CS 10:
Problem solving via Object Oriented Programming

Winter 2017

Tim Pierson
260 (255) Sudikoff

String finding

Agenda

1. Boyer-Moore algorithm
2. Tries
3. Suffix tries

Matching/recognizing patterns in sequences is a common CS problem

Example: Find pattern in DNA data

Task

Find a substring in this large string

A brute force approach is inefficient, O(nm)

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

- Start at index 0
- Loop over length of query string
- Look for match
- Move right one space if no match
- O(nm)

A brute force approach is inefficient, O(nm)

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

- Start at index 0
- Loop over length of query string
- Look for match
- Move right one space if no match
- O(nm)

A brute force approach is inefficient, O(nm)

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

- Start at index 0
- Loop over length of query string
- Look for match
- Move right one space if no match
- O(nm)

A brute force approach is inefficient, O(nm)

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

- Start at index 0
- Loop over length of query string
- Look for match
- Move right one space if no match
- O(nm)

A brute force approach is inefficient, O(nm)

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

Mismatch, slide query one space right and try again

- Start at index 0
- Loop over length of query string
- Look for match
- Move right one space if no match
- O(nm)

A brute force approach is inefficient, O(nm)

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						
1		A	B	C	D	E	F					

Mismatch, slide query one space right and try again

- Start at index 0
- Loop over length of query string
- Look for match
- Move right one space if no match
- O(nm)

A brute force approach is inefficient O(nm)

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						
1		A	B	C	D	E	F					
…												
n-m+1							A	B	C	D	E	F

- Start at index 0
- Loop over length of query string
- Look for match
- Move right one space if no match
- O(nm)

Match found after n-m+1 checks, each of length m, O(nm)

Boyer-Moore algorithm works backwards

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

- Start at index m
- Loop backward over query string
- Look for match

Boyer-Moore algorithm works backwards

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

- Start at index m
- Loop backward over query string
- Look for match

Boyer-Moore algorithm works backwards

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						

- Start at index m
- Loop backward over query string
- Look for match

On mismatch, slide query to last occurrence of text, or past mismatch

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						
1					A	B	C	D	E	F		

Z not in query, so move query beyond mismatch index, try again

- Start at index m
- Loop backward over query string
- Look for match
- Slide query to last occurrence of text, or past mismatch

On mismatch, slide query to last occurrence of text, or past mismatch

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						
1					A	B	C	D	E	F		

Mismatch, so move last occurrence of D in query to this index

- Start at index m
- Loop backward over query string
- Look for match
- Slide query to last occurrence of text, or past mismatch

On mismatch, slide query to last occurrence of text, or past mismatch

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						
1					A	B	C	D	E	F		
2							A	B	C	D	E	F

- Start at index m
- Loop backward over query string
- Look for match
- Slide query to last occurrence of text, or past mismatch

On mismatch, slide query to last occurrence of text, or past mismatch

Find query of length m, in text of length n

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	A	B	C	Z	E	F	A	B	C	D	E	F
Try 0	A	B	C	D	E	F						
1					A	B	C	D	E	F		
2							A	B	C	D	E	F

Match found

- Start at index m
- Loop backward over query string
- Look for match
- Slide query to last occurrence of text, or past mismatch

Boyer-Moore can be O(n)

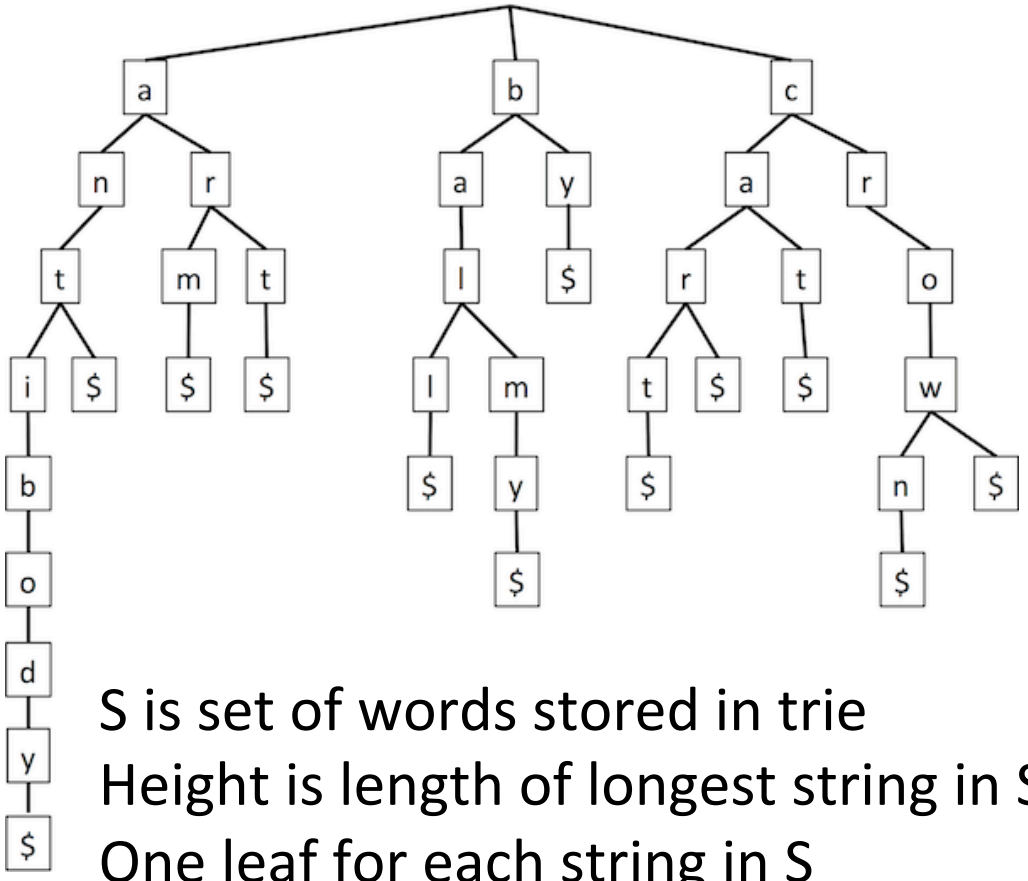
- Our version not quite O(n), but close
- O(m+n), but since normally n >> m, O(n) on “reasonable” text (e.g., not long strings of same character)
- Does require pre-processing step to store last index of each character in query. Easy way:
  - Loop over each character in text to be searched
  - Store characters in Map with index as value
  - At end, Map will have the last index for each character

Agenda

1. Boyer-Moore algorithm
2. Tries
3. Suffix tries

Tries can find all substrings in text that begin with a search string in O(dn)

Alphabet of d characters, and string length n

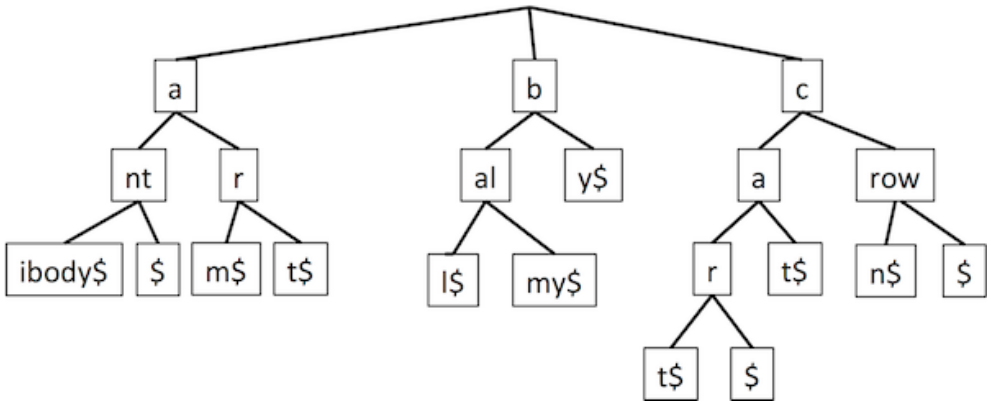


S is set of words stored in trie
Height is length of longest string in S
One leaf for each string in S
Can be used for Set or Map

- Trie is pronounced "Try" (from re*trie*ve)
- Multi-way tree
- Each node is a letter in the text to be searched
- To match string, start at root and follow children to first letter
- Repeat for next letter until find word or stop character ($ here)
- To find string of length n, must go down n levels
- If alphabet has d characters O(dn) to find or insert

Compressed tries save memory

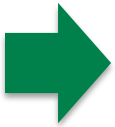
Alphabet of d characters, and string length n



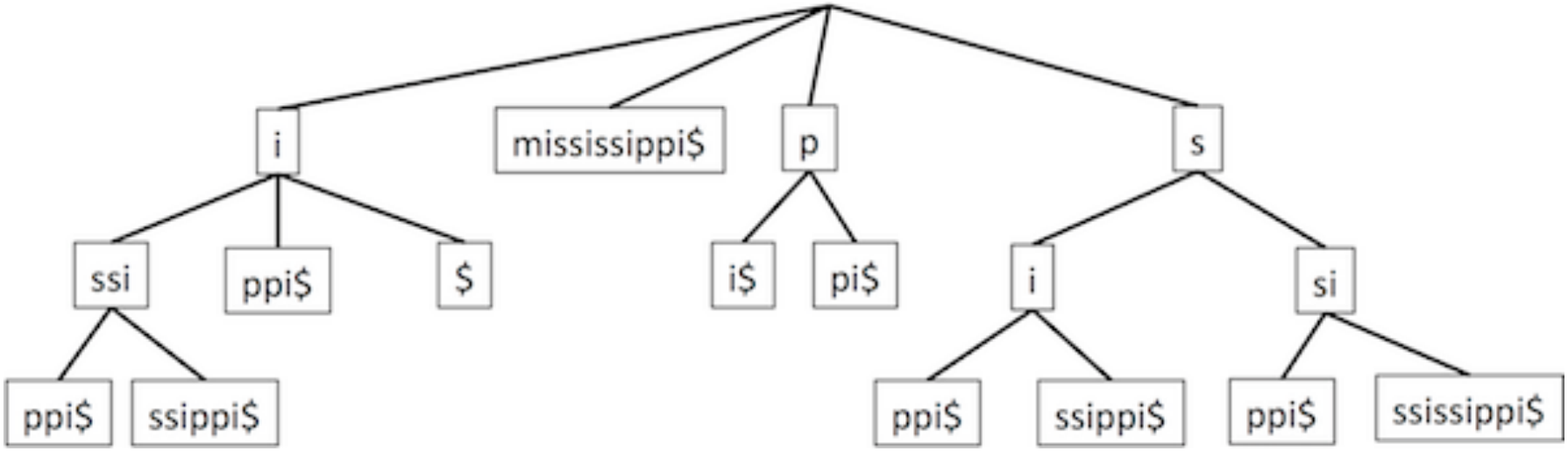
- Compressed trie stores substrings if no branches
- Number of nodes reduced from O(n) to O(s) – number of substrings
- Saves memory, book shows how to store indices
- Not necessarily faster than standard trie, still need to compute desired path
- Can be used for sorting
  - Add all words into trie
  - Do a pre-order traversal

Agenda

1. Boyer-Moore algorithm
2. Tries
3. Suffix tries



Tries works on prefixes, we can also work on suffixes with a Suffix trie



- Store data by suffixes (end of words)
- Add node for each substring X[j..n-1], for j=0,1,..n-1
- Use compressed trie (algorithm complicated, stores in O(n) time)
- Search for suffixes, start at root and work downward

You've come a long way in 10 weeks!

- You learned Java on your own
- Objects? No problem!
- ADTs:
  - Lists, Maps, Stacks, Queues, Trees, Graphs,…
- Data structures:
  - Arrays, Linked Lists, Trees, Hash Tables
- Advanced data structures/algorithms
  - 2-3-4 and Red-Black trees, Tries, Boyer-Moore
- Cutting-edge applications
  - Video processing, data compression, large graph traversal, pattern matching and pattern recognition

In my estimation, you are here