CS 10: Problem solving via Object Oriented Programming

String Finding

Agenda

1. Boyer-Moore algorithm

2. Tries

Matching/recognizing patterns in sequences is a common CS problem

Example: Find pattern in DNA data

AGGACGCCGCATTGACCATCTATGAGATGCTCCAGAACATCTTTGCTATTTTCAG
ACAAGATTCATCTAGCACTGGCTGGAATGAGACTATTGTTGAGAACCTCCTGGCT
AATGTCTATCATCAGATAAACCATCTGAAGACAGTCCTGGAAGAAAAACTGGAGA
AAGAAGATTTCACCAGGGGAAAACTCATGAGCAGTCTGCACCTGAAAAGATATTA
ATGACCAACAAGTGTCTCCTCCAAATTGCTCTCCTGTTGTGCTTCTCCACTACAG
CTCTTTCCATGAGCTACAACTTGCTTGGATTCCTACAAAGAAGCAGCAATTTTCA
GTGTCAGAAGCTCCTGTGGCAATTGAATGGGAGGCTTGAATACTGCCTCAAGCAC
AGGATGAACTTTGACATCCCTGAGGAGATTAAGCAGCTGCAGCAGTTCCAGAAGG
ATGACCAACAAGTGTCTCCTCCAAATTGCTCTCCTGTTGTGCTTCTCCACTACAG
CTCTTTCCATGAGCTACAACTTGCTTGGATTCCTACAAAGAAGCAGCAATTTTCA
GTGTCAGAAGCTCCTGTGGCAATTGAATGGGAGGCTTGAATACTGCCTCAAGCAC
AGGATGAACTTTGACATCCCTGAGGAGATTAAGCAGCTGCAGCAGTTCCAGAAGG
AGGACGCCGCATTGACCATCTATGAGATGCTCCAGAACATCTTTGCTATTTTCAG
ACAAGATTCATCTAGCACTGGCTGGAATGAGACTATTGTTGAGAACCTCCTGGCT
AATGTCTATCATCAGATAAACCATCTGAAGACAGTCCTGGAAGAAAAACTGGAGA
AAGAAGATTTCACCAGGGGAAAACTCATGAGCAGTCTGCACCTGAAAAGATATTA
TGGGAGGATTCTGCATTACCTGAAGGCCAAGGAGTACAGTCACTGTGCCTGGACC
ATAGTCAGAGTGGAAATCCTAAGGAACTTTTACTTCATTAACAGACTTACAGGTT
AGGACGCCGCATTGACCATCTATGAGATGCTCCAGAACATCTTTGCTATTTTCAG
ACAAGATTCATCTAGCACTGGCTGGAATGAGACTATTGTTGAGAACCTCCTGGCT
AATGTCTATCATCAGATAAACCATCTGAAGACAGTCCTGGAAGAAAAACTGGAGA
AAGAAGATTTCACCAGGGGAAAACTCATGAGCAGTCTGCACCTGAAAAGATATTA
TGGGAGGATTCTGCATTACCTGAAGGCCAAGGAGTACAGTCACTGTGCCTGGACC
ATAGTCAGAGTGGAAATCCTAAGGAACTTTTACTTCATTAACAGACTTACAGGTT

Task

Find a substring in this large string

Query string of length *m*

Text of length *n*

Generally assume $m << n$ (but doesn't have to be)

A brute force approach starts at index 0 and works forward

Find query of length m=6, in text of length n=12

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						

Brute force approach

- Start query string and text at index 0
- Loop over length of query string
- Look for match
- Move query string right one space if find mismatch

Compare each character in text and query string, move right if match

Find query of length m=6, in text of length n=12

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						

Brute force approach

- Start query string and text at index 0
- Loop over length of query string
- Look for match
- Move query string right one space if find mismatch

Compare each character in text and query string, move right if match

Find query of length m=6, in text of length n=12

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						

Brute force approach

- Start query string and text at index 0
- Loop over length of query string
- Look for match
- Move query string right one space if find mismatch

Compare each character in text and query string, move right if match

Find query of length m=6, in text of length n=12

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						

Brute force approach

- Start query string and text at index 0
- Loop over length of query string
- Look for match
- Move query string right one space if find mismatch

If find characters that do not match, move query right one space in text and try again

Find query of length m=6, in text of length n=12

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

Brute force approach

- Start query string and text at index 0
- Loop over length of query string
- Look for match
- Move query string right one space if find mismatch

Another mismatch, move query right one space again

Find query of length m=6, in text of length n=12

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 (and again…)

Brute force approach

- Start query string and text at index 0
- Loop over length of query string
- Look for match
- Move query string right one space if find mismatch

No need to keep checking if query string goes past length of text

Continue until hit end of text less length of query string or find match

Find query of length m=6, in text of length n=12

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							A	B	C	D	E	F

Here match found after n-m+1 checks
Each check of length m
Run time complexity?
O(nm)

A brute force approach is inefficient, O(nm)

BoyerMoore.java

Overall O(*nm*)
We can do better!

Look for *pattern* in *text*

```
17    public static int findBruteForce(char[] text, char[] pattern) {
18        System.out.println("Brute force looking for " + String.valueOf(pattern) + " in " + String.valueOf(text));
19        int n = text.length;
20        int m = pattern.length;
21
22        // Test for empty string
23        if (m == 0) return 0;
24
25        //brute force it -- loop over all characters in text O(n)
26        for (int i=0;i<=n-m;i++) { //index into the text
27            //loop over all characters in pattern while characters match O(m)
28            int k = 0; //index into the pattern
29            while (k<m && text[i+k] == pattern[k]) {
30                k++;
31            }
32            //if at end of pattern, then found  match starting at index i in text
33            if (k==m) {
34                System.out.println("\tFound match at index " + i);
35                return i;
36            }
37        }
38        //match not found
39        System.out.println("\tNo match found");
40        return -1;
41    }
42
```

- **Loop over all characters in *text* where *pattern* can fit**
- **No need to check beyond *n-m*, *pattern* of length *m* can't fit in remaining *text***
- **O(*n-m+1*) = O(*n*) if *n* >> *m***

Loop over all characters in pattern O(*m*)

If *pattern* matches *text*, then found match, return index in *text* where *pattern* found

Return -1 if loop over text and do not find pattern

Boyer-Moore algorithm is more efficient and works backwards

Find query of length m=6, in text of length n=12

i=5

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						

k=5

Check text at index *i=m-1=5*, query at *k=m-1=5*

Boyer-Moore

- Start at index m-1
- Loop backward
- If mismatch:
  - If text not in query string, move query past current index
  - If text in query string, move query to last occurrence of text

Boyer-Moore algorithm is more efficient and works backwards

Find query of length m=6, in text of length n=12

i=4

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						

k=4

Check text at index *i=m-1=5*, query at *k=m-1=5*
If match, then decrement *i=4* and *k=4*

Boyer-Moore

- Start at index m-1
- Loop backward
- If mismatch:
  - If text not in query string, move query past current index
  - If text in query string, move query to last occurrence of text

Boyer-Moore algorithm is more efficient and works backwards

Find query of length m=6, in text of length n=12

i=3

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						

k=3

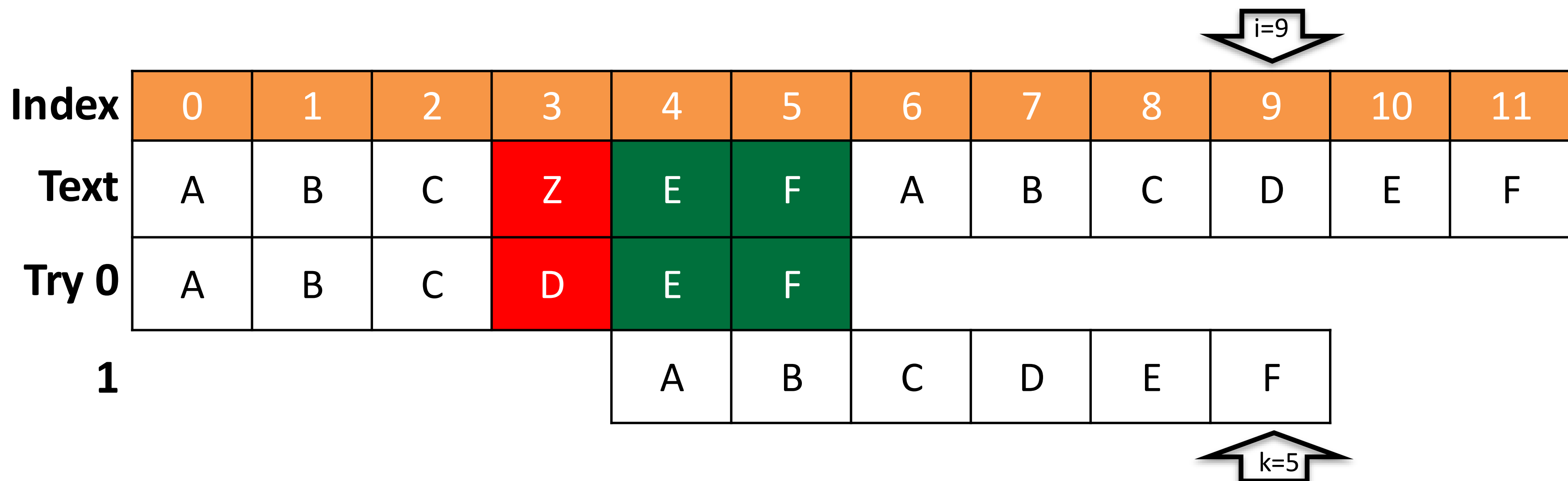
- **Z not in query, so any matches prior to Z must all fail**
- **No need to check those**
- **Move query string one space past character not in query string (Z here)**
- **Avoids checks at indices 0-2**
- **Move *i* to *i+m = 3+6 = 9* and *k=m-1=5***

Boyer-Moore

- Start at index m-1
- Loop backward
- **If mismatch:**
  - **If text not in query string, move query past current index**
  - If text in query string, move query to last occurrence of text

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

Find query of length m=6, in text of length n=12



Check text at *i=9* with query string at *k=5*

Boyer-Moore

- Start at index m-1
- Loop backward
- **If mismatch:**
  - **If text not in query string, move query past current index**
  - If text in query string, move query to last occurrence of text

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

Find query of length m, in text of length n

i=9

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		

k=5

Boyer-Moore

- Start at index m-1
- Loop backward
- **If mismatch:**
  - If text not in query string, move query past current index
  - **If text in query string, move query to last occurrence of text**

Mismatch, but D is in query string so move the last occurrence of D in query string to text index (e.g., move query so D is at index 9)
Don't go backward!

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

Find query of length m=6, in text of length n=12

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

i=11

k=5

Boyer-Moore

- Start at index m-1
- Loop backward
- If mismatch:
  - If text not in query string, move query past current index
  - If text in query string, move query to last occurrence of text

If had moved to first occurrence of text in query string, might cause a move too far right, have to move to last occurrence

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

Find query of length m=6, in text of length n=12

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

3 checks vs. 7 for brute force
Not greatly different for small strings, but very different for large strings!

Boyer-Moore

- Start at index m-1
- Loop backward
- If mismatch:
  - If text not in query string, move query past current index
  - If text in query string, move query to last occurrence of text

Boyer-Moore can be O(n)

- Our version is simplified version of original Boyer-Moore
- Full Boyer-Moore algorithm is 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 query string
  - Store characters in Map with current index as value
  - At end, Map will have the last index for each character

Boyer-Moore algorithm

BoyerMoore.java

```
49  public static int findBoyerMoore(char[] text, char[] pattern) {
50    System.out.println("Boyer-Moore looking for " + String.valueOf(pattern) + " in " + String.valueOf(text));
51    int n = text.length;
52    int m = pattern.length;
53
54    // Test for empty string
55    if (m == 0) return 0;
56
57    // Initialization, create Map of last position of each character = O(n)
58    Map<Character, Integer> last = new HashMap<>();
59    for (int i = 0; i < n; i++) {
60        last.put(text[i], -1);   // set all chars, by default, to -1
61    }
62    for (int i = 0; i < m; i++) {
63        last.put(pattern[i], i); // update last seen positions
64    }
65
66    // Start with the end of the pattern aligned at index m-1 in the text.
67    int i = m - 1;  // index into the text
68    int k = m - 1;  // index into the pattern
69    while (i < n) {
70        if (text[i] == pattern[k]) { // match! return i if complete match; otherwise, keep checking.
71            if (k == 0) {
72                System.out.println("\tFound match at index " + i);
73                return i; // done!
74            }
75            i--; k--;
76        }
77        else { // jump step + restart at end of pattern
78            i += m - Math.min(k, 1 + last.get(text[i]));  //move in text
79            k = m - 1; //move to end of pattern
80        }
81    }
82    System.out.println("\tNo match found");
83    return -1; // not found
84  }
```

Look for *pattern* in *text*

Preprocess: create Map *last* and set all distinct characters in *text* to -1

Update to hold last occurrence of character in *pattern*

Loop backward over *pattern*

Return index in *text* if *pattern* found

Jump past character not in *pattern* (*i += m-0*) or move by min of index into query (*k*) and last position of *text* character in *pattern* so do not go backward

Return -1 if not found

From Goodrich, Tamassia, Goldwasser

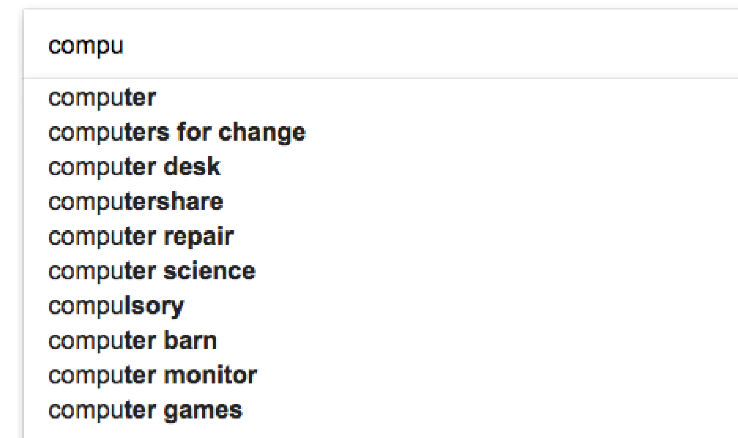
Agenda

1. Boyer-Moore algorithm

2. Tries

How would you implement autocomplete?

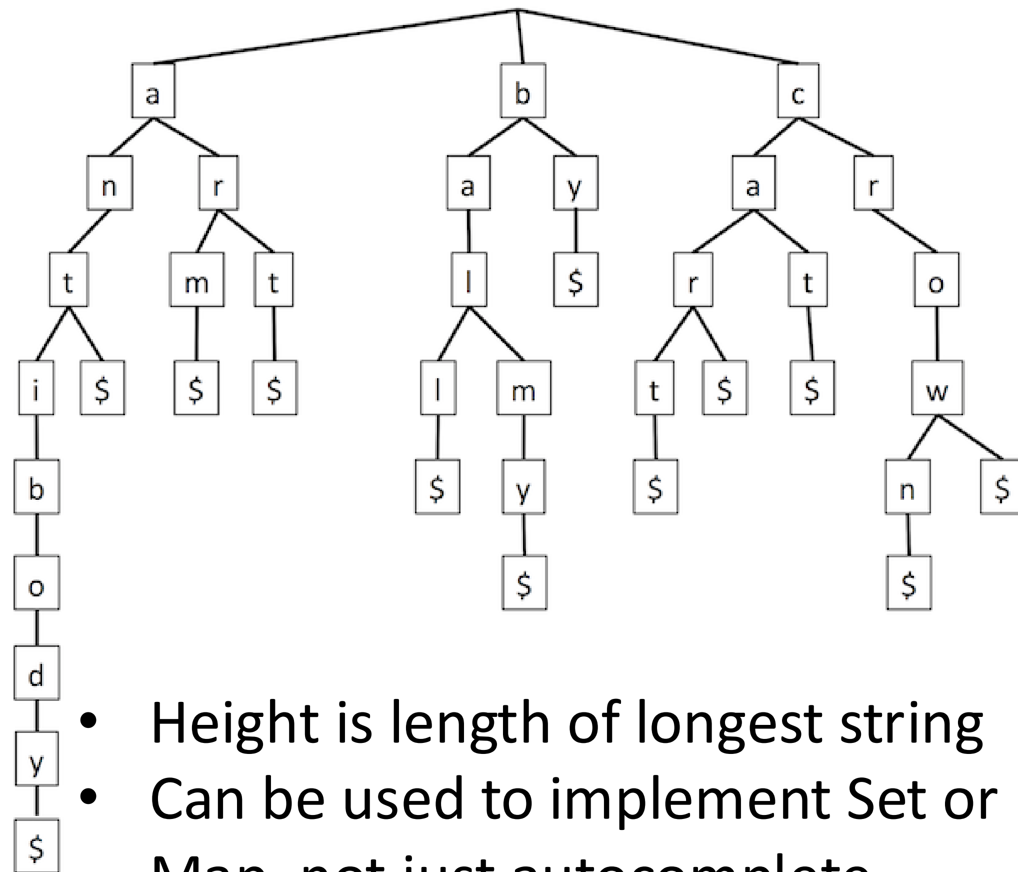
- Consider autocomplete text boxes
- A user starts typing, autocomplete shows possible words user might want given only a couple of characters
- How would you implement that?
- One way is with a Trie (pronounced "try" to differentiate from Tree, comes from "retrieve")



Typed in "compu" into Google, Google guesses what I want

Tries can find all substrings in text that _begin_ with a prefix string

Alphabet of d characters, and string length n

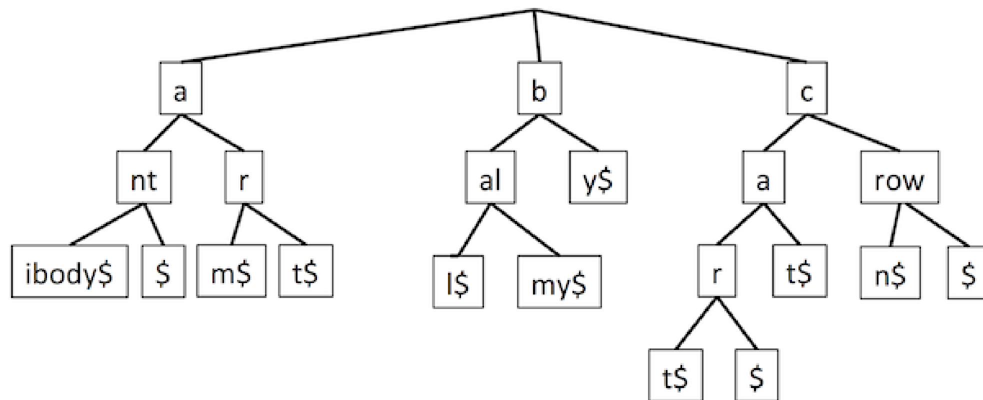


- Height is length of longest string
- Can be used to implement Set or Map, not just autocomplete

- Trie is a multi-way tree where each node is a letter
- Store set of words S in Trie with one node per letter and one leaf for each word
- To match prefix, start at root and follow children until find stop character ($)
- Example: type "ca" and find cart, car, and cat
- To find string of length *m*, must go down m levels
- If alphabet has $d = |\Sigma|$ characters, then O(*dm*) to find or insert

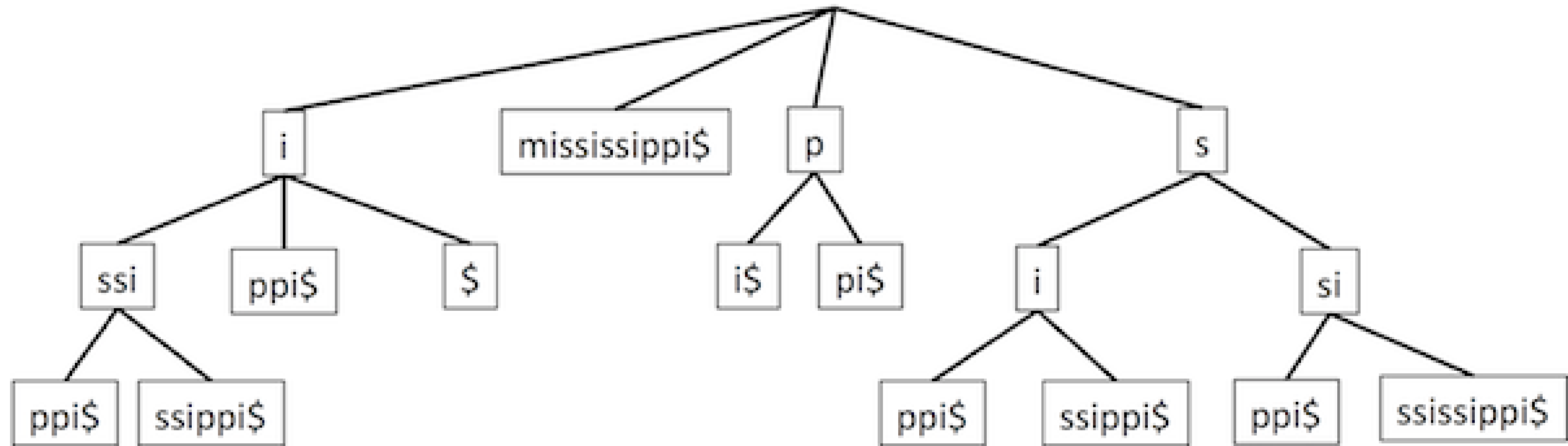
Compressed tries save memory

Alphabet of d characters, and string length n



- Compressed trie stores substrings if no branches (e.g., no branches after “ant” so put “ibody” in one node, not five)
- Number of nodes reduced from O(|n|) – total number of letters in S, to O(|s|) – number of words in S
- Saves memory, book shows how to store indices to make each node constant size
- Can be used for sorting
    - Add all words into trie
    - Do a pre-order traversal

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



Suffix tries

- 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
- See course web page for more details