Unit 6 - Tries

CS 201 - Data Structures II

Spring 2022

Habib University

Syeda Saleha Raza

Abundance of text data

- Some examples:
  - Snapshots of the World Wide Web, as Internet document formats HTML and XML are primarily text formats, with added tags for multimedia content
  - All documents stored locally on a user's computer
  - Email archives
  - Customer reviews
  - Compilations of status updates on social networking sites such as Facebook
  - Feeds from microblogging sites such as Twitter and Tumblr

Trie

- Tries is a tree-based data structure of storing strings that support fast pattern matching.
- Primarily support pattern matching and string matching.

Trie

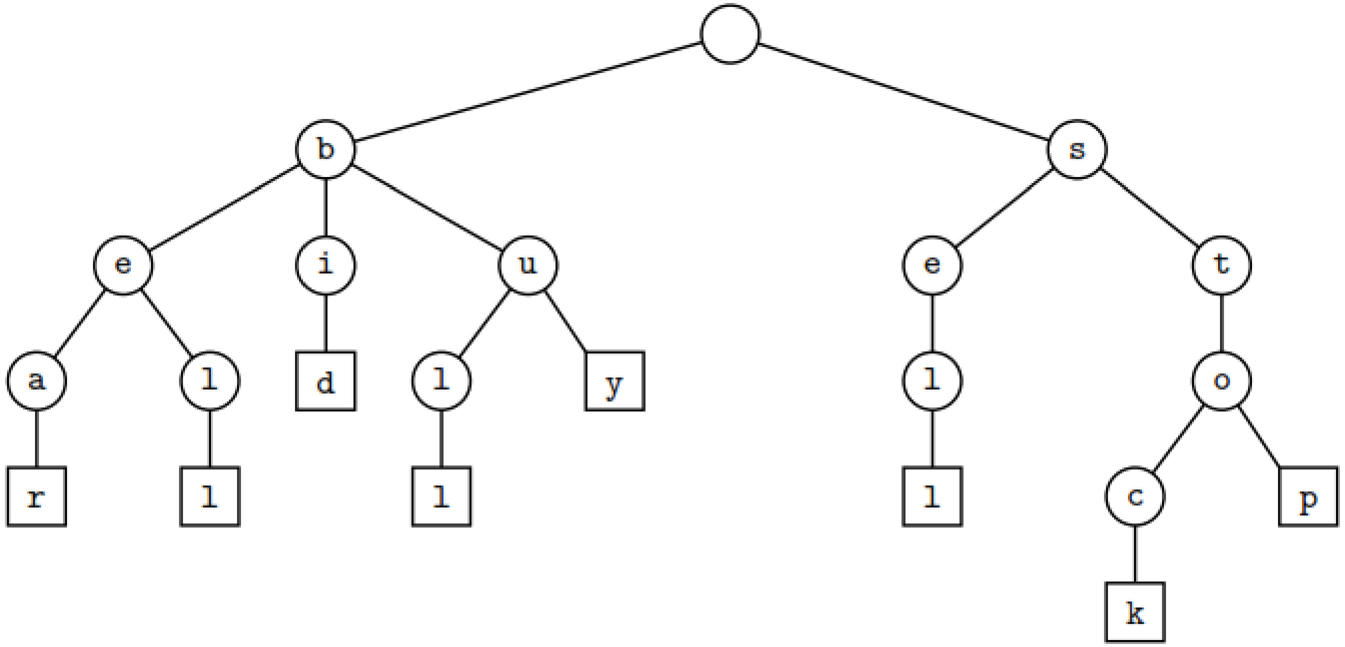


Figure 13.10: Standard trie for the strings {bear, bell, bid, bull, buy, sell, stock, stop}.

Let S be a set of s strings from alphabet Σ such that no string in S is a prefix of another string. A ***standard trie*** for S is an ordered tree T with the following properties (see Figure 13.10):

- Each node of T, except the root, is labeled with a character of Σ.
- The children of an internal node of T have distinct labels.
- T has s leaves, each associated with a string of S, such that the concatenation of the labels of the nodes on the path from the root to a leaf v of T yields the string of S associated with v.

Proposition 13.6: *A standard trie storing a collection S of s strings of total length n from an alphabet Σ has the following properties:*

- *The height of T is equal to the length of the longest string in S.*
- *Every internal node of T has at most $|\Sigma|$ children.*
- *T has s leaves*
- *The number of nodes of T is at most $n+1$.*

The worst case for the number of nodes of a trie occurs when no two strings share a common nonempty prefix; that is, except for the root, all internal nodes have one child.

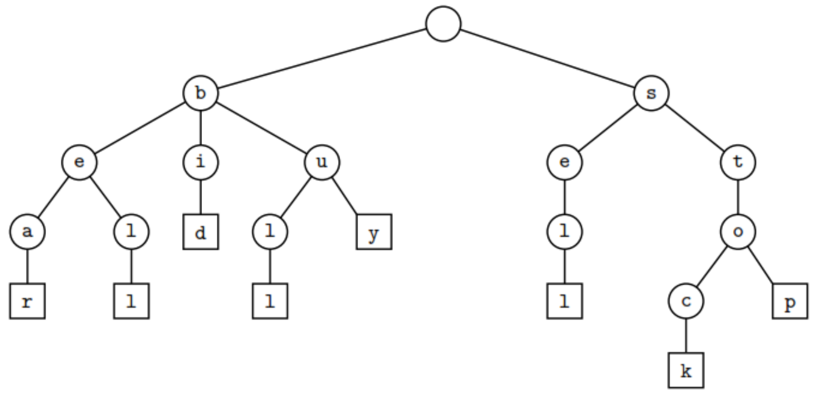
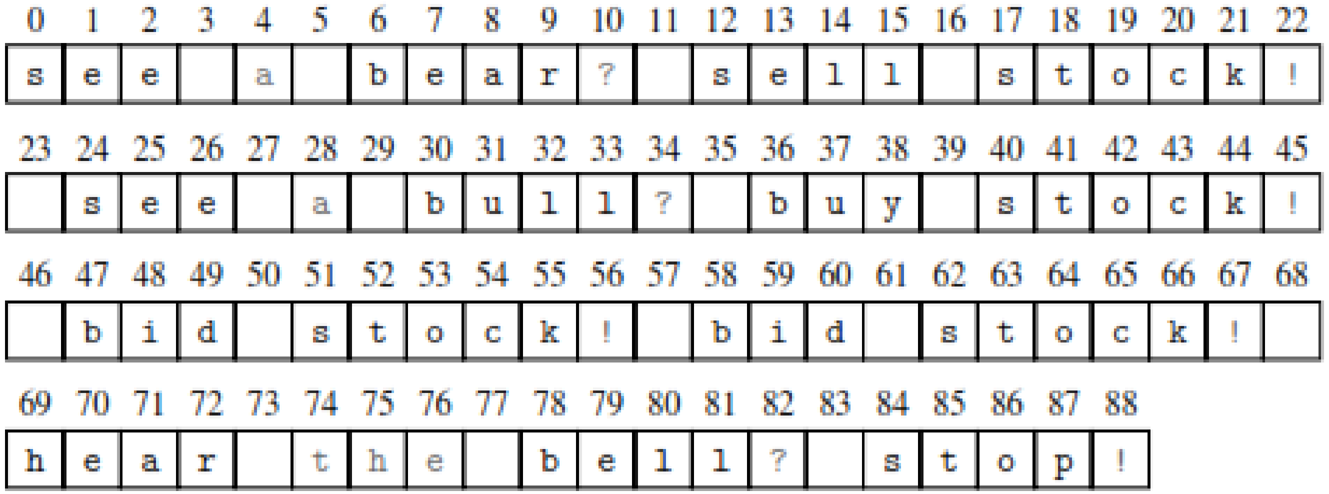


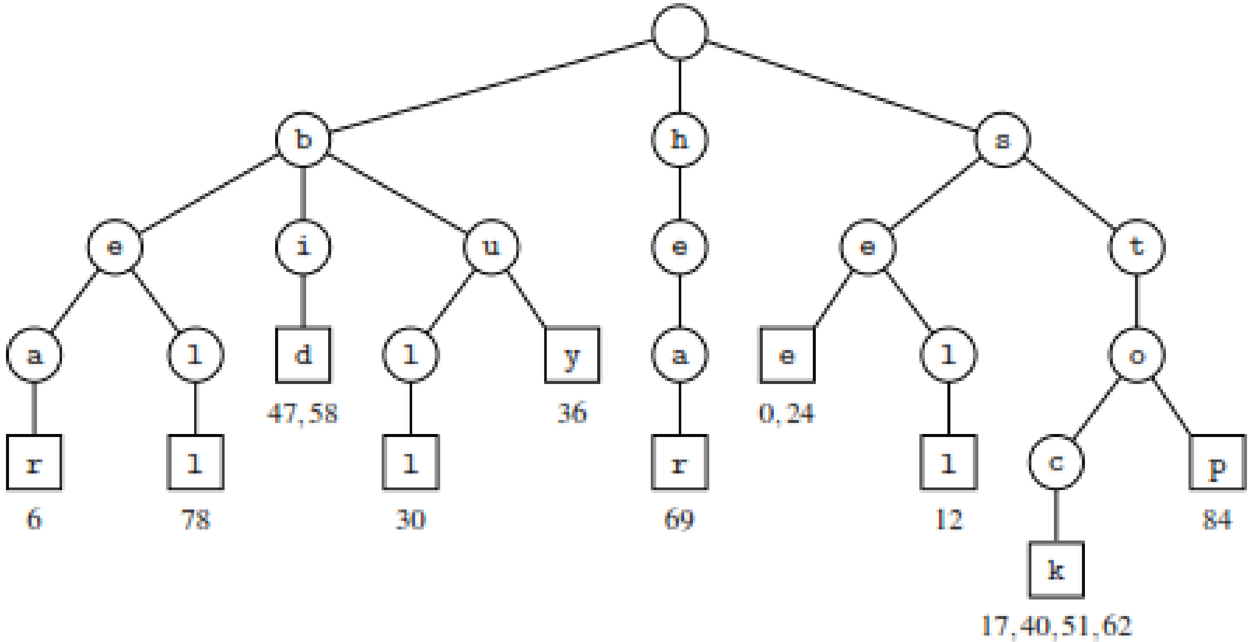
Figure 13.10: Standard trie for the strings {bear, bell, bid, bull, buy, sell, stock, stop}.

Example



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
s e e a b e a r ? s e l l s t o c k !
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
s e e a b u l l ? b u y s t o c k !
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
b i d s t o c k ! b i d s t o c k !
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
h e a r t h e b e l l ? s t o p !

(a)



b
h
s
e
i
u
e
e
t
a
l
d
l
y
a
e
l
o
r
l
47, 58
l
36
r
0, 24
l
c
p
6
78
30
69
12
84
k
17, 40, 51, 62

Exercise

- Let’s construct a trie for the given set of words:
  - {game, gamble, photos, blue, phone, gang, salute, bubble, salient, black, fear, blunt, fun}
- Build the compressed trie from the set of strings given above.

Trie – Delete a key

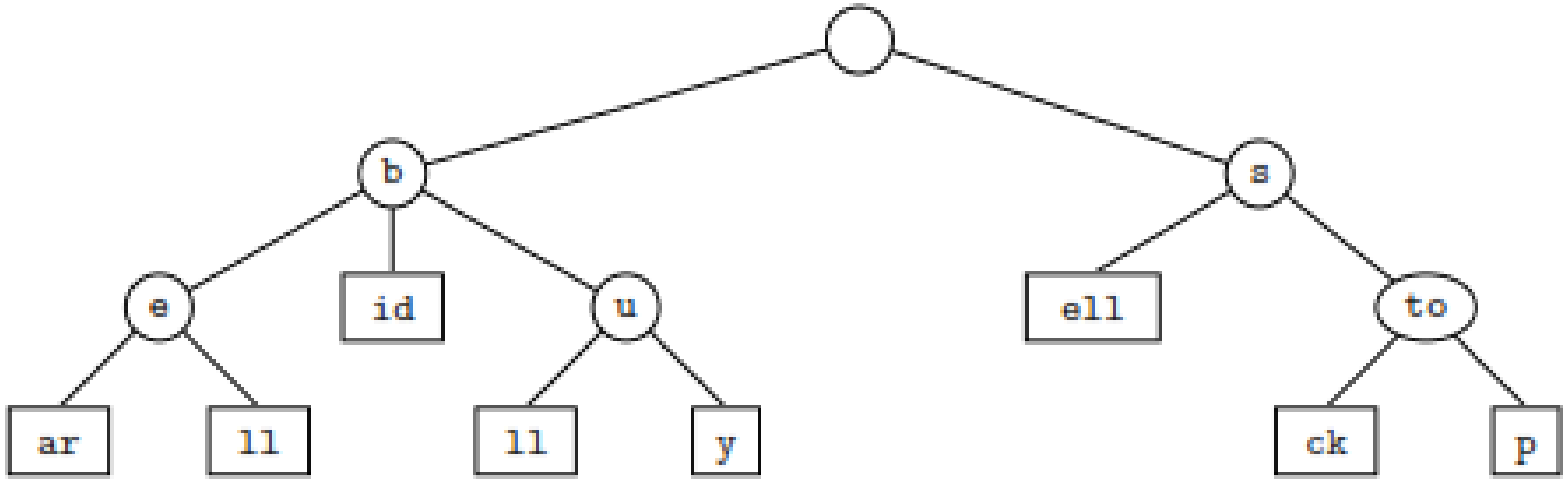
- During delete operation we delete the key in bottom up manner using recursion. The following are possible conditions when deleting key from trie,

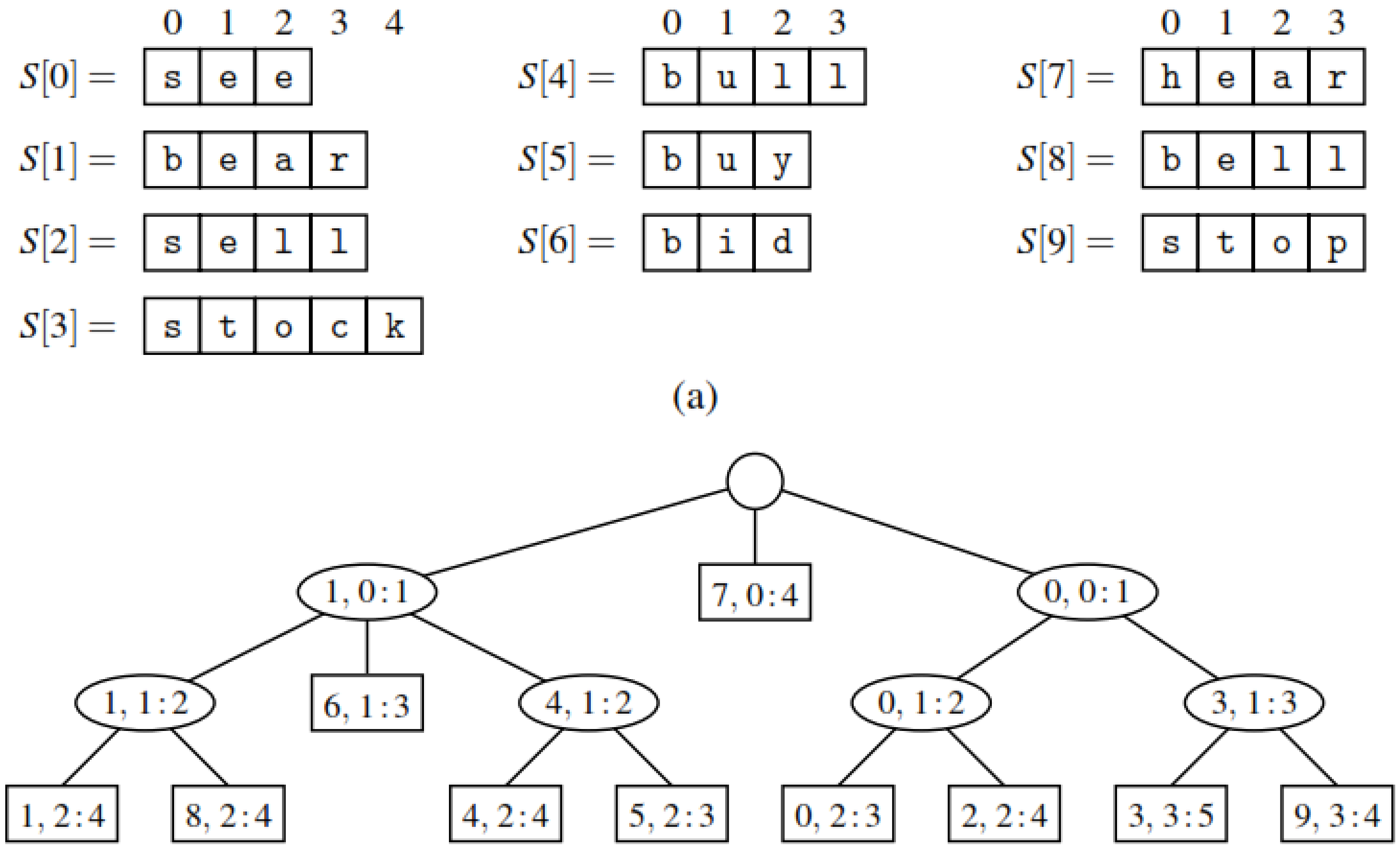
1. Key may not be there in trie. Delete operation should not modify trie.
2. Key present as unique key (no part of key contains another key (prefix), nor the key itself is prefix of another key in trie). Delete all the nodes.
3. Key is prefix key of another long key in trie. Unmark the leaf node.
4. Key present in trie, having atleast one other key as prefix key. Delete nodes from end of key until first leaf node of longest prefix key.

Complexity

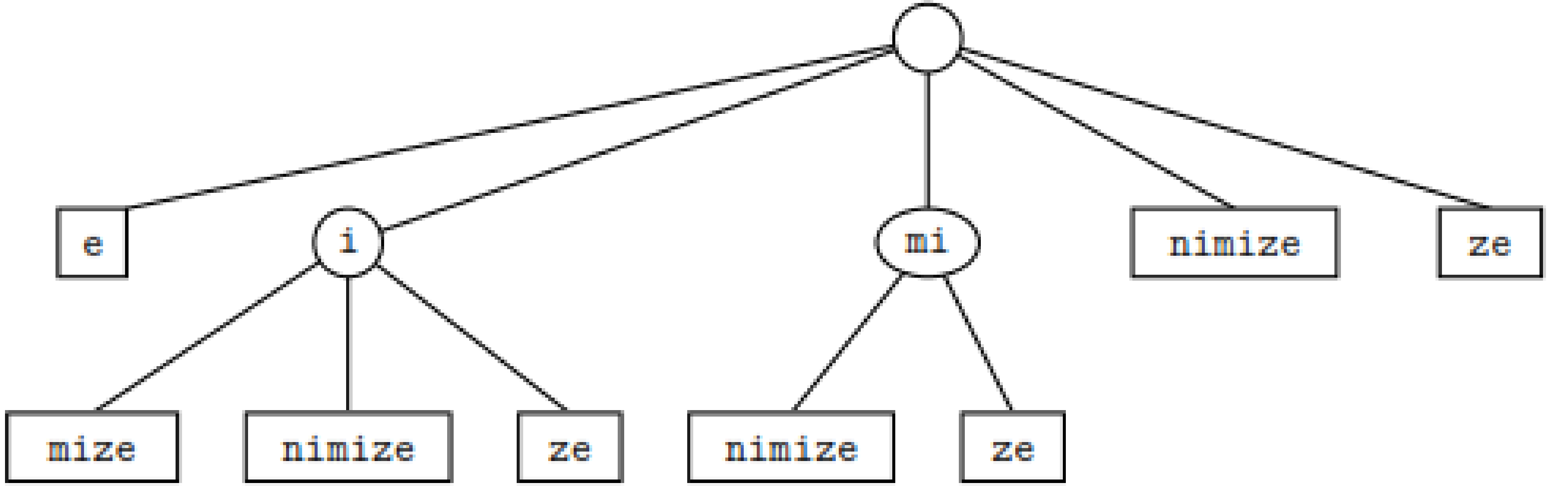
- Assumes no string is a prefix of another string.
- Search in trie
  - O(m.|alphabets|)
  - will further reduce for small alphabet size (like DNA string with {A,C,G,T})
  - Using secondary structure for each node (like a table or hashtable)
- Insertion

Compressed Trie




0 1 2 3 4
S[0] = s e e
S[1] = b e a r
S[2] = s e l l
S[3] = s t o c k
0 1 2 3
S[4] = b u l l
S[5] = b u y
S[6] = b i d
0 1 2 3
S[7] = h e a r
S[8] = b e l l
S[9] = s t o p
(a)
1, 0:1
7, 0:4
0, 0:1
1, 1:2
6, 1:3
4, 1:2
0, 1:2
3, 1:3
1, 2:4
8, 2:4
4, 2:4
5, 2:3
0, 2:3
2, 2:4
3, 3:5
9, 3:4

Suffix Trie



(a)

Suffix Trie

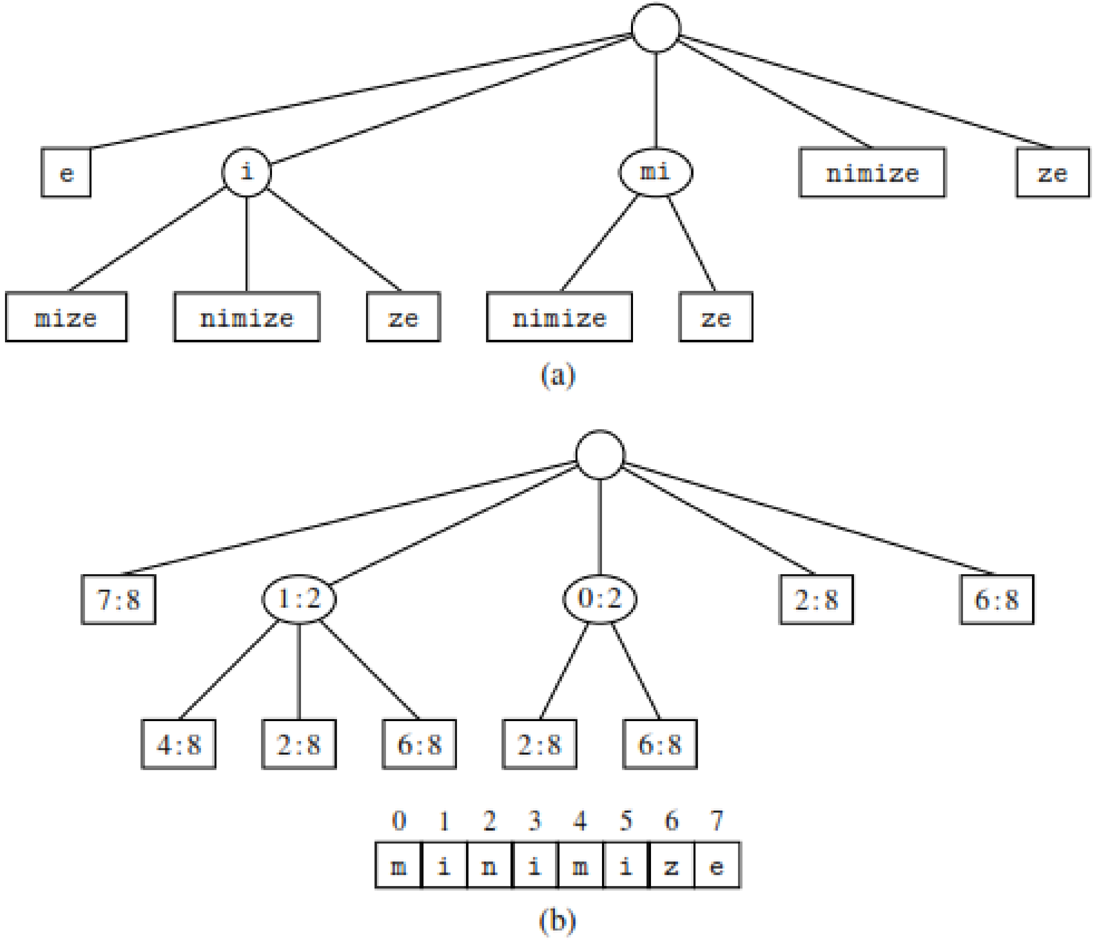


Figure 13.14: (a) Suffix trie T for the string X = "minimize". (b) Compact representation of T, where pair $j:k$ denotes slice $X[j:k]$ in the reference string.

Applications

- Search Engines
- Auto-complete
- Spell checker

Resources

- Open Data Structures (pseudocode edition), by Pat Morin. Available online at http://opendatastructures.org
- Data Structures and Algorithms in Python, by Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser. 2013. (1st. ed.). Wiley Publishing

Thanks