

Tries

CSc 372: Comparative Programming Languages





What are tries?

- A trie is a data structure meant for efficient data retrieval.
- The idea is to build a structure that is specific to the data set and then allows quick searches for specific items.
- Applications: searching texts, genomic databases

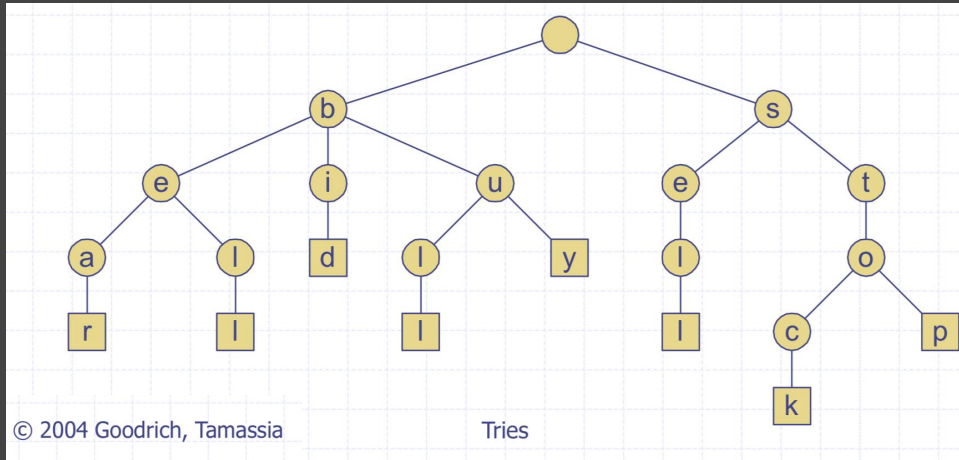


Tries tend to work well when...

- ... you have a more or less fixed text or database
- ... you have the expectation of many searches


This is because most of the work is done up front when building the trie. After that, searching for a string takes time proportional to the length of the string. It's worth doing all the work up front if the scenario meets the criteria above. Otherwise, it might make more sense to use a different algorithm or data structure.

Standard Tries




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

- Each node but the root is labeled with a character.
- 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 in S is a prefix of another string in S .



**Doesn't the restriction
that no string be a prefix
of another string
diminish the usefulness
of tries?**



Doesn't the restriction that no string be a prefix of another string diminish the usefulness of tries?

NOPE!

(There's a very simple way of handling this.)





Consider the text:

canaries can help detect stack overflows

**We should be able to search for both “can”
and “canaries”...**

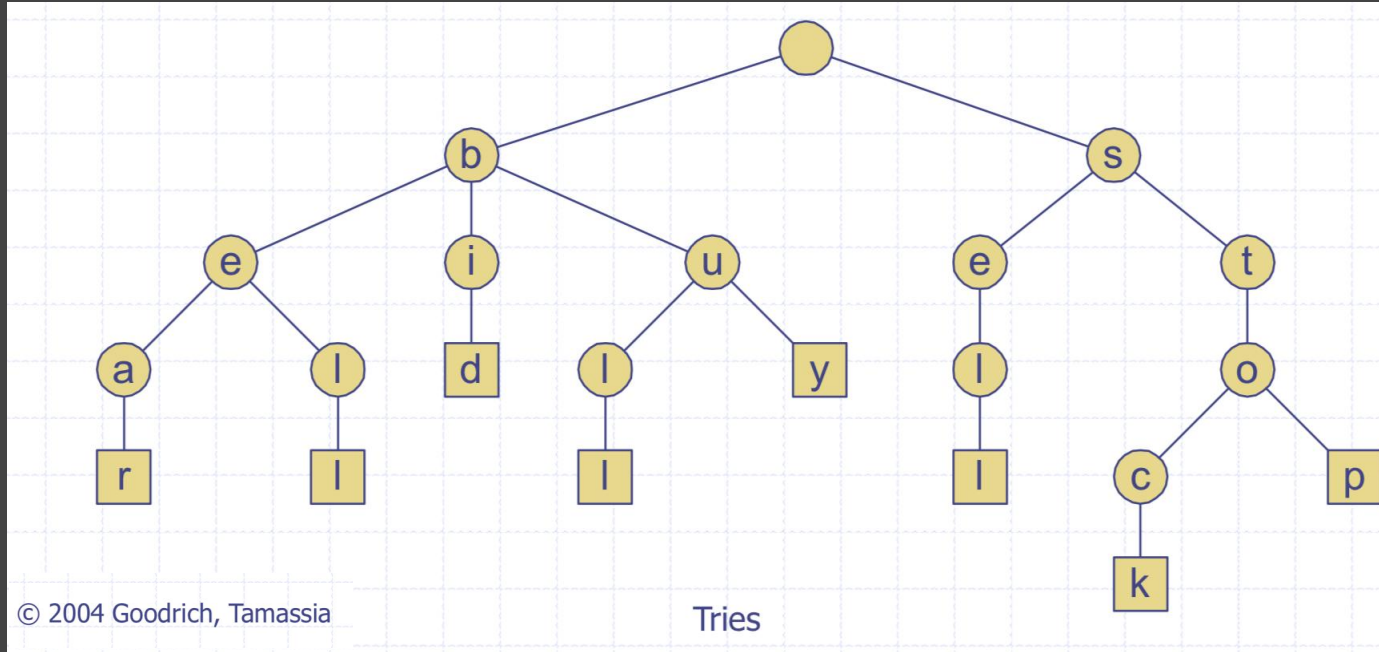
Consider the text:
canaries can help detect stack overflows

**We should be able to search for both “can” and
“canaries”...**

**...so we just add an arbitrary character not in the
original alphabet to the end of each string:
{canaries\$, can\$, help\$, detect\$, stack\$,
overflows\$}**

Standard Trie Example

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

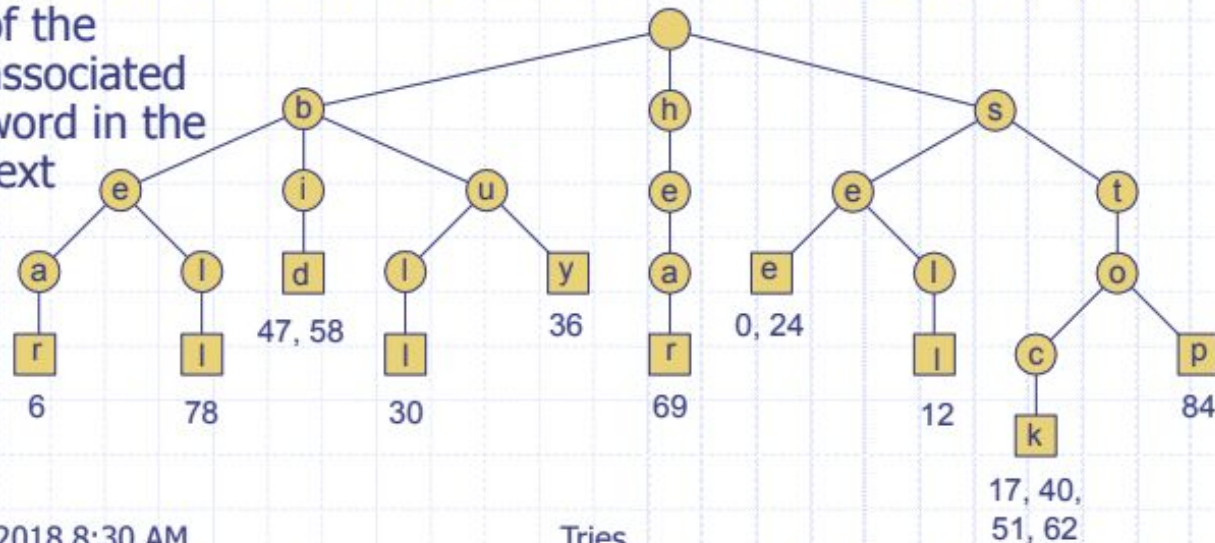


Word Matching with a Trie

- ◆ We insert the words of the text into a trie

- ◆ Each leaf stores the occurrences of the associated word in the text

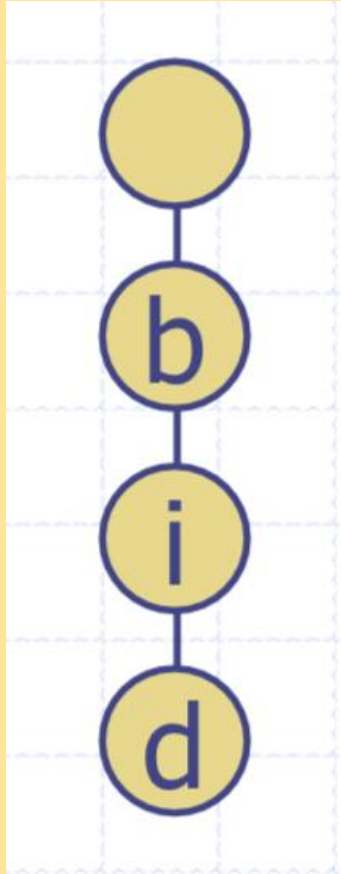
s	e	e		a		b	e	a	r	?		s	e	l	l		s	t	o	c	k	!		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
s	e	e		a		b	u	l	l	?		b	u	y		s	t	o	c	k	!			
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46		
b	i	d		s	t	o	c	k	!		b	i	d		s	t	o	c	k	!				
47	48	49	50	51	52	53	54	55	56	57	58	59	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					





**Example: Build a trie for
 $S = \{\text{bid}\}$**

Example: Build a trie for $S = \{\text{bid}\}$

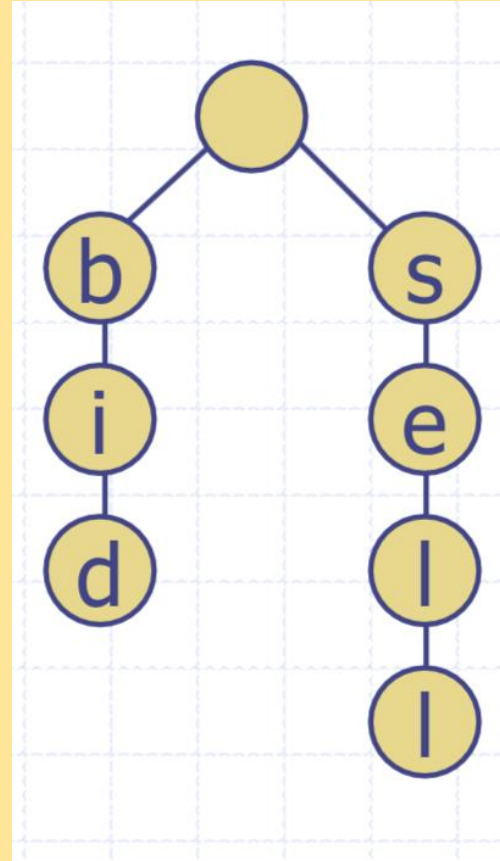


Example: add “sell”

$S = \{\text{bid}, \text{sell}\}$

Example: add "sell"

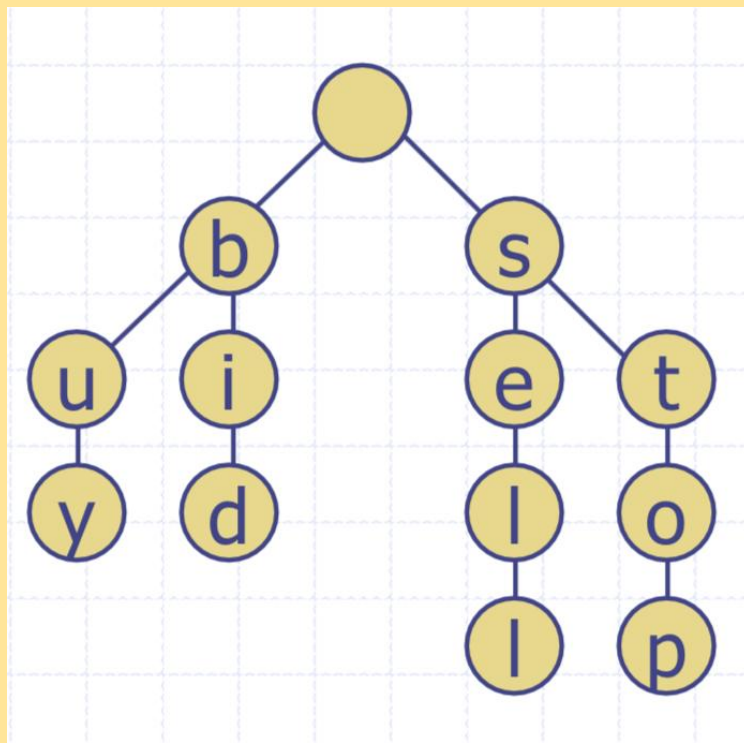
$S = \{\text{bid}, \text{sell}\}$



Example: Add “buy” and “stop”
 $S = \{\text{bid}, \text{sell}, \text{buy}, \text{stop}\}$

Example: Add “buy” and “stop”

$S = \{\text{bid}, \text{sell}, \text{buy}, \text{stop}\}$

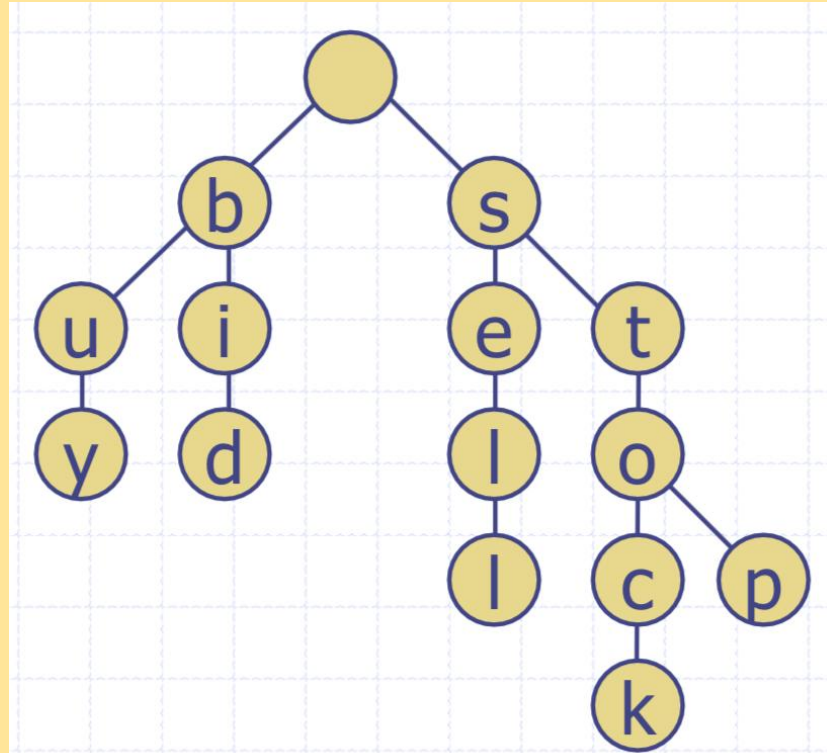


Example: Add “stock”

$S = \{\text{bid}, \text{sell}, \text{buy}, \text{stop}, \text{stock}\}$

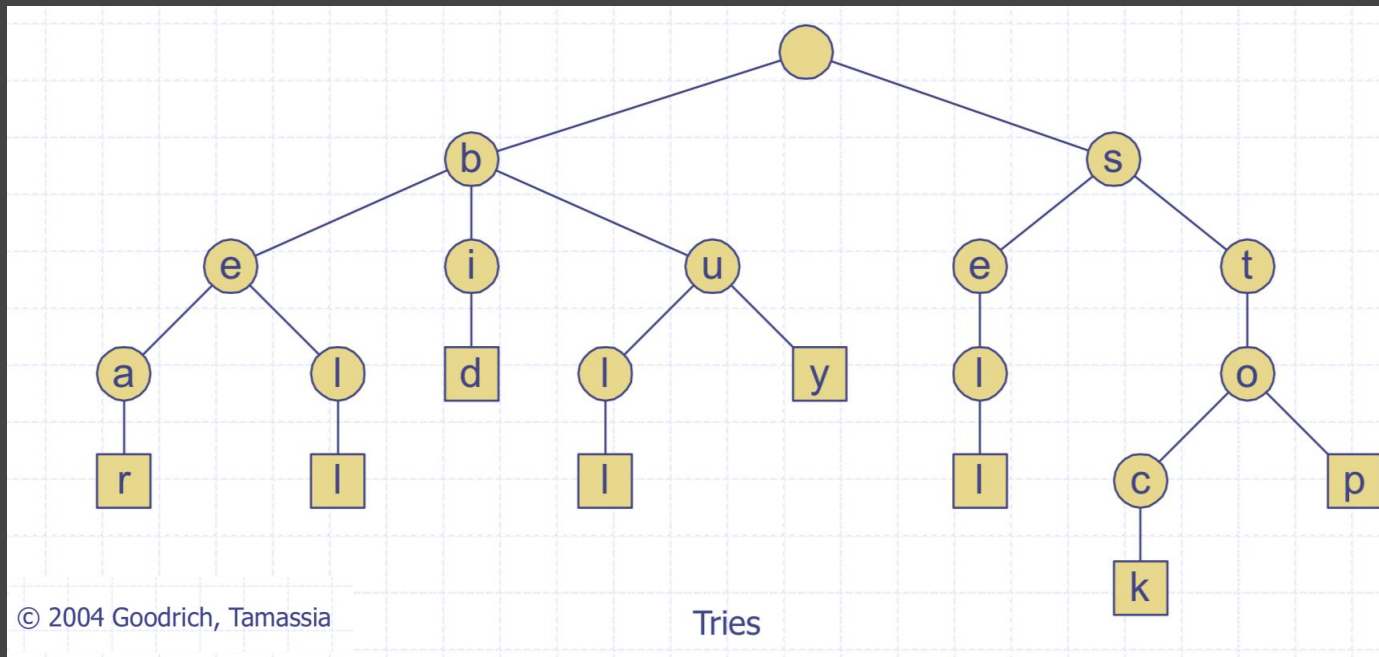
Example: Add “stock”

$S = \{\text{bid, sell, buy, stop, stock}\}$



Standard Trie Example

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



Programming Assignment 5 Question 6

- Implement a Trie class in Ruby.
- Implement an interactive program that
 - takes a filename from the user
 - opens the file and builds a trie where each word (just separate these by using split on a space character) is in the trie, which stores information about each occurrence of that word by line # and word #
 - allows the user to input a string
 - searches the trie for the string and produces one of the following three results
 - If the string is not in the text (and therefore not in the trie), then print a message indicating that the the string could not be found.
 - If the full string is a full word in the text, then print out all occurrences of the word in the text by line # and word #.
 - If the string is not a full word but a prefix of one or more words in the text, print out a list of all the words in the text with that prefix.



Example

simple.txt

```
sue sells  
seashells  
on the seashore  
seashore
```



Example

simple.txt

sue sells
seashells
on the seashore
seashore

Trie.rb example:

```
Enter a file:
> simple.txt
Enter a string to search:
> sally
String not found.
Enter a string to search:
> seashore
(2, 2), (3, 0)
Enter a string to search:
> sea
seashells
seashore
Enter a string to search:
:q
```



Required

- Implement the Trie class.
- Build a full Trie from the file that is passed in.
- Use the Trie class in the required program, and make sure it runs according to the expectations in the example given in the previous slide.

Recommended



- Implement a Node class to be used in the Trie class.
- Use a hash to keep track of the children (character \rightarrow Node).
- Use recursion in your method implementations.
- Use a hash to keep track of the line and word numbers, and only add this data to leaf nodes. For example: $(0 \rightarrow [0, 4])$ would denote that the word occurs at line 0, word 0 and word 4.
- Do an example or two by hand before starting your implementation.