Simple Prefix B+ Trees

References: Chapters 10.6 - 10.7 (Superficially 10.8 - 10.11)

A clarification regarding the book's treatment of B-trees and B + trees: the transition from understanding B trees to understanding the index set of the simple prefix B + tree may be facilitated by the following interpretation.

- Look at the B + tree index set as a B-tree in which the smallest element in a child is stored at the parent node in order:

key1, pointer1, key2, pointer2, ..., keyN, poinerN.

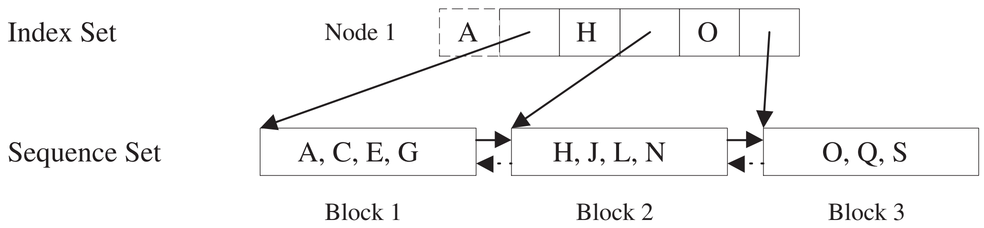
- Then, remember that each key is a separator. And remove key1 from the node's representation. It may help if you think that "key1" is still there, but is "invisible" for understanding purposes.

With this in mind, it should become clear that the updates on the B + tree index set are the same as in regular B-trees.

Simple Prefix B+ Tree Maintenance

Example:

- Sequence set has blocking factor 4
- Index set is a B tree of order 3



Note that "A" is not really there.

1. Changes localized to single blocks in the sequence set

   Insert "U":

   - Go to the root
   - Go to the right of "O"
   - Insert "U" to block 3:
     The only modification is

O, Q, S, U

     Block 3

   - There is no change in the index set

   Delete "O":

   - Go to the root
   - Go to the right of "O"
   - Delete "O" from block 3:
     The only modification is

Q, S, U

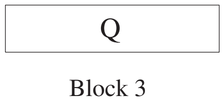
     Block 3

     There is no change in the index set: "O" is still a perfect separator for blocks 2 and 3.

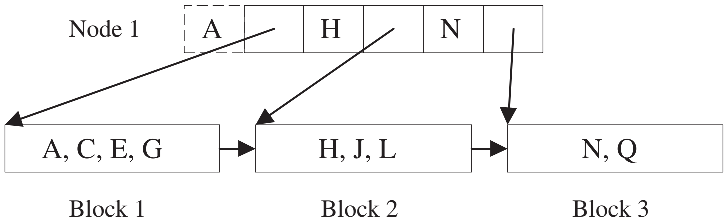
2. Changes involving multiple blocks in the sequence set.

   Delete “S” and “U”:

   Now block 3 becomes less than 1/2 full (underflow)

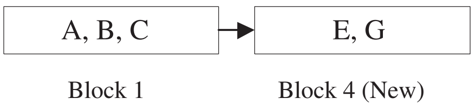


   Since block 2 is full, the only option is **re-distribution** bringing a key from block 2 to block 3:
   We must update the separator “O” to “N”.
   The new tree becomes:

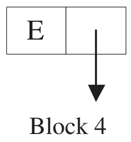


   Insert “B”:

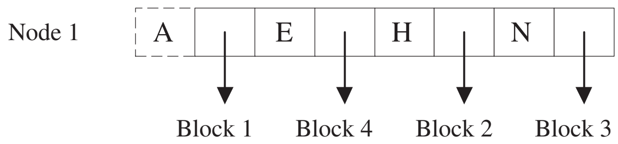
   - Go to the root
   - Go to the left of “H” to block 1
   - Block 1 would have to hold
     A,B,C,E,G
     But block can only hold 4 records.
   - Block 1 is split:



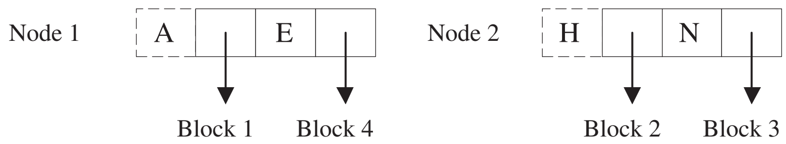
Promote new separator "E" together with pointer to new block 4



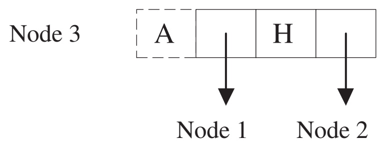
We wished to have:



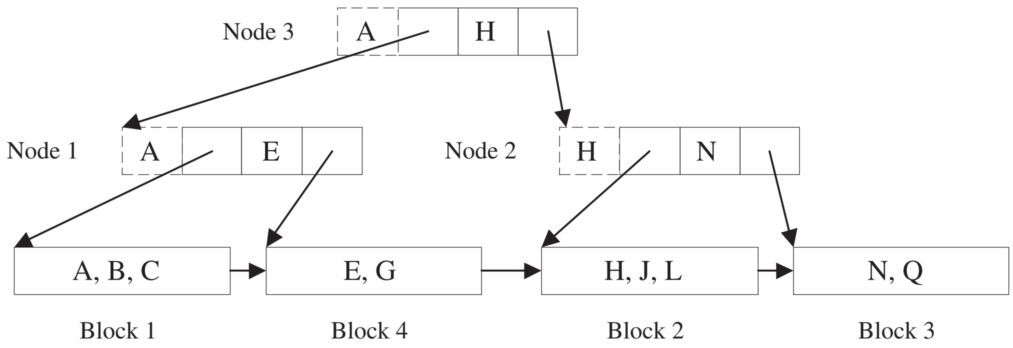
But the order of the index set is 3 (3 pointers, 2 keys).
So this causes node to split:



Create a new root to point to both nodes:



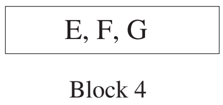
The new tree is:



Remember that “A” is not really present in nodes 1 an 3 and that “H” is not really present in node 2.

Insert “F”

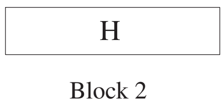
- Go to root
- Go to left of “H”
- Go to right of “E” in Node 1
- Insert “F” in block 4
- Block 4 becomes:



- Index set remain unchanged

Delete “J” and “L”

Block 2 would become:

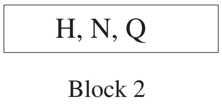


But this is an underflow.

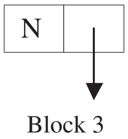
One may get tempted to redistribute among blocks 4 and 2: E, F, G and H would become E, F and G, H.

Why this is not possible ?
Block 4 and block 2 are not siblings! They are cousins.

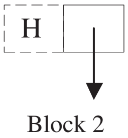
The only sibling of block 2 is block 3.
Redistribution in not possible between H and N,Q, so the only possibility is **merging** blocks 2 and 3 :



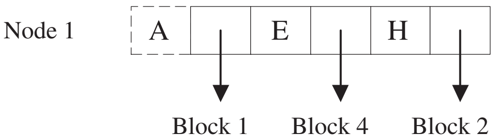
- Send block 3 to `AVAIL LIST`
- Remove the following from node 2



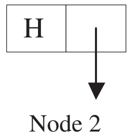
- This causes an underflow in Node 2



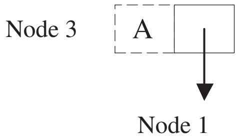
- The only possibility is a merge with its sibling (Node 1):



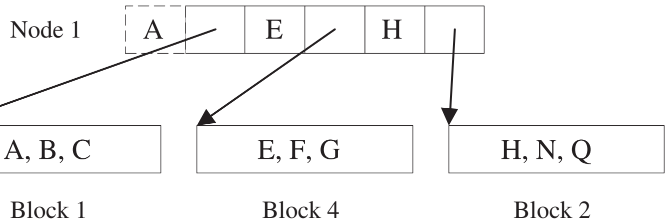
- In our interpretation "H" becomes "visible"; In the textbook interpretation "H" is brought down from the root.
- Send node 2 to `AVAIL LIST`
- Now remove



from the root (Node 3). This causes underflow on the root:



- Underflow on the root causes removal of the root (Node 3) and Node 1 becomes the new root :



Blocks were reunited as a big happy family again !!

Compare it with the original tree.

Note : Remember that a B+ tree may be taller, so that splitings or mergings of nodes may propagate for several levels up to the root.

Advanced Observations for Meditation

1. Usually a node for the index set has the same physical size (in bytes) than a block in the sequence set. In this case, we say "index set block" for a node.

2. Usually sequence set blocks and index set blocks are mingled inside the same file.

3. The fact that we use separators of variables length suggests the use of B trees of variable order. The concepts of underflow and overflow become more complex in this case.

4. Index set blocks may have a complex internal structure in order to store a variable length separators and allow for binary search on them (see Figure 10.12 on page 442).

5. Building a B+ tree from an existing file already containing many records can be done more efficiently than doing a sequence of insertions into an initially empty B+ tree. This is discussed in Section 10.9 (Building a Simple Prefix B+ Tree).

6. Simple prefix B+ trees or regular B+ trees are very similar. The difference is that the latter stores actual keys rather than shortest separators or prefixes.