2
3
4

- A K-d tree of size n induces a partition of the domain D into $n+1$ regions, each corresponding to a leaf in the K-d tree.
- The *bounding box* (or *bounds array*) of a node $\langle x, j\rangle$ is the region of the space delimited by the leaf in which x falls when it is inserted into the tree. Thus, the bounding box of the root $\langle y, i\rangle$ is $[0,1]^K$, the bounding box of the left subtree's root is $[0,1] \times \cdots \times [0, y_i] \times \cdots \times [0,1]$, and so on.

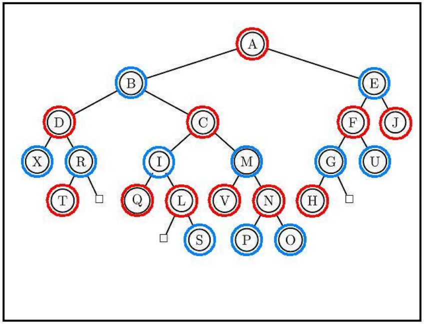
Standard K-d Trees

Definition (Bentley75)

A K-dimensional search tree T (K-d tree, for short) of size $n \geq 0$ stores a set of n K-dimensional records, each holding a key $x = (x_0, \ldots, x_{K-1}) \in D$, where $D = D_0 \times \cdots \times D_{K-1}$, and each D_j, $0 \leq j < K$, is a totally ordered domain. The K-d tree T is a binary tree such that

- either it is empty and $n = 0$, or
- its root stores a record with key x and has a discriminant $j =$ level of the root $\mod K$, $0 \leq j < K$, and the remaining $n - 1$ records are stored in the left and right subtrees of T, say L and R, in such a way that both L and R are K-d trees; furthermore, for any key $u \in L$, it holds that $u_j < x_j$, and for any key $v \in R$, it holds that $x_j < v_j$.

K-dimensional Binary Search Trees



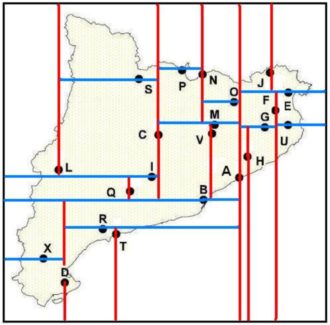
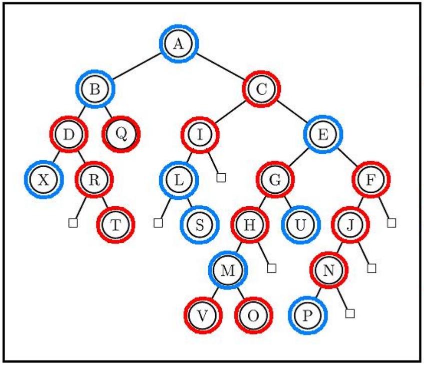


Figure: Standard K-d tree for the file of localities in Catalonia.

K-dimensional Binary Search Trees



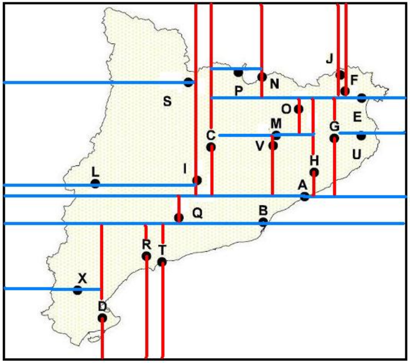


Figure: Relaxed K-d tree for the file of localities in Catalonia.

Partial Match Algorithm in Relaxed K-d Trees

Partial match search in relaxed K-d trees works as follows:

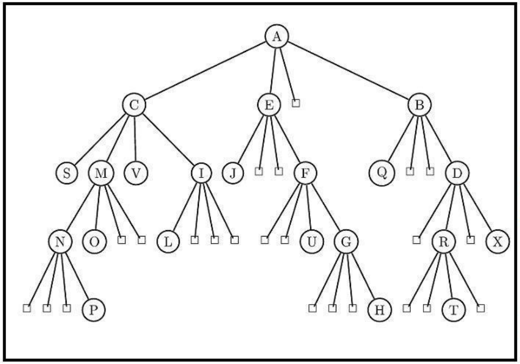
- At each node of the tree we verify if it satisfies the query and we examine its discriminant.
- If the discriminant is specified in the query then the algorithm recursively follows in the appropriate subtree depending on the result of the comparison between the key and the query.
- Otherwise the algorithm recursively follows the two subtrees of the node.

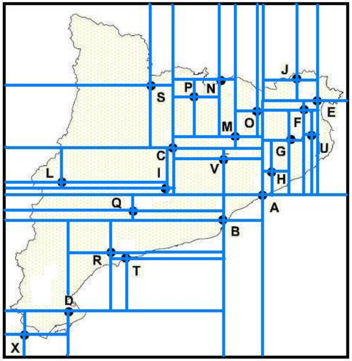
Definition of Standard Two-dimensional Quad Trees

A quad tree (Bentley and Finkel, 1974) for a file of 2-dimensional records, $F = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, is a quaternary tree in which:

1. Each node from F contains a 2-dimensional key and has associated four subtrees corresponding to the quadrants NW, NE, SE and SW.
2. For every node with key x the following invariant is true: any record in the NW subtree with key y satisfies $y_1 < x_1$ and $y_2 \geq x_2$; any record in the NE subtree with key y satisfies $y_1 \geq x_1$ and $y_2 \geq x_2$; any record in the SE subtree with key y satisfies $y_1 \geq x_1$ and $y_2 < x_2$; and, any record in the SW subtree with key y satisfies $y_1 < x_1$ and $y_2 < x_2$.

Example of Standard Two-dimensional Quad Trees





Definition of K-dimensional Quad Trees

Definition

A point quad search tree (or just “quad tree” throughout this work) T of size $n \geq 0$ stores a set of n K-dimensional records, each holding a key $x = (x_0, \ldots, x_{K-1}) \in D$, where $D = D_0 \times \cdots \times D_{K-1}$, and each D_j, $0 \leq j < K$, is a totally ordered domain. The quad tree T is a 2^K-ary tree such that

- either it is empty and $n = 0$, or
- its root stores a record with key x and has 2^K subtrees, each one associated to a K-bitstring $w = w_0 w_1 \ldots w_{K-1} \in \{0,1\}^K$, and the remaining $n-1$ records are stored in one of these subtrees, let's say T_w, in such a way that $\forall w \in \{0,1\}^K$: T_w is a quad tree, and for any key $y \in T_w$, it holds that $y_j < x_j$ if $w_j = 0$ and $y_j > x_j$ otherwise, $0 \leq j < K$.

A quad tree of size n induces a partition of the domain D into $(2^K - 1)n + 1$ regions, each corresponding to a leaf in the quad tree. The *bounding box* of a node in a quad tree is defined in a similar way as for K-d trees.

Deletion of nodes into two-dimensional quad trees is complicated. Finkel and Bentley [7] suggested that all nodes of the tree rooted at the deleted node must be reinserted, but this is usually expensive. A more efficient process developed by Sammet [16, 15] allows to reduce the number of nodes to be reinserted, although it is still an expensive and not straightforward process.

Algorithms for exact search, partial match, orthogonal range search and nearest neighbor search are similar to those for K-d trees already described.

The standard model for the probabilistic analysis of quad trees is that a *random* quad tree of size n is built by inserting n points independently drawn from some continuous probability distribution defined over $[0, 1]^K$.

The expected height H_n of a K-dimensional quad tree of size n is in probability asymptotic to $(c/K) \log n$, where $c = 4.31107\ldots$ [5]. It has been shown independently by Devroye and Laforest [6] and Flajolet et al. [8] that the expected cost of a random search in a random K-dimensional quad tree of size $n-1$ is $(2/K) \log n$. For two-dimensional random quad trees of size $n-1$ Devroye and Laforest [6] show that the variance of a random search is $\sqrt{\frac{2}{K^2} \log n}$. A complete characterization of random searches in K-dimensional quad trees is given by Flajolet and Lafforgue [9]. In particular, they show that the cost of a random search has logarithmic mean and variance, and that it is asymptotically distributed as a normal variable, although no closed forms are given for $K \geq 3$.

K-d Tries and Quad Tries

The idea of K-d trees can easily be generalized to digital search trees [4, 14] making a regular partition of the search space based on digits. The recursive partition of a region of the search space terminates when the region contains one (or no) data points. Searching in a binary K-d trie is as follows. At level 0 we use the first digit of the first key. If it is a zero the search proceeds to the left of the trie, and the search proceeds to the right if the bit is a one. The first bit of the second key is used in level 1, and so on, up to level $K - 1$. Then, in level K we use the second bit of the first key and so on. Note that both, the tree and the partition, depend on the given set of keys but not in the order in which they were inserted in the trie.

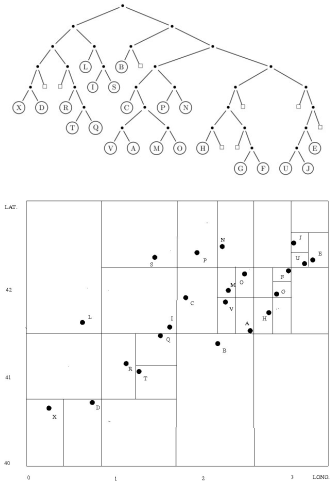


Figure: Graphic and geometric representation of a K-d trie built from the file of localities in Catalonia.

Range Trees

Range trees were introduced by Bentley [1]. They achieve the best worst-case search time for range search among all the structures described so far, but they have large preprocessing and storage cost. For most applications, the high storage required by range trees is prohibitive, but they are still interesting from a theoretical point of view.

Range Trees

Range trees are recursively defined in dimension: the K-dimensional structure is defined in terms of the $(K-1)$-dimensional one.

Range Trees: definition

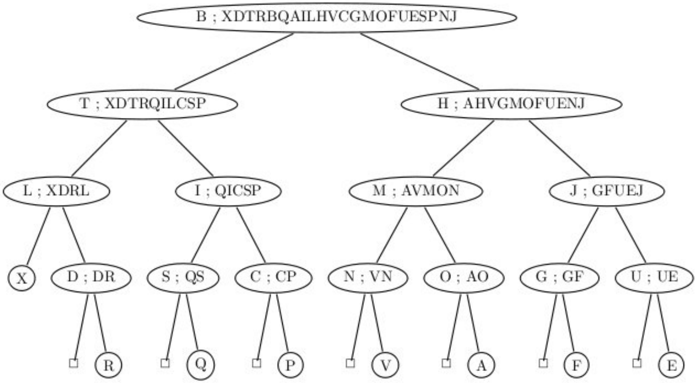
- A *range tree* for a set of one-dimensional records is a sorted list where the elements are stored by key ascending order.

Range Trees: definition

- A two-dimensional range tree is a rooted binary tree in which every node has associated a sorted array (one-dimensional range tree), a discriminant, and pointers to its left and right subtrees.
- The discriminant of every node is the median value of its records (sorted with respect to the first attribute), while arrays are sorted by the second attribute.
- The sorted array of the root contains all the nodes in the file. The root of its left subtree has a sorted array containing the records with first attribute smaller than the root's discriminant (and the right child the records with first attribute greater).
- This partitioning process continues until arrays consist of a single element.

The two-dimensional range of localities in Catalonia

Every node contains its discriminant value taken from the first attribute (longitude) and its associated list sorted with respect to the second attribute (latitude).



Range queries in range trees

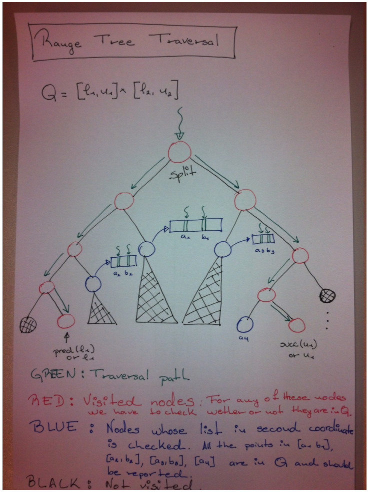
- To answer range queries in one-dimensional range trees we have to perform two binary searches over the list, in order to find the smallest and the greatest records that fall inside the range query.
- All the points in the array that lie between these two positions fall inside the range query and must be reported.

Range queries in range trees

Range searches in two-dimensional range trees are described recursively as follows.

- Each node in the data structure represents a range in the first dimension going from the smaller first attribute contained in the subtree to the greatest.
- When visiting a node, we compare the range of the first attribute of the query against the range of the node.
- If the range of the node is entirely within the range of the query, then we search the sorted list of that node for all the points satisfying the query, and we list the points found.
- Otherwise, we compare the range of the first attribute of the query to the discriminant of that node.
- If the range is entirely below (above) the discriminant, then we recursively visit the left (right) subtree; and if it overlaps the discriminant, then, we visit both subtrees recursively.

Two-dimensional range search in range trees I



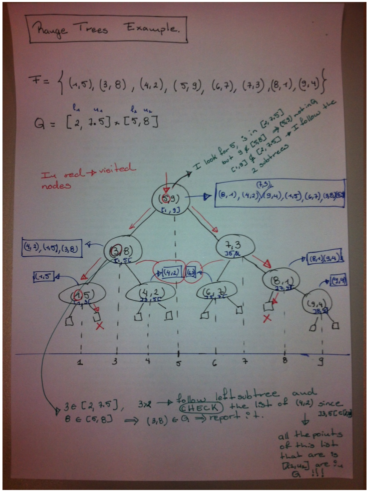
GREEN : Traversal path

RED : Visited nodes : For any of these nodes we have to check wether or not they are in Q.

BLUE : Nodes whose list in second coordinate is checked. All the points in $[a_1 b_1]$, $[a_2, b_2]$, $[a_3, b_3]$, $[a_4]$ are in Q and should be reported.

BLACK : Not visited.

Two-dimensional range search in range trees II



$F = \{(1,5), (3,8), (4,2), (5,9), (6,7), (7,3), (8,1), (9,4)\}$

$Q = [\overset{l_1}{2}, \overset{u_1}{7.5}] \times [\overset{l_2}{5}, \overset{u_2}{8}]$

In red → visited nodes

I look for 5, 5 is in [2, 7.5] but 9 ∉ [5,8] ⇒ (5,9) not in Q
[1,9] ⊄ [2, 7.5] → I follow the 2 subtrees

3 ∈ [2, 7.5], 3>2 → follow left subtree and CHECK the list of (4,2) since]3,5[⊂ [2,7.5]
8 ∈ [5, 8] ⟹ (3,8) ∈ Q ⟹ report it.

↓

all the points of this list that are in $[l_2, u_2]$ are in Q !!!

Higher Dimensional Range Trees

- We constructed a two-dimensional range tree by building a tree of one-dimensional structures. We can perform essentially the same operation to obtain a three-dimensional structure; we build a tree containing two-dimensional structures in the nodes.
- This process can be continued to higher dimensions.
- The result is a K-dimensional range tree with $\Theta(n \log^{K-1} n)$ storage, $\Theta(n \log^{K-1} n)$ construction time and $\Theta(\log^{K} n)$ range search cost [3].

Fractional Cascading

Idea = Add pointers!!!!

Performance of multidimensional data structures

Structure	Construction	Storage	Range Querie	Nearest Neighbor
List	$\Theta(Kn)$	$\Theta(Kn)$	$\Theta(Kn)$	$\Theta(Kn)$
Projection	$\Theta(Kn \log n)$	$\Theta(Kn)$	$\Theta(R + n^{1-\frac{1}{K}})$ (av.)	$\Theta(Kn)$
Cell	$\Theta(n)$	$\Theta(n)$	$\Theta(2^K F)$ (av.)	$\Theta(2^K F)$ (av.)
K-d tree	$\Theta(n \log n)$	$\Theta(n)$	$\Theta(R + n^{1-\frac{s}{K}+\theta(\frac{s}{K})})$ (av.)	$\Theta\left(n^\rho + \log n\right)$ (av.)
quad tree	$\Theta(n \log n)$	$\Theta(n)$	$\Theta(R + n^{1-\frac{s}{K}+\theta(\frac{s}{K})})$ (av.)	$\Theta\left(n^\rho + \log n\right)$ (av.)
range tree	$\Theta(n \log^{K-1} n)$	$\Theta(n \log^{K-1} n)$	$\Theta(\log^K n)$	

F denotes the average number of records per cell, R is the number of points within the range, and av. indicates average cost.

Random Relaxed K-d Trees

Definition

We say that a relaxed K-d tree of size n is *random* if the $n!^K \cdot K^n$ possible configurations of input file and discriminant sequence are equiprobable.

The Random Model for the Analysis of Partial Match

The assumptions for the analysis are:

- The relaxed K-d tree is random.
- The query is random: it is a multidimensional point randomly generated from the same distribution as that of the points in the tree, with an arbitrary specification pattern.

The Recurrence of Partial Match Searches

Following the previous random model at each node:

- With probability $\frac{s}{K}$ the discriminant will be specified in the query and the algorithm will follow one of the subtrees.
- With probability $\frac{K-s}{K}$ the algorithm will follow the two subtrees.
- Hence, the cost $M(T)$ of a Partial Match Search in a relaxed K-d tree T of size n with left subtree L of size ℓ and right subtree R is:

$$M(T \mid |L| = \ell\ ) = 1 + \frac{s}{K}\left(\frac{\ell+1}{n+1}M(L) + \frac{n-\ell}{n+1}M(R)\right) + \frac{K-s}{K}(M(L) + M(R)).$$

The Expected Cost of Partial Match

Theorem

The expected cost M_n (measured as the number of comparisons) of a partial match query with s out of K attributes specified in a random relaxed K-d tree of size n is

$$M_n = \beta n^{\alpha} + \mathcal{O}(1), \text{ where}$$

$$\begin{aligned} \alpha &= \alpha(s/K) = 1 - \frac{s}{K} + \phi(s/K) \\ \beta &= \beta(s/K) = \frac{\Gamma(2\alpha+1)}{(1-s/K)(\alpha+1)\Gamma^3(\alpha+1)} \end{aligned}$$

with $\phi(x) = \sqrt{9-8x}/2 + x - 3/2$ and $\Gamma(x)$ the Euler's Gamma function.

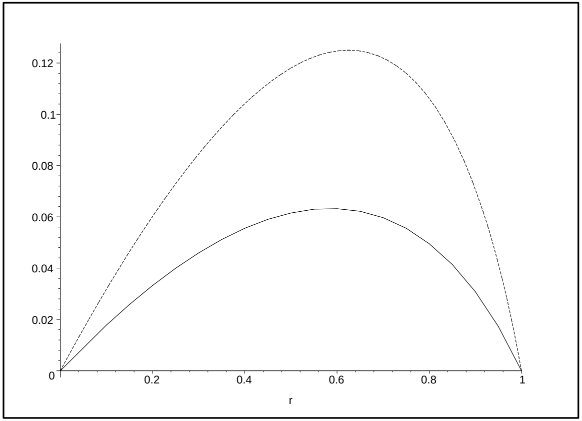
Solving the Recurrence of Partial Match Searches

In order to get the cost of partial match searches we follow the next steps:

- Take averages for all possible values of ℓ in the cost equation.
- Simplify by taking symmetries in the resulting recurrence.
- Translate the recurrence into a hypergeometric differential equation on the corresponding generating function.
- Solve the differential equation and obtain the generating function of the average cost of partial match.
- Use transfer lemmas to extract the coefficients of the average cost of partial match.

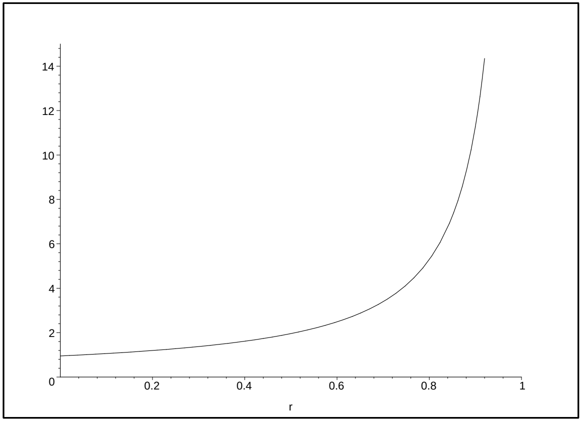
Exponent in the Average Cost of Partial Match Queries

Excess of the exponent α with respect to $1 - s/K$.



Constant in the Average Cost of Partial Match Queries

Plot of β.



Comparison with standard K-d trees

- The α coefficient for standard K-d trees is slightly smaller, but the analysis is more complicated since it involves the solution of a system of differential equations, one for each level of the tree (depending on the discriminant).
- The β coefficient for standard K-d trees is dependent on the query's specification pattern.
- Analysis: Ph. Flajolet and C. Puech (1986), H. Hwang (2003).

References I

J. L. Bentley.
Decomposable searching problems.
Information Processing Letters, 8(5):133–136, 1979.

J. L. Bentley and J. H. Friedman.
Data structures for range searching.
ACM Computing surveys, 11(4):397–409, 1979.

J.L. Bentley and J. H. Friedman.
Data structures for range searching.
ACM Computing Surveys, 11(4):397–409, 1979.

References II

W. A. Burkhard.
Hashing and trie algorithms for partial match retrieval.
ACM Transactions on Database Systems, 1(2):175–187, 1976.

L. Devroye.
Branching processes in the analysis of the height of trees.
Acta Informatica, 24:277–298, 1987.

L. Devroye and L. Laforest.
An analysis of random d-dimensional quadtrees.
SIAM Journal on Computing, 19(5):821–832, 1990.

R. A. Finkel and J. L. Bentley.
Quad trees: a data structure for retrieval on composite key.
Acta Informatica, 4(1):1–9, 1974.

References III

Ph. Flajolet, G. Gonnet, C. Puech, and J. M. Robson.
Analytic variations on quad trees.
Algorithmica, 10:473–500, 1993.

Ph. Flajolet and T. Lafforgue.
Search costs in quad trees and singularity perturbation analysis.
Discrete and Computational Geometry, 12(4):151–175, 1993.

M. Friedman, F. Baskett, and L. J. Shustek.
An algorithm for finding nearest neighbors.
IEEE Transactions on Computing, C–24(10):1000–1006, 1975.

References IV

D. E. Knuth.
The Art of Computer Programming: Sorting and Searching, volume 3.
Addison–Wesley, 2nd edition, 1998.

J. Nievergelt, H. Hinterberger, and K. C. Sevcik.
The grid file: An adaptable symmetric multikey file structure.
ACM Transactions on Database Systems, 1(9):38–71, 1984.

M. Regnier.
Analysis of the grid file algorithms.
BIT, 25(2):335–357, 1985.

References V

R. L. Rivest.
Partial-match retrieval algorithms.
SIAM Journal on Computing, 5(1):19–50, 1976.

H. Samet.
Deletion in two-dimensional quad-trees.
Communications of the ACM, 23(12):703–710, 1980.

H. Samet.
The Design and Analysis of Spatial Data Structures.
Addison-Wesley, 1990.

J. L. Bentley.
Decomposable searching problems.
Information Processing Letters, 8(5):133–136, 1979.

J. L. Bentley and J. H. Friedman.
Data structures for range searching.
ACM Computing surveys, 11(4):397–409, 1979.

J.L. Bentley and J. H. Friedman.
Data structures for range searching.
ACM Computing Surveys, 11(4):397–409, 1979.

W. A. Burkhard.
Hashing and trie algorithms for partial match retrieval.
ACM Transactions on Database Systems, 1(2):175–187, 1976.

L. Devroye.

Branching processes in the analysis of the height of trees.
Acta Informatica, 24:277–298, 1987.

L. Devroye and L. Laforest.
An analysis of random d-dimensional quadtrees.
SIAM Journal on Computing, 19(5):821–832, 1990.

R. A. Finkel and J. L. Bentley.
Quad trees: a data structure for retrieval on composite key.
Acta Informatica, 4(1):1–9, 1974.

Ph. Flajolet, G. Gonnet, C. Puech, and J. M. Robson.
Analytic variations on quad trees.
Algorithmica, 10:473–500, 1993.

Ph. Flajolet and T. Lafforgue.
Search costs in quad trees and singularity perturbation analysis.

Discrete and Computational Geometry, 12(4):151–175, 1993.

M. Friedman, F. Baskett, and L. J. Shustek.
An algorithm for finding nearest neighbors.
IEEE Transactions on Computing, C–24(10):1000–1006, 1975.

D. E. Knuth.
The Art of Computer Programming: Sorting and Searching, volume 3.
Addison–Wesley, 2nd edition, 1998.

J. Nievergelt, H. Hinterberger, and K. C. Sevcik.
The grid file: An adaptable symmetric multikey file structure.
ACM Transactions on Database Systems, 1(9):38–71, 1984.

M. Regnier.
Analysis of the grid file algorithms.
BIT, 25(2):335–357, 1985.

R. L. Rivest.
Partial-match retrieval algorithms.
SIAM Journal on Computing, 5(1):19–50, 1976.

H. Samet.
Deletion in two-dimensional quad-trees.
Communications of the ACM, 23(12):703–710, 1980.

H. Samet.
The Design and Analysis of Spatial Data Structures.
Addison-Wesley, 1990.