

Database Tuning, ITU, Spring 2010

Rasmus Pagh

March 9, 2010

1 Inverted indexes versus full-text indexes

An alternative to a full-text index is to supplement an inverted index with an full-text index on the vocabulary (set of words occurring in the documents). Since the vocabulary is likely to be much smaller than the total size of all documents, it might fit into internal memory. A search would first find the words that match one or more search terms, using the full-text index on the vocabulary. Then the inverted index would be used to retrieve the matching documents for each search term.

a) Argue that the time to search for documents containing the two strings **inverted** and **index** in this way will be about the same as in a full-text index (e.g., a suffix tree)?

b) Give an example of a search that could not be handled in this way (in fact, could not be handled by a simple inverted index) but is supported in a full-text index.

2 2D range searches

Suppose we are interested in 2D range searches of the form $(8 \leq a \text{ AND } a < 16) \text{ AND } (24 \leq b \text{ AND } b < 32)$, where a and b are 8-bit numbers. It is known that the range boundaries will always be divisible by 4, and we will never ask for a range that is not contained in a 32×32 square. Two possibilities are considered:

- A grid file.
- A simplified range tree, based on the idea of the lecture.

a) For the grid file, cell sizes between 4×4 and 32×32 are considered. Give an example of a point set and query where the smaller cell size is better, and one where the larger point set is better.

b) In the simplified range tree, argue that it suffices to store each point in 4 B-trees. What is the largest number of B-trees that may need to be searched?

c) Describe a point set and query where the grid file outperforms the range tree, and vice versa.

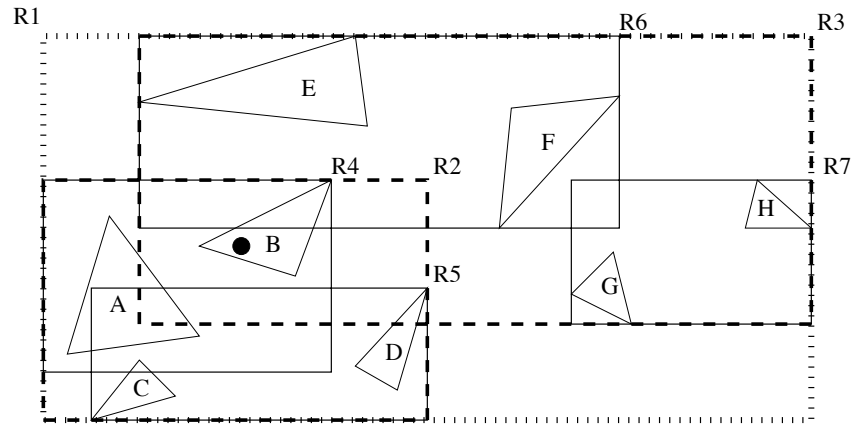
3 More inverted indexes

You are implementing a system that should support free text search for a number of words (not phrases), e.g., a query for “pays big bucks” should return all customers whose description contains these three words (not necessarily in sequence). You decide not to use the DBMS’s default text indexing, since the technical details are not well documented – instead, you want to implement a simple inverted list using relations, as you remember from a DBT lecture long ago. You intend to cluster the relation containing the occurrences such that Z occurrences of a word can be read in around Z/B I/Os. A special requirement is that commonly made searches should be very efficient. Therefore you intend to keep precomputed sorted lists of documents also for pairs of words. If a query contains two words for which there is a precomputed list, this list may be used when forming the query result.

1. Consider a query for three words. With a precomputed list for two of the words, we now need to merge only two lists rather than three. However, the time for this can be much smaller than the time needed to merge the lists for each of the three words. Explain why.
2. From an execution time perspective, the best thing would be to have a precomputed list for every pair of words. Suppose the maximum number of distinct words in any document is K , and there are N words in total. Argue that the precomputed lists may be of total length around $NK/2$.
3. In view of the above, you decide to keep lists only for pairs of the 10 most common query words (45 pairs in total). What is the extra space usage needed for this, from a worst-case perspective?
4. Compare the cost of updating this index when a new document is added to the cost of updating a standard inverted index. You may assume that the ID of the new document is higher than all other IDs, and that the set of most frequent words does not change. Your answer should consider both the case where internal memory can store a block for each distinct word, and the case where the size of internal memory is very small.

4 R-trees for triangles

In this problem we consider an R-tree that is used for storing non-overlapping triangles. We may draw a (binary) R-tree as follows:



Each rectangle corresponds to a node in the R-tree. In the drawing we have placed a name for each node at the upper right corner of its rectangle (except the root R1, whose name is at the top left corner). The leaf nodes containing the triangles are denoted A, B, C, D, E, F, G, H.

a) Consider a point query that asks for the triangle, if any, surrounding a point. In the example above, which nodes in the R-tree are visited when making the point query for the point shown as a black circle?

b) How many nodes in an R-tree with N leaves may need to be visited while performing a single point query? Give the worst example you can come up with, involving N triangles and 1 point. The example should be valid no matter how the rectangles of the R-tree are chosen.