# COMP9319 Web Data Compression and Search

Course revision,
Exam

1

## Announcements

- Final exam details and a sample exam released
- Additional consultations for week 12 (the week just before the exam)

2

## Revision

3



## COMP9319 – The foundations of

- how different compression tools work.

large amount of data on

abytes, terabytes or

search efficiently

d data repositories

4

4

## Course Aims

As the amount of Web data increases, it is becoming vital to not only be able to search and retrieve this information quickly, but also to store it in a compact manner. This is especially important for mobile devices which are becoming increasingly popular. Without loss of generality, within this course, we assume Web data (excluding media content) will be in XML and its like (e.g., HTML, JSON).

This course aims to introduce the concepts, theories, and algorithmic issues important to Web data compression and search. The course will also introduce the most recent development in various areas of Web data optimization topics, common practice, and its applications.

5

5

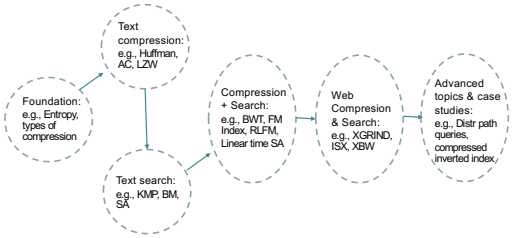
## Summarised schedule

- 1. Compression
- 2. Search
- 3. Compression + Search
- 4. “Compression + Search” on Web text data
- 5. Selected advanced topics

6

6

## Topics




7

## Topic Snapshots

8

## Questions to discuss (www)

- What (is data compression)
- Why (data compres
- Where

9

9

## Compression

- Minimize amount of information to be
  d
  ence of characters into a new bit sequence
  nt (for lossless)

10

10



## Terminology (Types)

- Block-block
  - source message and codeword: fixed length
  - e.g., ASCII
- Block-variable
  - source message: fixed; codeword: variable
  - e.g., Huffman coding
- Variable-block
  - source message: variable; codeword: fixed
  - e.g., LZW
- Variable-variable
  - source message and codeword: variable
  - e.g., Arithmetic coding

11

11

## Run-length coding

- Run-length coding (encoding) is a very widely used and simple compression technique
  - does not assume a memoryless source
  - replace runs of symbols (possibly of length one) with pairs of (symbol, *run-length*)

12

## Entropy

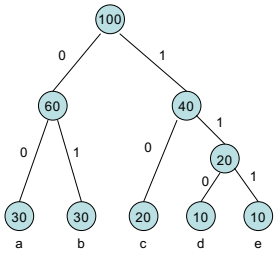
- What is the minimum number of bits per symbol?
- Answer: Shannon's result – theoretical minimum average number of bits per code work is known as Entropy (H)

$$\sum_{i=1}^{n} - p(s_i) \log_2 p(s_i)$$

13

## Huffman coding

| S | Freq | Huffman |
|---|---|---|
| a | 30 | 00 |
| b | 30 | 01 |
| c | 20 | 10 |
| d | 10 | 110 |
| e | 10 | 111 |




14

## Arithmetic coding (encode)

15

## Arithmetic coding (decode)

16



## LZW Compression

```
w = NIL;
 while ( read a character k )
    {
      if wk exists in the dictionary
        w = wk;
      else
        add wk to the dictionary;
        output the code for w;
        w = k;
    }
```

17

## LZW Decompression

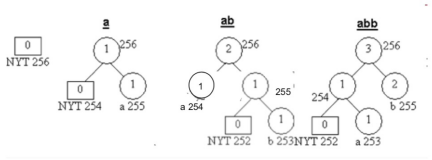
```
read a character k;
 output k;
 w = k;
 while ( read a character/code k )
    {
      entry = dictionary entry for k;
      output entry;
      add w + entry[0] to dictionary;
      w = entry;
    }
```

18

## Adaptive Huffman

abbbba: 0110000101100010011000100110001001100010011000100110000 1

abbbba: 0110000100110001011111 01




a: 01100001

b: 01100010

Modified from Wikipedia

19

## BWT(S)

**function** BWT (string s)

create a table, rows are all possible rotations of s

sort rows alphabetically

**return** (last column of the table)

20

## InverseBWT(S)

**function** inverseBWT (string s)

create empty table

**repeat** length(s) **time**

insert s as a column of table before first column of the table // first insert creates first column

sort rows of the table alphabetically

**return** (row that ends with the 'EOF' character)

21

## Other ways to reverse BWT

Consider L=BWT(S) is composed of the the transformed string ai:

– The number of symbols in the substring $V_0 \ldots V_i$. (i.e., Occ[ ])

– $V_i$, in L, the number lexographically less than

22

## Move to Front (MTF)

- Reduce entropy based on local frequency correlation
- Usually used for BWT before an entropy-encoding step
- Author and detail:
  - Original paper at cs9319/Papers
  - http://www.arturocampos.com/ac_mtf.html

23

## BWT compressor vs ZIP

ZIP (i.e., LZW based) BWT+RLE+MTF+AC

| File Name | Raw Size | PKZIP Size | PKZIP Bits/Byte | BWT Size | BWT Bits/Byte |
|---|---|---|---|---|---|
| bib | 111,261 | 35,821 | 2.58 | 29,567 | 2.13 |
| book1 | 768,771 | 315,999 | 3.29 | 275,831 | 2.87 |
| book2 | 610,856 | 209,061 | 2.74 | 186,592 | 2.44 |
| geo | 102,400 | 68,917 | 5.38 | 62,120 | 4.85 |
| news | 377,109 | 146,010 | 3.10 | 134,174 | 2.85 |
| obj1 | 21,504 | 10,311 | 3.84 | 10,857 | 4.04 |
| obj2 | 246,814 | 81,846 | 2.65 | 81,948 | 2.66 |

From http://marknelson.us/1996/09/01/bwt/

24

## Pattern Matching

- Brute Force
- Boyer Moore
- KMP

25

## Regular expressions

- 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(**ΣΣ**)* = {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.

26

## Theory of DFAs and REs

- RE. Concise way to describe a set of strings.
- DFA. Machine to re given string is in a g
- **Duality**: for any DFA, there exists a regular expression to describe the same set of strings; for any regular expression, there exists a DFA that recognizes the same set.

27

## DFA to RE: State Elimination

- Eliminates states of the automaton and s with regular
  cludes the behavior of
  tes.
- Eventually we get down to the situation
  node, and this is
  R

28



## Signature files

- Definition
  - Word-oriented index structure based on hashing.
  - Use liner search.
  - Suitable for not very large texts.
- Structure
  - Based on a Hash function that maps words to bit masks.
  - The text is divided in blocks.
    - **Bit mask of block is obtained by bitwise ORing the signatures of all the words in the text block.**
    - **Word not found, if no match between all 1 bits in the query mask and the block mask.**

29

## 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

30

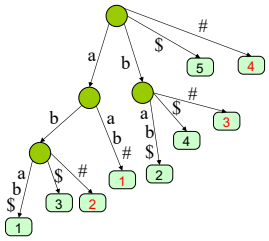
## Generalized suffix tree (Example)

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$ | |

}

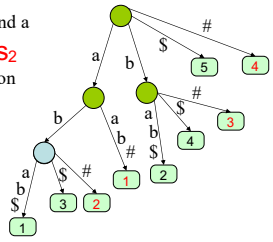


31

## 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"



32

## Suffix array

- We loose some of the functionality but we save space.

Let s = abab

Sort the suffixes lexic

ab, abab, b, bab

The suffix array gives the indices of the suffixes in sorted order

| 3 | 1 | 4 | 2 |
|---|---|---|---|

33



## Example

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

34

## BURROWS-WHEELER TRANSFORM

Reminder: Recovering T from L

1. Find F by sorting L
2. First char of T? m
3. Find m in L
4. L[i] precedes F[i] in T. Therefore we get mi
5. How do we choose the correct i in L?
   - The i's are in the same order in L and F
   - As are the rest of the char's
6. i is followed by s: mis
7. And so on….

| F | L |
|---|---|
| # | i |
| i | p |
| i | s |
| i | s |
| i | m |
| m | # |
| p | p |
| p | i |
| s | s |
| s | s |
| s | i |
| s | i |

35

## BACKWARD-SEARCH EXAMPLE

P = pssi

- i = 3
- c = 's'
- First = C['s'] + Occ('s',1) +1 = 8+0+1 = 9
- Last = C['s'] + Occ('s',5) = 8+2 = 10
- (Last – First + 1) = 2

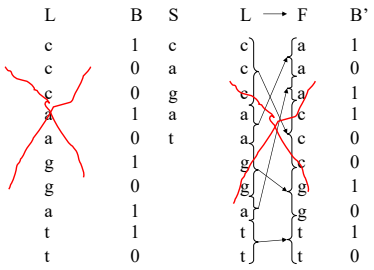
| F | L | |
|---|---|---|
| # | mississipp | i 1 |
| i | #mississip | p 2 |
| i | ppi#missis | s 3 |
| i | ssippi#mis | s 4 |
| i | ssissippi# | m 5 |
| m | ississippi | # 6 |
| p | i#mississi | p 7 |
| p | pi#mississ | i 8 |
| s | ippi#missi | s 9 |
| s | issippi#mi | s 10 |
| s | sippi#miss | i 11 |
| s | sissippi#m | i 12 |

C[ ] = 1 (i) 5 (m) 6 (p) 8 (s)

Algorithm backward_search($P[1,p]$)
(1) $i \leftarrow p$, $c \leftarrow P[p]$, First $\leftarrow C[c]+1$, Last $\leftarrow C[c+1]$;
(2) **while** ((First ≤ Last) **and** ($i \geq 2$)) **do**
(3) $c \leftarrow P[i-1]$;
(4) First $\leftarrow C[c] + \text{Occ}(c, \text{First}-1) + 1$;
(5) Last $\leftarrow C[c] + \text{Occ}(c, \text{Last})$;
(6) $i \leftarrow i - 1$;
(7) **if** (Last < First) **then return** "no rows prefixed by $P[1,p]$" **else return** (First, Last)

36

## Compressed SA: Run-Length FM-index...




37

## Changes to formulas

- Recall that we need to compute $C_T[c]+rank_c(L,i)$ in the backward search.
- **Theorem:** $C[c]+rank_c(L,i)$ is equivalent to $select_1(B',C_S[c]+1+rank_c(S,rank_1(B,i)))-1$, when $L[i] \neq c$, and otherwise to $select_1(B',C_S[c]+rank_c(S,rank_1(B,i)))+ i-select_1(B,rank_1(B,i))$.

38

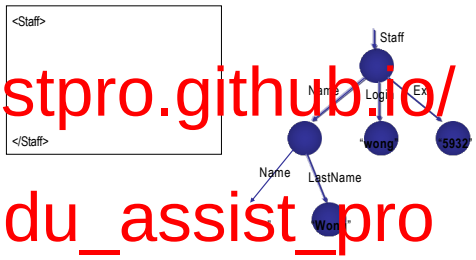
## Linear time suffix array construction

- Consider the popular example string S:
- **bananainp**

1. Construct the suffi
   linear time algorithm
2. Then compute the BWT($)
3. What's the relationship between the suffix array and BWT ? (e.g., SA -> BWT vs BWT -> SA )

39

## Semistructured Data: Tree/HTML/XML/JSON/RDF...




40

## XPath for XML

/bib/book[@price < "60"]

/bib/book[author/@age < "25"]

/bib/book[author/text()]

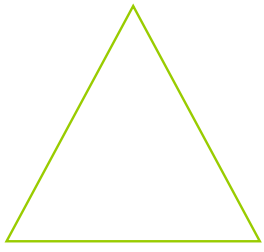
41

## Path query evaluation

Top-down

Bottom-up

Hybrid

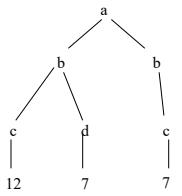


42

## XPath evaluation

<a><c>12</c><d>7</d><c>7</c></a>

/ a / b [c = “12”]




43

## Path indexing

- Traversing graph/tree almost = query processing for semistructured / XML data
- Normally, it requires to traverse the data from the root and return all nodes X reachable by a path matching the given regular path expression
- Motivation: allows the system to answer regular path expressions without traversing the whole graph/tree

44

## Two techniques

- Based on the idea of language-equivalence
- Data Guide

45

## XMill

46



## XGRIND

**Original Fragment:**

```
<student name=“Alice“>
      <a1>78</a1>
      <a2>86</a2>

   <midterm>91</midterm>
      <project>87</project>
</student>
```

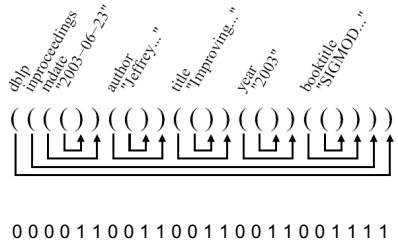
**Compressed Fragment:**

```
T0 A0 nahuff(Alice)
   T1 nahuff(78) /
   T2 nahuff(86) /
   T3 nahuff(91) /
   T4 nahuff(87) /
/
```

XML Compression Techniques 47

47

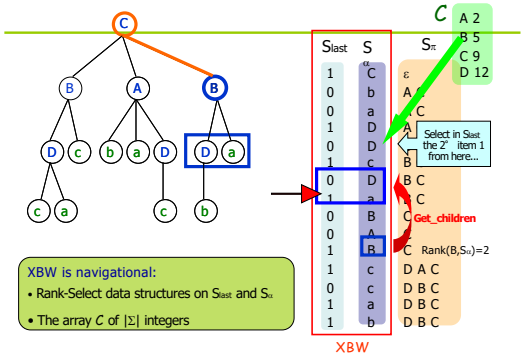
## ISX: Balanced Parenthesis Encoding




0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1

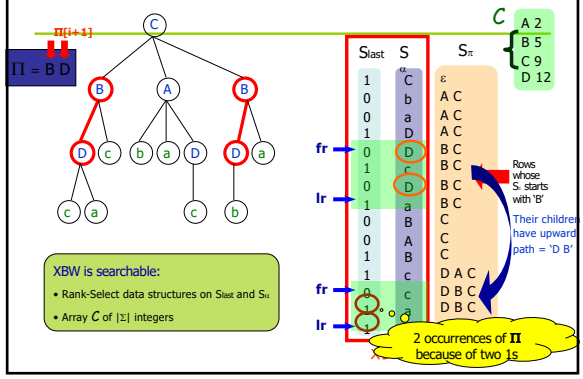
48

## XBW is navigational



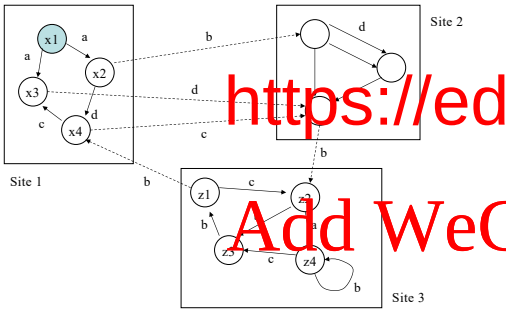

49

## XBW is searchable (count subpaths)




50

## Distributed path queries




51

## Distributed path query processing

- Given a query, we compute its automaton
- rocess at each site
- Stop(n, s) and Result(n,
- to a central
- union

52



## Compression for inverted index

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk
  - Note: Large search engines keep significant part of postings in memory
- We will devise various compression schemes for dictionary and postings.
- VB code, Gamma code

53

## Web Graph Compression

The compression techniques are specialized for Web Graphs.

The average link size decreases with the increase of the graph.

The average link access time increases with the increase of the graph.

The ζ-codes seems to have the best trade-off between avg. bit size and access time.

54

## Case studies: e.g., Content optimization

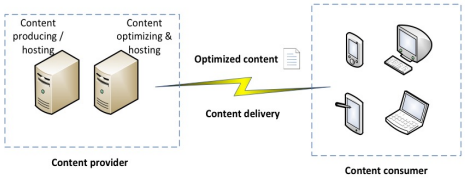



Figure 2. Delivery of content with content optimization

55

## Covered Topics

1. Entropy & basic compressions (RLE, Huffman, AC, LZW, Adaptive Huffman)
2. Pattern matching (Brute Force, KMP, BM); Regular Expression & Finite Automata; Inverted Index & Signature Files.
3. Suffix tree, Suffix array, BWT, MTF, FM Index, RLFM, O(n) SA construction.
4. Semistructured Data, XML & XPath; Path indexing; Tree/XML compressions (XMill, XGrind, ISX, XBW).
5. Querying distributed data.
6. Inverted index & its compression; variable length coding; Web graph compression.
7. Case studies: Google Bigtable; Cloud data optimization.

56

## What have not been covered?

- Multimedia data compression (e.g., images, videos)
- Lossy compression



57

## Future Directions

- Lossy (text) compression: summarization,

58

## Learning outcomes

- have a good understanding of the fundamentals of text compression
- be introduced to advanced data compression techniques such as those based on Burrows Wheeler Transform
- have programming experience in Web data compression and optimization
- have a deep understanding of XML and selected XML processing and optimization techniques
- understand the advantages and disadvantages of data compression for Web search
- have a basic understanding of XML distributed query processing
- appreciate the past, present and future of data compression and Web data optimization

59

59

## Learning outcomes (a compressed version)

- have a different perception on:
  - ”information” (e.g., entropy) & its represention
  - string manipulation (compare, substring, etc)
  - semistructured text data manipulation
- have experience in practical considerations on:
  - efficient algorithms vs efficient implementations
  - computations with limited resources (e.g., when dealing with big text data)

60

60

## Assignment 2

- We are still waiting for submissions with special considerations this week
- Marking script and test cases to be released next week (week 11)
- Aim to finish marking and release results in/by week 12 – before your exam in week 13

61

## Assignment 2 (BWT specific)

- Understand deeply how BWT backward search & decoding work
- Understand the relationship between L and F columns
- C[ ] for F column
- How Occ[ ] and C[ ] work

62

## Assignment 2 (general)

- What index structures to build
- Time & space to bui
- Size to keep the da
- Storage, memory, considerations & trade-offs
- "Feel" the practical implementation bottlenecks vs algorithmic complexity

63



## Assessment

64

64

## The final exam

- One final exam (50 pts).
- If you are ill on the day of the exam, do not attend the exam – c.f. fit-to-site policy. Apply for special consideration asap.
- It's a 2.5 hr online exam (13:30-16:00 Aug 24, 2022 AEST).
- Read the sample exam to get familiar.
- Supp exam covers the same scope & CLOs but may be of a different format, e.g., an oral exam.

65

65

## One attempt only! (for the Moodle Qns)

Sample Final Exam

**COMP9319 Web Data Compression & Search 2021 T2**

**Final Exam (Sample)**

**The actual final exam has more & different questions but will be of the same style and format.**

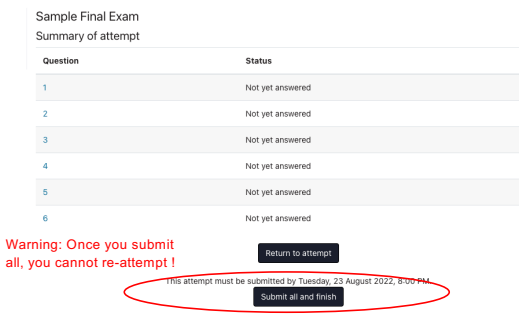
**Please refer to the Live Lecture of Week 10 regarding further info and hints about the final exam.**

Exam Conditions (please read carefully)

- You can attempt this exam from 13:30 to 16:00 Wednesday 24 August 2021 Sydney time.
- You have a maximum of 2.5 hours within the above period to complete and submit the exam.
- You may change your Moodle answers many times before you click "Finish Attempt" and before the exam ends. **You are not allowed to make any changes to your Moodle answers after you click "Finish Attempt" or the exam ends.**
- When the exam ends at 16:00 Wednesday 24 August 2021 Sydney time, any attempting exam in Moodle will be closed and submitted automatically.
- For the programming question, you need to submit it manually using give on a CSE machine. Only submissions before 16:00 Wednesday 24 August 2021 Sydney time will be marked.

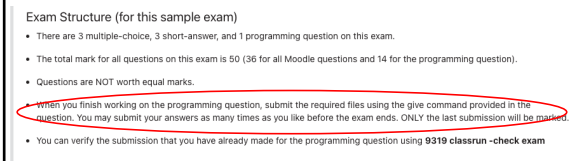
66

## One attempt only! (for the Moodle Qns)




67

## Multiple “give”




68

## Hints

- Total 50 pts:
  - 20 M.C. (20 pts)
  - 10 short-answer qu
  - 1 short programmi

69

## Hints (con’t)

- M.C. and short-answer questions:

eptual qns (avg 2min),
ut (avg 5min)
ulator (make sure you have

ected fr a qn bank, so if
e exam, please
qn in your email

70



## Hints (con’t)

- M.C. and short-answer questions:
  - Should take approx 1.5 hr to complete.
  - You will run out of time if you try to study or “search for an answer”.
  - So make sure you study lectures and practise the exercises before hand.
  - Go to the wk10/wk11/wk12 consultations if you have qns

71

## Hints (con’t)

- Same as normal exam papers, this exam has been checked by at least one other staff member. We rarely got questions during the previous exams, and if we did, the answer would usually be " *the question is as it is written* ". We're not expecting it to be any different in this exam.

72

## Hints (con't)

- The programming question:
  - In C or C++, you need to provide a makefile
  - Test on CSE machines
  - No specific performance requirements
  - Will manually check code for some partial marks (if your program doesn't work & your code is readable)

73

## Hints (con't)

- The programming question:
  - A short program, should be able to finish and test it in 30-60mins.
  - Test cases will be small and you don't have to use dynamic memory management such as pointers.

74

## Hints (con't)

- Overall, the exam questions are:
  - Direct & straightfo
  - Most Moodle ques exercises' style
- Exam preparation:
  - Study the lectures (slides & recordings)
  - Exercises
  - **Come to the consultations if any questions**

75

## Finally

due very soon

**Please** give some constructive

**u** !

76



## The End – *my last exercise for you* ☺

edoy$ogb

kuo$cdogl

77