CS 7960 *Alex Clemmer, u0458675*

Streaming Problem Set

1 Building B -Trees in External Memory

The key insight of the B^+-Tree is that it stores records ("satellite information") at the leaf level of the tree, which allows for a larger branching factor, which makes the hight of the tree lower, which allows for fewer IOs. So how many IOs does it take to construct such a tree from $O(N/B)$ contiguous blocks of memory?

1.1 The algorithm

The insertion case of B-Trees (*i.e.*, the general case of B-Trees, not just the B^+-Tree specifically) is programmatically similar to the insertion case of BSTs, but with a few significant added complications. We still begin by looking for the leaf position to insert the key at, but actually inserting the key is restricted by the fact that the B-Tree must remain balanced.

This problem more or less breaks down into a couple basic cases. If the leaf block we're planning to insert to is *not* full, then we can simply add the record. If it actually is full, we want to run a `split` procedure on the leaf.

`split` is conceptually simple: we allocate a new leaf, and move half the records from the current leaf into the new leaf. We then want to take this new leaf's smallest key and insert it into the parent, in addition to the middle key. If the parent is full, we `split` that too.

Notably, there is a way to do this in a single pass (as opposed to one pass both to find the leaf position, and n more IOs to split all parents that need to be split), and while this will reduce our constants, it will not reduce the order of our IOs.

1.2 Analysis

There are $O(N/B)$ contiguous (unsorted) blocks of memory. Just reading this data will thus take $O(N/B)$ IOs at a minimum.

As noted above, an insertion (like most operations on B-Trees) is proportional to the height. Thus the asymptotic bound of IOs will depend on this factor.

The root contains at least 1 key and all the other nodes should contain at least $d-1$ keys. Any B-Tree will then have at least 2 nodes at depth 1 (for obvious reasons we disallow the minimum degree $d = 1$), and $2d$ at depth 2, and $2d^2$ nodes at depth 3, until we come to the height h of the tree, at which point the total will be $2d^{h-1}$. Thus the height of the tree grows at $O(\log n)$. Or, more precisely, for any n-key B-Tree where $n \geq 1$, we can say that for some minimum degree $d \geq 2$, the height $h \leq log_d \frac{n+1}{2}$.

From this we can trivially see that it takes $O(h) = O(\log_{\frac{M}{B}} \frac{N}{B})$ IOs to actually do the insertion (the order of the logarithm is generated by number of B-sized blocks we can fit into memory of size M). There are $O(N/B)$ such insertions, so the total IOs is $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$

2 Aggressive Approximate Counts

1. Which of the following courses have you taken: Differential calculus; Integral calculus; Multivariate calculus; Linear algebra; Probability and statistics; Artificial intelligence; Algorithms; Computer vision; Image processing; Natural language processing; Robotics; Optimization (linear, quadratic, convex, etc.)
2. List a few (research/CS/math/whatever) topics that interest you.
3. How would you rate your programming skills (1-10, 10 best)? How would you rate your math skills?
4. What are your goals in this class?
5. Please provide a 4-8 character identifier that we can use to post grades only pseudo-anonymously. Use only alpha-numerics (no spaces).
6. Please be sure that you have subscribed to the class mailing list.