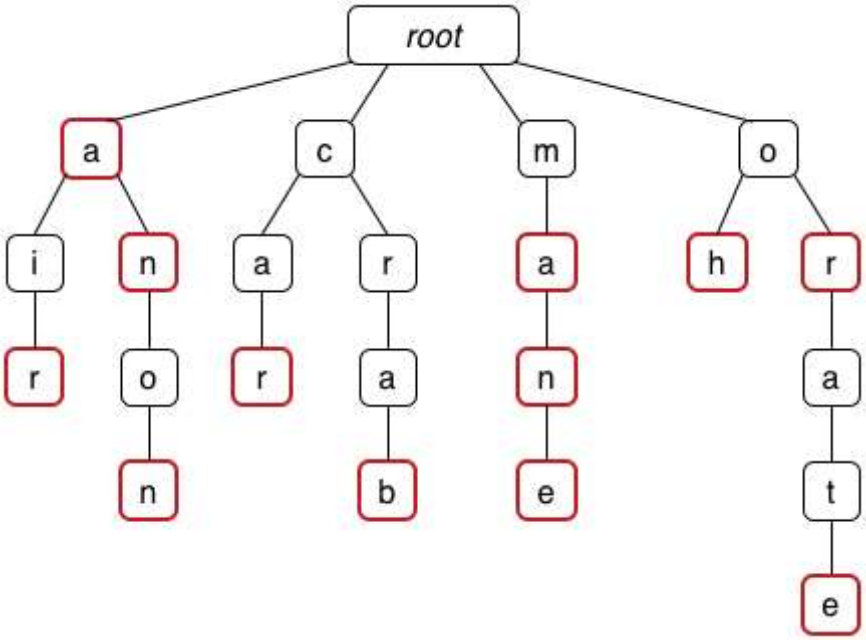
Week 07 Problem Set: Tries, Pattern Matching and Huffman Coding

1. Consider the following trie, where finishing nodes are shown in red:



What are the keys?

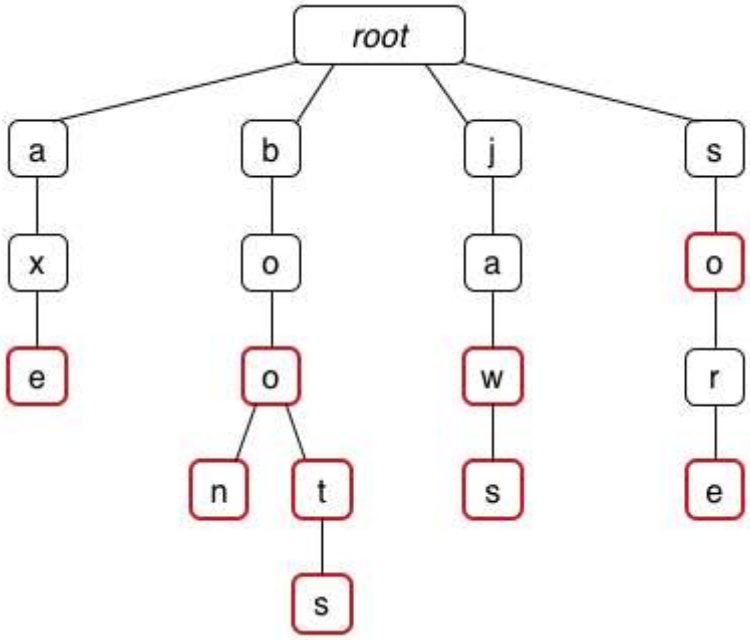
In alphabetical order: a, air, an, anon, car, crab, ma, man, mane, oh, or, orate.

2. If the following keys were inserted into an initially empty trie

```
jaws boots axe boo so sore boot boon
```

what would the final trie look like? Does the order of insertion matter?

The trie after all keys are inserted:



The order of insertion does not matter. The same trie will always result from insertion of the same set of words.

3. For every character in the following pattern, calculate the last-occurrence function (L) used in the Boyer-Moore algorithm and display it as a table:

   ```
   xywapaswyy
   ```

   See: Visualizing String Matching. Enter the pattern in both "Text" and 'Pattern", and click on "Search", see "Last" array.

4. Compute a table representing the Knuth-Morris-Pratt failure function for the following pattern:

   ```
   cgtacgttcgtac
   ```

   See: Visualizing String Matching. Click the bottom "Build failure function" in the top panel.

5. Consider the following *text* and *pattern*:

   Text:

   ```
   ABCACBBDABCADD
   ```

   Pattern:

   ```
   ABCAD
   ```

   Identify the sequence of character comparisons required for the Boyer-Moore algorithm. You also need to report total number of character comparisons required.

   See: Visualizing String Matching. Use the bottom panel to "Pause" and "Step Forward" to **slowly visualise** each step.

6. Consider the following *text* and *pattern*:

   Text:

   ```
   aaabaadaabaaa
   ```

   Pattern:

   ```
   aabaaa
   ```

   Identify the sequence of character comparisons required for the Knuth-Morris-Pratt algorithm. You also need to report total number of character comparisons required.

   See: Visualizing String Matching. Use the bottom panel to "Pause" and "Step Forward" to **slowly visualise** each step.

7. For the following string, calculate frequency array and draw Huffman tree:

   ```
   ababbcdcbaabcacbbbs
   ```

   Also provide the huffman code table for each character.

   See: Visualizing String Matching. Use the bottom panel to "Pause" and "Step Forward" to **slowly visualise** each step.
