Dynamic Indexing

Up to now we always assumed for static collections (given n documents, index them and stop), while in reality documents come and are modified over time.

This means that the postings for terms already in the dictionary need to be updated, and that new terms are added/deleted.

1° Approach: AUXILIARY INDEX

Maintain a "big" main index and a "small" auxiliary one: insert the new terms in the small one and periodically re-index in the big one.

In case of search, scan both index and merge the results. In case of deletion, keep a bit-vector for the deleted docs and filter the search results by that vector.

However, in order to make the reindexing efficient, we should keep a separate file for each posting list: this way merging the two indexes is just an append.

BUT FOR OS, keeping a lot of file is inefficient

File system indexing problem

2° Approach: Logaritmic Merge

Take an index Z_0 stored in memory with max size M: when Z_0 is full, move its content to a new index I_0, stored in disk, of max size M. The next time Z_0 is full again, we move $Z_0 + I_0$ to a new index I_1, stored in disk, of size $2M$.

Going this way, we will have a series of indexes in disk, each of size twice as large as the previous one $(M, 2M, 4M, \dots)$, and a small index Z_0 in memory, of size M.

Each posting participates to no more than $\log\left(\frac{\#\text{posting}}{M}\right)$ mergings, because each merge move the posting to the next index, and we have at most $\log\left(\frac{\#\text{posting}}{M}\right)$ indexes.

Whereas in the previous approach, each posting participates to at most $\frac{\#\text{posting}}{M}$ mergings, because we have a merge of the two indexes every M-size document.