CMPT 225 - B-trees insert/delete

Insertion

Do a search and the place where it would be, add it.

If inserting a key makes a leaf too big, then we need to split the leaf.
We split it into 2.
Now the parent node gets another child and one more key.
The new key on the parent is the smallest key of the new (bigger) child leaf.

If the internal (parent) node becomes too big, we need to split it.
When we split an internal node, the middle node becomes a key of the parent of the splitted internal node after the split.

If the root gets too big, we need to split the root.
This is how the tree gets taller
This is similar to the internal node split.
The middle key in the root becomes the new root.
The root will now have 2 children exactly. This is ok because our definition allows this.

Even though the height increased, the depth of the leaves are all the same because they just get all moved down one.

Insertion Implementation

```
Find the leaf where the new key k belongs, call it v
insert k into v

while(v is over-full and not the root):
  u = parent(v)
  split(v)  # also adds the middle key to the parent u
  v = u

if(v is the root and over-full):
  split v into two
  add a new root with two children
```

Insertion can be done in O(logn) where n is the number of keys in the tree.

Removal

If we want to remove a key, we search for it, find it, and delete it.
If the leaf now has too few keys, we merge it with a neighbour (adjacent) sibling.

But this can't always be done.
If the neighbours both have more than the minimum number of keys then we can't merge.

But if this is the case, then the neighbours must have more than the minimum number of keys so we can steal one.

We may remove a key and the parent of the merged node might have too few children.
Then we have to do a rotation.

If the parent has a sibling that has strictly more than the minimum then we can take one of it's keys and it's child and add them to the sibling that needs another key.

If both the sibling has the minimum number of allowable nodes and the parent has one less, then we can merge the parent and the sibling and between them, the middle node of the parent and sibling's parent.

If the root gets too small.
One child is too small and the other child is of minimum size.
So we just merge the child-root-child together.
So the tree loses height.

```
find the leaf L with the key k to be removed
delete k from L

if(k was the smallest key in L):
  correct the key in an ancestor

v = L

while(v has too few and is not root):
  if(v has an adjacent sibling u with more than ciel(m/2) children):
    shift a key and child from u to v
  else:
    u = parent(v)
    merge v with an adjacent sibling
    v = u

if(v is root and only one child c):
  delete v
  c is the new root.
```

B-Tree removal is O(logn)

A tight analysis of B-tree height

Claim: An order-m B-tree of height h has at least (m/2)^h keys if L >= 2m.

so h <= log_{m/2}n

B-tree facts

B-trees were designed to exploit operation of disk cache to compensate for disk latency.
Most books still consider b-trees to be data structures for external storage.
But b-trees are also a very good in-memory data structure also: they exploit CPU data cache to compensate for main memory latency.