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 height of the tree smaller, 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. (Note that 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. When $k = 1$, $S(x)$ will return either a_m (the last element in the stream) or m/k (the "default" value). This is guaranteed by the `else` condition: when we have only one counter-index pair, that pair is always c_{min}, and therefore we will always increment c_i and set $t_i = a_j$. Any query that is not the last number in the stream will yield m/k.

2. When $k = 2$, $c_1 + c_2 = m$. No matter what, we will always increment either c_1 or c_2. There is no way to avoid this. Thus, adding them gives us m.

3. If $k = 2$ and $f_x > m/2$, then $S(x) > m/2$. We can trivially see that if the stream is split between 2 numbers, p and q such that count$(p) >$ count(q), then one of the counts c_i will be at least 1 greater than the other. At this point, we could split up q's count between any number of unique numbers. Unfortunately, there is no way to divide count(q) such that count$(p) <$ count(q). Since each of these numbers will have a smaller count than q, this property must hold no matter how the count is divided. Thus, $S(x) > m/2$ for any case where $f_x > m/2$.

4. Under these circumstances, $|S(x_r) - f_{x_r}| = 0$. The reason is, if there are k counter-index pairs and exactly k numbers $\{x_1 \ldots x_k\}$ for which it is true that $f_{x_i} > 0$, then every x_i will be tracked by one counter-index pair (c_i, t_i). This is equivalent to brute-forcing the problem.

5. The maximum for c_i at that point will be $m' + 1$. If $k = 1$, the total c_i is m'. If we process x, then we have no choice but to set $t_i = x$ and increment c_i to $m' + 1$.