COMP9319 Web Data Compression and Search

Course revision, a2,
exam

Agenda

- Course revision
- Solutions to a2 challenges
- About the final exam

Announcements

- Final exam, in-person, 2hrs, 1:45-4:00pm August 24 Thursday
- Final consultations - week 11 (help before the exam): all online (refer to the WebCMS timetable for detail).
- a2 marking has started, but each test set takes long – results to be released possibly in week 12.

Revision

COMP9319 – The foundations of

- how different compression tools work.
- how to manage a large amount of data on small devices.
- how to search gigabytes, terabytes or petabytes of data.
- how to perform full text search efficiently without added indexing.
- how to query distributed data repositories efficiently (optional).

Course Aims

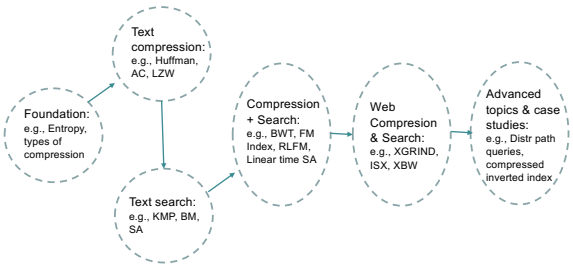
As the amount of Web data increases, it is becoming vital to not only be able to search and retrieve this information quickly, but also to store it in a compact manner. This is especially important for mobile devices which are becoming increasingly popular. Without loss of generality, within this course, 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.

Summarised schedule

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

Topics



Topic Snapshots

Questions to discuss (www)

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

Compression

- Minimize amount of information to be stored / transmitted
- Transform a sequence of characters into a new bit sequence
  - same information content (for lossless)
  - as short as possible

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

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*)

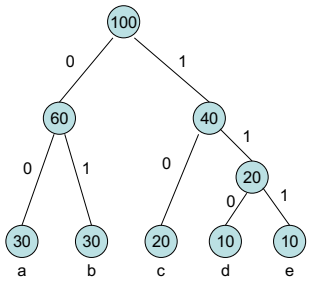
Entropy

- 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)

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

Huffman coding

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



Arithmetic coding (encode)

New Character	Low value	High Value
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPACE	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

Arithmetic coding (decode)

Encoded Number	Output Symbol	Low	High	Range
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	SPACE	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.2	0.3	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

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;
    }
```

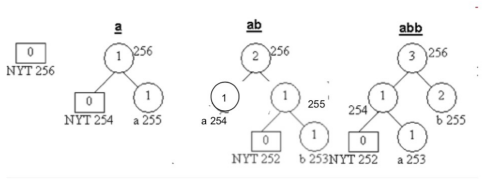
LZW Decompression

```
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;
     }
```

Adaptive Huffman

abbbbba: 0110000101100010011000100110001001100010011000100110001

abbbbba: 011000010011000101111101



a: 01100001

b: 01100010

Modified from Wikipedia

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)

InverseBWT(S)

function inverseBWT (string s)

create empty table

repeat length(s) **times**

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)

Other ways to reverse BWT

Consider L=BWT(S) is composed of the symbols $V_0 \ldots V_{N-1}$, the transformed string may be parsed to obtain:

- The number of symbols in the substring $V_0 \ldots V_{i-1}$ that are identical to V_i. (i.e., Occ[ ])
- For each unique symbol, V_i, in L, the number of symbols that are lexicographically less than that symbol. (i.e., C[ ])

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

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/

Pattern Matching

- Brute Force
- Boyer Moore
- KMP

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(**∑∑**)* = {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.

Theory of DFAs and REs

- RE. Concise way to describe a set of strings.
- DFA. Machine to recognize whether a given string is in a given set.
- **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.

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

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.**

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

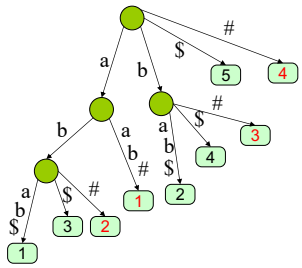
Generalized suffix tree (Example)

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$	

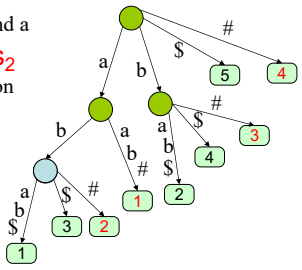
}



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"



Suffix array

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

Let s = abab

Sort the suffixes lexicographically:
ab, abab, b, bab

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

3	1	4	2

Example

Let S = mississippi

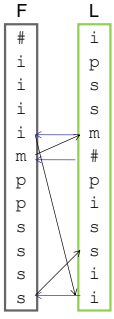
Let P = issa

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

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….



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

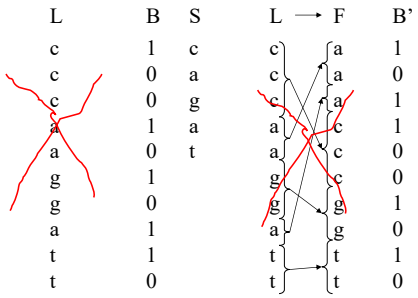
	F	L
First →	#	mississipp i
	i	#mississip p
	i	ppi#missis s
	i	ssippi#mis s
Last →	i	ssissippi# m
	m	ississippi #
	p	i#mississi p
	p	pi#mississ i
	s	ippi#missi s
	s	issippi#mi s
	s	sippi#miss i
	s	sissippi#m i

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 $\leq$ 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⟩

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



Changes to formulas

- Recall that we need to compute $C_T[c]+\text{rank}_c(L,i)$ in the backward search.
- **Theorem:** $C[c]+\text{rank}_c(L,i)$ is equivalent to $\text{select}_1(B',C_S[c]+1+\text{rank}_c(S,\text{rank}_1(B,i)))-1$, when $L[i] \neq c$, and otherwise to $\text{select}_1(B',C_S[c]+\text{rank}_c(S,\text{rank}_1(B,i)))+ i-\text{select}_1(B,\text{rank}_1(B,i))$.

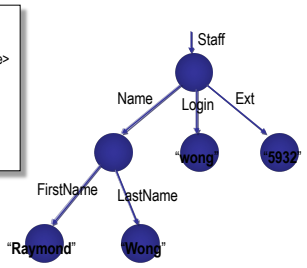
Linear time suffix array construction

- Consider the popular example string S:
- **`bananainpajamas$`**

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

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

```
<Staff>
  <Name>
    <FirstName> Raymond </FirstName>
    <LastName> Wong </LastName>
  </Name>
  <Login> wong </Login>
  <Ext> 5932 </Ext>
</Staff>
```



XPath for XML

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

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

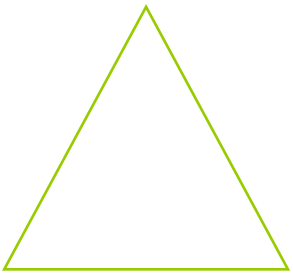
/bib/book[author/text()]

Path query evaluation

Top-down

Bottom-up

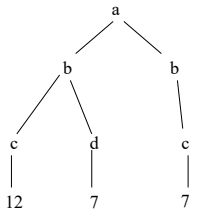
Hybrid



XPath evaluation

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

/ a / b [c = “12”]



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

Two techniques

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

XMill

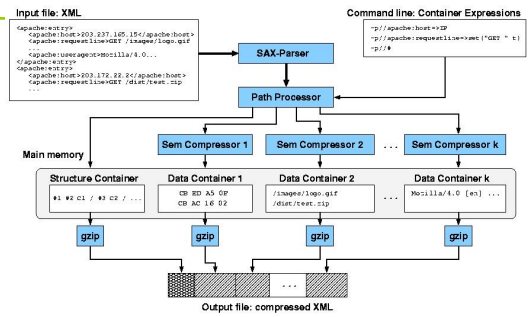


Figure 4: Architecture of the Compressor

XGRIND

Original Fragment:

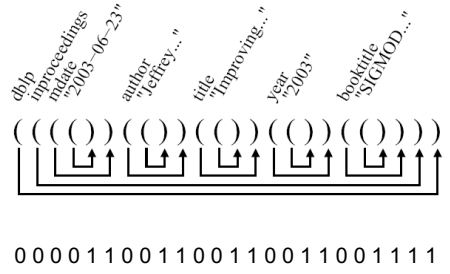
<student name=“Alice“>
<a1>78</a1>
<a2>86</a2>

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

Compressed Fragment:

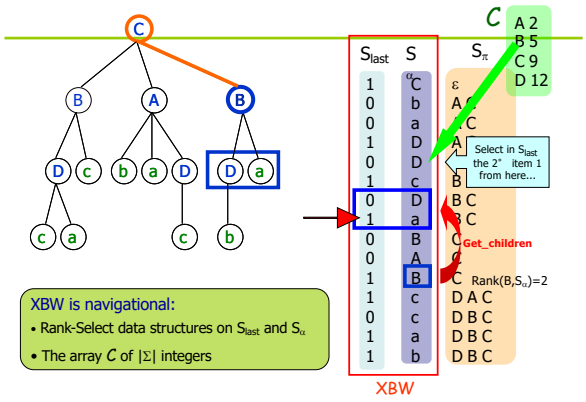
T0 A0 nahuff(Alice)
T1 nahuff(78) /
T2 nahuff(86) /
T3 nahuff(91) /
T4 nahuff(87) /
/

ISX: Balanced Parenthesis Encoding



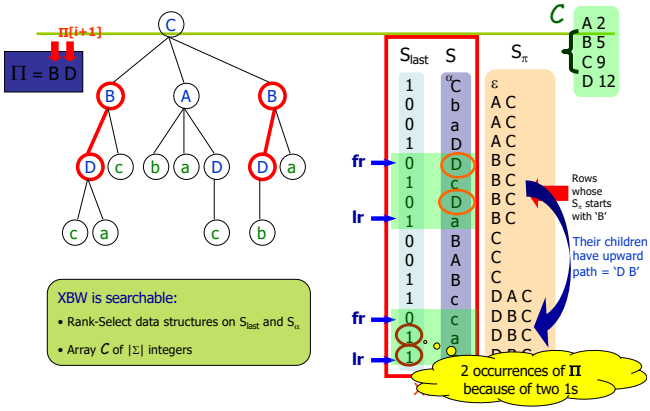
0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1

XBW is navigational



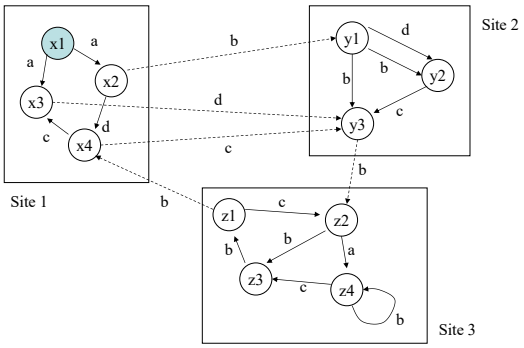
XBW is navigational:
- Rank-Select data structures on S_{last} and S_α
- The array C of $|\Sigma|$ integers

XBW is searchable (count subpaths)



XBW is searchable:
- Rank-Select data structures on S_{last} and S_α
- Array C of $|\Sigma|$ integers

Distributed path queries



Distributed path query processing

- Given a query, we compute its automaton
- Send it to each site
- Start an identical process at each site
- Compute two sets Stop(n, s) and Result(n, s)
- Transmits the relations to a central location and get their union

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

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.

Case studies: e.g., Content optimization

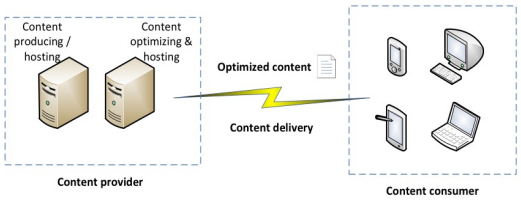


Figure 2. Delivery of content with content optimization

Covered Topics

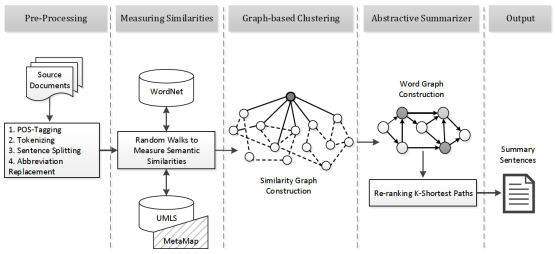
1. Entropy & basic compressions (RLE, Huffman, AC, LZW, Adaptive Huffman)
2. Pattern matching (Brute Force, KMP, BM); Regular Expression & Finite Automata; Inverted Index & Signature Files.
3. Suffix tree, Suffix array, BWT, MTF, FM Index, RLFM, O(n) SA construction.
4. Semistructured Data, XML & XPath; 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.

What have not been covered?

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

Future Directions

- Lossy (text) compression: summarization, topic modeling, …



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

Learning outcomes (a succinct 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
  - scalable computations (with limited resources) - e.g., when dealing with big text data

Assignment 2

- We have started the marking process this week
- Final marking script and test cases to be released later next week (week 11)
- Aim to finish marking and release results in week 12 – before your exam in week 13

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

Assignment 2 (general)

- What index structures to build
- Time & space to build index
- Size to keep the data + index
- Storage, memory, CPU, time considerations & trade-offs
- Practical implementation bottlenecks vs algorithmic complexity

Assignment 2 (The settings)

- “Backward” search $\Longleftrightarrow$ decode
- Dealing with the runlengths
- Gaps for Occ
- Dealing with blocks
- The unique [record id]
  - Consecutive but can start from any +ve int
- The last [record id]

Assignment 2 (The settings)

[5]apple[6]orange[7]banana

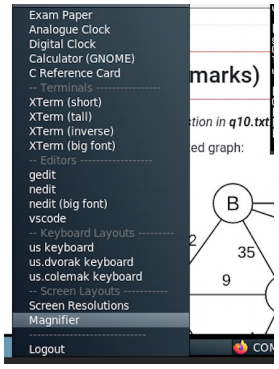
Assessment

```
a1        = mark for assignment 1      (out of 15)
a2        = mark for assignment 2      (out of 35)
asgts     = a1 + a2                    (out of 50)
exam      = mark for final exam        (out of 50)
okEach    = exam > 20                  (after scaling)
mark      = a1 + a2 + exam
grade     = HD|DN|CR|PS  if mark >= 50 && okEach
          = FL           if mark <  50 && okEach
          = UF           if !okEach
```


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-sit policy. Apply for special consideration asap.
- It's a 2 hr in-person exam (13:45-16:00 Aug 24, 2023).
- Supp exam covers the same scope & CLOs but may be of a different format, e.g., an oral exam.

Exam environment



Exam paper

COMP9319 23T2 Final Exam

Thursday 24th August 2023

Marks: 100

Time: 10 minutes reading + 2 hours working

Please read all of the instructions below while you are waiting.

Instructions

Start-of-exam Instructions

- Place your student card/other photo ID on the desk.
- Wait for the supervisor to announce the start of reading time.
- Once reading time has started, click the button below to reveal the questions.

End-of-exam Instructions

- Stop working when your supervisor tells you to do so.
- Log out from your workstation.
- Take all of your belongings.

Exam paper

Exam Instructions

- All the files you need to work on the exam are in your home directory.
- Right click on the desktop to bring up the menu and open a terminal.
- Follow the submission instructions under each question to submit your work.
- You can submit your answers multiple times. Only your last submission will be marked.
- **Do not wait until just before the end of the exam to submit all your answers. Submit each question as you finish working on it.**
- To check what you have submitted so far, use the `submit` command (with no arguments).
- If you have any questions during the exam, you must raise your hand and ask a supervisor.
- The following conditions apply to programming questions:
  - Solutions which do not attempt to solve the question generally but instead only hardcode return values for specific tests will receive zero marks.
  - Code style is not marked (*except* that global variables and static variables are strictly forbidden). However, good style may help a marker understand your code better, which will give you a greater chance of being awarded partial marks if your code does not work.
  - We will not measure the timing of your program, however, any test that runs more than 10 seconds will be terminated (e.g., in case it may have gone into an infinite loop).
  - Memory leaks/errors will not be penalised. **However**, we advise that you ensure there are no memory errors in your code, as programs containing memory errors are not guaranteed to behave correctly or consistently. A program that works when you test it during the exam but contains memory errors may not work during autotesting.

During the exam

Same as previous exam papers, this exam has been checked by at least one other staff member. We rarely got questions during previous COMP9319 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.

Exam structure

Structure

This exam consists of two parts:

- A programming question (18 marks)
- Written short-answer questions (82 marks)

There are **13 questions** in total.

Resources

The lecture slides from this term are available for you to use. You do not need to reference them.

You can access them via this page: **lecture slides**

Exam question

Question 5 (5 marks)

*Write your answers for each of the following questions (a)-(b) in **q5.txt**.*

Given the text string below:

jejunojejunostomy

(a) (2 marks)

What is its entropy? (round to 2 decimal places)

(b) (3 marks)

What is the average number of bits needed for each letter, using static Huffman code? (round to 2 decimal places)

Submission

Submit using the following command:

```
$ submit q5
```

q5.txt

Question 5

Please replace "X" inside the square brackets with your answer. Do not remove the square brackets.

(a)

ANSWER: [X]

(b)

ANSWER: [X]

q5.txt

Question 5

Please replace "X" inside the square brackets with your answer. Do not remove the square brackets.

(a)

ANSWER: [2.98]

(b)

ANSWER: [3.00]

q5.txt

Question 5

Please replace "X" inside the square brackets with your answer. Do not remove the square brackets.

(a)

ANSWER: [2.98]

(b)

ANSWER: [3]

Finally

MyExperience: due very soon

Please give some constructive feedback, **thank you** !

The End – *my last exercise for you* ☺

edoy$ogb

kuo$cdogl