UNIT 37 - QUADTREE ALGORITHMS AND SPATIAL INDEXES

This unit is very long and deals with more advanced algorithms. Depending on the abilities and interests of your students, you may want to omit the third and fourth algorithms included or consider providing this as extra handouts. Advanced students may be pleased to have the opportunity to examine the more subtle, complex nature of these advanced algorithms. The later section on indexes does not depend on material covered in the earlier sections.

UNIT 37 - QUADTREE ALGORITHMS AND SPATIAL INDEXES

A. INTRODUCTION

- the previous unit defined the basic idea of a quadtree
- this unit examines how quadtrees are used in several simple processes, including:
  - measurement of area
  - overlay
  - finding adjacent leafs
  - measuring the area of contiguous patches
- in addition, this unit will look at how quadtrees can be used to provide indexes for faster access to vector-coded objects
- finally, alternative forms of spatial indexing will be reviewed

Definition

- to traverse a quadtree:
  - begin by moving down the leftmost branch to the first leaf
  - after processing each leaf in this branch, move back up to the previous branching point, and turn right
  - this will either lead down to another leaf, or back to a previous branching point

diagram

overhead - First map

  - several of the following examples use this simple raster and its associated quadtree

B. AREA ALGORITHM

Procedure

- to measure the area of A on the map:

traverse the tree and add those leafs coded A, weighted by the area at the level of the leaf

Example

- in the example quadtree, elements at level 0 have area 16, at level 1 - area 4, at level 2 - area 1
    - thus, area of A is:

      1 (leaf 00) + 1(leaf 02) + 1 (leaf 03) + 4 (leaf 2) + 1 (leaf 32)

      = 8 units

C. OVERLAY ALGORITHM

overhead - Second map
- note: this overhead can be physically overlayed on First map

Procedure

- to overlay the two maps:
    - traverse the trees simultaneously, following all branches which exist in either tree
    - where one tree lacks branches (has a leaf where the other tree has branches), assign the value of the associated leaf to each of the branches
    - e.g. node 3 is branched on map 1, not on map 2
        - the leafs derived from this node (30, 31, 32 and 33) have values B, B, A and B on map 1, all 2 on map 2
- the new tree has the attributes of both of the maps, e.g. A1, B2

Result

overhead - First map + Second map

D. ADJACENCY ALGORITHM

Problem

- find if two leafs (e.g. 03 and 2) are adjacent

  Corollary: find the leafs adjacent to a given leaf (e.g. 03)

- note that in arc based systems adjacencies are coded in the data structure (R and L polygons), so this operation is simpler with vector based systems

Definition

- here adjacent means sharing a common edge, not just a common point

diagram

Two cases

- leaf codes are: 1. same length (same size blocks, e.g. 01 and 02) or 2. one is longer than the other (different size blocks, e.g. 03 and 2)

- solving this problem requires the use of: 1. conversion from base 4 to binary and back
    - base 4 because of the "rule of 4" used in constructing quadtrees 2. bit interleaving 3. a new concept called Tesseral Arithmetic

Tesseral Arithmetic

- tesseral arithmetic is an alternate arithmetic useful for working with the peculiarities of quadtree addressing

- to add binary numbers normally, a "carry" works to the position to the left
    - e.g. adding 1 to 0001 gives 0010
    - this is the same as decimal arithmetic except that carries occur when the total reaches 2 instead of 10

- in tesseral arithmetic, a "carry" works two positions to the left
    - e.g. adding 1 to 0001 gives 0100

- the reverse happens on subtraction
    - 1000 less 1 is 0010 not 0111, as the subtraction affects only the alternate bits

- in other words, if we number the bits from the left starting at 1
    - adding or subtracting 1 affects only the even- numbered bits
    - adding or subtracting 2 (binary 10) affects only the odd-numbered bits

Determining Adjacency

handout - Determining adjacency

1. same size blocks:

- two leafs are adjacent if their binary representations differ by binary 1 or 10 (decimal 1 or 2) in tesseral arithmetic
- example: 01 and 03 are adjacent because 0001 and 0011 differ by binary 10, or decimal 2
- example: 033 and 211 are adjacent because in tesseral arithmetic

  001111 + 10 = 100101, or 100101 - 10 = 001111

2. different size blocks:
    - taking the longer of the two codes:
        - convert it from base 4 to binary
        - tesseral-add and -subtract 01 and 10 to create four new codes

reject any cases where subtracting was not possible (a "negative" code would have resulted, or a "carry" would have been necessary to the left of the leftmost digit)
        - discard the excess rightmost digits in the resulting transformed longer codes
        - convert back to base 4 to get the leaf
    - the two blocks are adjacent if any of the transformed and truncated codes are equal to the shorter code

- example: Are 02 and 2 adjacent?
    - convert 02 to binary = 0010

      0010 + 1 = 0011 0010 + 10 = 1000 0010 - 1 (impossible) 0010 - 10 = 0000

        - truncating gives 00 and 10
        - these are equal to 0 and 2 in base 4
    - therefore, 02 and 2 are adjacent (also 02 and 0 are adjacent)

- example: Are 033 and 2 adjacent?
    - convert 033 to binary = 001111

      001111 + 1 = 011010 001111 + 10 = 100101 001111 - 1 = 001110 001111 - 10 = 001101

        - truncating to two digits gives 01, 10 and 00
        - these are equal to 1,2 and 0 in base 4
    - therefore, 033 and 2 are adjacent

- example: Find leafs adjacent to 03 in the first map above

- method: find the codes of adjacent blocks of the same size, then work down the tree to find the appropriate leaf
    - (note: can only find equal or shorter codes - equal or bigger leaf blocks)

      0011 + 1 = 0110 = 12 : leaf 1 0011 + 10 = 1001 = 21 : leaf 2 0011 - 1 = 0010 = 02 : leaf 02 0011 - 10 = 0001 = 01 : leaf 01

Length of common boundary

- the length of common boundary between the two blocks is determined by the level of the longer code
    - can use this to construct an algorithm to determine the perimeter of a patch
        - e.g. the length of the A/B boundary in the first example map

diagram

E. AREA OF A CONTIGUOUS PATCH ALGORITHM

Problem

find the area of a contiguous patch of the same value, e.g. all A

Corollary: How many separate patches of A are there?

- note: this is a general method which can be used in both quadtree and vector data structures
  - i.e. find contiguous sets of quadtree blocks or irregularly shaped polygons, given that adjacencies are known or can be determined

- the following example uses the original raster map
  - note that there are only two contiguous patches; the areas of A and B form only one patch each

Method

- handout - Area of a contiguous patch (2 pages)

- create a list of leafs, with their associated codes, by traversing the tree

  - allow space for a "pointer" for each leaf, and give it an initial value of 0 (see handout)

Algorithm

- for each leaf i:
  - find all adjacent leafs j with equal or shorter length codes (4 maximum)
  - if the adjacent leaf j has the same value, determine which of i and j has the higher (larger value) position in the list, and set its pointer to the lower position
  - (note: if a pointer has already been changed, it may be changed again or left, the result is the same)

- this produces the final pointer list

Results

1. the number of contiguous patches will be equal to the number of zeros
   - in the example, two pointers are zero, indicating two contiguous patches

2. the value of each patch can be obtained by looking up the values of leafs with 0 pointers
   - in the example, leafs 00 and 01 have 0 pointers
   - these have the values A and B respectively

3. to find the area of each patch, select one of the zeros and sum its area plus the areas of any leafs which point to it directly or indirectly
   - the component leafs of each patch can be found by starting at with a leaf at the end (or beginning) of the list and following the pointers until a 0 is found
     - e.g. leaf at position 10 (code 33) points to 8, which points to 7, which points to 5, which points to 2, which has a zero pointer

        - therefore, leaf position 10 (code 33) is part of the same patch as leaf 2 (code 01) and has the value B
    - the areas can be found by summing the leaf areas
        - for the example:

          A leafs: 00 02 03 2 32 A positions: 1 3 4 6 9 Area of A: 1 + 1 + 1 + 4 + 1 = 8

          B leafs: 01 1 30 31 33 B positions: 2 5 7 8 10 Area of B: 1 + 4 + 1 + 1 + 1 = 8

F. QUADTREE INDEXES

Indexing using quadtrees

- indexes are used in vector systems to get fast access to the objects in a particular area of a map
    - very useful in searching for potentially overlapping or intersecting objects
    - therefore, they are an essential part of a polygon overlay operation

- in Unit 34, looked at the usefulness of a simple sort of objects on one axis (e.g. x) in the moving band operation for intersection calculations
    - now will look at methods which can be thought of as sorting on both axes simultaneously

- these use 2D coding systems and a simple one dimension sort

Setting up the index

overhead - Quadtree indexes

- steps are:

  1. for each object (point, line, area) in the database, find the smallest quadtree leaf which encloses the object

      - some large objects will have to be classified as NULL, as they span more than one of the four leafs in the first branching (0, 1, 2 and 3)
      - other smaller objects may be enclosed within a small leaf, e.g. 031

  2. sort or index the objects by the enclosing quadtree leafs

Using the index

- to find all objects which might intersect an area, line or point of interest
    - find the quadtree leaf enclosing the object of interest
    - starting at this point follow up the quadtree through all branching points that contain the original cell and down the quadtree to all branching points and leafs below the cell

- example: the area of interest is enclosed in leaf 31 of the original example quadtree (overhead - First map)
    - the objects which may intersect the area of interest are those in leaf 31 and all leafs above it
        - thus, these are 3 and the null leaf
        - objects in other (remote) leafs cannot intersect the area of interest, so need not be checked
- example: the area of interest is enclosed in leaf 0
    - the objects which may intersect the area are in leaf 0, the null leaf and all leafs below 0 - 00, 01, 02, 03
    - there may be other leafs below these as well such as 010, 011, 012, 013, etc

Generalizations

- quadtree indexing is most effective for small objects, particularly points
    - large objects tend to require large enclosing leafs even though they may not fill much of the space (i.e. highway corridors)
        - these objects will always need to be checked for intersection
    - it may pay to subdivide objects so that the pieces fall entirely within smaller leafs
- indexing in this way is intuitively more efficient than indexing by x or y alone since the quadtree index is effectively two-dimensional
- the divisions at each branching need not be equal in size
    - it may pay to have some blocks of smaller area and some of larger area, rather than four equal squares at each branching
    - however, for general efficiency the blocks should be rectangular

G. R-TREE INDEXES

- R-tree indexes are a response to the problem of indexing large areas
    - R stands for "range", a concept similar to MER

Method

overhead - R-tree

- find two, possibly overlapping, rectangles (aligned with x and y axes) such that:
    - as many objects as possible are wholly within one or the other rectangle
    - there are roughly equal numbers of objects wholly enclosed in each rectangle
    - the overlap between the rectangles is minimum
- indexing is determined by the rectangle in which the object is contained
    - objects which are wholly within a rectangle are associated with that rectangle
    - objects which are not wholly within either of the two rectangles are associated with the undivided map

- apply the procedure recursively, finding two new smaller rectangles within each existing rectangle

- this creates a tree structure similar to the quadtree
    - every object is associated with some node in the tree

- to find the objects which might intersect a given area of interest:
    - find the smallest rectangle used in the indexing procedure which wholly encloses the area of interest
    - the objects are those associated with this rectangle and all nodes above and below it in the tree

Problem

- although benchmark tests have shown that R-trees are generally more efficient than quadtrees and simple 1-D sorts, they are computationally intensive to construct

REFERENCES

Buchmann, A., O. Gunther, T.R. Smith and Y.-F. Wang. Design and Implementation of Large Spatial Databases, Unit Notes in Computer Science 409, Springer Verlag, Berlin. Contains a collection of papers on spatial data indexing.

Guttman, A, 1984. "R-trees: A dynamic index structure for spatial searching," ACM SIGMOD, pp. 47-57.

Mark, D.M., and J.P. Lauzon, 1984. "Linear quadtrees for Geographic Information Systems," Proceedings, International Symposium on Spatial Data Handling, Zurich, 2:412-430.

Noronha, V., 1988. "A survey of Hierarchical Partitioning Methods for Vector Images," Proceedings, Third International Symposium on Spatial Data Handling, Sydney, Australia, pp. 185-199.

Oosterom, P. van, 1990. "A modified binary space partitioning tree for geographic information systems," International Journal of Geographical Information Systems 4(2):133-46.

The two Samet books listed as references for Unit 36 contain useful discussions of quadtree algorithms.

DISCUSSION AND EXAM QUESTIONS

1. Compare the formal methods of indexing (quadtree, R-tree, 1-D sort) with informal methods in common use (e.g. continents, nation-states, major civil divisions, ZIP codes, etc.).

2. How would you design a study to compare the effectiveness of different indexing schemes? What data would you use? What measures would you compare?

3. Current vector-based systems use a wide variety of indexing schemes. Why is there no

consensus as to the best? What methods are best for what purposes and area of application?

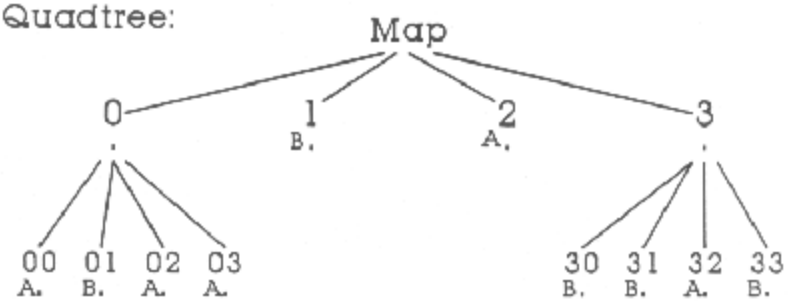
4. Devise a means of measuring the Manhattan distance between two quadtree blocks (assume the codes have the same length).

Last Updated: August 30, 1997.

UNIT 37 IMAGES

First Map:
A A A B
A A B B
A A B B
A B B B

Quadtree:



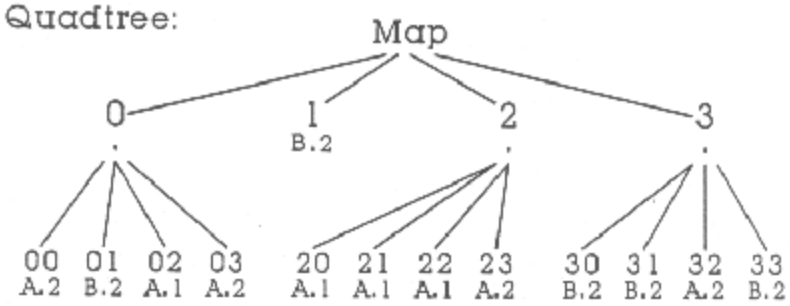
Second Map:
1 2 2 2
1 1 2 2
1 2 2 2
2 2 2 2

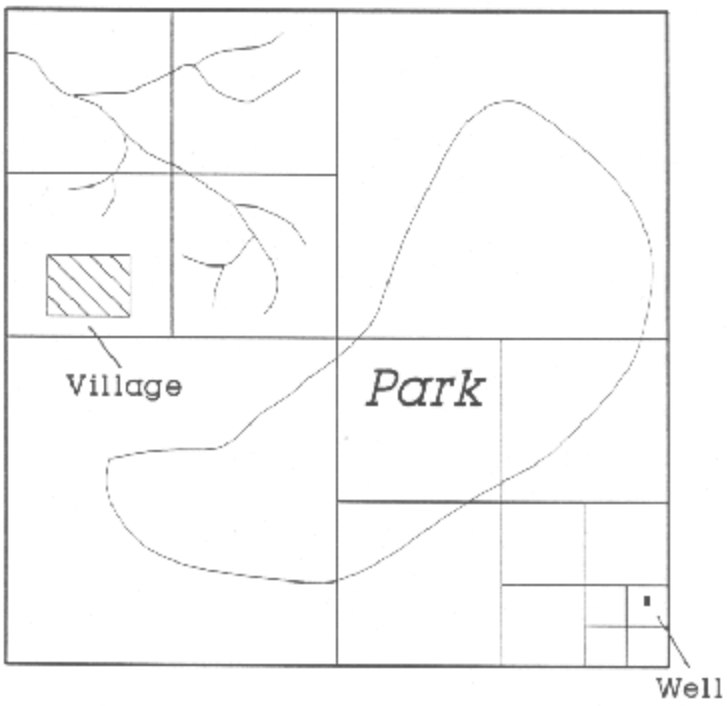
Quadtree:

First Map:
A A A B
A A B B
A A B B
A B B B

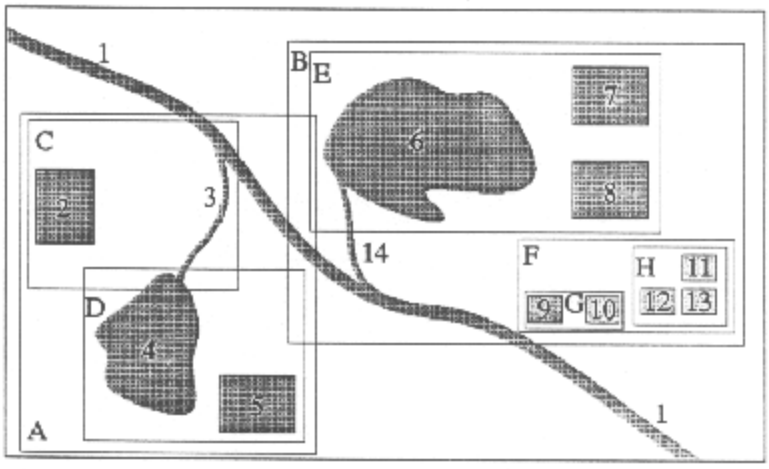
Second Map:
1 2 2 2
1 1 2 2
1 2 2 2
2 2 2 2

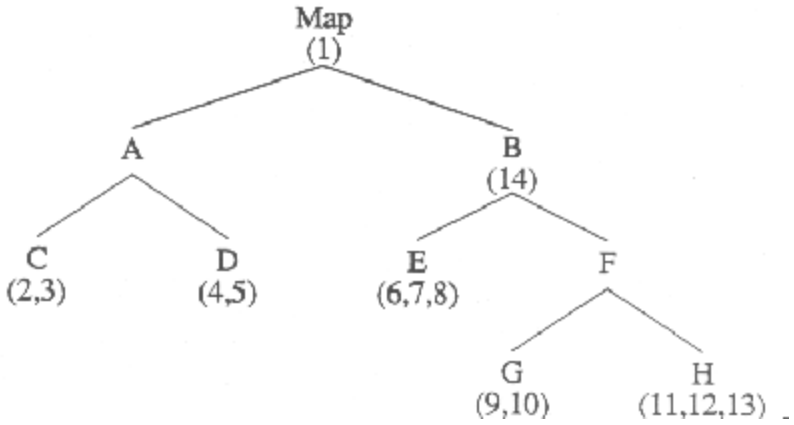
Quadtree:





	Index
Park	NULL
River	2
Village	20
Well	1113


1
B
E
7
C
6
2
3
8
14
F
H
11
D
9
G
10
12
13
4
5
A
1


Map
(1)
A
B
(14)
C
(2,3)
D
(4,5)
E
(6,7,8)
F
G
(9,10)
H
(11,12,13)