15-853:Algorithms in the Real World

String Searching I
- Tries, Patricia trees
- Suffix trees

Exact string searching

Given a text T of length m and pattern P of length n

"Quickly" find an occurrence (or all occurrences) of P in T

A Naïve solution:

Compare P with T[i…i+n] for all i --- O(nm) time

How about O(n+m) time? (Knuth Morris Pratt)

How about O(m) preprocessing time and O(n) search time?

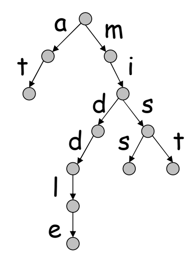
Notation:

Capital letters for strings : A, B, S

Lower case letters for characters : a, b, c, x, y, …

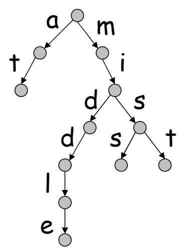
TRIEs

Dictionary = {at, middle, miss, mist}



TRIEs (searching)

Consider an alphabet Σ, with $|\Sigma| = k$
Total of m nodes in trie.
Consider searching a string of length n to see if it is a **prefix** of an element in the dictionary.



Search("mid",T)

TRIEs (searching)

Consider an alphabet Σ, with $|\Sigma| = k$
Total of n nodes in trie.
Consider searching a string of length m to see if it is a **prefix** of an element in the dictionary.
Implementation choices:

- Array per node: O(nk) space, O(m) time search
- Tree per node: O(n) space, O(m log k) time search
- Hash children: O(n) space, O(m) time
  can hash node pointer and child character

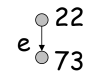
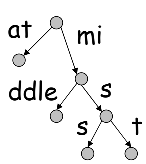


Table.Lookup((22,e)) = 73

PATRICIA Trees

PATRICIA: *Practical Algorithm to Retrieve Information Coded in Alphanumeric* (1968)
Also called radix trees or compressed TRIEs
All nodes with single child are collapsed.

Dictionary = {at, middle, miss, mist}



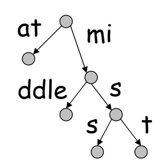
Take less space in practice

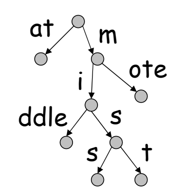
Insertion

Inserting string S into a PATRICIA tree

- Find longest common prefix
- Split edge if needed
- Add suffix

Insert("mote",T)





Takes O(|S|) time

Using Suffixes

If we want to search for any substring within a string we can store all suffixes of the string in a TRIE or PATRICIA tree.

S = mississippi

Dictionary =

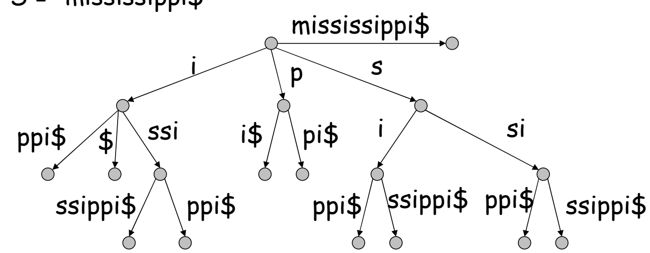
{mississippi, ississippi, ssissippi, sissippi, issippi, ssippi, sippi, ippi, ppi, pi, i}

Typically use special character ($) at the end of a string to make sure every entry has its own leaf

Suffix Trees

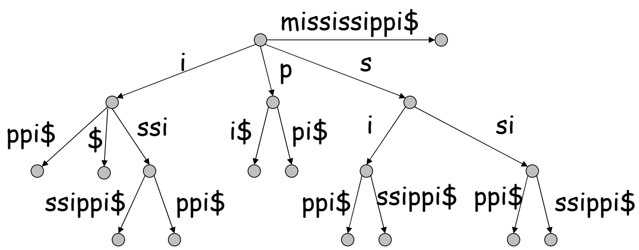
Patricia tree on all suffixes of a string.

S = “mississippi$”



Suffix Tree Space

How do we store a suffix tree in O(n) space?



Suffix Tree Construction

Simple algorithm:

T = empty
for i = 1 to n
  insert(S[i:n],T)

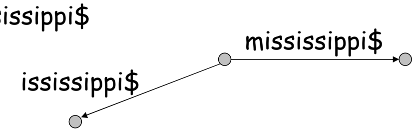
Takes $O(n^2)$ time.

Suffix Tree Construction

mississippi$

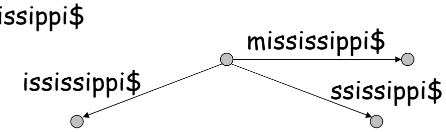
Suffix Tree Construction

ississippi$



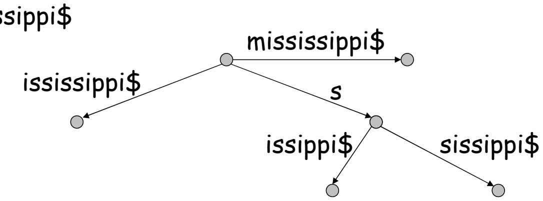
Suffix Tree Construction

ssissippi$



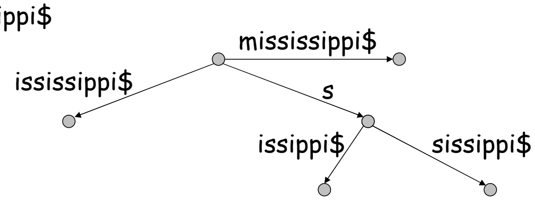
Suffix Tree Construction

sissippi$



Suffix Tree Construction

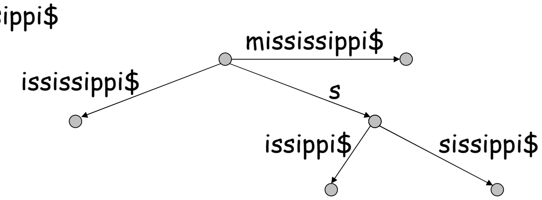
issippi$

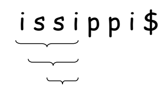


When we look up "issi" can we make looking up "ssi" for the next step cheaper?

Suffix Tree Construction

issippi$



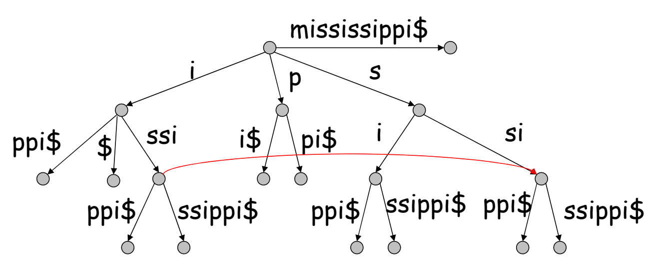


When we previously "looked up" "issi" didn't we then also look up "ssi", "si", "s" on later steps

Suffix Links

For every internal node for a string "aS", keep a pointer to the node for "S"

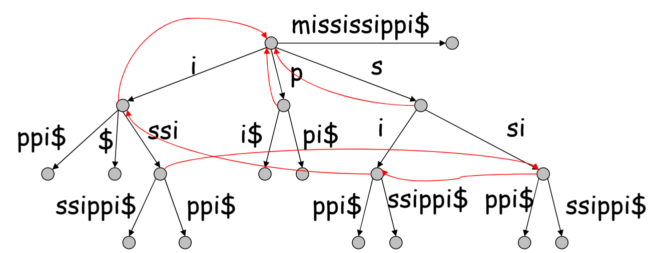
Why must it exist?



Suffix Links

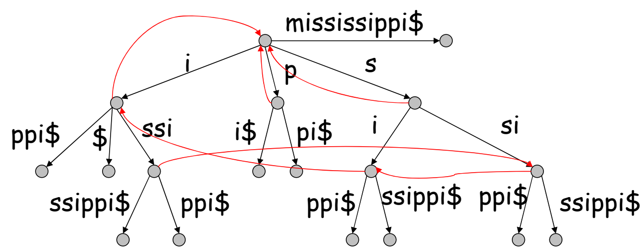
For every internal node for a string "aS", keep a pointer to the node for "S"

Why must it exist?



Suffix Links

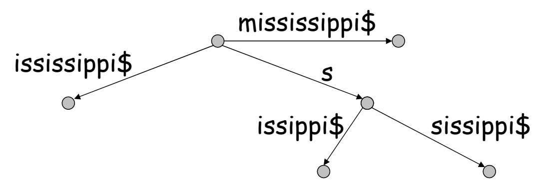
Now if I have found "issi" finding "ssi" is easy, and then finding "si".



Suffix Tree Construction

m i s s i s s i p p i $

insert finger (i) search finger (j)



Suffix Tree Construction

m i s s i s s i p p i $

↑ i ↑ j

Algorithm:

Repeat until (i == n)

Search incrementing j until no match.

- i.e. found S[i:j-1] in tree but not S[i:j]

If in middle of an edge:

- Then split edge at S[i:j-1] and add suffix S[j:n]
- Else add new child to S[i:j-1] with suffix S[j:n]

Use parent's suffix link to find S[i+1:j-1]

If split edge, add suffix link from S[i:j-1] to S[i+1:j-1]

i = i + 1

Almost Correct Analysis

Each increment of j takes O(1) time

- Just search one more character

Each increment of i takes O(1) time

- Just follow suffix link

Total time is O(n) since i and j are each incremented O(n) times.

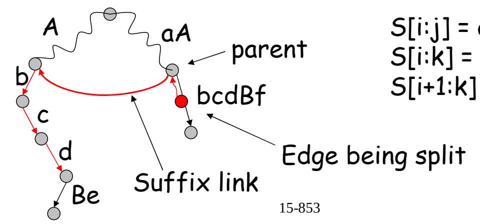
What is wrong?

Following Suffix Links

1. Go to parent of edge that is being split
   - S[i:k] for some k < j
2. Follow link to S[i+1:k]
3. Search down for S[i+1:j-1]
   - This step might not be O(1) time

Following Suffix Links

1. Go to parent of edge that is being split
   - S[i:k] for some k < j
2. Follow link to S[i+1:k]
3. Search down for S[i+1:j-1]
   - This step might not be O(1) time

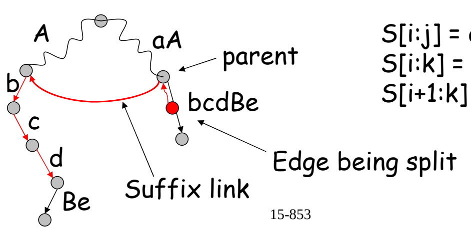


S[i:j] = aAbcdBe
S[i:k] = aA
S[i+1:k] = A

Following Suffix Links

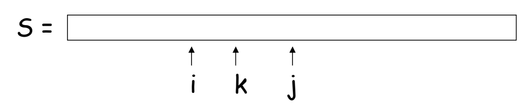
Note that searching edge Be to find B takes constant time even if B is long.

Why?



S[i:j] = aAbcdBe
S[i:k] = aA
S[i+1:k] = A

The "Three Finger" Analysis



Note: there is no counter for k, it is the location of the next node up (inclusive) of S[i:j-1] in the search

Each increment of j takes O(1) time
Following suffix link to increment i takes O(1) time
Each "increment" of k to find S[i+1:j-1] takes O(1) time

TOTAL TIME = O(n)

Summary

Really the only change over the naïve $O(n^2)$ algorithm is the use of suffix links to speed up search when inserting each suffix.

i.e. the key is linking S[i:j] to S[i+1:j] and just doing this for internal nodes in the tree is sufficient.

Suffix trees have many applications beyond string searching.

Extending to multiple lists

Suppose we want to match a pattern with a dictionary of k strings with a total length m.

Concatenate all the strings (interspersed with special characters) and construct a common suffix tree

Time taken = $O(m + k)$

Unnecessarily complicated tree; needs special characters

Multiple lists – Better algorithm

First construct a suffix tree on the first string, then insert suffixes of the second string and so on

Each leaf node should store values corresponding to each string

$O(m)$ as before

Longest Common Substring

Find the longest string that is a substring of both S_1 and S_2

Construct a common suffix tree for both

Any node that has descendants labeled with S_1 and S_2 indicates a common substring

The “deepest” such node is the required substring

Can be found in linear time by a tree traversal.

Common substrings of M strings

Given M strings of total length n, find for every k, the length l_k of the longest string that is a substring of at least k of the strings

Construct a common suffix tree labeling each leaf with the string it came from

For every internal node, find the number of distinctly labeled decendants

Report l_k by a single tree traversal

O(Mn) time – not linear!

Lempel-Ziv compression

Recall that at each stage, we output a pair (p_i, l_i) where $S[p_i .. p_i+l_i] = S[i .. i+l_i]$

Find all pairs (p_i, l_i) in linear time

Construct a suffix tree for S

Label each internal node with the minimum of labels of all leaves below it – this is the first place in S where it occurs. Call this label c_v.

For every i, search for the string S[i .. m] stopping just before c_v, i. This gives us l_i and p_i.