B^+ Trees

Robert Soulé

1 B^+ Trees

A tree is an abstract data type, consisting of a set of linked nodes. The nodes are hierarchical, and the top most node is called the *root*. An *internal* node is a node that has at least one child. A *leaf* node is a node with no children. A B tree is a tree with the following properties:

- It is *rooted*, meaning it has a root.
- It is *directed*, meaning that the order of the children matter.
- It is a *balanced* tree, meaning that all paths from the root to the leaves have the same length.
- For some parameter m:
  - All internal nodes have between $\lceil m/2 \rceil$ and m children.
  - The root has between 2 and m children.

A B^+ tree is a special case of a B tree. In a B^+ tree, the internal nodes contain only *keys* and *pointers* to subtrees. Values are stored at the leaves. The *keys* serve as separation values for the subtrees. For example, if an internal node has n children, then it must have $n-1$ keys, such that all values stored in the left-most subtree will be no greater than k_0. The values stored in the left-most + 1 subtree will be between k_0 and k_1, and so on.

2 What is the value of m?

Remember that we want to *minimize* the number of block accesses, so we get as much benefit from reading a single block as possible. Therefore, each node should be as big as possible, while still fitting in a single block. Recall that each node in the B-tree will contain m pointers (each storing a block address) and $m-1$ keys. In other words:

$$(m \times \text{size of pointer}) + ((m-1) \times \text{size of key}) \leq \text{size of block}$$

Let's work through an example. Suppose we have a 1 terabyte ($\approx 2^{40}$ bytes) disk, and that each block is 4096 ($= 2^{12}$) bytes.

How many blocks do we have? We simply divide the size of the disk by the size of the blocks:

$$\text{size of disk/size of block} = \text{number of blocks}$$

So, we have:

$$2^{40}/2^{12} = 2^{28}\text{blocks}$$

How big does each pointer need to be? We calculated that there are 2^{28} blocks, and each block needs an address. If we number the blocks $0, \ldots, 2^{28}-1$, then we need 28 bits for each pointer to store the largest address.

How big is the pointer really? In practice, this is determined by the operating system, and it is usually 32 or 64 bits (or 4 or 8 bytes).

How big is the key? The size of the key depends on what we are storing. Suppose we are storing integers as the key. The size of an integer can vary from system to system, but it is most often 32 bits (4 bytes).

What is m? If we assume that each block is 4096 bytes, pointers are 4 bytes, and the keys are 4 byte integers, then we want the largest number such that:

$$(m \times \text{size of pointer}) + ((m-1) \times \text{size of key}) \leq \text{size of block}$$

or, in our case:

$$(m \times 4) + ((m-1) \times 4) \leq 4096$$

$$(4m) + ((4m-4)) \leq 4096$$

$$8m \leq 4100$$

$$m \leq 512.5$$

$$m = 512$$

So, our value of m is 511.