CMSC 420 PROGRAMMING ASSIGNMENT 1: A DATA STRUCTURE FOR POINT DATABASES*

Spring 1997

Abstract

In this assignment you are required to implement an information management system for handling multidimensional data similar to that used in database management systems. In such an environment the primary entities are points in a multidimensional space and the problem in which we are interested is how to manage a large collection of them. In the following, we describe a variant of the quadtree data structure that has been found to be useful for such a problem. Your task is to implement this data structure in such a way that a number of operations can be efficiently handled.

This assignment is divided into three parts. You may program in PASCAL C, or C++. For the first two parts, you must read the attached description of the problem and data structure. A detailed explanation of the assignment including the specification of the operations which you are to implement is found at the end of the description. After you have done this, you are to turn in a proposed implementation of the data structure using PASCAL's record (or C or C++'s structure) definition facility. One week later you must turn in a program for the command decoder (i.e., a scanner for the commands corresponding to the operations which are to be performed on the data structure). For the third part, you are to write a program to implement the data structure and the specified operations.

1 Background

To do this assignment, you will need to learn about Quadtrees. The material presented here gives a brief introduction. Please refer to [Samet, 1990] or other references for more details. Note that this material will not be covered in class until well *after* the assignment is due.

1.1 POINT QUADTREES

The point quadtree is a data structure for storing multidimensional point data developed by [Finkel and Bentley, 1974]. It should be easy for you to understand it because it is simply a multidimensional generalization of a binary search tree. In two dimensions each node has four subtrees corresponding to the directions NW, NE, SW, and SE. Each subtree is commonly referred to as a quadrant or subquadrant. For example, see Figure 2.4[1] where the correspondence of a quadtree of 8 nodes to the data of Figure 2.1 is presented. In our presentation we shall only discuss two-dimensional quadtrees although it should be clear that what we say can be easily generalized to more than two dimensions. For the point quadtree, the points of decomposition are the data points themselves (i.e., in Figure 2.4, Chicago at location (35,40) subdivides the two dimensional space into four rectangular regions). Note that the regions are usually not of equal size. Requiring the regions to be of equal size leads to a variant of the region quadtree of [Klinger, 1971]. This data structure was developed for representing homogeneous spatial data and is used in computer graphics, image processing, geographical information systems, pattern recognition, and other applications. Its use in computer graphics is rooted in the work of [Warnock, 1969] who applied it to hidden surface elimination. We shall not say more about the region quadtree. For more details on quadtrees, see [Samet, 1990].

Records are inserted into point quadtrees in a manner similar to that done for binary search trees. In essence, we search for the desired record based on its x and y coordinates. At each node a comparison is made by use of procedure **PTCOMPARE** and the appropriate subtree is chosen for the next test. Upon reaching the bottom of the tree (i.e., a null pointer is encountered), we have found the location where the record is to be inserted. For example, the tree in Figure 2.4 was built for the sequence Chicago, Mobile, Toronto, Buffalo, Denver, Omaha, Atlanta, and Miami. Figures 2.5a, 2.5b, 2.5c, and 2.5d show how the tree was constructed in an incremental fashion for Chicago, Mobile, Toronto, and Buffalo.

```
quadrant procedure PTCOMPARE(P,R);
/* Return the quadrant of the point quadtree rooted at node R in
   which node P belongs.  The record structure for the point quadtree
   has fields XCOORD and YCOORD (among others) that indicate the value
   of the corresponding $x$ and $y$ coordinates. */
begin
  value pointer node P,R;
  return(if XCOORD(P) < XCOORD(R) then
           if YCOORD(P) < YCOORD(R) then 'SW'
           else 'NW'
         else if YCOORD(P) < YCOORD(R) then 'SE'
         else 'NE');
end;
```

In order to cope with data points that lie directly on one of the quadrant lines emanating from a data point, say P, we adopt the following conventions. Quadrants NE and SE of P are closed with respect to the x coordinate, and quadrants NW and NE of P are closed with respect to the y coordinate. This means that quadrants NW and SW of P are open with respect to the x coordinate and quadrants SW and SE of P are open with respect to the y coordinate. Equivalently, we say that the lower and left boundaries of each block are closed while the upper and right boundaries of each block are open. For example, in Figure

[1] All figure numbers and page numbers refer to [Samet, 1990].

2.4, insertion of Memphis with coordinates (35,20) would lead to its placement somewhere in quadrant SE of the tree rooted at Chicago (i.e., at (35,40)).

Deletion of nodes from a point quadtree is not simple. [Finkel and Bentley, 1974] suggest that all nodes of the tree rooted at the deleted node must be reinserted - a sometimes expensive process. Nevertheless, you may (and most likely will want to!) use this process. In order to see the difficulties involved in finding a better solution, we note that ideally, we want to replace the deleted node (say A) with a node (say B) that leaves an empty shaded (i.e., hatched) region as in Figure 2.6. Unfortunately, this involves a considerable amount of search. In fact, it is not uncommon that such a node fails to exist (e.g., when deleting node A in Figure 2.7). [Samet, 1990] suggests an algorithm that proceeds in a manner analogous to the method for binary search trees. For example, for the binary search tree of Figure 2.8, when node A is to be deleted, it can be replaced by one of nodes D or G - i.e., the "closest" node in value. In the case of a quadtree, e.g., Figure 2.7, it is not clear which of the remaining nodes should replace A. This is because no definition of "closest" exists. We notice that no matter which of the nodes is chosen, some of the nodes will have to shift position. Therefore, in Figure 2.7, if L replaced A, then J would no longer occupy the NW quadrant with respect to L and thus J would need to be reinserted along with some of its subtrees in the quadrant rooted at F. Similarly, E and I would have to be reinserted along with some of their subtrees in the quadrant rooted at N. For more details about such a solution, see pp. 54-64 in [Samet, 1990].

The point quadtree is especially attractive in applications that involve search. A typical query is one that requests the determination of all nodes within a specified distance of a given data point - i.e., all cities within 50 miles of Washington, D.C. The efficiency of the point quadtree data structure lies in its role as a pruning device on the amount of search that is required. Thus many records will not need to be examined. For example, suppose that in the hypothetical database of Figure 2.1 we wish to find all cities within 8 units of a data point with coordinates (83,10). In such a case, there is no need to search the NW, NE, and SW quadrants of the root (i.e., Chicago with coordinates (35,40)). Thus, we can restrict our search to the SE quadrant of the tree rooted at Chicago. Similarly, there is no need to search the NW and SW quadrants of the tree rooted at Mobile (i.e., coordinates (50,10)).

As a further clarification of the amount of pruning of the search space that is achievable by use of the point quadtree, we make use of Figure 2.17. In particular, given the problem of finding all nodes within radius r of point A, use of the Figure indicates which quadrants need not be examined when the root of the search space, say R, is in one of the numbered regions. For example, if R is in region 9, then all but its NW quadrants must be searched. Similarly, if R is in region 7, then the subsequent search can be restricted to the NW and NE quadrants of R. Similar techniques can be used to search for data points in any connected figure. For example, [Finkel and Bentley, 1974] give algorithms for searching within a rectangular window of arbitrary size.

2 ASSIGNMENT

This assignment has three parts. It is to be programmed in PASCAL or C or C++. The first part is concerned with data structure selection. The second part requires the construction of a command decoder. The third part requires that you implement a given set of operations.

The first part is to be turned in at the beginning of class on Thursday, February 6th. It is worth 10 points. There will be ABSOLUTELY no late submissions accepted for this part of the assignment. At that time, you will be handed the correct data structure which you are to use. The second part of the assignment is to be turned in at the beginning of class on Thursday February 13th. It is worth 10 points. Again, there will be NO late submissions accepted for this part of the assignment. While doing parts one and two you are also to start thinking and coding the program necessary to implement the operations. This should be done in such a way that the data structure itself is a BLACK BOX. Thus you may want to specify your primitives in such a way that they are independent of the data structure finally chosen. You are strongly advised to begin implementing some of the operations. For example, you should implement an output routine so that you can see whether your program is working properly.

For the third part of the assignment, you are to write a PASCAL (or C or C++) program to implement the data structure and the specified operations. It is worth 60 points. In order to facilitate grading and your task, you are to use the data structure implementation that will be given to you in class when you turn in the first part of the assignment. This part will be due February 27th.

In order to facilitate your programming as well as lend some realism to your output, you can assume that the coordinate values of the points are integers that range between 0 and 999.

One class meeting date before the due date of each part of the project you will be informed of the availability of and name of the test data file which you are to use in exercising your program for grading purposes. You should also prepare your own test data.

2.1 Part 1: DATA STRUCTURE SELECTION

You are to select a data structure to implement the point quadtree. Turn in a definition in the form of a set of PASCAL (or C or C++) records. In doing this part of the assignment you should bear in mind the type of data that is being represented and the type of operations that will be performed on it. In order to ease your task, remember that the primitive entity is a point. We specify a point by giving the x and y coordinates of its endpoints. The rest of your task is to build on this entity adding any other information that is necessary to efficiently support the operations described in Section 2.3.

From the description of the operations you will see that a name is associated with each point. At times the operations are specified in terms of these names. Thus you will also need a mechanism to efficiently keep track of these names. It should be integrated with the rest of your data structures.

2.2 Part 2: COMMAND DECODER

You are to turn in a working command decoder written in PASCAL, C, or C++ for all the commands (including the optional ones) given in Section 2.3. This should take as input from the user a command name with arguments in parenthesis. It should dispatch to a function (to be written in Part 3) which actually applies that function to the arguments. You are not expected to do error recovery and can assume that the commands are syntactically correct. All commands will fit on one line. However, your data structure should make a provision for names to be of arbitrary lengths (i.e., character strings of arbitrary length). The names can be any combination of letters or digits (e.g., A, 1, 2A, B33, etc.). However, for your own safety you may wish to incorporate some primitive error handling. Test data for this part of the assignment will be found in a file specified by the TA.

2.3 Part 3: OPERATIONS

1. Initialize the quadtree. The command INIT_QUADTREE(WIDTH) is always the first command in the input stream. WIDTH determines the length of each side of the square are covered by the quadtree. Each side has the length 2^{WIDTH}. It also has the effect of starting with a fresh data set.

2. Generate a display of a $2^{\mathrm{WIDTH}} \times 2^{\mathrm{WIDTH}}$ square from the point quadtree. It is invoked by the command DISPLAY(). To draw the point quadtree, you are to use the drawing routines provided. An appendix to the project description covers their use, and the utilities SHOWQUAD and PRINTQUAD, that can be used to render the output of your programs on a screen or a printer. A dashed (broken) line should be used to draw quadrant lines, but the points should be solid filled circles. Point names should be output somewhere near the point. Make sure to avoid overwriting names that are already on the output.

3. List all the names in the data base in alphanumerical order. This means that letters come before digits in the collating sequence. Similarly, shorter identifiers precede longer ones. For example, a sorted list is A, AB, A3D, 3DA, 5. It is invoked by the command LIST_NAMES() and yields for each point its name, and the values of its x and y coordinates.

4. Create a point by specifying the values of its x and y coordinates and its name and add it to the point quadtree. It is invoked by the command ADD(N,PX,PY) where N is the name to be associated with the point while PX and PY are the x and y coordinates respectively of the point. Output an appropriate message indicating that the point has been created as well as its name and coordinates.

5. Search for the location of a point whose name is specified. It is invoked by the command SEARCH(N) where is N is the name. Output the values of the x and y coordinates of the point associated with the name as well as the name itself. An appropriate message is to be output if the name is not present in the database.

6. Remove a point from the point quadtree. It is invoked by the command REMOVE(N) where N is the name associated with the point. Output the values of the x and y coordinates of the point associated with the name as well as the name itself. An appropriate message is to be output if the name is not present in the database.

7. List all the names associated with points on the map that are within a given radius of the point associated with a given name. It is invoked by the command SEARCH_CIRCLE(N,R) where N is the name associated with the point and R is

the radius. Output the values of the x and y coordinates of the points that are found as well as the names associated with them. Also, output the name and location of the point, as well as the radius, around which the search is centered. An appropriate message is to be output if the name is not present in the database.

8. List all the names associated with points on the map that are within a given box that surrounds the point associated with a given name. It is invoked by the command SEARCH_BOX(N,H,V) where N is the name associated with the point and H and V are the horizontal and vertical extents of the box. This means that the box is of width $2 \times$ H and length $2 \times$ V and centered at the point associated with N. Output the values of the x and y coordinates of the points that are found as well as the names associated with them. Also, output the name and location of the point, as well as the radius, around which the search is centered. An appropriate message is to be output if the name is not present in the database.

9. Find the nearest K points to a given point. It is invoked by the command NEAREST_NEIGHBOR(K,N) where K is the number of points that are desired and N is the name associated with the point about which the search is to proceed, Output the value of the x and y coordinates of the point that are found as well as the name associated with it. An appropriate message is to be output if the name is not present in the database.

10. Find the nearest K points to a given point and output them in the order in which they were found. In other words, you are to find them in an incremental fashion so that you output the nearest point once you have found it. Then you look for the second nearest point and output it, the third nearest point and output, etc. You are not to find all K points at once and then output them. It is invoked by the command INCREMENTAL_NEAREST_NEIGHBOR(K,N) where K is the number of points that are desired and N is the name associated with the point about which the search is to proceed. Output the value of the x and y coordinates of the point that are found as well as the name associated with it. An appropriate message is to be output if the name is not present in the database.

References

[Finkel and Bentley, 1974] R.A. Finkel and J.L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 1:1–9, 1974.

[Klinger, 1971] A. Klinger. Patterns and search statistic. In J.S. Rustagi, editor, *Optimizing Methods in Statistics*, pages 303–337. Academic Press, 1971.

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

[Warnock, 1969] J.L. Warnock. A hidden surface algorithm for computer generated half tone pictures. TRAINS Technical Note 4-15, Computer Science Dept. University of Utah, 1969.