Data Compression Techniques (Spring 2012)
Course Exam, 01 March 2012. Solutions.

1. [*2+2+2+2+2+2 points*] What is the *main difference* between the concepts in the following pairs:

   (a) prefix codes versus arithmetic coding

   Solution. Prefix codes have integer lengths and (typically) map each source symbol to one code. Arithmetic coding maps the whole source string to one code. The resulting coding can then be seen as prefix codes having fractional code lengths.

   (b) gamma code versus delta code

   Solution. Gamma codes use unary coding to encode the length of the binary representation, whereas delta codes use a gamma code to represent the length.

   (c) entropy versus zeroth order empirical entropy

   Solution. The entropy is defined for an arbitrary probability distribution, whereas the 0-th order empirical entropy is defined on the frequency of symbols in a given string.

   Grading. Full points for stating the main difference. One point subtracted for just defining the consept(s) and/or stating a secondary difference.

   Define the following concepts:

   (d) zero frequency problem

   Solution. When an adaptive encoding process encounters a source symbol for the first time, the symbol has a probability equal to zero (since it has not occurred in the previously compressed part). This leads to problems because, in general, symbols having a zero probability cannot be encoded. The problem can be avoided by e.g. adding one to the counts of all symbols.

   (e) coarse optimality

   Solution. A text compression algorithm is called coarsely optimal if the compressed size of $T[0, n)$ is bounded by $nH_k(T) + o(n \log \sigma)$ bits for any $k = o(\log_\sigma n)$, where $H_k(T)$ is the k-th order entropy and σ is the alphabet size (more details on Slide 102).

   (f) searchable prefix sums

   Solution. Let $L[0, n)$ be a sequence of positive integers that sum up to u. A searchable prefix sums data structure supports the operations $sum_L(j) = \sum_{i<j} L[i]$ for $j \in [0, n]$, and $search_L(i) = \max\{j \in [0, n] | sum_L(j) \leq i\}$ for $i \in [0, u]$.

   Grading. Full points for stating the correct definition. One point subtracted for some errors.

   A few lines for each part is sufficient.

2. [*6+6 points*] Compress the text $T =$ `senselessness` using zeroth order semiadaptive Huffman coding.

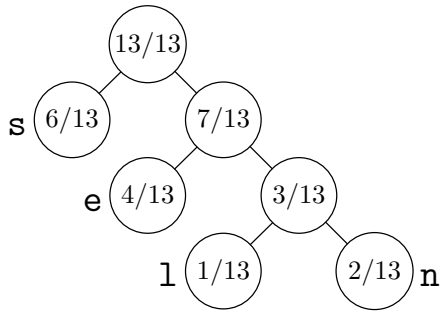
   (a) Construct the Huffman code using the symbol frequencies in the text. Encode T using the code.

   Solution.

The symbol frequencies are:

symbol	`e`	`l`	`n`	`s`
frequency	4/13	1/13	2/13	6/13

The Huffman tree looks like this:



and the codes are:

symbol	`e`	`l`	`n`	`s`
code	10	110	111	0

The encoded text is 0 10 111 0 10 110 10 0 0 111 10 0 0.

Grading.

Most received full points. The most common error was to use adaptive Huffman instead of semiadaptive. 1 point was awarded for these.

(b) The encoded text must be stored with additional information that is needed for decoding the text. Describe what information must be stored, and how to store it using as few bits as possible.

You may assume that the alphabet is fixed to {`e`, `l`, `n`, `s`} and the alphabetical order is `e` < `l` < `n` < `s`.

Solution.

The information that must be stored is the length of the text and the codes for all symbols. The length can be encoded using gamma or delta code. For the codes, there are several alternatives:

- Encode the symbol counts using, for example, gamma coding in the alphabetical order. The decoder can use the same algorithm as the encoder to reconstruct the code. In this case, the text length does not need to be encoded, since it is the sum of the symbol counts.
- Encode the text length n and then the symbol counts using codes of fixed length $\lceil \log(n+1) \rceil$. Even better, encode the symbol counts using interpolative encoding.
- It is enough to store the lengths of the codes. For any set of code lengths (satisfying Kraft's inequality), it is possible to construct a code with those lengths. The compressor and the decompressor just have to agree on the algorithm. In particular, they can agree to use the canonical codes.
  The code lengths can be encoded using unary or Golomb–Rice codes and listed in the alphabetical order of the symbols. There may be symbols that do not occur in the text and have no code assigned to them. (In this case there are none.) The code length 0 can be used for such symbols.

Grading.

Two points for text length and four points for the codes with half of the points for what and half for how.

3. [*6+6 points*] Let L = `rttrraa$ii` be the Burrows–Wheeler transform for a text T. The order of the symbols is `$` < `a` < `i` < `r` < `t`. The last character of T is `$`.

(a) What is T? Explain step-by-step the inverse Burrows–Wheeler transform, i.e., how T can be recovered from L.

Solution. $T = \texttt{ritaritar\$}$.

T can be recovered as follows. First, we stable sort the string L to get a new string F and an *LF-mapping* that maps from the j-th $s \in \Sigma$ in L to the j-th s in F. The LF-mapping can be used to reverse the Burrows–Wheeler transform: let $i_\$$ denote the position such that $L[i_\$] = \$$, then the original T is recovered in reversed order by

$$L[i_\$],\ L[LF[i_\$]],\ \ldots,\ L[LF^{n-1}[i_\$]],$$

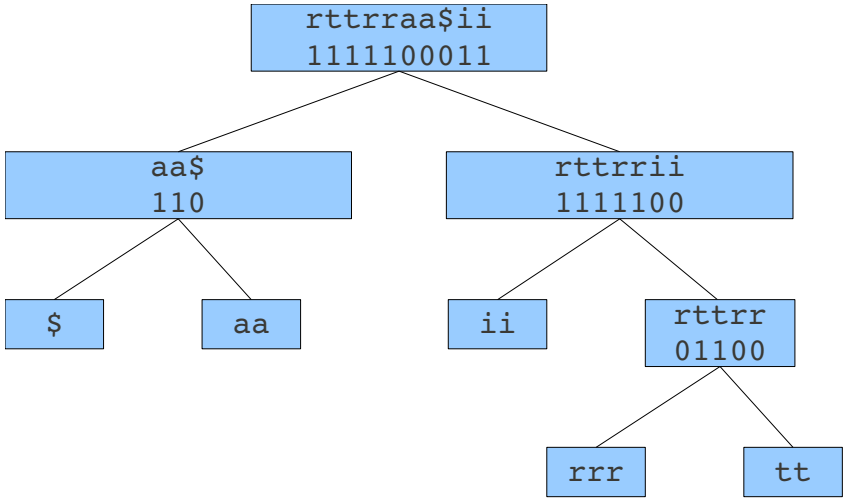
where $LF^j[i]$ denotes the recursion $LF^j[i_\$] = LF[LF^{j-1}[i_\$]]$.

Grading.

Full points for providing a correct solution and clear explanation. 1–3 points subtracted for missing explanation or errors.

(b) Give the wavelet tree of L. Explain how the wavelet tree can be used in the inverse Burrows–Wheeler transform.

Solution.



The wavelet tree supports the rank and select operations. Since the LF-mapping can be computed with e.g. $LF[i] = select_F(s, rank_L(s, i))$ where $s = access_L(i)$ (more details on Slide 132), we can use wavelet trees to compute the inverse BWT.

Grading.

Three points for giving the wavelet tree (any shape). Up to three additional points for explaining the connection to the inverse BWT.

4. [*7+7 points*] Compare LZ77 (without distance or length limits) and LZ78 parsings.

(a) Give an example of a text, where the number of phrases in the LZ78 parsing is much larger than in the LZ77 parsing.

Solution.

Let $T = \texttt{a}^n$. The LZ77 parsing has just two phrases $\texttt{a}$ and $\texttt{a}^{n-1}$. The phrases in the LZ78 parsing are

$$\texttt{a}, \texttt{aa}, \texttt{a}^3, \texttt{a}^4, \ldots, \texttt{a}^{k-1}, \texttt{a}^k, \texttt{a}^{n-k(k+1)/2}$$

where k is the largest integer such that $k(k+1)/2 < n$. Thus the number of phrases is $\Theta(\sqrt{n})$.

Grading.

Five points for a text with the main part periodic, i.e., of the form x^k for some string x and integer k. Full points for observing that the number of LZ77 phrases is constant with respect to k while the number of LZ78 phrases grows without bound. One point subtracted for some errors.

(b) Prove that the number of phrases in the LZ78 parsing is never smaller than in the LZ77 parsing.

Solution.

Informally, the idea is that any LZ78 phrase is of the form xs, where x is an earlier phrase and thus a subtring of the text. Therefore, xs is also a legal (but not necessarily the longest possible) LZ77 phrase. Thus an LZ77 phrase starting at position i is always at least as long as an LZ78 phrase starting at position i. The formal proof below covers also the general situation, where the starting positions are not synchronized.

Let T be a text with LZ78 parsing

$$T[i_0..i_1)T[i_1..i_2)T[i_2..i_3)\ldots$$

and LZ77 parsing

$$T[j_0..j_1)T[j_1..j_2)T[j_2..j_3)\ldots$$

We will show by induction that $i_k \leq j_k$ for all k.

Base case: $i_0 = 0 = j_0$.

Induction step: The assumption is that $i_{k-1} \leq j_{k-1}$ and we need to show that $i_k \leq j_k$.

Since $j_k \geq j_{k-1} + 1$, we have the following trivial cases:

- If the phrase $T[i_{k-1}...i_k)$ is a single symbol, $i_k = i_{k-1} + 1 \leq j_{k-1} + 1 \leq j_k$.
- $i_k \leq j_{k-1} + 1 \leq j_k$.

Otherwise, the phrase $T[i_{k-1}..i_k)$ is of the form $T[i_{l-1}...i_l)s$, where $l < k$ and s is a single symbol. Then $T[i_{l-1} + (j_{k-1} - i_{k-1})...i_l) = T[j_{k-1}...i_k - 1)$, i.e., $T[j_{k-1}...i_k - 1)$ has an earlier occurrence at $T[i_{l-1} + (j_{k-1} - i_{k-1})...i_l)$. Thus $T[j_{k-1}...i_k)$ is a possible LZ77 phrase (but not necessarily the longest possible). Therefore, $j_k \geq i_k$.

Grading.

Four points for the informal observation in the beginning of the solution above or something equivalent. Full points for dealing with the unsynchronized situation too.