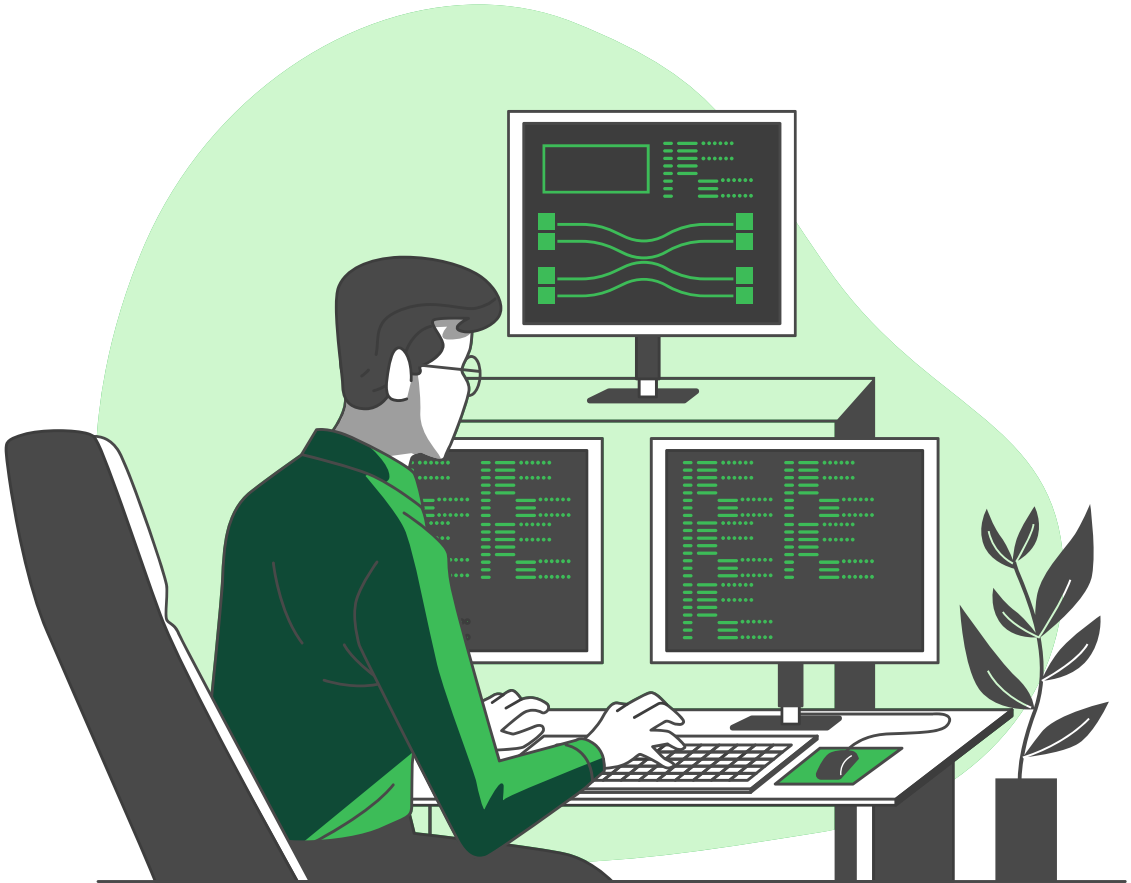
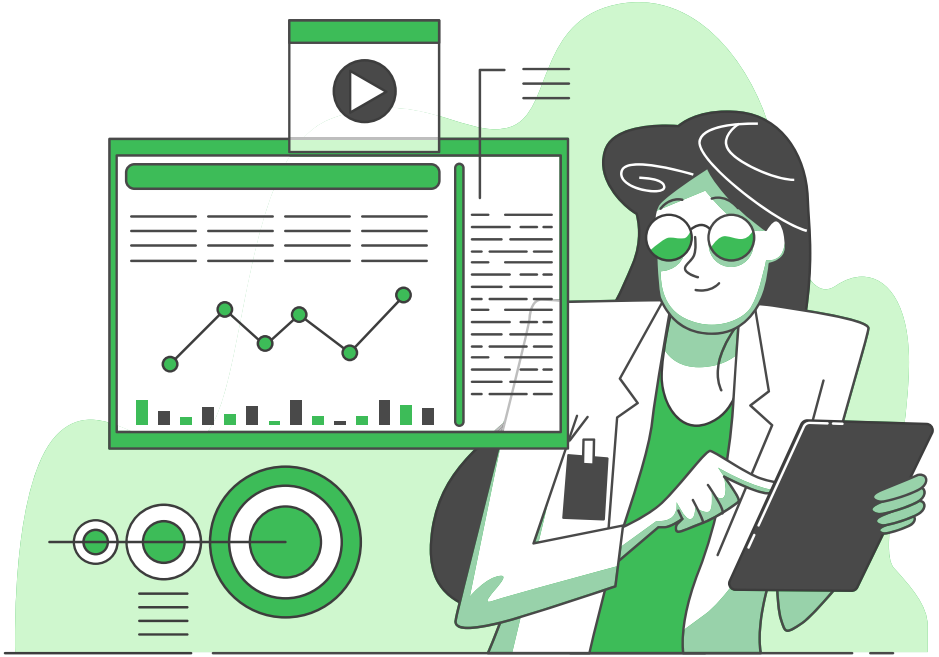


Trie

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents

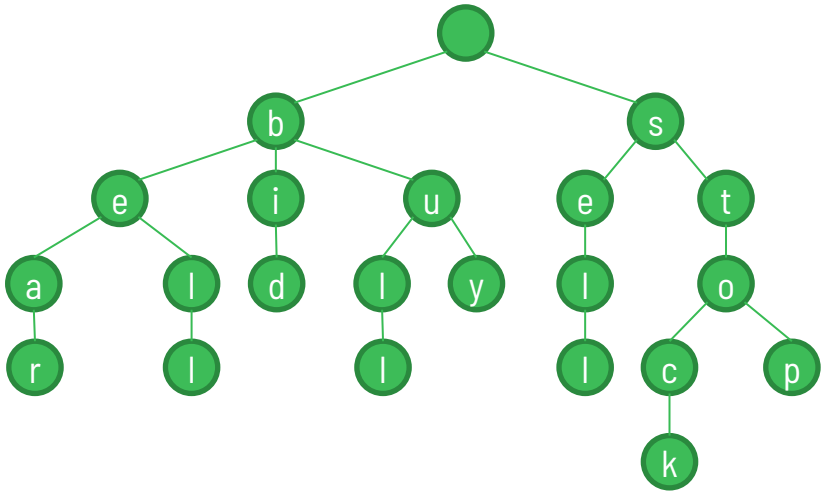


01

Standard Trie

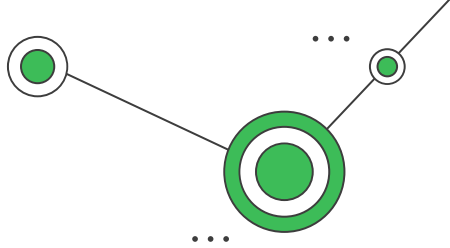
Pattern matching data structure

Trie



A **trie** (pronounced “try”) is a tree-based data structure for storing strings in order to support **fast pattern matching**.

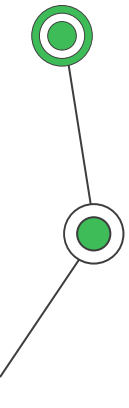
Trie Operations



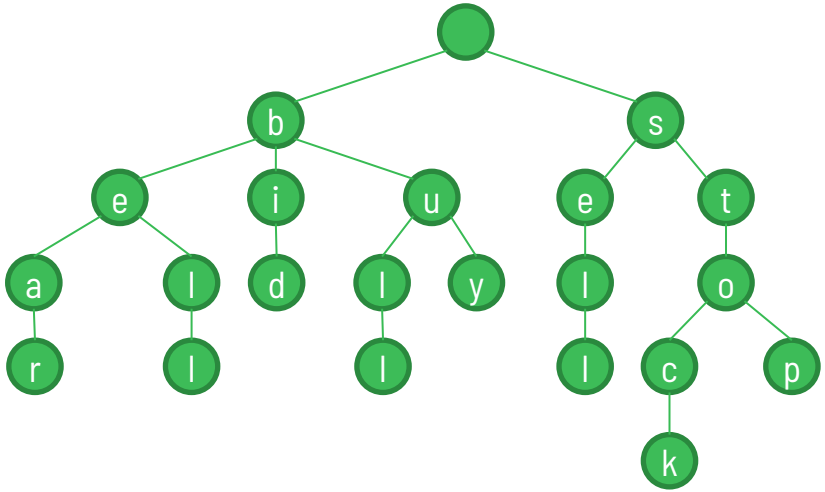
Given a collection S of strings, all defined with the same alphabet Σ:

1. Efficiently search for a pattern string P (i.e., **pattern matching**)

2. Efficiently search for all strings in S that contains a pattern string P as a prefix (i.e., **prefix matching**)



Standard Trie



A standard trie for a set of strings S is an ordered tree such that:

- The children of a node are alphabetically ordered.
- The paths from the external nodes to the root yield the strings of S.
- **Assumption:** no string $w \in S$ is a prefix of another string in S.

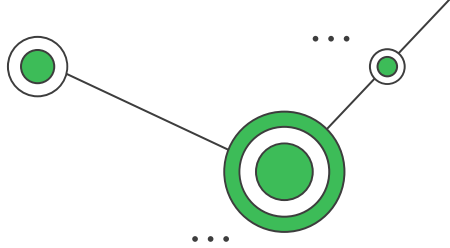
Constructing a Standard Trie

The number of children of the root node is the number of distinct first letters in all the words in the input string.

Example: How many children would the root of a standard trie have given the following sets of strings?

- S = {apple, aardvark, animal, awesome}
- S = {xylophone, zebra, penguin, violin, yellow}
- S = {CAGT, AGTC, GATC}

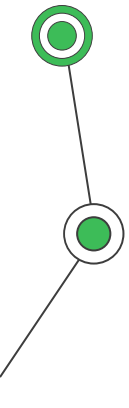
Constructing a Standard Trie



Recall the assumptions that no string in S is a prefix of another string in S.

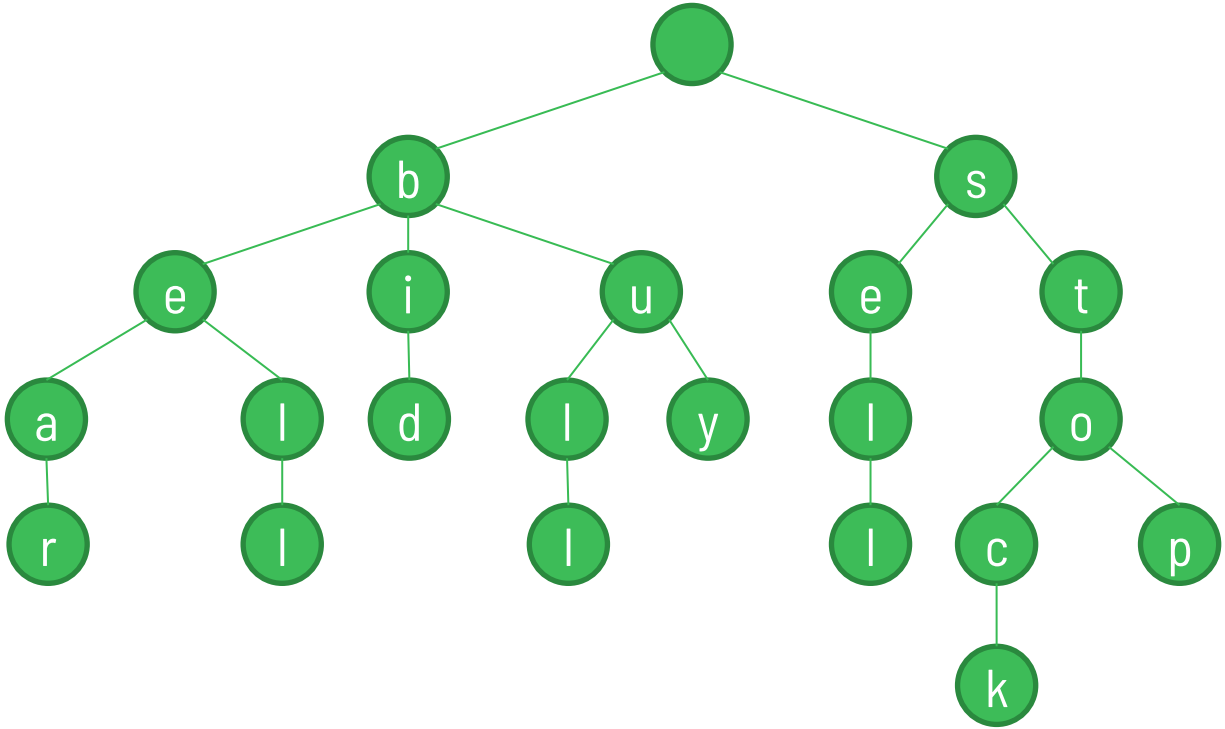
To insert a string X in a trie T:

- Try to trace the path associated with X in T.
- If you reach an external node, you have found X, so update the node to reflect the location of this instance.
- Else, you are stopped at an internal node, and you must create a new chain of node descendants for the rest of X.

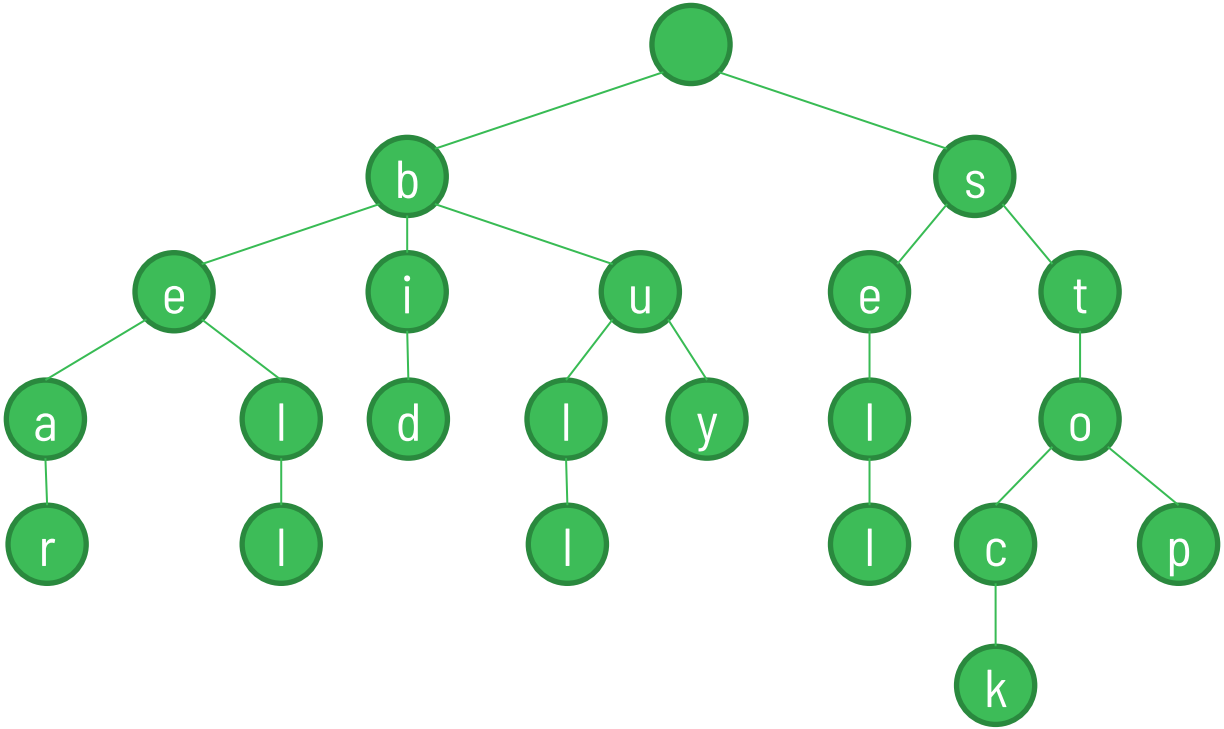


Build it!

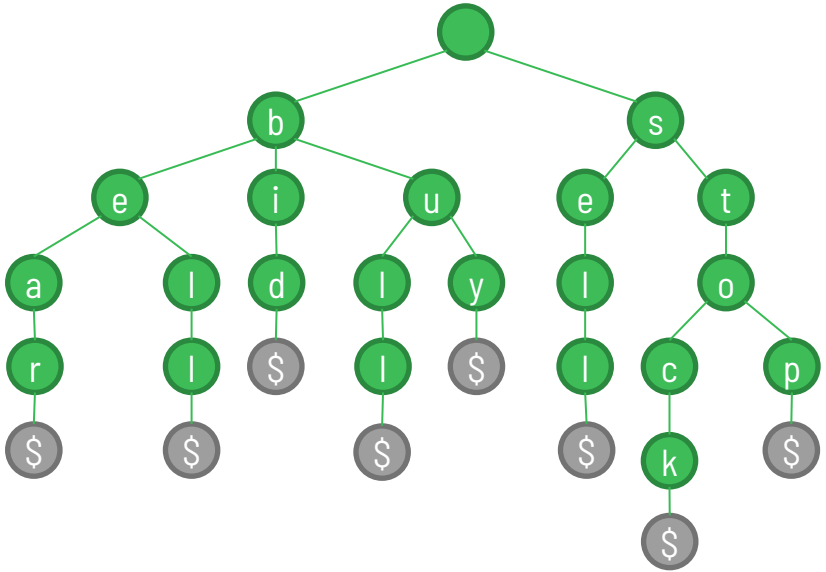
S = {bear, bell, bid, bull, buy, sell, stock, stop}



Add “be”



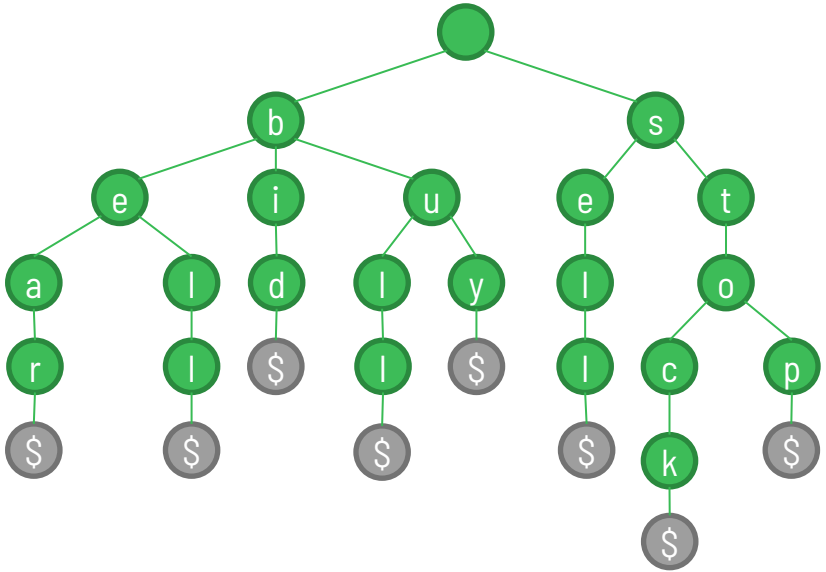
Problematic Assumption



Assumption: no string in $w \in S$ is a prefix of another string in S.

Q: What if we have a string that is a prefix of another string in S?
A: Append an '$' to each string in $w \in S$.

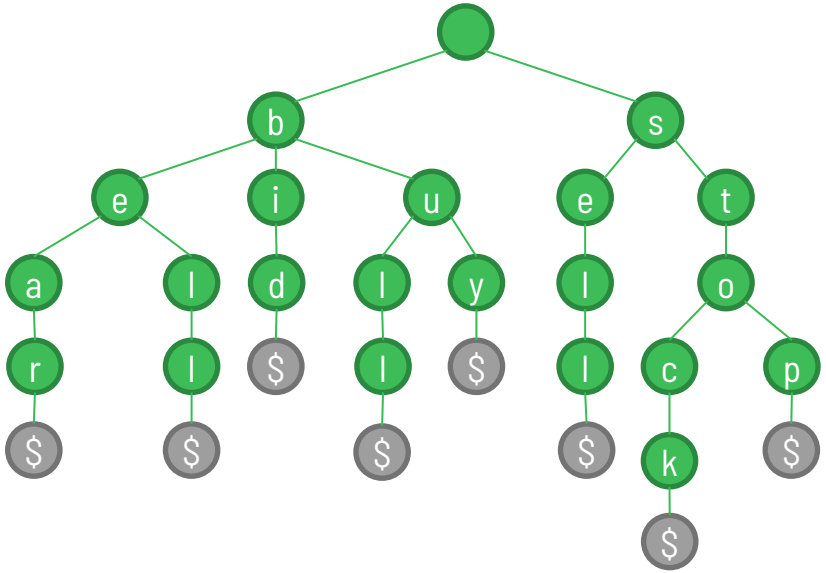
Trie Analysis (Space)



Let T be a standard trie storing a collection S of $|S|$ strings of total length $n = \sum_{w \in S} |w|$ from an alphabet Σ of size d.

- Every internal node of T has at most d children.
- T has $|S|$ external nodes.
- The height of T is equal to the length of the longest string in S.
- The number of nodes of T is $O(n)$.

Trie Analysis (Time)



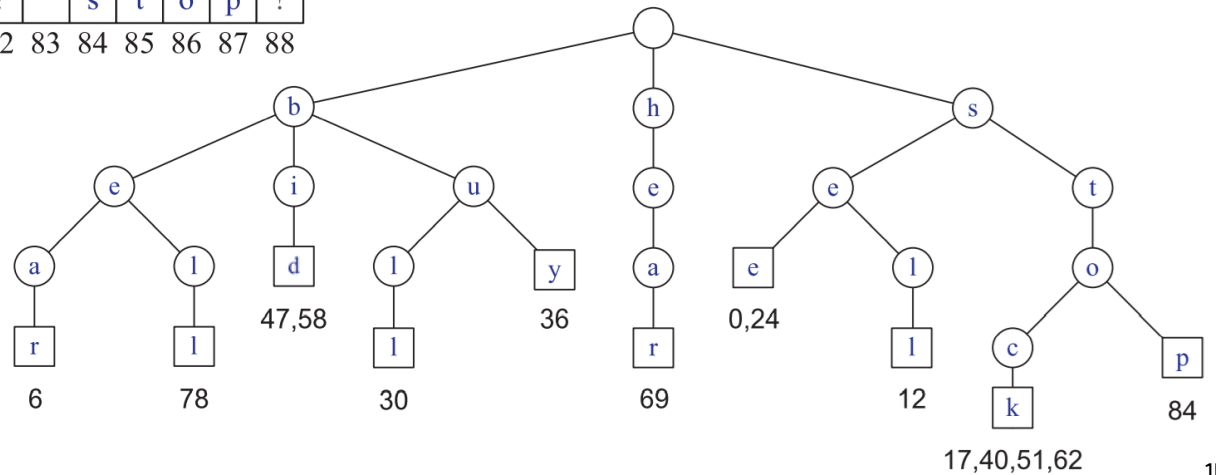
Search, Insert, and delete time complexity?

Generally speaking: $O(L)$, where L is the average length of the strings in T.

For a single string w: $O(|w|)$

Word Matching in a Trie

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

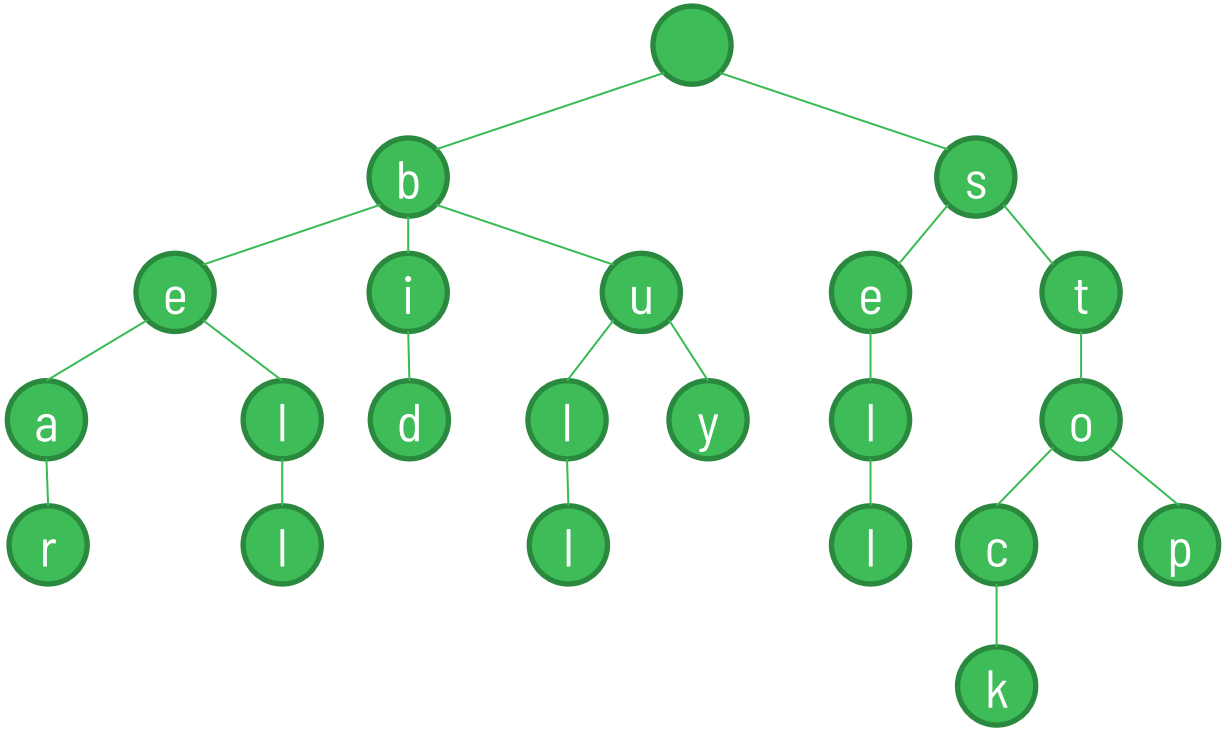


02

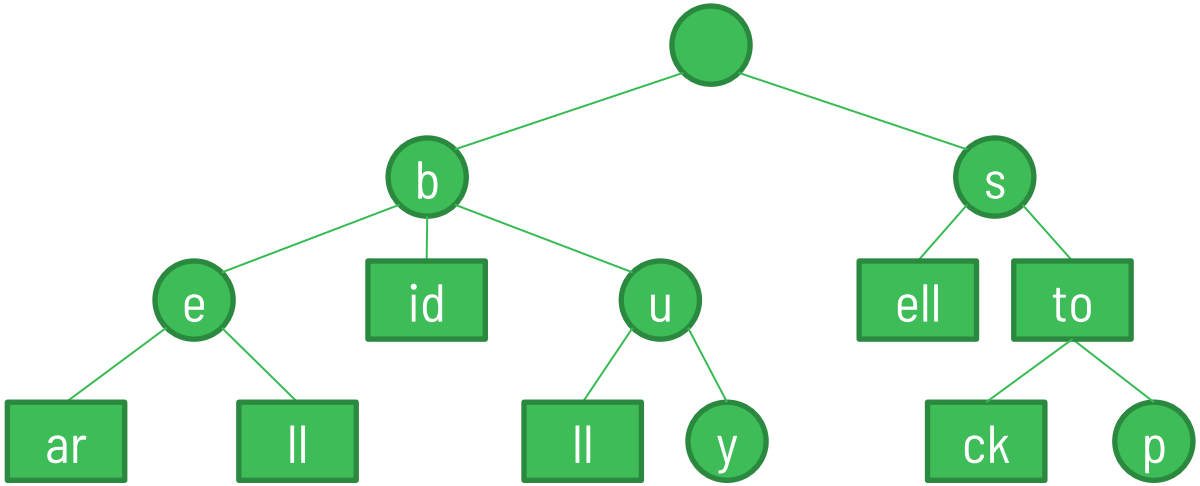
PATRICIA

Yep, that is the name

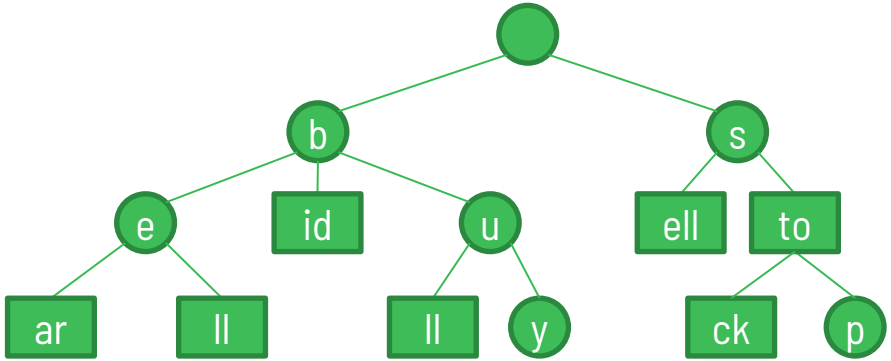
How would you improve this trie?



How would you improve this trie?



PATRICIA Trie



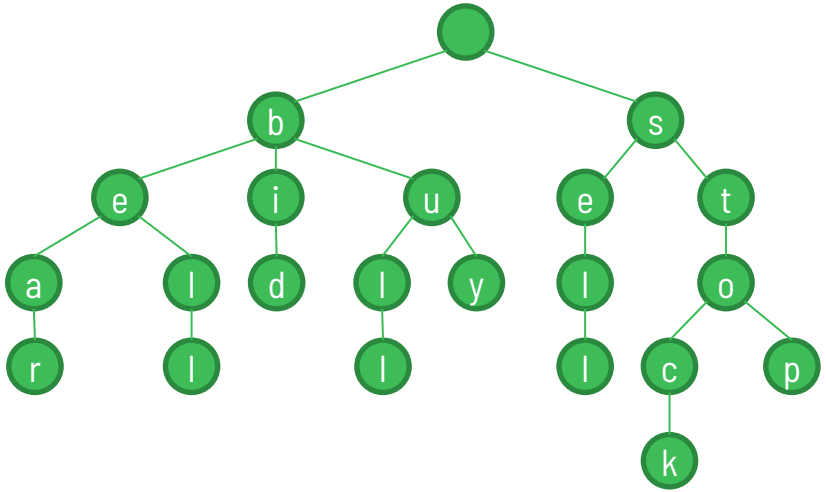
Practical **A**lgori**T**hm to **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric

Also known as Compressed trie. PATRICIA is a variant of a Radix Tree.

Motivation: Ensure that each internal node has at least two children.

Method: Compress chains of single-child nodes into individual edges.

PATRICIA Definitions



Let T be a standard trie. An internal node $v \in T$ is redundant if v has one child and is not the root.

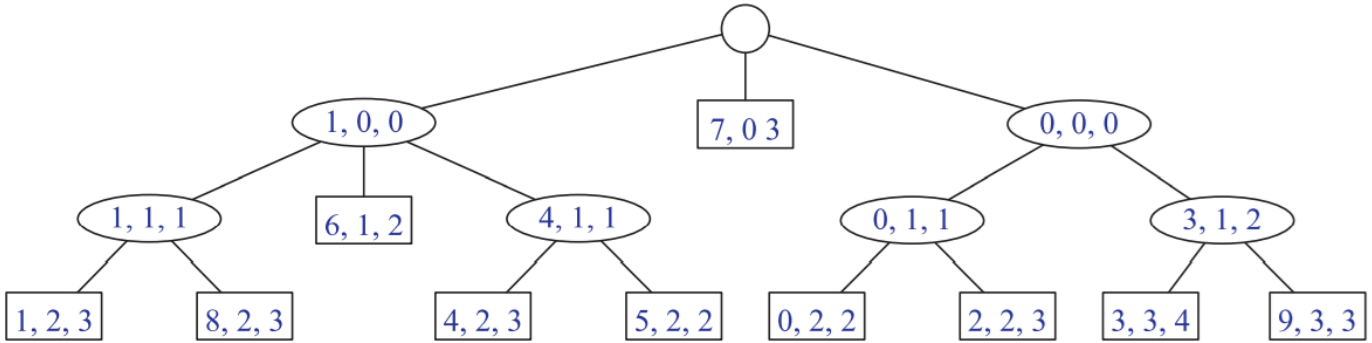
How many redundant nodes there are in this trie?

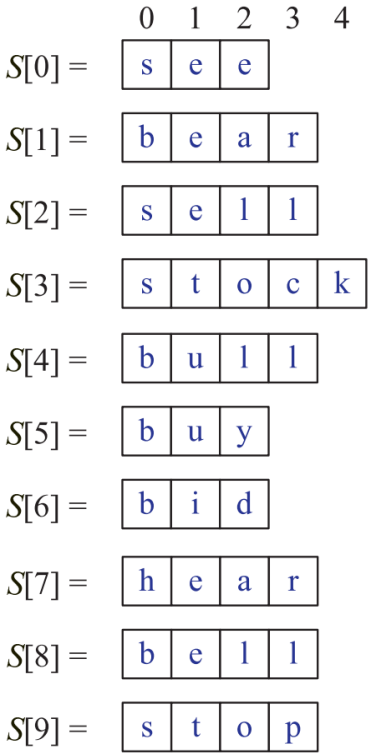
A chain of $k \geq 2$ edges $(v_0, v_1)(v_1, v_2) \ldots (v_{k-1}, v_k)$ is redundant if:

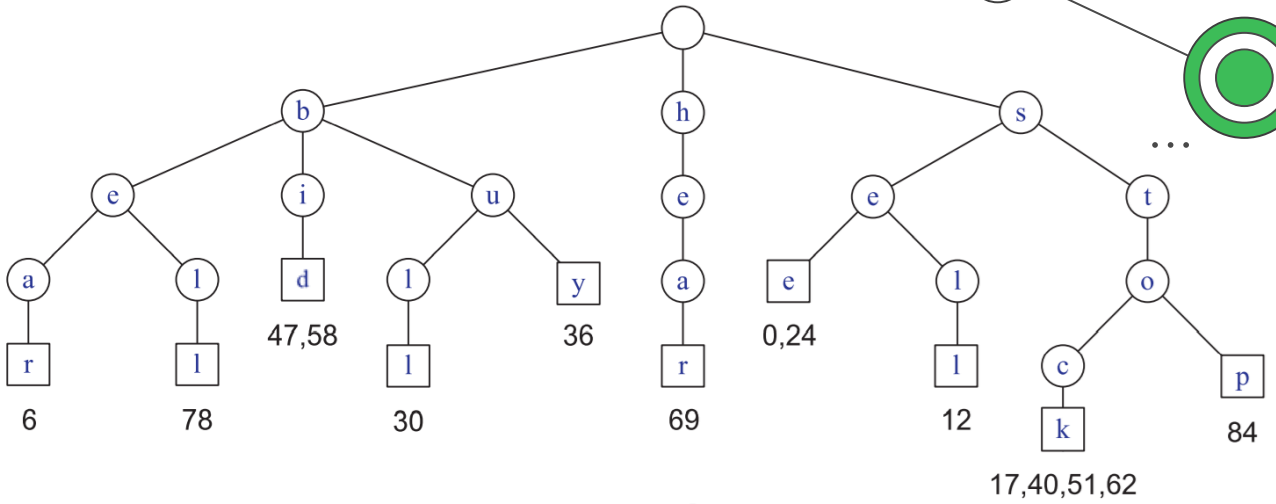
- v_i is redundant for $i = 1 \ldots k-1$
- v_0 and v_k are not redundant.

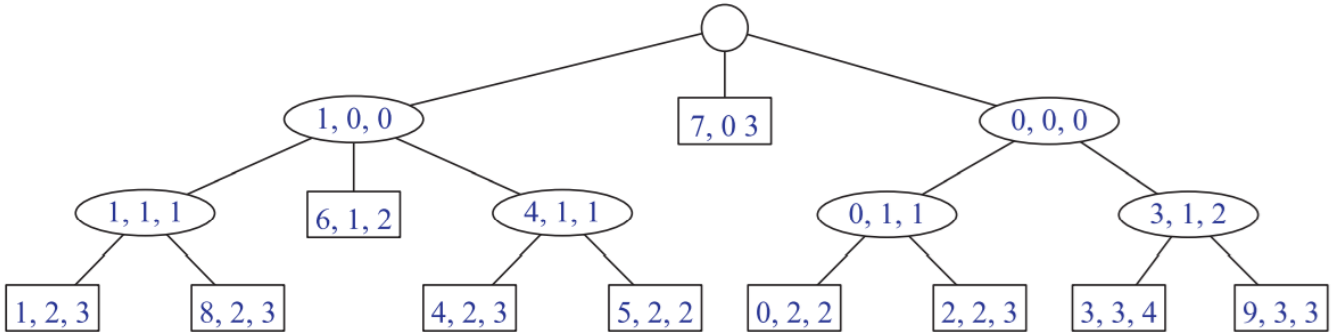
Compact Representation

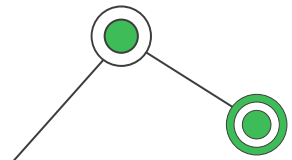
	0	1	2	3	4
$S[0]$ =	s	e	e		
$S[1]$ =	b	e	a	r	
$S[2]$ =	s	e	l	l	
$S[3]$ =	s	t	o	c	k
$S[4]$ =	b	u	l	l	
$S[5]$ =	b	u	y		
$S[6]$ =	b	i	d		
$S[7]$ =	h	e	a	r	
$S[8]$ =	b	e	l	l	
$S[9]$ =	s	t	o	p	



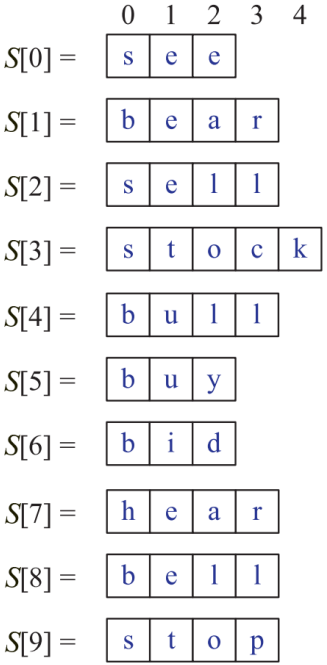

0 1 2 3 4
S[0] = s e e
S[1] = b e a r
S[2] = s e l l
S[3] = s t o c k
S[4] = b u l l
S[5] = b u y
S[6] = b i d
S[7] = h e a r
S[8] = b e l l
S[9] = s t o p

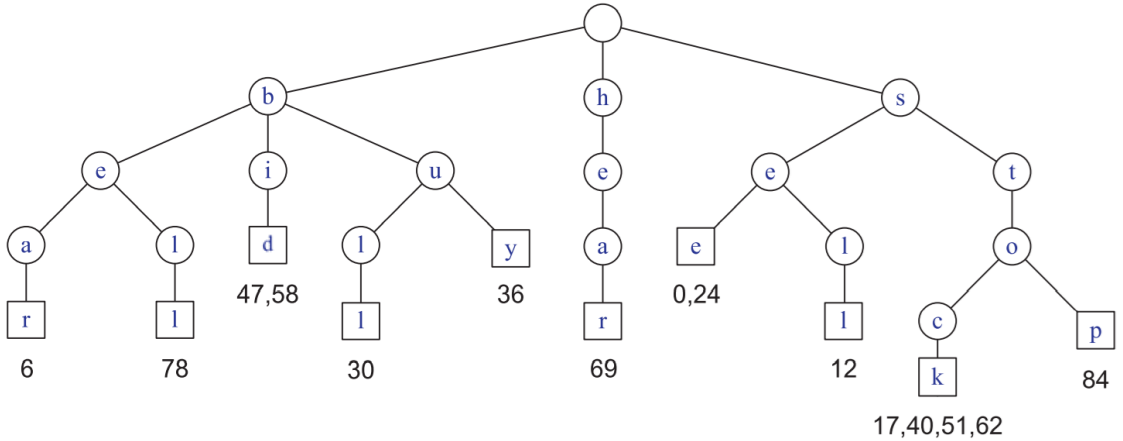

b
h
s
e
i
u
e
e
t
a
l
d
l
y
a
e
l
o
r
l
47,58
l
36
r
0,24
l
c
p
6
78
30
69
12
k
84
17,40,51,62

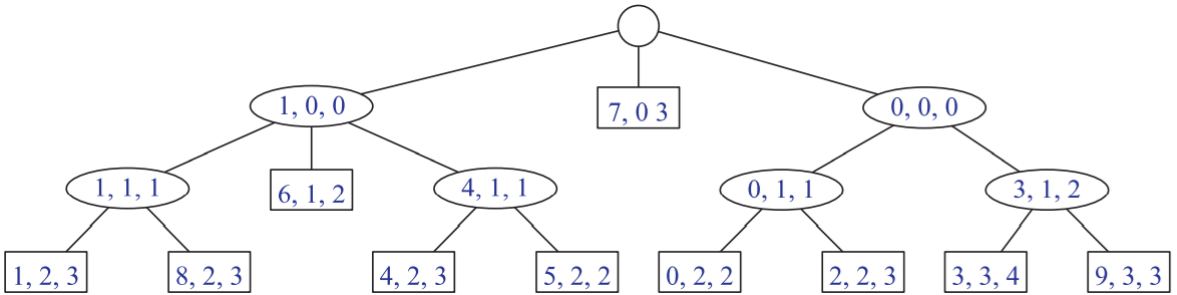

1, 0, 0
7, 0 3
0, 0, 0
1, 1, 1
6, 1, 2
4, 1, 1
0, 1, 1
3, 1, 2
1, 2, 3
8, 2, 3
4, 2, 3
5, 2, 2
0, 2, 2
2, 2, 3
3, 3, 4
9, 3, 3



Search “ear”, “to”, “id”, “top”







We tried!

Do you have any questions?