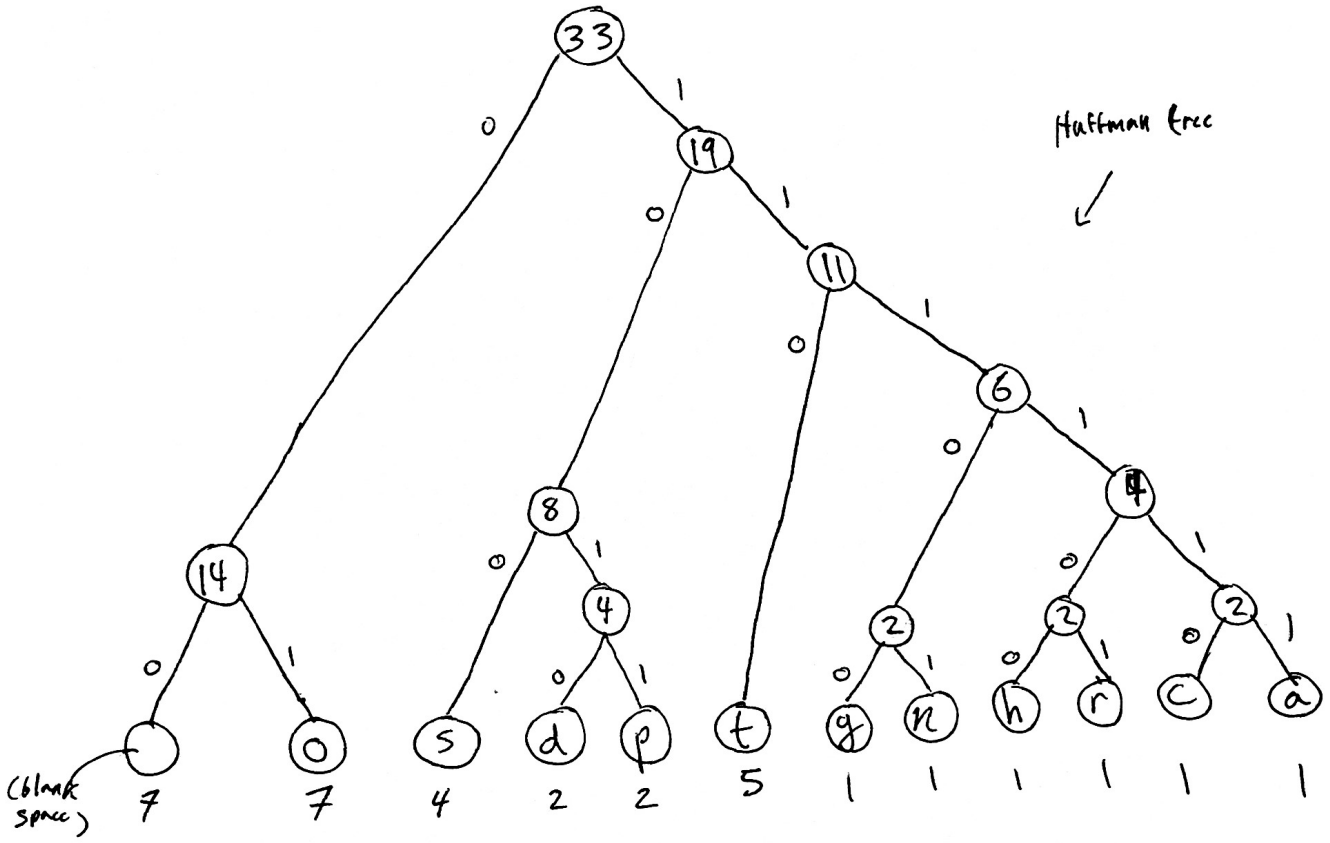
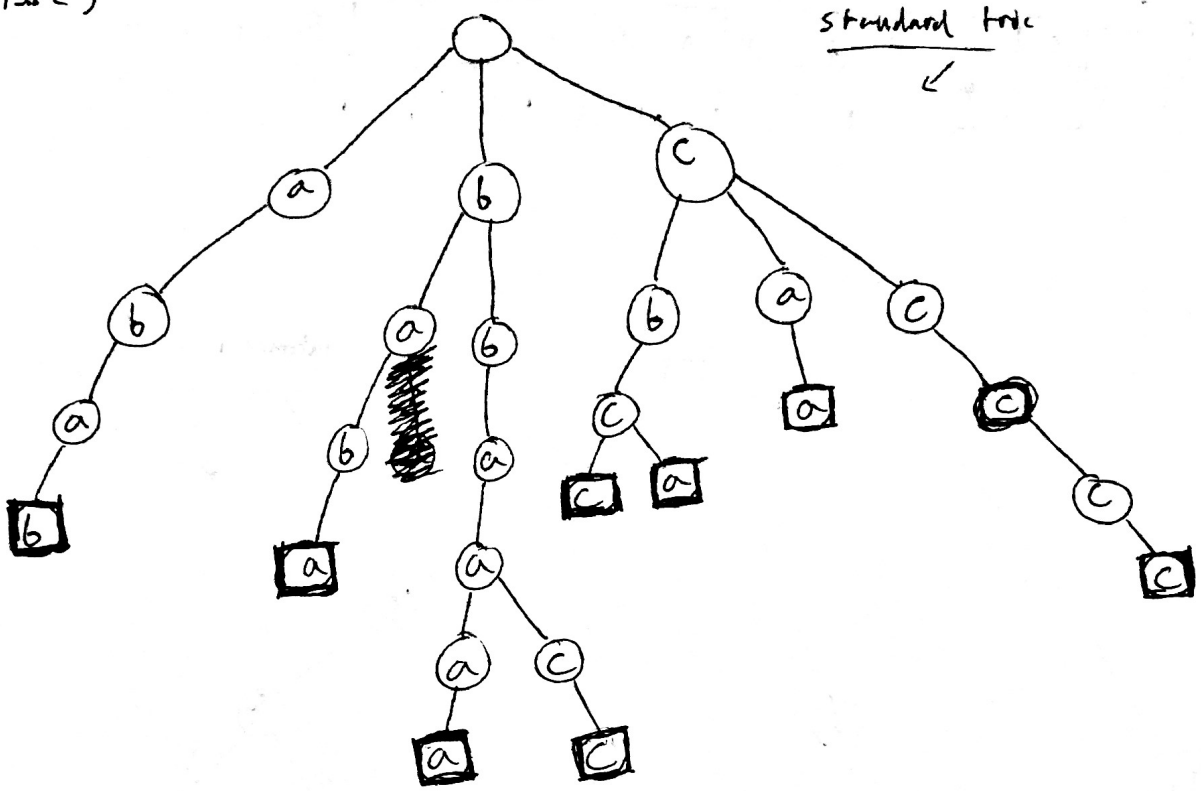
R.-13.11)

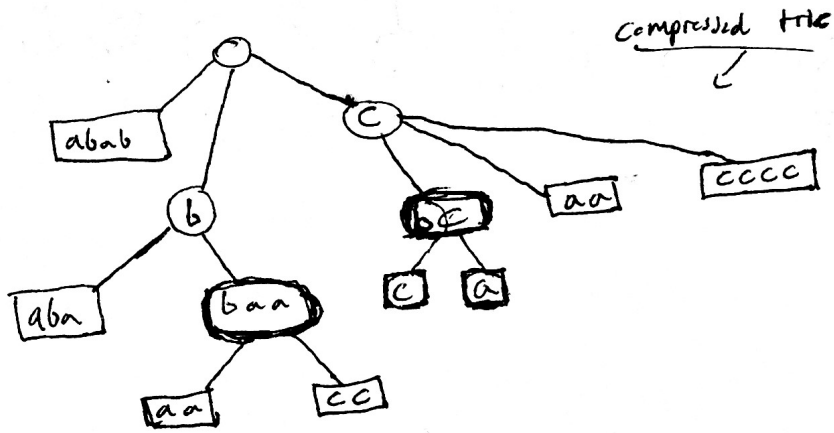
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| c | ' ' | o | s | d | p | t | g | n | h | r | c | a |
| f | 7 | 7 | 4 | 2 | 2 | 5 | 1 | 1 | 1 | 1 | 1 | 1 |



R-13.12)



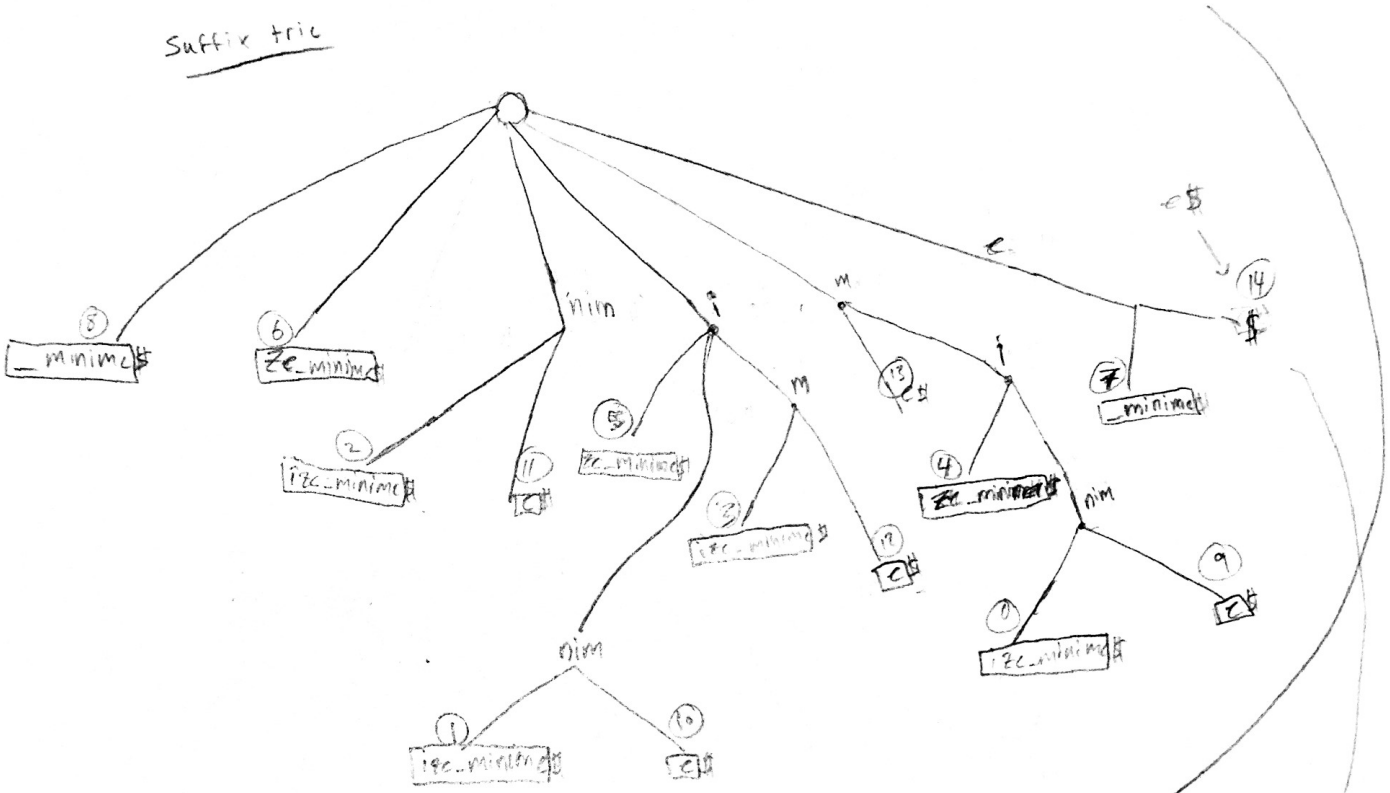

R-13.13)



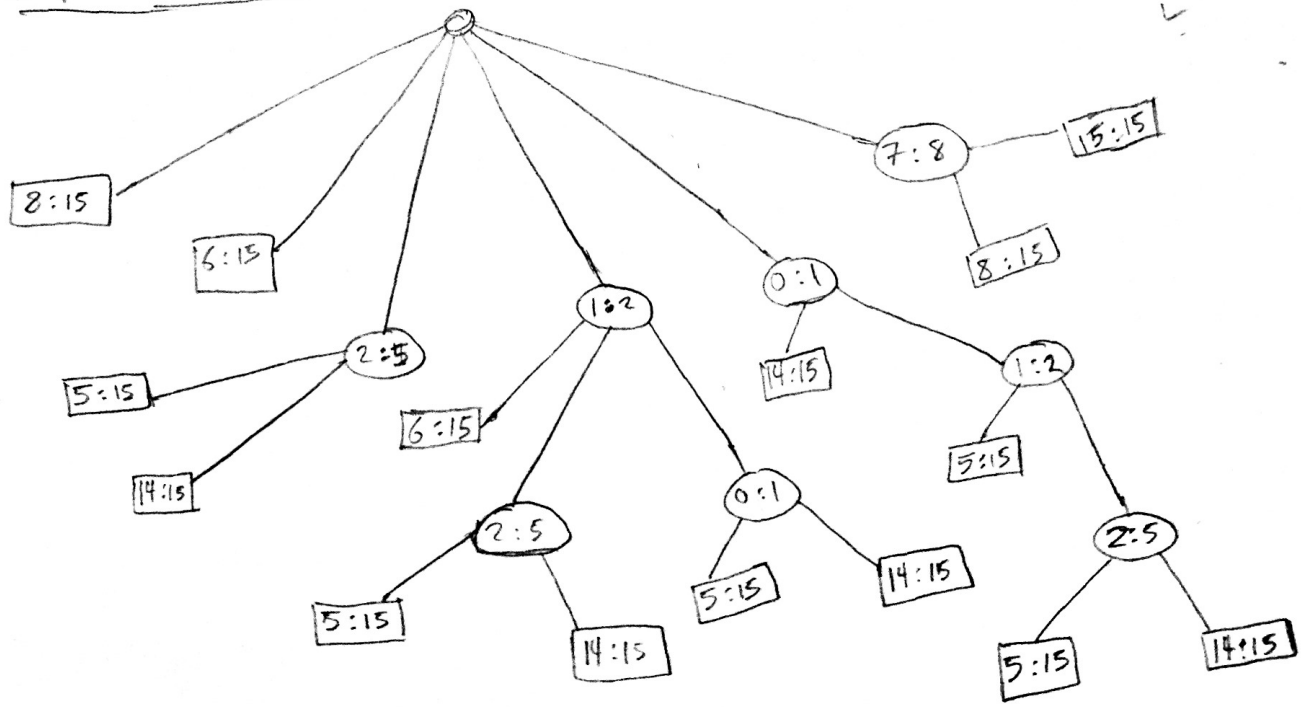
R.-(3.14)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m | i | n | i | m | i | z | c | _ | m | i | n | i | m | r | $ |

Suffix trie




Compressed Suffix trie



A. -13.5)

Given a standard trie $T$ and a string $s$ to delete from the trie, search the trie for the given string (traverse paths of the trie checking for char's in order of given string).

If the search was unsuccessful, return the original trie.

Else, let $u$ be node where $s$ was found.

- check if $s$ is not equal to string ending at $u$, or has a child.
  - if so return trie; else:
  - set $v$ = $u$'s parent.
    - delete $u$ from trie
  - if $v$ has a child
    - set $v$'s string to concatenation of $v$'s and child strings
      (v.string += child.string)
    - then delete $v$'s child node
- Return the new trie w/ string removed.

Time complexity: $O(n \cdot m)$, where $n$ = alphabet size and $m$ = depth of leaf representing the string to be removed.