**Data Compression Techniques**
Renewal/Separate Exam, 13 April 2012 at 16-20 Lecturer: Juha Kärkkäinen

*Please write on each sheet: your name, student number or identity number, signature, course name, exam date and sheet number. You can answer in English, Finnish or Swedish.*

1. [*2+2+2+2+2 points*] Define the following concepts:

   (a) zeroth order empirical entropy

   **Solution.** The zeroth order empirical entropy of a text $T$ over an alphabet $\Sigma$ is

   $$H_0(T) = -\sum_{s \in \Sigma} f_s \log f_s$$

   where $f_s$ is the relative frequency of the symbol $s$ in $T$.

   (b) grammar compression

   **Solution.** Grammar compression is a text compression method that represents a text $T$ using a context free grammar that generates $T$ and only $T$.

   (c) balanced parentheses sequence

   **Solution.** A balanced parenthesis sequence is a binary sequence $B[0..2n)$ satisfying $\text{excess}_B(2n) = 0$ and $\text{excess}_B(i) \geq 0$ for all $i \in [0..2n]$, where $\text{excess}_B(i) = \text{rank-1}_B(i) - \text{rank-0}_B(i)$. The 1-bits are interpreted as opening parentheses and the 0-bits as closing parentheses.

   What is the *main difference* between the concepts in the following pairs:

   (d) LZW versus original LZ78

   **Solution.** In LZW

   - initial dictionary contains all individual symbols (strings of length one)
   - each text phrase is encoded as a pointer to the dictionary
   - the text phrase *plus* the next symbol is added to dictionary.

   In LZ78

   - initial dictionary contains only the empty string.
   - each text phrase is encoded as a pointer to the dictionary *plus* one symbol
   - the text phrase is added to dictionary as such.

   (e) adaptive versus semiadaptive compression model

   **Solution.** An adaptive model is computed for every text prefix $T[0..i)$, $i \in [0..n)$, and then used for encoding $T[i]$. A semadaptive model is computed from $T[0..n)$ and then used for encoding each text symbol. In other words, the same semiadaptive model is used for all text symbols but the adaptive model can be different for each symbol.

   A few lines for each part is sufficient.

2. [*5+5 points*] Consider the following prefix code:

| symbol | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| code | 1010 | 111 | 00101 | 000 | 01 | 1011 | 110 | 0011 | 00100 |

(a) Show that the code is redundant, i.e., satisfies Kraft's inequality with strict inequality.
**Solution.**

$$2^{-4}+2^{-3}+2^{-5}+2^{-3}+2^{-2}+2^{-4}+2^{-3}+2^{-4}+2^{-5} = \frac{2+4+1+4+8+2+4+2+1}{32} = \frac{28}{32} < 1.$$

(b) Modify the code by deleting some bits in the codewords so that the result is a complete prefix code, i.e., satisfies Kraft's inequality with equality. You may not add or change any bits, only delete them.
**Solution.** Delete the third bit in the codewords for a and f:

| symbol | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| code | 100 | 111 | 00101 | 000 | 01 | 101 | 110 | 0011 | 00100 |

Then

$$2^{-3}+2^{-3}+2^{-5}+2^{-3}+2^{-2}+2^{-3}+2^{-3}+2^{-4}+2^{-5} = \frac{4+4+1+4+8+4+4+2+1}{32} = \frac{32}{32} = 1.$$

3. [*10 points*] Let $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ be the alphabet with the probability distribution

| symbol | a | b | c | d |
|---|---|---|---|---|
| probability | 0.4 | 0.2 | 0.1 | 0.3 |

Encode the string "bad" as a binary sequence using *exact* arithmetic coding. Give the intermediate steps in the encoding process. You may assume that the length of the string is known and does not need to be encoded.

**Solution.** First compute the source interval:

| string | interval |
|---|---|
| $\epsilon$ | $[0, 1)$ |
| b | $[0.4, 0.6)$ |
| ba | $[0.4 + 0.2 \cdot 0.0, 0.4 + 0.2 \cdot 0.4) = [0.4, 0.48)$ |
| bad | $[0.4 + 0.08 \cdot 0.7, 0.4 + 0.08 \cdot 1.0) = [0.456, 0.48)$ |

Then find the code interval, which is the largest dyadic interval that is completely contained by the source interval.

| code | interval |
|---|---|
| $\epsilon$ | $[0, 1)$ |
| 0 | $[0, 1 - 2^{-1}) = [0, 0.5)$ |
| 01 | $[0 + 2^{-2}, 0.5) = [0.25, 0.5)$ |
| 011 | $[0.25 + 2^{-3}, 0.5) = [0.375, 0.5)$ |
| 0111 | $[0.375 + 2^{-4}, 0.5) = [0.4375, 0.5)$ |
| 01110 | $[0.4375, 0.5 - 2^{-5}) = [0.4375, 0.46875)$ |
| 011101 | $[0.4375 + 2^{-6}, 0.46875) = [0.453125, 0.46875)$ |
| 0111011 | $[0.453125 + 2^{-7}, 0.46875) = [0.4609375, 0.46875)$ |

The code string is 0111011.

4. [*10 points*] What are the properties of the Burrows–Wheeler transform that make it a useful tool for higher order text compression? Make your answer as complete as possible. Use examples to illustrate your answer.

   **Solution.** The BWT is a transformation of a text that can be computed in linear time. It is invertible and the inverse transform can be computed in linear time too.

   The BWT is permutation of the text where the symbols are sorted by their right context. This means that the symbols with the same context are consecutive in the BWT. In a text, there is often a strong correlation between symbols and their contexts, which means that the consecutive symbols sharing a context have a strong correlation between each other. This correlation can be used for compression.

   Formally, one can prove the following result. For any text $T$ and any $k$

$$\sum_{w \in \Sigma^k} |L_w| H_0(L_w) = |T| H_k(T)$$

   where $L_w$ (a context block) is the symbols of $T$ with the right context $w$. In other words, zeroth order compression of the context blocks achieves $k$th order compression of the text. This is known as compression boosting.

   Furthermore, for most symbols $s$ in the BWT, the nearest preceding symbols share a long context with $s$, symbols that are a little further away share a short context with $s$, and symbols far away share no context at all. Thus an adaptive model that predicts each symbol using the preceding symbols and gives a higher weight to the nearest symbols is often a good predictor. Such models are called context oblivious and give rise to simple compression methods such run length encoding and move-to-front encoding.

5. [*10 points*] Let $M$ be a $n \times n$ sparse matrix that contains $m$ non-null entries. The non-null entries are integers from the interval $[0..\sigma)$. Design a compressed representation for $M$. The representation should support the following operations:

   - access$(i, j)$ returns the value at $M[i, j]$ (which may be null). The time complexity should be constant.
   - row$(i)$ returns all non-null values on the row $i$. The time complexity should be $\mathcal{O}(k + 1)$, where $k$ is the number of values returned.
   - column$(j)$ returns all non-null values on the column $j$. The time complexity should be $\mathcal{O}(k + 1)$, where $k$ is the number of values returned.

   The space complexity should be as small as possible. You may use any of the compressed data structures described on the lectures.

   **Solution.**

   The non-null entries are concatenated into an array $A[0..m)$ in row major order. For each row, there is a bit vector of length $n$ marking the non-null entries, and these are concatenated into a one bit vector $R$ of $n^2$ bits. Now

$$\text{access}(i, j) = A[\text{rank-1}_B(in + j)]$$

   and

$$\text{row}(i) = A[\text{rank-1}_R(in)..\text{rank-1}_R((i + 1)n)$$

   To implement column-operation, we store bit vectors for columns marking non-null entries and concatenated into a single bit vector $C$. Then

$$\text{column}(j) = \{\text{access}(\text{select}_C(i), j) \mid i \in [\text{rank-1}_C(jn)..\text{rank-1}_C((j + 1)n)\}$$

The bit vectors $R$ and $C$ are processed to support rank and in constant time. Then the operations have the required time complexities. Each bit vector can be stored in

$$n^2 H_0(R) + o(n^2) = m \log(n^2/m) + \mathcal{O}(m) + o(n^2)$$

bits. In addition, we need the space for the $m$ non-null entries.