Homework Set #5

Due 12 November 2008, before 12:00 noon, INR 032 or INR 038

Problem 1

Consider the finite state machines shown in Fig. 1, with input alphabet $\mathcal{X} = \{a, b\}$ and binary output alphabet.

(a) For each of the finite state machines, determine whether it is

- uniquely docodable.
- information lossless.

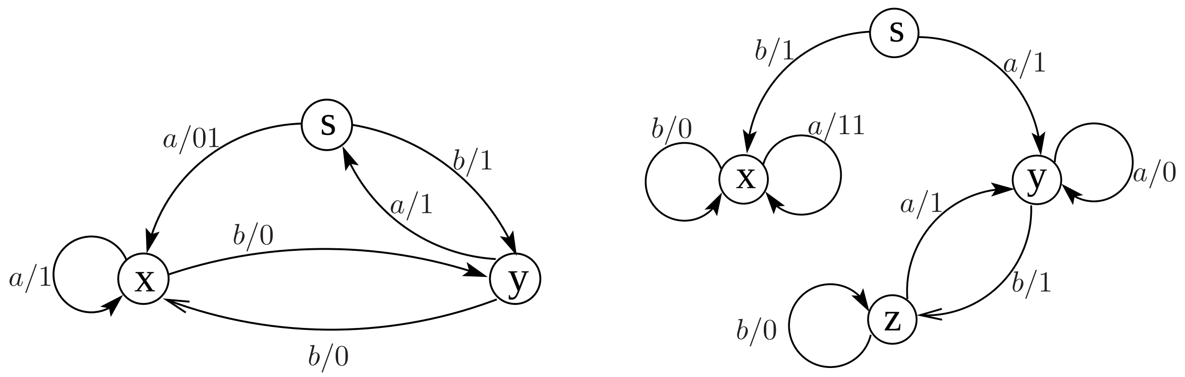
Let we feed the input sequence $bbabaababa$ to the FSM1 and assume that the initial state of the machine is “S”.

(b) Find the output of the FSM. What is the length of the output?

(c) Find the maximum number of distinct words the sequence can be parsed into.

(d) Apply the Lempel-Ziv compression algorithm studied in the class to this sequence, with the initial dictionary $\mathcal{D}_0 = \{a, b\}$. What is the length of the output?

(e) Find the number of distinct words the sequence is parsed into using the LZ algorithm.



(a) FSM1 (b) FSM2

Figure 1: Finite state machines

Problem 2 (Coding with Lempel-Ziv)

(a) Let symbols $x_1, x_2, \ldots, x_n$ come from alphabet $\mathcal{X} = \{a, b, c, d\}$. Using the Lempel-Ziv algorithm presented in class and the output alphabet $\{0, 1\}$, encode the sequence $abadcdadd$.

(b) Decode the sequence $0010000010001111 1$ which was encoded using Lempel-Ziv algorithm.

Consider the following variant of Lempel-Ziv called "sliding window Lempel-Ziv" or LZ77.

Assume that the string $x_1, x_2, \ldots, x_n$ is already compressed until position $i-1$. Then, look for the longest substring $x_i, \ldots, x_{i+l-1}$ starting at i such that there exists some $j \in \{i-W, \ldots, i-1\}$ and the substring $x_j, \ldots, x_{j+l-1}$ is the same as $x_i, \ldots, x_{i+l-1}$, where W is a positive integer. Hence, we are looking at the longest match that started in the window of length W before the current position of the cursor. Then, return a pointer to this past string (how many symbols before the current symbol the match occurred) and the length of the match (i.e. return $i-j$ and l).

For example, if $W = 4$ the string $ABBABBABBBAABABA$ will be parsed as follows: $A, B, B, ABBA, BB, BA, A, BA, BA$, which we can represent as $(0, A), (0, B), (1, 1), (3, 4),\ (3, 2), (4, 2), (1, 1), (3, 2), (2, 2)$. The first number indicates how many symbols ago the match started, with the convention that 0 means there was no match in the window (in which case the symbol is sent uncompressed). Hence, we need $\lceil \log(W+1) \rceil$ bits to represent the first number. The second number indicates the length of the match. In general this length can be longer than the size of the window but here we restrict ourselves to the case where $l \leq W$. We would begin the encoding by: $(000, 0), (000, 1), (001, 001), (011, 100), \ldots$.

(c) Assume $W = 7$ and $\mathcal{X} = \{a, b\}$. Using sliding window Lempel-Ziv algorithm, compress the sequence $ababababbbaabbaab$.

(d) Decode the sequence $0000000101001001111010 1100$ which was encoded using the sliding window algorithm.

Problem 3 (Lempel Ziv Welch algorithm and GIF compression)

In problem 2, both the finite sequences you encode and decode were in a "good" form; i.e., the last phrase could be parsed and be added to the dictionary or was already in the dictionary (in LZ78) or equivalently, the last phrase could be parsed as seen previously in the sequence (in LZ77). Though it is obvious that the last phrase does not affect the asymptotic optimality of Lempel-Ziv, it might be of interest to see how the practical versions of Lempel-Ziv terminates the encoding of finite sequences in GIF compression method for example. GIF is designed based on a variant of LZ78 modified by Welch (and known as LZW), and we focus only on this variant in this problem.
The basic encoding algorithm used in GIF is as follows:

```
1)   Initialize the dictionary by $\mathcal{X}$;
2)   [.c.] := empty;
3)   K := next character in charstream;
4)   Is [.c.]K in the dictionary?
     YES:{
         [.c.] := [.c.]K;
         go to [3];
```

```
}
NO:{
    output the code for [.c.] to the codestream;
    add [.c.]K to the dictionary;
    [.c.] := K;
    go to step 3;
}
```

Charstream is the sequence to be encoded, codestream is the encoded sequence, and [.c.] is the "current prefix" which is empty at the beginning and is formed to contain the largest sequence of symbols which is already in the dictionary. Afterwards, this prefix in encoded to codestream, [.c]K (K being the next character) is added to he dictionary and [.c] is set to K. The algorithm stops in step [3] when there is no next character in the charstream and just outputs the code for *[.c.]*. Note that since nothing is deleted from the dictionary (as opposed to the method studied in class), the last phrase would either be added to the dictionary or is already in the dictionary.
The decoding algorithm is as follows:

```
1)  Initialize the dictionary by $\mathcal{X}$;
2)  get the first code: CODE;
3)  output the string for CODE to the charstream;
4)  OLDCODE := CODE;
5)  CODE := next code in codestream;
6)  does CODE exist in the dictionary?
    YES:{
        output the string for CODE to the charstream;
        [...] := translation for OLDCODE;
        K := first character of translation for CODE;
        add [...]K to the dictionary;
        OLDCODE:= CODE;
    }
    NO:{
        [...] := translation for OLDCODE;
        K := first character of [...];
        output [...]K to charstream and add it to the dictionary;
        OLDCODE := CODE;
    }
7)  go to step 5;
```

Use the above algorithms to encode and decode the sequence $ABACABAB$ ($\mathcal{X} = \{A, B, C, D\}$).

Problem 4 (Lempel-Ziv Algorithm is Asymptotically Optimal)

Consider a first order 2-state Markov process with the transition matrix $\begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$.

(a) Find the stationary distribution of this Markov process ($[p_0, p_1]$).

(b) Imagine that the Markov process is in the state 0. How many steps does it take on average for the process to return to the state 0 again? (See that it is equal to $\frac{1}{p_0}$)

(c) Now consider blocks of length n of X_i's (X_0^{n-1}) to construct an extended Markov process. Imagine that the a block of length n moves 1 bit at a time to form the extended Markov process states. Forexample for $n = 3$, the sequence 00111010110 consists of the following extended states: $001, 011, 111, 110, 101, 010, 101, 011, 110$. Find the stationary distribution of this extended Markov process.

(d) How many steps does it take on average for the extended Markov process to return to the state x_0^{n-1} starting from the state x_0^{n-1}? (Use (b) to guess the answer even if you didn't prove it.)

(e) Consider each sequence which is to be encoded as a state of an extended Markov process and assume a LZ77 algorithm with infinite-length sliding window. Then to encode the block $x_0x_1\cdots x_{n-1}$, the last time we have seen these n symbols should be communicated. Call it $R_n(x_0x_1\cdots x_{n-1})$. Explain that the requested average number of steps in (d) is indeed

$$\mathbf{E}\{R_n(X_0X_1\cdots X_{n-1})|(X_0X_1\cdots X_{n-1}) = x_0x_1\cdots x_{n-1}\}.$$

(f) Verify the following inequalities and equalities.

$$\begin{aligned}
\lim_{n\to\infty}\frac{1}{n}\mathbf{E}l(X_0^{n-1}) &= \lim_{n\to\infty}\frac{1}{n}\mathbf{E}(\log R_n + 2\log\log R_n + O(1)) \\
&= \lim_{n\to\infty}\frac{1}{n}\sum_{x_0^{n-1}} p(x_0^{n-1})\mathbf{E}(\log R_n(X_0^{n-1})|X_0^{n-1} = x_0^{n-1}) \\
&\leq \lim_{n\to\infty}\frac{1}{n}\sum_{x_0^{n-1}} p(x_0^{n-1})\log\mathbf{E}(R_n(X_0^{n-1})|X_0^{n-1} = x_0^{n-1}) \\
&= H(\mathcal{X})
\end{aligned}$$

Hint: If $k \leq m$ is to be encoded, $\log m$ bits are needed. How about the case when there is no upper bound on k? This is exactly the case for encoding R_n. Think about the following encoding for R_n:

$$C(R_n) = 00\cdots 01x_1x_2\cdots x_l$$

where there are $\lceil\log R_n\rceil$ zeros preceding 1, and $x_1\cdots x_l$ is R_n in binary. The length of this code is then $2\lceil\log R_n\rceil + 1$. But $00\cdots 01$ is the most inefficient code to describe $\lceil\log R_n\rceil$!! What if we use the proposed encoding scheme, to encode $\lceil\log R_n\rceil$ (instead of coding by $00\cdots 01$)?! How many bits in total would then be required to describe R_n?