## Chapter 3.6 Tree Structure of Multi-Dimension Data

***Instructions:***

Considering range search and the nearest neighbor search of multi-dimensional data, there are four types, which includes:

* Multi-key index
* kd – tree
* quad tree
* r – tree

The former three are used in the point collection while r – tree is used to present the range collection and is used to present the point collection.

### Chapter 3.6.1 Multi-key index

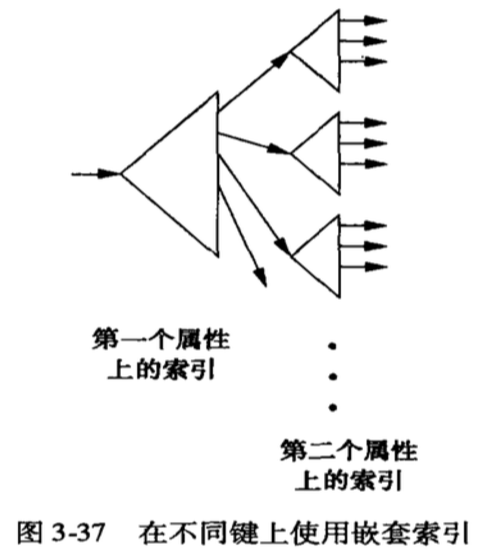
***Instructions:***

Several properties are used to present the dimension of data points, the range search and the nearest neighbor search should be supported. Multi-key index is a tree, points of each level is indexes of one property.

***Principles:***

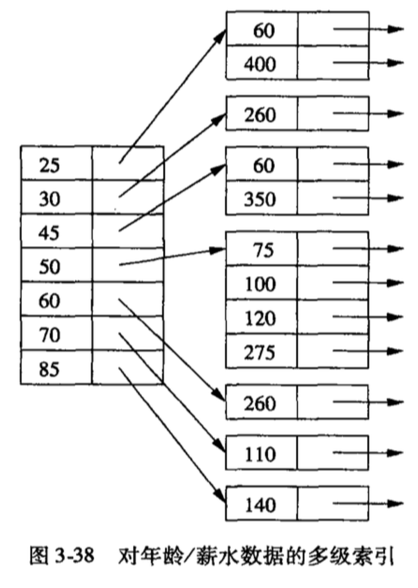
Two properties’ situation:

* The root is the first index of attribute among two attributes. The root can be any type, such as B-tree or hash list. (This means that each index in this structure can be a type of structure, either be a B-tree or hash list.)
* The index of each key relates to the pointer of another index. The first attribute should be V, while the second attribute can be a random value.



***Examples:***

As the image below shown, the first attribute is age while the second attribute is salary. The root is the index of age, while in the right, seven indexes are supported to access other indexes. To make it clear, the first index is the root index, including age (25, 30, 45, 50, 60, 70, 85). The second type of index is the salary indexes, including salary (60k, 400k), (260k), (60k, 350k), (75k, 100k, 120k, 275k), (260k), (110k), (140k), totally 7 indexes.



From image above, the root contains 7 indexes which represents age and in the right, salary.

* Age = 25, then it relates to two salary pointers, including 60k and 400k.
* Age = 30, then it relates to one salary pointer, 260k. (one key-value pair index)
* Age = 45, then it relates to two salary pointers, including 60k and 350k.
* Age = 50, then it relates to four salary pointers, including 75k, 100k, 120k and 275k.
* Age = 60, then it relates to one salary pointer, 260k. (one key-value pair index)
* Age = 70, then it relates to one salary pointer, 110k. (one key-value pair index)
* Age = 85, then it relates to one salary pointer, 140k. (one key-value pair index)

### Chapter 3.6.2 The performance of multi-key index

***Principles:***

The performance of multi-key index, mainly considering two properties.

* ***Partial matching search:***
* Assign the first attribute, then search one of child index through root index and this index will lead to the wanted point. The process could be very efficient and effective.

*(This place can be misleading, since each index of multi-key index is a structure, as introduced before, it can be a hash list or Btree structure.)*

* If the first attribute is unknown, then search every child index, this is a time-consuming procedure.
* **Ranging search:**
* If a single index does support range searching, such as B-tree or index sequence file, then multi – key index for ranging search works great.
* Use root index and the range of first attribute can find all sub-indexes that including all answers. After that use second attribute to search each child – index.
* **Nearest ranging search:**
* This has been discussed before, just as the chapter 3.4.3.

### Chapter 3.6.3 kd – tree

***Instructions:***

kd – tree (k – dimensional searching tree) is a main memory data structure which popularizes binary searching tree to multi-dimensional data structure.

***Principles:***

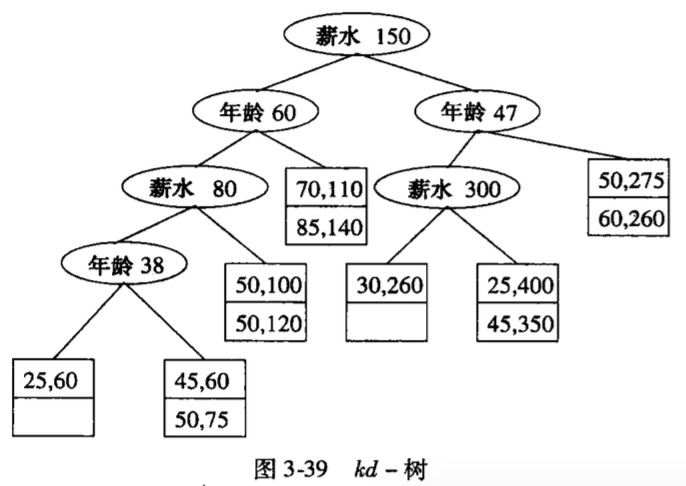
kd – tree is a binary tree, the inner point has correlated attribute a and value V, it separates two parts of the data point set: the part in which value a is less than V ( < V ) and in which value a is bigger than or equal to V ( >= V ).

*(The attribute in all dimensions alternates with each other, so in different tree level, the attributes are different.)*

In the normal kd – tree, data points are stored in the nodes, which are like stored in the binary search tree.

* Every inner node has one attribute, the divided value on this attribute, and the pointers which are used to point to left and right tree.
* Leaf node is the block structure, there are stored as much as possible records in the block space.

***Examples:***



The kd – tree includes 12 nodes, and in each block only saves two records. The contents are saved in leaf nodes. The inner nodes are oval. *Three mainly principles:*

1. Root level is using salary property to split.

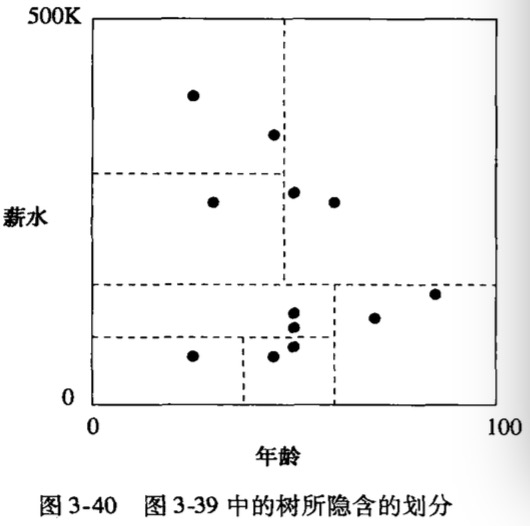
* In left part are all records of which salary < 150k.
* In right part are all records of which salary >= 150k.

1. The second level is using age property to split.

* In the left – child tree all records are in the range of age < 60 and salary < 150k.
* In the right – child tree all records are in the range of age >= 60 and salary >= 150k.

1. The inner nodes are separating the point space into the leaf nodes.

* The horizontal line salary = 150k is used to separate the root node. The upper part is the part in which salary is larger than 150k.
* The vertical line age = 47 is used to separate the second level right – child tree. The lower part is the part in which the salary is less than 150k.
* The vertical line age = 60 is used to separate the second level left – child tree which we can tell from the picture.



### Chapter 3.6.4 Operations on kd – tree

***Instructions:***

Finding a tuple in which values in each dimension are given is just like dealing them in the binary tree. We decide the flow of inner node and then it will lead us to the block of single leaf node.

***Principles:***

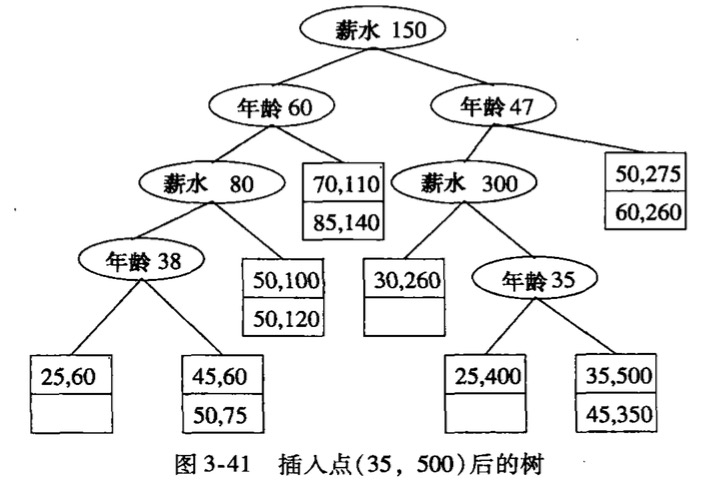
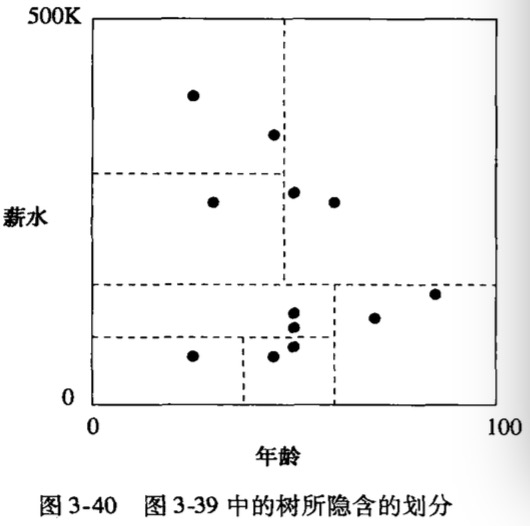
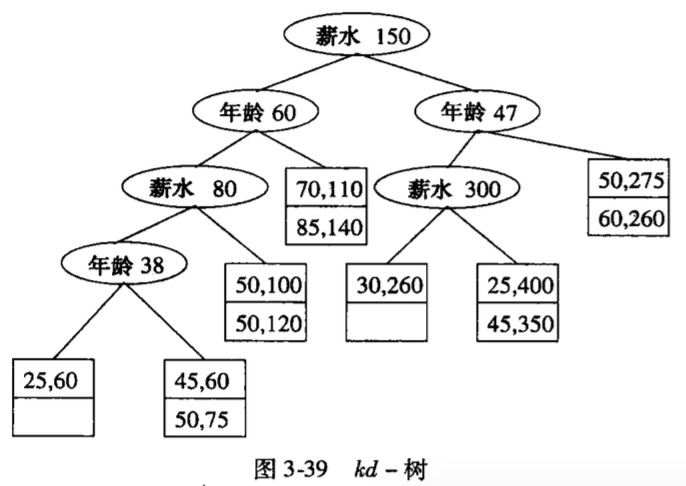
1. Finding a leaf node to insert.

* If there does exist block space in the leaf node, then store new data in empty space;
* If there doesn’t exist block space, then separate the leaf node content according to the attribute property.

1. Create a new inner node.

* The leaf node is separated into two new blocks, and give the inner node a division value.

***Example:***



Customer (age = 35, salary = $500)

1. Find a block to insert:

* The root node (salary = $150k), $500k > $150k, go to the right child tree.
* The second level, node age = 47, 35 < 47, go to the left child tree.
* The third level, node salary = $300k, $500k > $300k, go to the right child tree.
* In the right block, exists two nodes, one node (age = 25, salary = $400k) and another node (age = 45, salary = $350k). The node (age = 35, salary = $500k) should be inserted into this block.

1. Insert node into kd – tree:

* Using age = 35 to divide the block including two nodes (age = 25, salary = $400k) and (age = 45, salary = $350k)
* In left – child node, only exists one node (age = 45, salary = $350k)
* In the right – child node, only exists two nodes (age = 25, salary = $400k) and (age = 35, salary = $500k)

***Key Thinking:***

* *Partial Matching Search:*
* Given one value of attribute, and if the tree level is the current attribute, then choose one direction to proceed according to the value.
* Given one value of unknown attribute, then considering two sub-nodes.
* *Example:*

If searching one node with age equals to 50, then considering two sub-nodes of root, because the root is divided salary. But, in the left – node of root, just can go left; In the right – node of root, just can go right.

* *Ranging Search:*
* *Example: search age (35 - 55) and salary ($100k, $200k)*
* In root, salary range exceeds $150k, so two child trees are all needed to be considered.
* In left child tree, salary range ($100k, $150k) totally belongs to it. Proceeds to left child tree, age range (35, 55) is totally belongs to the left tree.
* In right child tree, salary range ($100k, $150k) is bigger than $80k. At last, find two leaf nodes, (50, $100k) and (50, 120k).
* Back to the first level from root level, then age = 47 tells us to check two child trees.
* In the level, salary equals to $300k, then proceeds to left, find the point (30, $260k), it beyond the range.
* In the level, age equals to 47, then proceeds to right, two nodes (50, $275k) and (60, $260k) are all beyond range.

### Chapter 3.6.5 Make kd – tree adapt to be an auxiliary storage

Assuming that a kd – tree includes n leaf nodes to store file, then average path length from root to leaf node equals to .

1. *Multiple branches of Inner Node -> Solve the Long Road Route*

The inner node of kd – tree has multiple key – pointer pairs, which looks like B – tree. The attribute A can be divided into range of n + 1 child trees. If there are n + 1 pointers, then child tree can be reached which just include attribute A in the appropriate range.

1. *Store inner node to Block -> Solve un-used Space*

In order to decrease the number of blocks for traversing, save all leaf nodes of several levels into one block. After using this method, once search the block of this node, then must using other nodes in the block, and save disk I/O.

### Chapter 3.6.6 Quad Tree

***Principles:***

1. In a quad tree, every inner node presents a square area in a two – dimension space or k – dimension cube of a k – dimension space. In current chapter, mainly considering the two – dimension scenario.
2. If the number of points in a square is less than the limit number of points in a block space, then treat this square as a tree leaf, and this tree leaf will be decided by it’s block which is used to store it’s points.
3. If too much points exist in the square so that one block can not save, then the square will be regarded as the inner node, and the child nodes will be corresponded to its four quadrants.

***Example:***

In the picture, salary range is limited during ($0k, $400k). Assuming that each block is only used to save two records.

* For quadrant and sub-node of nodes, using compass labeling method. Each inner pointer points the central coordinate of the area.
* There exists twelve points in the whole space, and in one block, there is only allowed to save two points, so the space needs to be divided into quadrant.
* For southwest and northeast, there only exist two nodes. So one leaf node can be used to represent and do not need to be further split.
* For southeast and northwest, there are more than two points in the left quadrants. They need to be divided into sub-quadrants, then in each divided quadrant remains two or less than two points, and no more division are needed.
* Since in the inner node of k – dimensional quadrant tree, there exists two to k power sub-node, so there exists the range k that could put nodes into the block.
* The situation can not be ensured in which central coordinate can be used to divide the area evenly so that in remaining block, there exists only two points.
* If there exists a lot of empty pointers when dimension number is too big, then just keep all non – empty pointers and the signal used to present the quadrant, this can save us a large of time.



### Chapter 3.6.7 R – Tree

***Principles:***

R – tree is a data structure which uses some basic B – Tree features to deal with multi – dimensional data. Recall that the node of B – tree has a collection of keys, then these keys separate line into segments. Then along the line, the point only belongs to one segment. So it will be easier to reach the point.



***Definition:***

* Data Area: R – tree that presents the data consisting of two – dimension or higher dimension, is called data area.
* One inner node corresponds to one inner area, or just ‘area’. Principally, the area can be any shape, although it is usually rectangular or other simple shape.
* In R – tree, there exists sub – area in the *key* location of R – tree node, and it is used to represent sub – node of node.

***Example:***

The picture below shows one R – tree node which can relates to large rectangular. Four dotted - line rectangulars are used to present four areas which relate to the original big one.

*Pay attention:* As long as including all data areas among the big aone into some small area, then it counts.



### Chapter 3.6.8 Operations on R – Tree

***Scenario:***

For the “where – am - I” typical search, R – Tree is useful. This type of search points to a pointer p and ask for data area where the pointer p locates.

***Principle:***

1. Searching:

* Start from the root node, it relates all area. Through searching sub – area of the root, make sure which sub – area has the related pointer P. Notice that there may has 0, 1 or several such areas.
* If there has no area, then the pointer P doesn’t exist in any data area, done; If at least one inner area includes pointer P, then must search pointer P in each area recursively.
* When we find one or multi – area that includes pointer P, then make sure we find the real data area, or the whole records of every data area, or the pointers point to the record.

1. Insertion:

* Find an appropriate sub – area from root. If there exist multi – areas, then just choose one, enter into the corresponding sub – area and repeating the finding process; If there has no such area, then just

Example:

Example: