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 (which will become the current leaf's right neighbor), 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 $O(n)$ 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 $O(h) = O(\log_B N)$ IOs to actually do an insertion. Over multiple insertions, however, we will be bounded to some extent by the number of such blocks we can fit in memory, which puts us squarely in $O(h) = O(\log_{\frac{M}{B}} \frac{N}{B})$. There are $O(N/B)$ IOs to read the elements, and each of these requires an insertion, so the total IOs is $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$.

## 2 Aggressive Approximate Counts

NOTE: This may be (by the standards of the directions) a bit verbose, as I am not a math major. However, I really did try to make it as concise as possible, and I didn't just make things up. So, please be patient in reading the proofs. Thanks.

(A) 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 $c_{\text{min}}$, 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 $m/k$.

(B) 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 $m$.

(C) If $k = 2$ and $f_x > m/2$, then $S(x) > m/2$. We can trivially see that if the $m$-length stream consists of only repetitions of two numbers $p$ and $q$, and if $f_p > f_q$, the algorithm will terminate with $c_p$ being at least one greater than $c_q$. Here we can split up $c_q$ between any number of unique numbers (*i.e.*, increase the unique numbers in $m$, but making sure that $c_p > m/2$). But there is no way to distribute fractions of $c_q$ in two buckets in such a way that they will not eventually be overwhelmed by the sheer magnitude of $c_p$.

(D) These circumstances provide a maximum $|S(x_r) - f_{x_r}| = 0$. The reason is, if there are $k$ counter-index pairs and exactly $k$ numbers $\{x_1 \dots x_k\}$ for which it is true that $f_{x_i} > 0$, then every $x_i$ will be tracked by one counter-index pair $(c_i, t_i)$, which means that $S(x)$ should always be precise.

(E) In this case, at most, $c_i = m/k$. There are $k$ counter-index pairs, and we must pick the minimum. The smallest of these counters will at most be $m/k$ (or else the sum of counters will be $> m$).

(F) In this case, the maximum is $|S(x) - f_x| = m/k$. Let's say there are $k+1$ total elements in the stream, that the last two elements in the stream are equal, (*i.e.*, $a_k = a_{k+1}$), and that their value is not found anywhere else in the stream. When we encounter $a_k$, we are forced by the `else` clause to increment $c_{min}$ and replace $t_{min} = a_k$. Then, when we encounter $a_{k+1}$, we increment this count again. From above we know that at most $c_{min} = m/k$, which gives the upper bound $c_{a_{k+1}} = m/k + 2$. Since at minimum $f_{a_{k+1}} = 2$, at most $|S(x) - f_x| = m/k + 2 - 2 = m/k$.

(G) No. Reassigning $t_i = x'$ to $t_i = x$ requires that the corresponding $c_i$ is actually one of the $c_{min}$'s. Reassignment also involves incrementing $c_i$, which makes this count bigger than current $f_{x'}$. Encountering $x'$ again will cause us to reassign to one of the pairs with count $c_{min}$; since $c_{x'}$ had to be a $c_{min}$ to be overwritten to begin with, the current pool of $c_{min}$'s is at least as large as $c_{x'}$ was. Thus there is no way to have processed more than $c_i$ instances of $x'$ than the variable indicates.

(H) No. (G) shows that there is no possible way for us to have processed more $x'$ instances than denoted by counter $c_{x'}$. Put another way, it is never the case that some $f'_x > c'_x$. Thus if there is a count at all for $c'_x$ for $S(x)$ to return, it is at least $\geq f'_x$.

(I) No. If some $f_x > m/k$, then we will encounter $x$ *at least* one time more than some other corresponding element $y$ for which it must be true that $f_y < m/k$. So even if there is no $t_i = x$ every time we see $x$, we will still end up incrementing some $c_{min}$ once more than it is possible for $y$ to do, because $x$ occurs at least one time more than $y$, and if the last occurrence of $y$ happens after the last occurrence of $x$, then $x$ will not be $c_{min}$ (since it would have at worst incremented some $c_{min}$), and the $y$ thus can't select it to replace. If $y$ could replace it, it can't be the case that $f_x > m/k$.

(J) $|S(x) - f_x| \leq m/k$. This is the maximum bound we can construct from the cases above.

## 3 Randomized Approximate Range Searching

For any query interval, the goal of $S$ is for $P(|S(R) - \text{size}(R)| > \epsilon m) < \delta$. We're maintaining $k$ independent samples, which will determine exactly what our error rate is. We'll see how to control this in a minute.

Ensuring this bound is pretty straightforward:

```
for $s_i \in R$ with probability $(size(R)/m)$ do
    $Y_i \leftarrow \{1$ if $s_i \in R, 0 else\}$
    $X_i \leftarrow (Y_i - size(R)/m)/k$ if $s_i \in R, 0 else\}$
end for
```

Notably, the error count estimate of $S$ is $M = \sum X_i$. The Chernoff bound is useful here:

$$P(|M - \sum \mathbb{E}[X_i]| > \alpha) < 2 \cdot exp(-2\alpha^2 / \sum_i (\Delta_i)^2)$$

$$P(|M| > \epsilon) < 2 \cdot exp(-2\epsilon^2/(k(1/k^2)))$$

Note here that usually $\mathbb{E}[X_i] = 0$. From here, we solve for $k$:

$$\begin{aligned} 2exp(-2\epsilon^2 k) &< \delta \\ exp(2\epsilon^2 k) &> 2/\delta \\ 2\epsilon^2 k &> \ln(2/\delta) \\ k > (1/2)(1/\epsilon^2)\ln(2/delta) \\ = O((1/\epsilon^2)\log(1/\delta) \end{aligned}$$

Thus $k \in O(1/\epsilon^2 \log(1/\delta))$, which determines how many $k$ we'll need to control the error to the extent we wish.