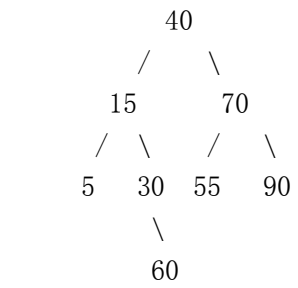
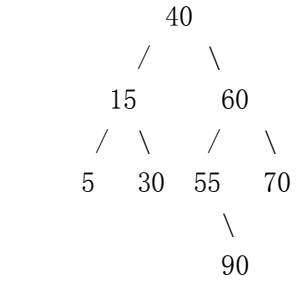
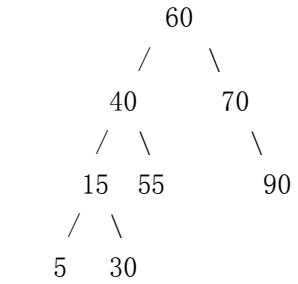
1)
First, you insert 64 in the right place,
which is as the right child of node 55





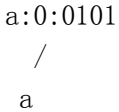


2)

It is hard to draw the backward pointers using ASCII characters. So I will put a label on every node, and use that to set up the backward pointer. And we follow the left branch if the bit is 0.

So a node looks like: <label>:<bitnumber>:<key>.

After inserting 0101:

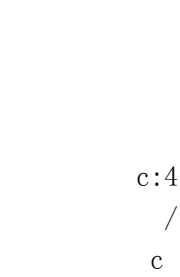


After inserting 1010:

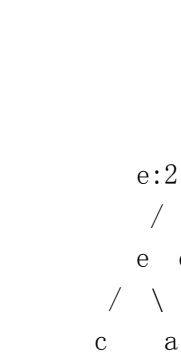
a:0:0101

After inserting 0100:

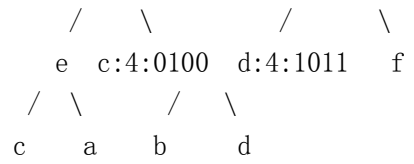
After inserting 1011:



After inserting 0001:



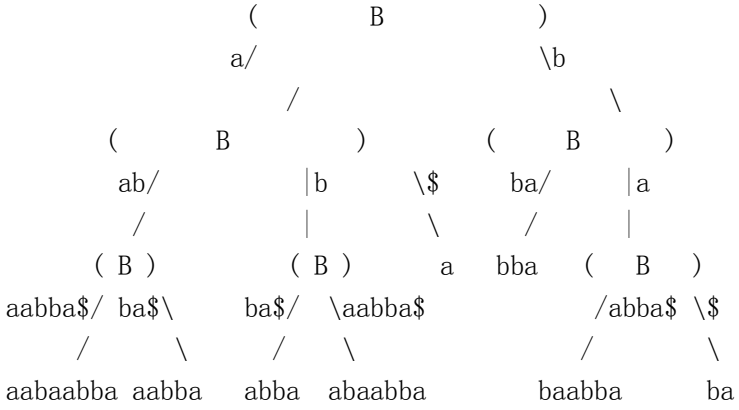
After inserting 1110:



3)

(a)

S = aabaabba Suffix(S) = {a, ba, bba, abba, aabba, baabba, abaabba, aabaabba}



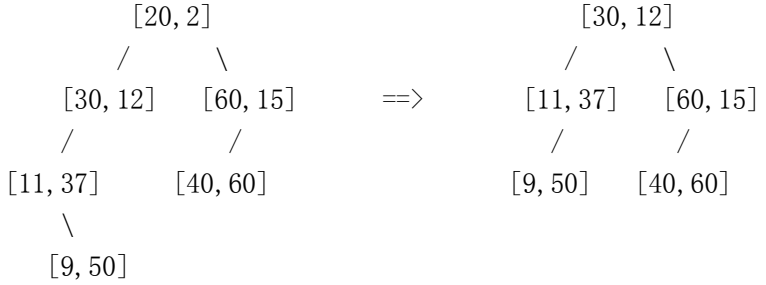
(b)

Among the branch nodes that are parents of end nodes,
choose the one whose length of the labels on the edges leading to
that branch node is largest.
Then the sequence of labels on the path from root to the chosen one
is the longest repeating substring.

The longest repeating substring = aab

4)

(a) after insert operations (b) after delete



5)

5. 2D range tree

Assume we have N records with 2 keys, x and y.

A 2-D range tree is a binary tree using the range of x, while in each node we keep a sorted list based on y. The tree always splits the records in half, using a median x key value as the discriminator.

Preprocessing time P:

we need to build a sorted list on y, and another on x. Since we have the sorted list on x, we can find the discriminator easily. And since we have the sorted list on y, we can split the list in linear time on each level (there are logN levels overall). So the preprocessing time is:

$$P = NlogN + NlogN + N*logN = O(NlogN)$$

Space required S:

In each node we store an array of the records. At each level the number of records is N. So the space required is:

$$S = N*logN = O(NlogN)$$

Query time Q:

Upon each query, we start at the root, decide which branch to take by comparing the boundaries of x with the discriminator. Once we arrive a node, we use binary search on y in the array. Then we report those records that fit the query. So the query time is:

$$Q = logN*logN + F = O((logN)^2 + F)$$

where F is the number of records reported.