Akshay Kumar
COSC 160
Professor Montgomery
9 December 2018

# B-Tree Implementation

The methods that posed significant challenges were insert and remove. To make it easier, I tried to model insert and remove he same way the I would perform them by hand, which meant the functions were recursive and divided into many steps, instead of magically arriving at the end result, which would have been a lot more difficult if not impossible.

The BNode struct was given its own methods, as that would help split up the work flow and make the project more object oriented. These functions were quite straightforward to implement, as their scope was very limited, and they weren't recursive.

For insert, the only case that needed checking was if the operation resulted in an overflow, which would cause the creation of a new key in the parent, and the splitting of the overflowing node. This was simple enough to implement, as the recursive nature made it easy.

Remove, on the other hand, was much more difficult as there were so many cases I needed to account for, such as underflow, whether to borrow or merge, if the root underflowed, finding and removing the immediate successor when deleting an internal node's key, etc. The logic is messy at parts, but I believe the resulting code still maintains clear readability. Yes, it does have some repetition of code, but I hesitated to early-optimize as that might cause more errors. I have added comments that you should find helpful.

---

- Insert: the invariant is that all leaf are on the same level, and all nodes except the root contain [ceil(M/2)-1] to [M-1] keys, and [numKeys + 1] links. All values are passed by value, as pointers themselves do not need to change since no operations are performed on null pointers themselves, so we always have the parent to reference.

- Remove: the same invariant as above is held. Remove could also have been pass by value since the parent is always available in an operation, but I had it pass by reference, and decided "if it ain't broke, don't fix it."

- DeleteTree(…) is a recursive delete, and is only to be used in the destructor. This is supposed to clear the tree. It can be modified so it does not delete the root, but for now that is not necessary.

* printTree() and search(...) don't modify the tree at all, they just traverse.