COMP9319 Web Data Compression and Search

Regular expressions and basic indexed search

Including some slides modified from comp3402 at cs.ucf.edu, haimk, Tel-Aviv U. and the slides from Andrew Davison.

1

Regular Expressions

- Notation to specify a language
  - Declarative
  - Sort of like a programming language.
    - Fundamental in some languages like Perl and applications like grep or lex
  - Capable of describing the same thing as an NFA
    - The two are actually equivalent, so RE = NFA = DFA
  - We can define an algebra for regular expressions

2

Definition of a Regular Expression

- R is a regular expression if it is:
  1. **a** for some *a* in the alphabet Σ, standing for the language {a}
  2. ε, standing for the langua
  3. Ø, standing for the empty la
  4. R_1+R_2 where R_1 and R_2 signifies union (sometim
  5. R_1R_2 where R_1 and R_2 are regular expressions and this signifies concatenation
  6. R* where R is a regular expression and signifies closure
  7. (R) where R is a regular expression, then a parenthesized R is also a regular expression

This definition may seem circular, but 1-3 form the basis
Precedence: Parentheses have the highest precedence, followed by *, concatenation, and then union.

3

Using Regular Expressions

- Regular expressions are a standard
  l.
  erl, Unix, Python, . . .

4

RE Examples

- L(**001**) = {001}
- L(**0+10***) = { 0, 1, 10, 100, 1000, 10000, … }
- L(**0*10***) = {1, 01, 10, 010, 0010, …} i.e. {w | w has exactly a single 1}
- L($\Sigma\Sigma$)* = {w | w is a string of even length}
- L((**0(0+1)**)*) = { ε, 00, 01, 0000, 0001, 0100, 0101, …}
- L(**(0+ε)(1+ ε)**) = {ε, 0, 1, 01}
- L(1Ø) = Ø ; concatenating the empty set to any set yields the empty set.
- Rε = R
- R+Ø = R

- Note that R+ε may or may not equal R (we are adding ε to the language)
- Note that RØ will only equal R if R itself is the empty set.

5

Exercise 1

- Let Σ be a finite set of symbols
- Σ = {10, 11}, Σ^* = ?

6

Answer

Answer: $\sum^*$ = {ϵ, 10, 11, 1010, 1011, 1110, 1111, …}

7

Exercise 2

- L1 = {10, 1}, L2 = {011, 11}, L1L2 = ?

8

Answer

- L1L2 = {10011, 1011, 111}

9

Exercise 3

- Write RE for

and 1's

and 1's with at least 2

  – All strings of 0's and 1's beginning with 1 and
utive 0's

10

Answer

- (0|1)*

All strings of 0's and 1's.

- (0|1)*00(0|1)*

All strings of 0's and 1's with at least 2 consecutive 0's

- (1+10)*

All strings of 0's and 1's beginning with 1 and not having two consecutive 0's

11

More Exercises

- 1) (0|1)*011
- 2) 0*1*2*
- 3) 00*11*22*

12

More Exercises (Answers)

1) (0|1)*011
Answer: all strings of 0's and 1's ending in 011
2) 0*1*2*
- Answer: any number of 0's followed by any number of 1's followed by any number of 2's
- 3) 00*11*22*
Answer: strings in 0*1*2 with at least one of each symbol

13

Deterministic Finite Automata (DFA)

- Simple machine with N states.
- Begin in start state.
- Read first input symbol.
- Move to new state, depending on current state and input symbol.
- Repeat until last input symbol read.
- Accept or reject string depending on label of last state.

14

DFA

15

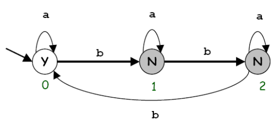
Theory of DFAs and REs

- RE. Concise way to describe a set of
recognize whether a
a given set
- **Duality**: for any DFA, there exists a
describe the same
regular expression,
at recognizes the same set.

16

Duality Example

- DFA for multiple of 3 b's:



- RE for multiple of 3 b's:

```
(a*ba*ba*ba*)* a*
```

17

Fundamental Questions

- Which languages CANNOT be described by any RE?
- Set of all bit strings with equal number of 0s and 1s.
- Set of all decimal strings that represent prime numbers.
- Many more. . . .

18

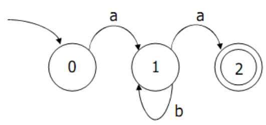
Problem 1

- Make a DFA that accepts the strings in the language denoted by regular expression ab*a

19

Solution

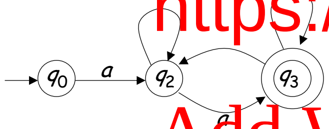
- ab*a:



20

Problem 2

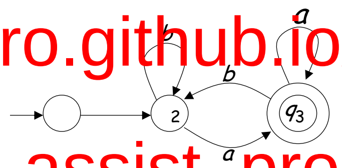
- Write the RE for the following automata:



21

Solution

- a(a|b)*a



22

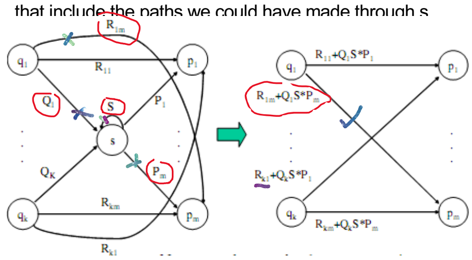
DFA to RE: State Elimination

- Eliminates states of the automaton and replaces the edges with regular expressions that includes the behavior of the eliminated states.
- Eventually we get down to the situation with just a start and final node, and this is easy to express as a RE

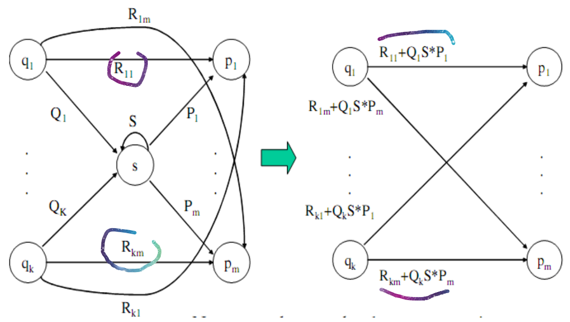
23

State Elimination

- Consider the figure below, which shows a generic state s about to be eliminated.
- The labels on all edges are regular expressions.
- To remove s, we must make labels from each qi to p1 up to pm that include the paths we could have made through s



24



25

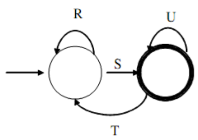
DFA to RE via State Elimination (1)

- Starting with intermediate states and then moving to accepting states, apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges.
- The result will be a one or two state automaton with a start state and accepting state.

26

DFA to RE State Elimination (2)

- If the two states are different, we will have an automaton that looks li



- We can describe this automaton as: (R | SU*T)*SU*

27

DFA to RE State Elimination (3)

- If the start state is also an accepting state, then rm a state elimination from on that gets rid of every state This leaves the following:

utomaton as simply R*

28

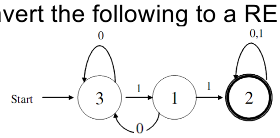
DFA to RE State Elimination (4)

- If there are n accepting states, we must repeat the above steps for each accepting states to get n different regular expressions, R1, R2, … Rn.
- For each repeat we turn any other accepting state to non-accepting.
- The desired regular expression for the automaton is then the union of each of the n regular expressions: R1 U R2… U RN

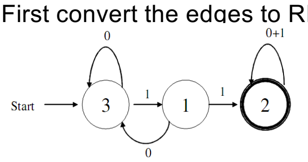
29

DFA->RE Example

- Convert the following to a RE:



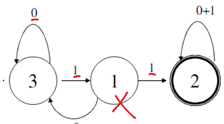
- First convert the edges to RE’s:



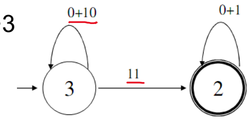
30

DFA -> RE Example (2)

- Eliminate State 1:



- Note edge from 3->3

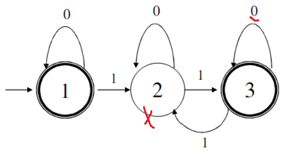


- Answer: (0+10)*11(0+1)*

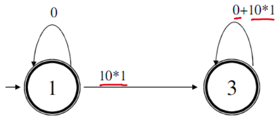
31

Second Example

- Automata that accepts even number of 1's



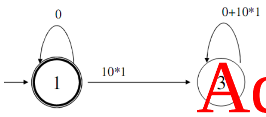
- Eliminate state 2:



32

Second Example (2)

- Two accepting states,



- This is just 0*; can ignore going to state 3 since we would "die"

33

Second Example (3)

- This is just 0*10*1(0|10*1)*
- Combine from previous slide to get 0* | 0*10*1(0|10*1)*

34

Text search

- Pattern matching directly
  - Brute force
  - BM
  - KMP
- Regular expressions
- Indices for pattern matching
  - Inverted files
  - **Signature files**
  - **Suffix trees** and **Suffix arrays**

35

35

Inverted Index

For each term *t*, we store a list of all documents that contain *t*.

BRUTUS	→	1	2	4	11	31	45	173	174	
CAESAR	→	1	2	4	5	6	16	57	132	...
CALPURNIA	→	2	31	54	101					

⋮

dictionary postings

36

36

36

Create postings lists, determine document frequency

term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

⟹

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

37

37

Positional indexes

- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions

38

38

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

TO, 993427:

‹ 1: ‹7, 18, 33, 72, 86, 231›;
2: ‹1, 17, 74, 222, 255›;
4: ‹8, 16, 190, 429, 433›;
5: ‹363, 367›;
7: ‹13, 23, 191›; . . . ›

BE, 178239:

‹ 1: ‹17, 25›;
4: ‹17, 191, 291, 430, 434›;
5: ‹14, 19, 101›; . . . › Document 4 is a match!

39

39

Signature files

- Definition
  - Word-oriented index structure based on hashing.

ry large texts.

function that maps words to bit masks.

cks.

ined by bitwise ORing the rds in the text block.

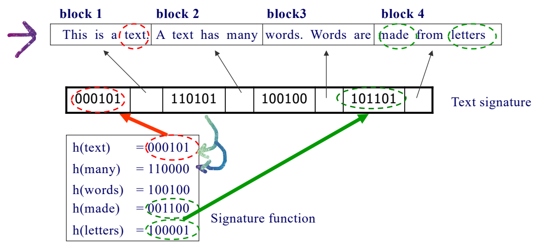
ch between all 1 bits in the ck mask.

40

40

Signature files

- Example:



41

41

Signature files

- False drop Problem
  - The corresponding **bits** are **set** even though the word is **not there**!
  - The **design** should **insure** that the **probability** of false drop is **low**.
    - Also the Signature file should be as short as possible.
  - **Enhance** the **hashing function** to minimize the error probability.

42

42

Signature files

- Searching
  1. For a single word, Hash word to a bit mask W.
  2. For phrases,
     1) Hash words in query to a bit mask.
     2) Bitwise OR of all the query masks to a bit mask W.
  3. Compare W to the bit masks B_i of all the text blocks.
     - If all the bits set in W are also in B_i, then text block may contain the word.
  4. For all candidate text blocks, an online traversal must be performed to verify if the actual matches are there.
- Construction
  1. Cut the text in blocks.
  2. Generate an entry of the signature file for each block.
     - This entry is the bitwise OR of the signatures of all the words in the block.

43

43

Suffix trees and suffix arrays

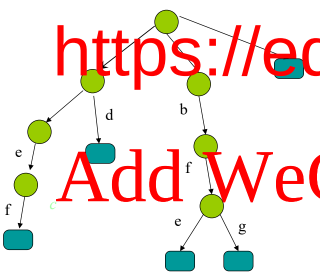
44

44

Trie

- A tree representing a set of strings.

{
aceef
ad
bbfe
bbfg
c }

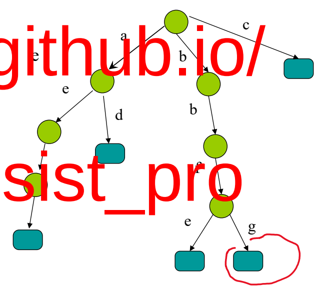


45

45

Trie (Cont)

- Assume no string is a prefix of another

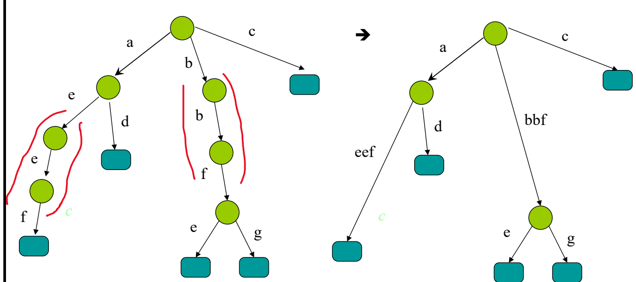


46

46

Compressed Trie

- Compress unary nodes, label edges by strings



47

47

Suffix tree

Given a string s a suffix tree of s is a compressed trie of all suffixes of s

To make these suffixes prefix-free we add a special character, say $, at the end of s

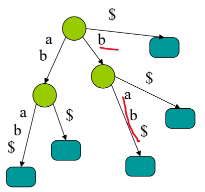
48

48

Suffix tree (Example)

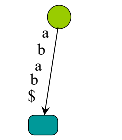
Let s=abab, a suffix tree of s is a compressed trie of all suffixes of s=abab$

{
$
b$
ab$
bab$
abab$
}

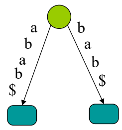


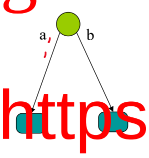
Trivial algorithm to build a Suffix tree

Put the largest suffix in

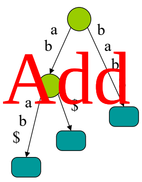


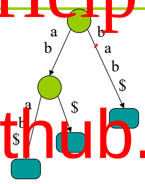
Put the suffix bab$ in

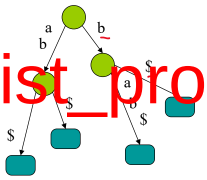


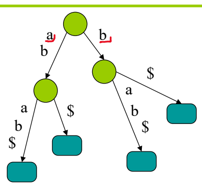


Put the suffix ab$ in

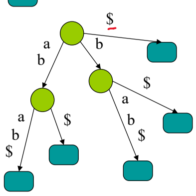


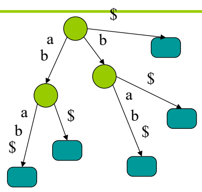






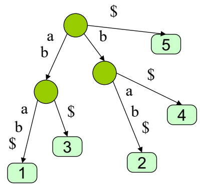
Put the suffix $ in





abab$
1 2 3 4 5

We will also label each leaf with the starting point of the corresponding suffix.



Analysis

Takes O(n^2) time to build.

*It can also be constructed in O(n) time.

55

55

What can we do with it ?

Exact string matching:

Given a Text T, |T| = n, preprocess it such that when a pattern P, |P|=m, arrives you can quickly decide when it occurs in T.

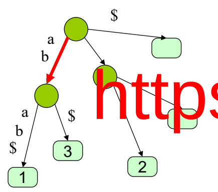
W e may also want to find all occurrences of P in T

56

56

Exact string matching

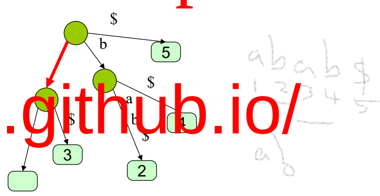
In preprocessing we just build a suffix tree in O(n) time



Given a pattern P = ab we traverse the tree according to the pattern.

57

57



...hen the pattern occurs in the text.

...reach corresponds to an occurrence.

By traversing this subtree we get all k occurrences in O(n+k) time.

58

58

Generalized suffix tree

Given a set of strings S a generalized suffix tree of S is a compressed trie of all suffixes of s ∈ S

To make these suffixes prefix-free we add a special char, say $, at the end of s

To associate each suffix with a unique string in S add a different special char to each s

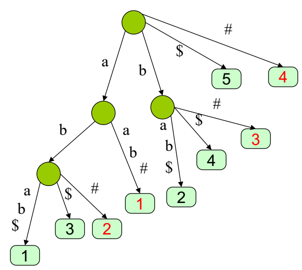
59

59

Generalized suffix tree (Example)

Let s_1=abab and s_2=aab here is a generalized suffix tree for s_1 and s_2

{
$ #
b$ b#
ab$ ab#
bab$ aab#
abab$
}



60

60

So what can we do with it ?

Matching a pattern against a database of strings

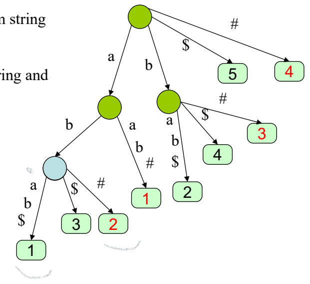
61

61

Longest common substring (of two strings)

Every node with a leaf descendant from string S_1 and a leaf descendant from string S_2 represents a maximal common substring and vice versa.

Find such node with largest "string depth"

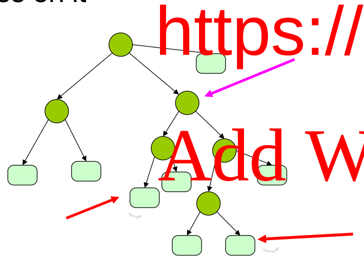


62

62

Lowest common ancestor

A lot more can be gained from the suffix tree if we preprocess it so LCA queries on it

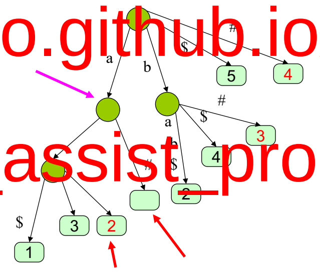


63

63

Why?

The LCA of two leaves represents the longest
P) of these 2 suffixes



64

64

Finding maximal palindromes

- A palindrome: caabaac, cbaabc
- Want to find all maximal palindromes in a string s

Let s = cbaaba

The maximal palindrome with center between i-1 and i is the LCP of the suffix at position i of S and the suffix at position m-i+1 of S^r

65

65

Maximal palindromes algorithm

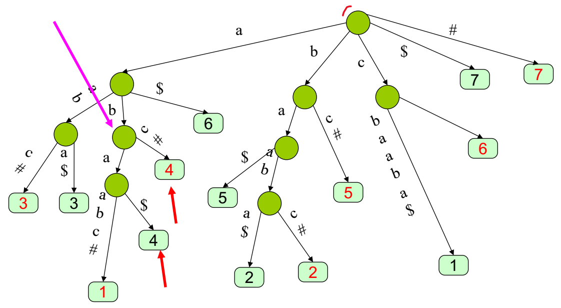
Prepare a generalized suffix tree for
s = cbaaba$ and s^r = abaabc#

For every i find the LCA of suffix i of s and suffix m-i+1 of s^r

66

66

Let s = cbaaba$ then s^r = abaabc#



67

67

Analysis

O(n) time to identify all palindromes

68

68

Drawbacks

- Suffix trees consume a lot of space
- It is O(n) but the co

- Notice that if we indeed want to traverse an edge in O(1) time then we need an array of ptrs. of size |Σ| in each node

69

69

Suffix array

- We loose some of the functionality but we

1234

xicographically:

3 1 4 2

e indices of the

3	1	4	2

70

70

How do we build it ?

- Build a suffix tree
- Traverse the tree in DFS, lexicographically picking edges outgoing from each node and fill the suffix array.
- O(n) time

71

71

How do we search for a pattern ?

- If P occurs in T then all its occurrences are consecutive in the suffix array.
- Do a binary search on the suffix array
- Takes O(mlogn) time

72

72

Example

Let S = mississippi

Let P = issa

P = issi

L	11	i
	8	ippi
M	5	issippi
	2	ississippi
	1	mississippi
M	10	pi
	9	ppi
	7	sippi
	4	sisippi
	6	ssippi
R	3	ssissippi

Supra index

- Structure
  - **Suffix arrays** are **space efficient** implementation of **suffix trees**.
  - Simply an array **containing all the pointers** to the text suffixes listed in lexicographical order.
  - **Supra-indices:**
    - If the suffix array is **large**, this binary search can perform **poorly** because of the number of random disk accesses.
    - Suffix arrays are designed to allow **binary searches** done by comparing the contents of each pointer.
    - To remedy this situation, the use of **supra-indices** over the suffix array has been proposed.

Supra index

- Example