Design And Implementation An Iraqi Cities Database Using K-D TREE

Makia k. Hamad **Mahmood A. Othman**
Department of Computer Science / College of science / University of Baghdad

Abstract

This research include design and implementation of an Iraqi cities database using spatial data structure for storing data in two or more dimension called k-d tree .The proposed system should allow records to be inserted, deleted and searched by name or coordinate. All the programming of the proposed system written using Delphi ver. 7 and performed on personal computer (Intel core i3).

تصميم وبناء قاعدة بيانات ضمن خارطة العراق باستخدام (k-d tree)

مكية كاظم حمد **محمود احمد عثمان**
قسم علوم الحاسبات/ كلية العلوم / جامعة بغداد

الخلاصة

يـهدف النظام المقترح لتصميم وبنـاء قاعدة بيانات ضمن خارطة العراق باستخدام نوع خاص من هياكل البيانات والمسماة (k-d tree) الخاصة بخزن المعلومات ذات البعدين أو أكثر . يوفر النظام المقترح إمكانية الأضافه والحذف للقيود بالأضافه إلى البحث باستخدام الاسم أو الإحداثيات للمدينـة .

1. Introduction

A multidimensional search key presents different concepts. Imagine that we have a database of city records, where each city has a name and an xy-coordinate. A BST(binary search tree) provides good performance for searches on city name, which is a one-dimensional key. Separate BSTs could be used to index the x-and y-coordinates. This would allow us to insert and delete cities, and locate them by name or by one coordinate. However, search on one of the two coordinates is not a natural way to view search in a two-dimensional space. Another option is to combine the xy-coordinates into a single key, say by concatenating the two coordinates, and index cities by the resulting key in a BST. That would allow search by coordinate, but would not allow for efficient two-dimensional range queries such as searching for all cities within a given distance of a specified point.

The problem is that the BST only works well for one-dimensional keys, while a coordinate is a two-dimensional key where neither dimension is more important than the other.

Multidimensional range queries are the defining feature of a spatial application. Because a coordinate gives a position in space, it is called a spatial attribute. To implement spatial applications efficiently requires the use of spatial data structures. Spatial data structures store data objects organized by position and are an important class of data structures used in geographic information systems, computer graphics, robotics, and many other fields.

2. Spatial Data Structures

All of the search trees such as — BSTs, AVL trees, B-trees, and tries—are designed for searching on a one-dimensional key. A typical example is an integer key, whose one-dimensional range can be visualized as a number line. These various tree structures can be viewed as dividing this one dimensional number line into pieces.

Some databases require support for multiple keys, that is, records can be searched based on any one of several keys. Typically, each key has its own one dimensional index, and any given search query searches one of these independent indices as appropriate.[1]

3. K-D Tree For Multidimensional Keys Searching

A k-d tree (short for k-dimensional tree) is a space-partitioning data structure for organizing points in a k-dimensional space figure(1). K-d trees are a useful data structure for several applications, such as searches involving a multidimensional search key (e.g. range searches and nearest neighbor searches). K-d trees are a special case of binary space partitioning trees .[2]

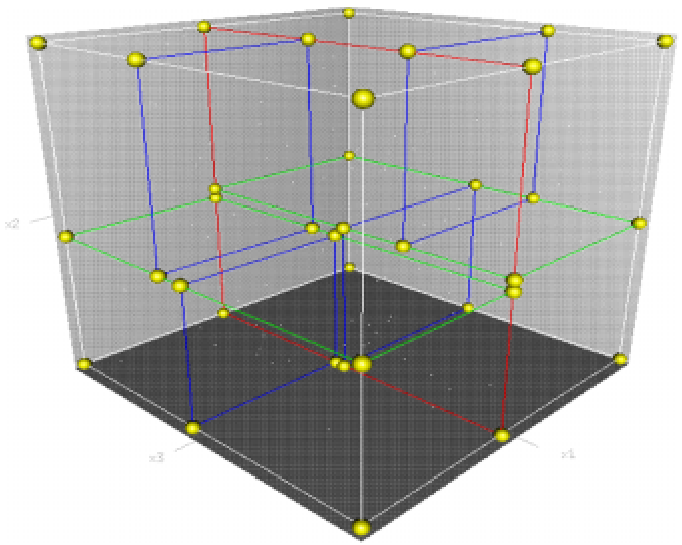


Figure (1). k-d tree(3 dimension)

3.1 Creating

The k-d tree is a modification to the BST that allows for efficient processing of multidimensional keys. The k-d tree differs from the BST in that each level of the k-d tree makes branching: decisions based on a particular search key for that level, called the discriminator. the discriminator can be at level i to be i mod k for k dimensions. For example, assume that we store data organized by : xy – coordinates , in this case, k is 2 (there are two coordinates), with the x-coordinate field arbitrarily designated key 0, and the y-coordinate field designated key 1.. At each level, the discriminator alternates between x and y. Thus; a node n at level 0 (the root) would have ,in its left subtree only nodes whose x values are less than N_x (since x is search key 0, and 0 mod 2= 0).The right subtree would contain nodes whose x values are greater than N_x. A node M at level would have in its left subtree only, nods whose y values are less than M_y. There is no restriction on the relative values of M_x and the x values of M's descendants, since branching decisions made at M are based solely on the y coordinate. Figure 2 shows an example of how a collection of two-dimensional points would be store in k-d tree.[1] [3]

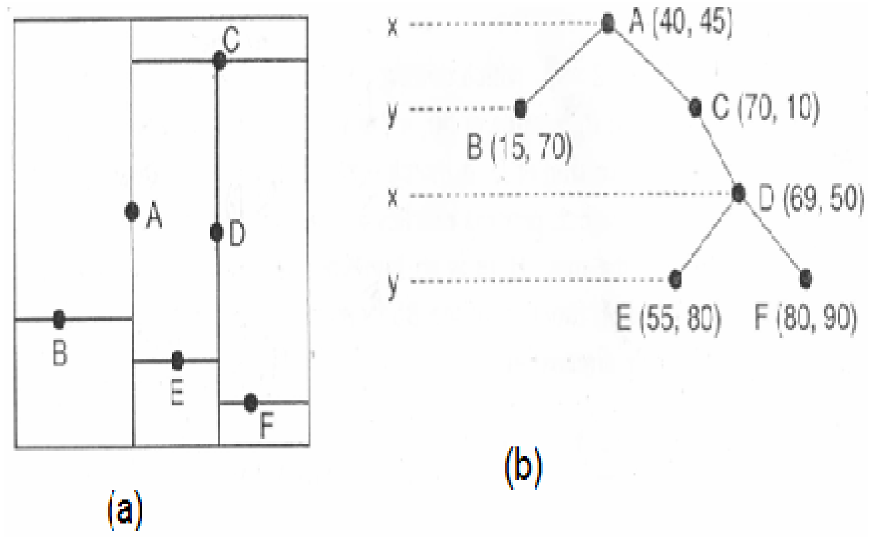


Figure (2).
Example of a k-d tree. (a) The k-d tree decomposition for a 100 x 100-unit region containing six data points. (b) The k-d tree.

In Figure (2) the region containing the points is (arbitrarily) restricted to a 100 x 100 square, and each internal node splits the search space. Each split is shown by a line, vertical for nodes with x discriminators and horizontal for nodes with y discriminators. The root node splits the space into two parts; its children further subdivide the space into smaller parts. The children's split lines do not cross the root's split line. Thus, each node in the k-d tree helps to decompose the space into rectangles that show the extent of where nodes may fall in the various subtrees .Searching a k-d tree for the record with a specified xy-coordinate is like searching a BST, except that each level of the k-d tree is associated with a particular discriminator .Consider searching the k-d tree for a record located at P : (69, 50).First compare P with the point stored at the root record A in Figure (2).[1][3]

If P matches the location of A, then the search is successful. In this example the positions do not match (A's location (40, 45) is not the same as (69, 50)), so the search must continue .The x value of A is compared with that of P to determine in which direction to branch. Since Ax's value of 40 is less than P's value of 69, we branch to the right subtree (all cities with x value greater than or equal to 40 are in the right subtree).

A_y does not affect the decision on which way to branch at this level. At the second level, P does not match record C's position, so another branch must be taken. However, at this level we branch, based on the relative y values of point P and record C (since 1 mod 2= I, which corresponds to the y-coordinate). Since C_y value of 10 is less than P_y value of 50, it branch to the right. At this point, P is compared against the position of D. A match is made and the search is successful .As with a BST, if the search process reaches a NULL pointer, then the search point is not contained in the tree.[1][3]

3.2 Insertion In K-D Tree

Inserting a new node into the k-d tree is similar to BST insertion. The k-d tree search procedure is followed until a NULL pointer is found, indicating the proper place to insert the new node. For example, inserting a record at location (10, 50) in the k-d tree of Fig 2 first requires a search to the node containing record B. At this point, the new record is inserted into B's left subtree.[4]

One adds a new point to a k-d tree in the same way as one adds an element to any other search tree. First, traverse the tree, starting from the root and moving to either the left or the right child depending on whether the point to be inserted is on the "left" or "right" side of the splitting plane. Once you get to the node under which the child should be located, add the new point as either the left or right child of the leaf node, again depending on which side of the node's splitting plane contains the new node.[4]

Adding points in this manner can cause the tree to become unbalanced, leading to decreased tree performance. The rate of tree performance degradation is dependent upon the spatial distribution of tree points being added, and the number of points added in relation to the tree size. If a tree becomes too unbalanced, it may need to be re-balanced to restore the performance of queries that rely on the tree balancing, such as nearest neighbor searching.[4]

3.3 Deleting From K-D Tree

Deleting a node from a k-d tree is similar to deleting from a BST, but slightly harder. As with deleting from a BST, the first step is to find the node (call it N) to be deleted. It is then necessary to find a descendant of N which can be used to replace N in the tree. If N has no children, then N is replaced with a NULL pointer. Note that if N has one child that in turn has children, we cannot simply assign N's parent to point to N's child as would be done in the BST. To do so would change the level of all nodes in the subtree; and thus the discriminator used for a search would also change. The result is that the- subtree would no longer be a k-d tree since a node's children might now violate the BST property for that discriminator.[4]

Similar to BST deletion, the record stored in N should be replaced either by ,the record in N's right subtree with the least value of N's discriminator, or by the record in N's left subtree with the greatest value for this discriminator. Assume that n was at an odd level and therefore y is the discriminator. N could then be replaced by the record in its right subtree with the least y value (call it y_{min}). The problem is that y_{min} is not necessarily the leftmost node, as it would be in the BST.[4]

Note that we can replace the node to be deleted with the least-valued node from the right subtree only if the right subtree exists. If it does not, then a suitable replacement must be found in the left subtree. Unfortunately, it is not satisfactory to replace l's record with the record having the greatest value for the discriminator in the left subtree, because this new value might be duplicated. If so, then we would have equal values for the discriminator in N's left subtree, which violates the ordering rules for the k-d tree. Fortunately, there is a simple solution to the problem. We first move the left subtree of node N to become the right subtree (i.e., we simply swap the values of N's left and right child pointers). At this point, we proceed with the normal deletion process, replacing the record of N to be deleted with the record containing the least value of the discriminator from what is now n's right subtree.[4]

3.4 **Search In K-D Tree**

Assume that we want to print out a list of all records within a certain distance d of a given point P. [5]

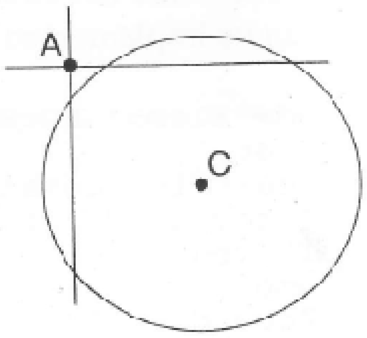


Figure (3). check the Euclidean distance between a record and the query point. It is possible for a record A to have x and y coordinates each within the query distance of the query point C, yet have A itself lie outside the query circle.

We will use Euclidean distance Figure (3). that is, point is defined to be within distance d of point N if .

$\sqrt{(Px - Nx)^2 + (Py - Ny)^2}$ >=d …………(1)

If the search process reaches a node whose key value for the discriminator is more than d above the corresponding value in the search key, then it is not possible that any record in the right subtree can be within distance d of the search key since all key values in that dimension are always too great. Similarly, if the current node's key value in the discriminator is d less than
that for the search key value, then no record in the left subtree can be within the radius. In such cases' the subtree in question need not be searched, potentially saving much time. In general the number of nodes that must be visited during a range query is linear on the number of data records that fall within the query circle.[1][5]

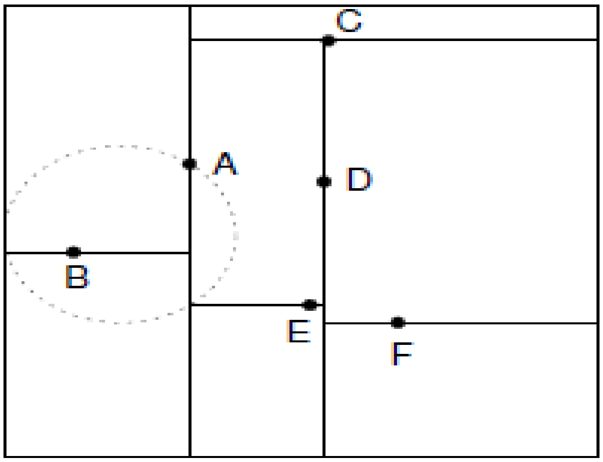


Figure (4) . search in k-d tree.

For example, assume that a search will be made to find all cities in the k-d tree of Figure (4) within 25 units of the point (25, 65). The search begins with the root node, which contains record A. Since (40, 45) is exactly 25 units from the search point , it should be reported. The search procedure then determines which branche of the tree to take. The search circle extends' to both the left and the right of a (vertical) dividing line, so both branches of the tree must be searched The left subtree is processed first, Here, record B is checked and found to fall within the search circle. Since the node storing B has no children processing of the left subtree is complete. processing of A's right subtree now begins. The coordinates of record c are checked and found not to fall within the circle . Thus, it should not be reported. However, it is possible that cities within c's subtrees could fall within the search circle even if C does not. As C is at level 1, the discriminator at this level is the y-coordinate .since 65-25>10, no record in c's left subtree (i,e'., records above c) could possibly be in the search circle. Thus, c's left subtree (if it had one) need not be searched. However, cities inc's right subtree could fall within the circle. Thus, search proceeds to the node containing record D. Again, D is outside the search circle. Since 25 + 25 < 69, no record in D's right subtree could be within the search circle. Thus, only D's left subtree need be searched. This leads to comparing record E's coordinates against the search circle. Record E falls outside the search circle, and processing is complete. So we see that we only search subtrees whose rectangles fall within the search circle figure (5), figure (6) .[1][5]

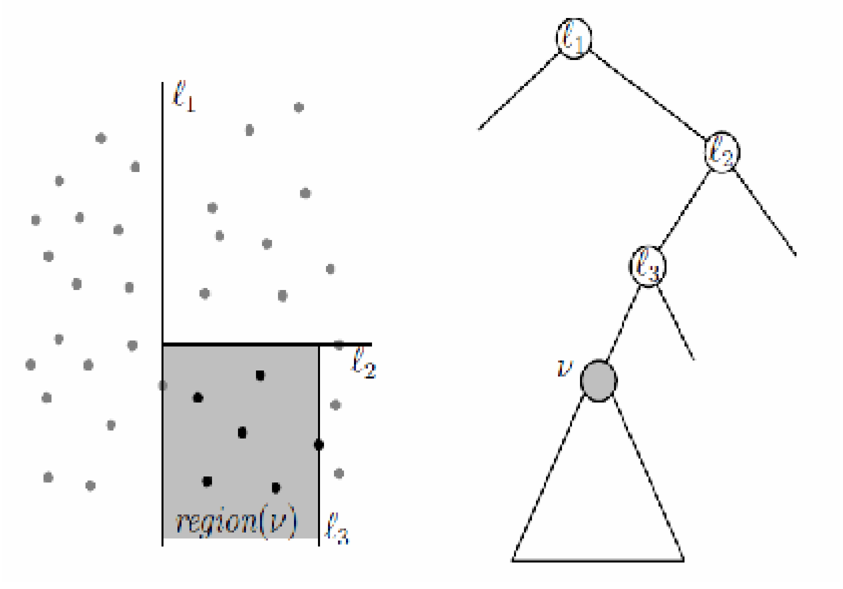


Figure (5). nearest neighbor searching.

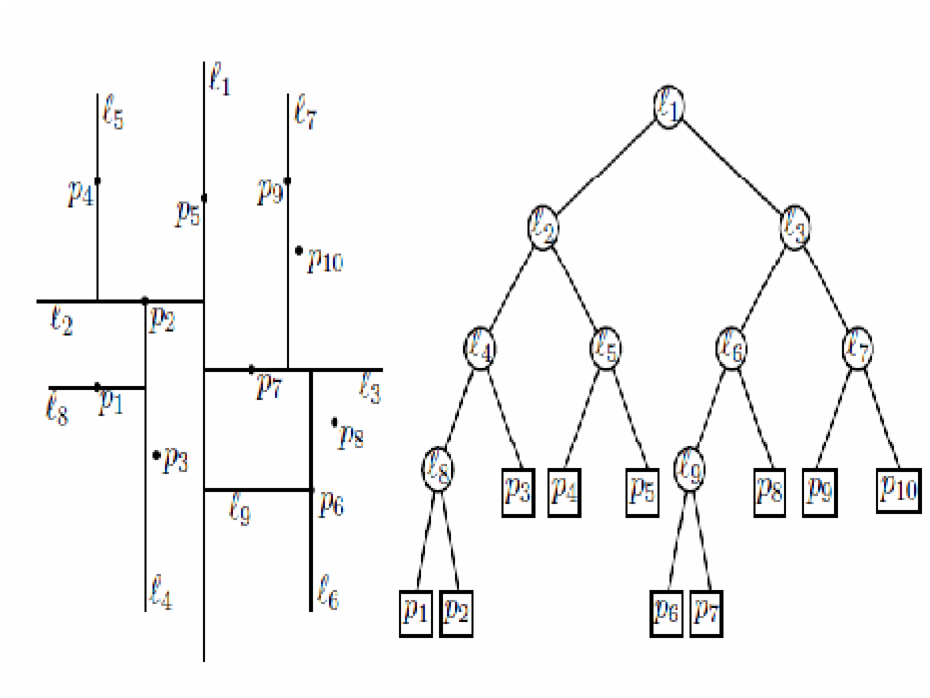


Figure (6). build k-d tree

4. Practical Part

This section include practical work step by step through this research. The purpose of this research is to implement the K-d tree search method to find the shortest path between any two points on any map stored in the database and the research supports a request to print all records within a given distance of a specified point .

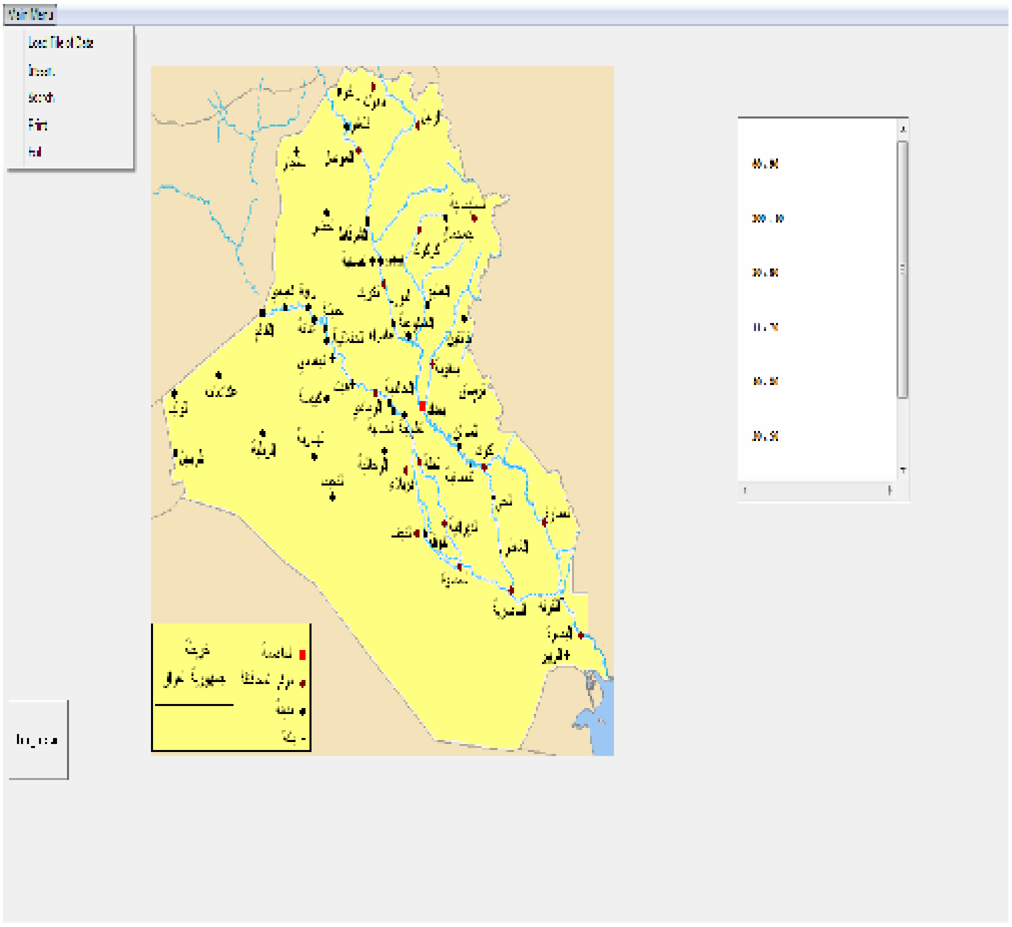


Figure (7). the main interface

Using Iraq map Figure (7) before describing the interface of the proposed system , we have to talk about the database that will be build before running the system. The database in our work contains records, each contains the name of the province or main city and the coordinates are expressed as integer x- and y-coordinates. Our database allows records to be inserted and searched by the coordinate. In our case, the database used is a text file that stores the coordinates of all the provinces and main cities of Iraq and our only root is Baghdad, and if we need to change the root, we have to design the database of that root (i.e Basrah).

The coordinates represent the location of the city on the map according to a fixed point in the map, that is (0,0), that represents the top left corner of the image (i.e map). the interface of the program contained more than one frame will be more friendly use , so we optimized the system to one frame as shown in Figure (8).

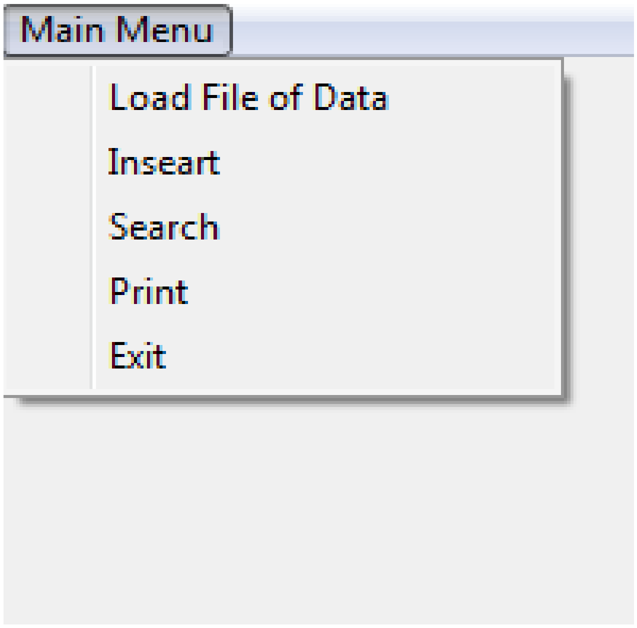


Figure **(8)** :main menu and its operation.

The proposed system is included the following operation

1. Load File Of Data

This function enables the user to load the database of the selected map (i.e all the records that contain the coordinates on the Iraqi map). Then it takes those coordinates and pass them to the insert subprogram build all the nodes of the K-d tree related to the selected map, In figure (9) Flow chart to build k-d tree.

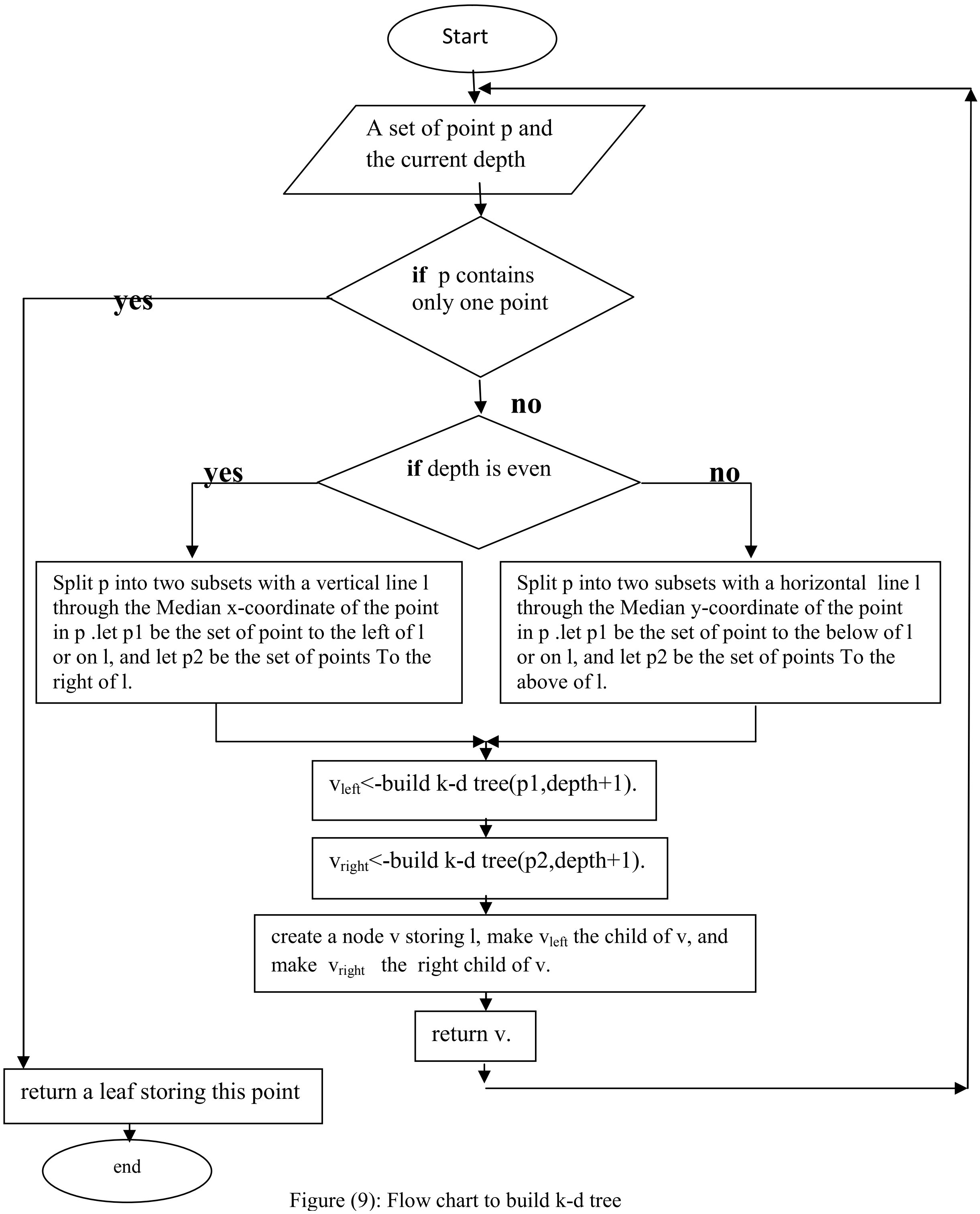

Start
A set of point p and the current depth
if p contains only one point
yes
no
if depth is even
yes
no
Split p into two subsets with a vertical line l through the Median x-coordinate of the point in p .let p1 be the set of point to the left of l or on l, and let p2 be the set of points To the right of l.
Split p into two subsets with a horizontal line l through the Median y-coordinate of the point in p .let p1 be the set of point to the below of l or on l, and let p2 be the set of points To the above of l.
v_left<-build k-d tree(p1,depth+1).
v_right<-build k-d tree(p2,depth+1).
create a node v storing l, make v_left the child of v, and make v_right the right child of v.
return v.
return a leaf storing this point
end

Figure (9): Flow chart to build k-d tree

2. Insert

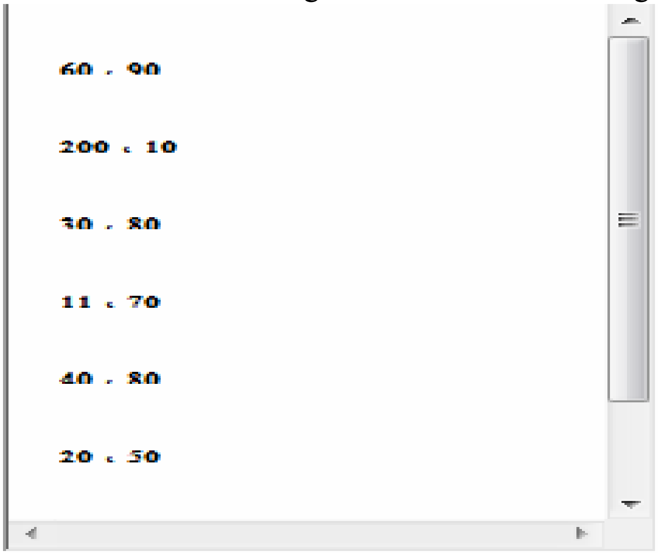
This function enables to insert a new city coordinate as a node in the K-d tree

3. Search

This function to find the shortest path between the root ,assumed in this case is Baghdad and any other city that exists in the database. On pressing this command, the system will ask to enter the (X,Y) coordinates through two input windows that will appear immediately and after entering those coordinates the system will check if those coordinates exists in the database, if so, a message will appear on the screen and a path will be drown on the map, highlighted in red, and if the coordinates do not exists in the database, the system will show an error message to the user.

4. Print

This function prints the coordinated of the K-d tree in an ascending order in the output window that exists on the right side of the interface figure (10).



```
60 , 90

200 , 10

30 , 80

11 , 70

40 , 80

20 , 50
```

Figure (10):output window.

5. Exit

This function terminates the whole project.

Conclusions

1. K-d tree are not suitable for efficiently finding the nearest neighbor in high dimensional spaces , that means if the dimension increase the efficiency is decreased.

2. Inserting a new point into a balance k-d tree take same time as removing a point from it .
3. Insert any new point to a k-d tree as a new leaf same as insertion in ordinary binary search tree.
4. Balancing a k-d tree requires care , because k-d tree are stored in multiple dimensions .the tree rotation technique can be used to balance them.
5. K-d tree can be use with many application such as indexing quickly and accurately in large collection of images .

References

[1]. Clifford A. Shaffer A Practical Introduction to Data Structures and Algorithm Analysis Third Edition (C++ Version) Department of Computer Science Virginia Tech Blacksburg, VA 24061January 19, 2010.

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

[3]. Cormen, Thomas H. ; Leiserson , Charles E.; Rivest, Ronald L.; Stein, Clifford, "Introduction To Algorithms", Third Edition, MIT Press,(2009).

[4]. Chandran, Sharat. Introduction to kd-trees. University of Maryland Department of Computer Science.

[5]. H. M. Kakde. Range searching using kd tree. pp. 1-12, 2005.