Department of Computer Science and Engineering
University of California at San Diego

CSE 102
Spring 1996

Practice Questions

1. Guttman claims that exhaustive search for an optimal split is impractical but you might think that with fast and large central memories, and even faster CPUs you can get away with it. Well... let's see.

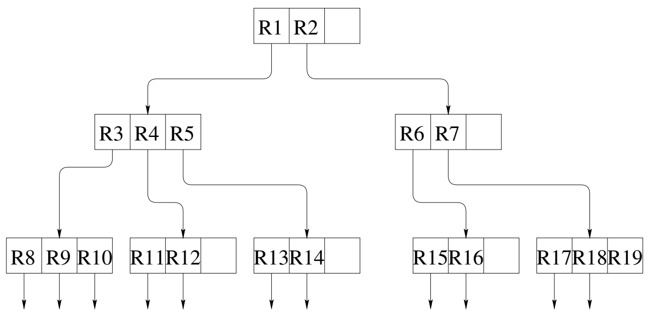
   Let us say that we have $m = 8$ and $M = 64$. Your processor can examine one million possible splits per second. Would it be practical to make an exhaustive search? (just to give you a reference: 1 day $\approx 2^{16} sec$, 1 year $\approx 1.8 \times 2^{25} sec$, life expectation at birth of the average American: $1.05 \times 2^{31} sec$, age of the universe: $1.36 \times 2^{62} sec$.)

   Would your answer change if your processor could examine *10 billions (*10^{10}*)* possible splits per second?

   Answer: If $M = 64$, the number of possibilities we must consider for exhaustive search is 2^{64}. In a second we make a million of them (that is, rougly, 2^{20}) so we need 2^{44} seconds to finish it all. While this is considerably less that the age of the universe, it is still about 2^{20} years, or 1 million years. I call this impractical!

   If you can do 10 billion comparisons per second (10,000 times what you did before), you need "only" 100 years.

2. Consider the following R-tree (fig 3.1 on the paper), and the query region indicated by a gray square.

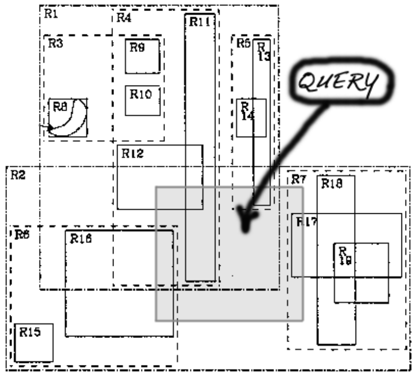


   (a) What regions will the query retrieve?
   (b) What branches of the tree will be traversed, and what regions will be checked for intersection?

   Answer: the query will retrieve regions R12, R11, R16, R13, R17. It will explore all the tree except for the children of R3.

3. Suppose that in the same tree we remove the region R8. What tree will we obtain? What intervals will be affected?

   Answer: Since the node containing R8 is full, removing R8 will not require the reorganization of the tree, so the structure will remain the same. Intervals R3 and R1, though, will have to be changed.



4. We have an R-tree with 4096 leaves, m=2, M=16. What are the minimum and maximum heights for the tree?

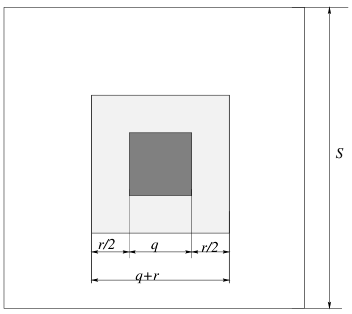
   Answer: The minimum height is when every node has M children, This is $\log_M 4096 = 3$. The maximum height is when the root has only two children and every other node has m: $1 + log_2 2048 = 12$.

5. in B-tree, we have an exact criteron for almost everything: we know how to order the children of a node, how to distribute elements in a node, which children to move and where when we split a node, and whatnot. In R-trees we have no such precise criteria, but only some heuristics which, by definition, sometimes will work, and sometimes won't. The reason, of course, is not that Guttman is not good enough to come up with optimal methods, but has something to do with a fundamental difference between one-dimensional data (like those we have in B-trees) and multidimensional data (like those we have in R-trees). What difference is that?

   Answer: The difference is that data in a B-tree can be ordered, while data in a multidimensional tree cannot. Although we can define several partial orders (e.g. inclusion of a region into another), there is no total order for two-dimensional regions.

6. (this is strictly for math lovers). The original objective of R-trees is to minimize the number of regions that intersect a given region (the "query" region), but what Guttman really does is to try to keep regions "small." You are asked to prove that this is a good idea.

   Let us see how. Suppose everything happens inside a square of side S. Your query is a square of side q there in the middle (see Fig.). Let us say for now that the R-tree contains only one region: a

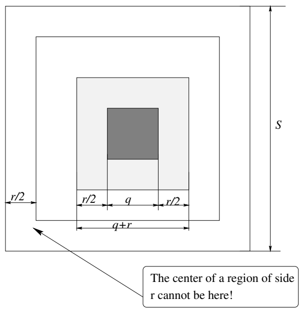


   square of side r that can be placed everywhere inside the square of side S with equal probability. The probability that the query intersect the region is equal to the probability that the center of the region falls within the square of size $q + r$ in the center of the square S (the square shown in light gray in Fig.). What is that probability (remember to assume that the region can be everywhere in S with equal probability).

What if you have n regions (all of them squares of side r) in the square S? What is the probability that at least one of them intersect the query? (Hint: P(at least one intercept) = 1 - P(no intersection) = 1 - P(All the centers of the regions out of the light gray square in Fig.))

Based on this, do you think Guttman's strategy is sensible?

Answer: The center of a region can occupy a square of side $S - r$. The probability that this center be in the central square is given by the ratio of the areas of the central $q + r$ square divided by the



area of the region $S - r$ where the center can be:

$$p = \left(\frac{q+r}{S-r}\right)^2 \approx \left(\frac{q+r}{S}\right)^2$$

where the approximation is due to the fact that, in general, $S >> r$.

The probability that one particular region does not intersect the query is given by:

$$p_1 = 1 - \left(\frac{q+r}{S}\right)^2$$

therefore the probability that none of n regions intersect the query is:

$$p_1^n = \left[1 - \left(\frac{q+r}{S}\right)^2\right]^n$$

and the probability that at least one intersect is:

$$p_2^n = 1 - \left[1 - \left(\frac{q+r}{S}\right)^2\right]^n$$

you can see, by graphing this as a function of r (for a fixed n) that the probability decreases as r decreases. Therefore Guttman's strategy is quite sensible.

7. Given M, what values can m assume to have the R-tree algorithms work properly and avoid performace degradation? What happens if m is too big? What happens if m is too small?

   Answer: $2 \leq m \leq \frac{M}{2}$. If $m = 1$ (too small) the tree could degenerate into a linear list, and the search time becomes linear rather than logaritmic. if $m > \frac{M}{2}$ it is impossible to split a node: if a node with $M + 1$ elements is split in two parts, at least one of the parts will have less than $\frac{M}{2} + 1$ nodes. If $m > \frac{M}{2}$ that node is illegal and cannot be created.

8. Would your answer to the previous question change (and, if yes, how) if instead of R-trees we were considering B-trees?

   Answer: No.

9. Guttman claims (pag. 51, last paragraph of section 3.3) that insertions can prevent deterioration of the tree structure. Can you explain intuitively why?

10. Suppose you are building an R-tree for a data structure that will never change but will be interrogated very often. Would you use the linear split or the quadratic split?

    Answer: Quadratic split is slow but, in general, gives a tree with a better structure that is better for search. Since insertion is a one-time operation in this case, I would go for it. I would run the long and computing-intensive insertion algorithm at night, when only students are using the machines...