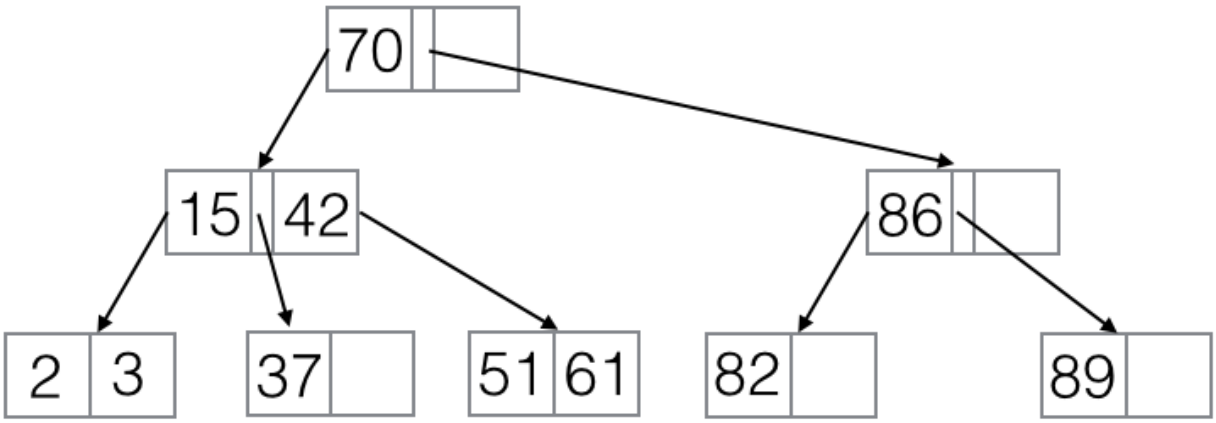
CS150A Quiz 3 Solutions

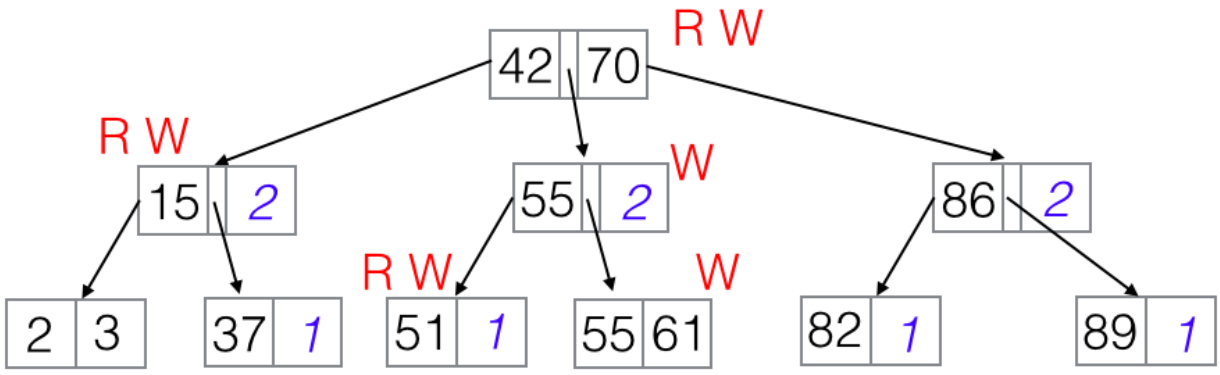
Q1: Suppose that all nodes in our B+ tree have an order of 1605. What's the MAXIMUM number of records we can index with a B+ tree of height 4?
(3210 pointers/page + 1 pointer) **^4** * (3210 leaf pointers) = **341244966965525610** records (a lot)

Q2: We want to bulk-load a B+ tree, and we reduce the fill factor of this bulk load. Which of the following applies, in general?
The key fact is that a smaller fill factor means each page will hold fewer records on the initial bulk load. This means we'll have to write more pages to disk, which means (**1**) our bulk loads take more time to complete, (**2**) pages consume more space on disk, and (**3**) the tree is (in general) deeper because we'll have more leaf pages, requiring us to perform more I/Os to get to any one page. The upshot is that since we've "pre-allocated" pages for additional records, insertions should trigger **fewer disk writes**.

Original tree:



Q3: We insert 55 into the B+ tree in figure A. How many I/Os (page reads and writes) does this operation take?
After inserting 55:



tl;dr **8** read and write ops (3 read, 5 write)

1. **Read** the root [70]. 55 < 70, so follow the leftmost pointer and
2. **read** the page containing [15, 42]. 55 > 42, so follow the rightmost pointer and
3. **read** the page containing [51, 61]. We're at the leaf, so we can perform an insertion here. However, our leaf page is full! We therefore have to split it on the key 55, so
4. we create a page containing [55, 61], and we **write** it to disk. (Note that we're at a leaf page, so we must copy 55 to its parent.) We also
5. delete 61 from the page containing [51, 61] and **write** that to disk ([51]). Now we recursively insert 55 into our parent [15, 24], which is full, so we need to split it on the key 42, and
6. we create a page containing [55], setting its leftmost pointer to be the page containing [51] and its rightmost pointer to be the page containing [55, 61]. (Note that since this is an internal node, we don't need to keep 42, as we're recursively pushing it up to the root!) We **write** this to disk, and now
7. we delete 42 and the rightmost pointer from the page containing [15, 42], and we **write** that to disk ([15]). Finally,
8. we insert the separator 42 to the root node, setting its left pointer and the pointer between 42 and 70. We **write** this node ([42, 70]) to disk as the new root.

Q4: After performing the insert in Q3, what's the maximum number of keys we can insert into the B+ tree in figure A without splitting the ROOT?

Each leaf node which isn't full yet can support an additional key, and we have 4 of these empty slots. In addition, each node with spare space on level 1 of the tree can accommodate an additional reference to a whole index leaf page, so each of these empty slots increases our capacity by 2. (Exercises for the reader: 1. What if we had extra space in the root? 2. How many extra keys can an empty slot on level (k) reference? Make sure to account for the extra leftmost pointer!)

Q5: What is the number of cache hits if we use an LRU replacement policy?

8

Q6: What is the number of cache hits if we use an MRU replacement policy?

5

Q7: What is the number of cache hits if we use a CLOCK replacement policy?

8

Q8: What is the number of set reference bits at the end of Q6?

4