COS 488	**Collaborator 1** : last name, first name
Spring 2013	**Collaborator 2** : last name, first name

Solution to HW 6, Problem I.23

Al-Naji, Nader

Alice wants to communicate n bits of information to Bob over a channel (a wire, an optic fiber) that transmits 0,1-bits but is such that any occurrence of 11 terminates the transmission. Thus, she can only send on the channel an encoded version of her message (where the code is of some length $l \geq n$ that does not contain the pattern 11.

Here is a first coding scheme: given the message $m = m_1 m_2 ... m_n$, where $m_j \in \{0, 1\}$, apply the substitution: $0 \to 00$ and $1 \to 10$; terminate the transmission by sending 11. This scheme has $l = 2n + O(1)$, and we say that its *rate* is 2. Can one design codes with better rates? with rates arbitrarily close to 1, asymptotically?

Let C be the class of allowed code words. For words of length n, a code of length $L \equiv L(n)$ is achievable if and only if there exists a one-to-one mapping from $\{0, 1\}^n$ into $\bigcup_{j=0}^{L} C_j$, ie, $2^n \leq \sum_{j=0}^{L} C_j$. Working out the OGF of C, one finds that necessarily:

$$L(n) \geq \lambda n + O(1)$$
$$\lambda = \frac{1}{log_2 \phi} = 1.440420$$
$$\phi = \frac{1 + \sqrt{5}}{2}$$

Thus, no code can achieve a rate better than 1.44; ie a loss of at least 44% is unavoidable.

Solution:
We fill in the details above to verify the claim. In what follows, we ignore the added cost of inserting 11 to end the transmission. We begin by computing the number of words that do not contain any string 11, which we can find by defining and computing coefficients for the class of all such strings. Begin by denoting the class of all strings that do not contain an instance of 11 by C. This class can be recursively defined as an empty string, a string that contains a single one, or a string that contains a zero or an instance of 10, followed by another string from the class. Denote the class containing the empty string by ϵ, the class containing the single string 1 by Z_1, the class containing the single string 0 by Z_0, and the class containing the single string 10 by Z_{10}. The OGF's for these classes are then 1, z, z,

and z^2 respectively. Using the symbolic method yields:

$$\begin{aligned}
C &= \epsilon + Z_1 + (Z_0 + Z_{10}) \times C && \text{(Symbolic method)} \\
\Rightarrow C(z) &= 1 + z + (z + z^2)C(z) && \text{(Converting to OGF)} \\
\Rightarrow \sum_{n\geq 0} (C_n)\, z^n &= \sum_{n\geq 0} (\delta_0 + \delta_1 + C_{n-1} + C_{n-2})\, z^n && \text{(Rewriting } C(z)\text{)} \\
\Rightarrow C_n &= \delta_0 + \delta_1 + C_{n-1} + C_{n-2} && \text{(Matching coefficients)} \\
\Rightarrow C_n &= C_{n-1} + C_{n-2};\ C_0 = 1;\ C_1 = 2
\end{aligned}$$

This recurrence looks very familiar to Fibonacci but with different initial conditions. We can use Theorem 2.2 on page 55 to solve the recurrence. Theorem 2.2 is a general strategy for solving linear recurrences with constant coefficients. It states that all solutions to the recurrence:

$$a_n = x_1 a_{n-1} + x_2 a_{n-2} + ... + x_t a_{n-t}$$

for $n \geq t$ can be expressed as a linear combination of terms of the form $n^j \beta^n$ where β is a root of the characteristic polynomial:

$$q(z) \equiv z^t - x_1 z^{t-1} - x_2 z^{t-2} - ... - x_t$$

and j is such that $0 \leq j < \nu$ if β has multiplicity ν.

The characteristic polynomial for the recurrence described above is therefore:

$$\begin{aligned}
&q(z) = z^2 - z - 1 \\
&\Rightarrow z = \frac{1+\sqrt{5}}{2} \equiv \phi \text{ and } z = \frac{1-\sqrt{5}}{2} \equiv \hat{\phi}
\end{aligned}$$

Theorem 2.2 says that the solution is $F_n = c_0 \phi^n + c_1 \hat{\phi}^n$. Applying the initial conditions:

$$\begin{aligned}
C_0 &= 1 = c_0 + c_1 \\
C_1 &= 2 = c_0 \phi + c_1 \hat{\phi}
\end{aligned}$$

yields the solution

$$\begin{aligned}
c_0 = \frac{1}{\sqrt{5}}\phi^2,\ c_1 &= -\frac{1}{\sqrt{5}}\hat{\phi}^2 \\
\Rightarrow C_n &= \frac{1}{\sqrt{5}}\left(\phi^{n+2} - \hat{\phi}^{n+2}\right)
\end{aligned}$$

Now that we have the number of code words of length n that do not contain the string 11, we can solve the given inequality:

$$\sum_{j=0}^{L} C_j \geq 2^n \qquad \text{(Want to solve for } L\text{)}$$

$$\begin{aligned}
\sum_{j=0}^{L} C_j &= \sum_{j=0}^{L} \frac{1}{\sqrt{5}} \left(\phi^{n+2} - \hat{\phi}^{n+2} \right) \\
&= \sum_{j=0}^{L} \frac{1}{\sqrt{5}} \phi^{n+2} - \sum_{j=0}^{L} \frac{1}{\sqrt{5}} \hat{\phi}^{n+2} \\
&= \phi^2 \frac{\phi^{L+1} - 1}{\sqrt{5}(\phi - 1)} - \hat{\phi}^2 \frac{\hat{\phi}^{L+1} - 1}{\sqrt{5}(\hat{\phi} - 1)} \\
&= \phi^2 \frac{\phi^{L+1} - 1}{\sqrt{5}(\phi - 1)} + \hat{\phi}^2 \frac{1 - \hat{\phi}^{L+1}}{\sqrt{5}(\hat{\phi} - 1)}
\end{aligned}$$

$$\Rightarrow \phi^2 \frac{\phi^{L+1} - 1}{\sqrt{5}(\phi - 1)} + \hat{\phi}^2 \frac{1 - \hat{\phi}^{L+1}}{\sqrt{5}(\hat{\phi} - 1)} \geq 2^n \qquad \text{(Want to solve for L)}$$

$$\Rightarrow (\phi^{L+1} - 1) \frac{\phi^2}{\sqrt{5}(\phi - 1)} \geq 2^n - (1 - \hat{\phi}^{L+1}) \frac{\hat{\phi}^2}{\sqrt{5}(\hat{\phi} - 1)}$$

$$\Rightarrow c_0 \phi^{L+1} - c_0 \geq 2^n + c_1 \qquad \text{(Where } c_0 \geq 0 \text{ and } c_1 \geq 0\text{)}$$

$$\Rightarrow c_0 \phi^{L+1} \geq 2^n + c_2 \qquad \text{(Where } c_2 = c_0 + c_1 \geq 0\text{)}$$

$$\Rightarrow L \lg \phi + \lg \phi + \lg c_0 \geq \lg(2^n + c_2) \qquad \text{(Log both sides)}$$

$$\Rightarrow L \geq \frac{\lg(2^n + c_2)}{\lg \phi} - \frac{\lg c_0 + \lg \phi}{\lg \phi} \qquad \text{(Solve for } L\text{)}$$

$$\Rightarrow L \geq \frac{\lg(2^n)}{\lg \phi} - \frac{\lg c_0 + \lg \phi}{\lg \phi} \qquad \text{(Since } c_2 \geq 0\text{; weakens bound)}$$

$$\Rightarrow L \geq \frac{n}{\lg \phi} - \frac{\lg c_0 + \lg \phi}{\lg \phi} \qquad \text{(Simplifying)}$$

$$\Rightarrow L \geq \frac{n}{\lg \phi} + O(1) \qquad \text{(Final result)}$$

Thus, a code cannot achieve a rate of better than $\frac{1}{\lg \phi}$ asymptotically.

Solution to HW 6, Problem I.43

Al-Naji, Nader

Logarithmic differentiation of $H(z)$ provides for the H_n a recurrence by which one computes H_n in time polynomial in n.

Solution:
From the book, we have:

$$\mathcal{H} = \mathcal{Z} \times MSet(\mathcal{H})$$
$$\Rightarrow H(z) = z \prod_{m=1}^{\infty} (1 - z^m)^{-H_m}$$

Taking the log and differentiating yields:

$$\log H(z) = \log(z) - \sum_{m=1}^{\infty} H_m \log(1 - z^m)$$
$$\frac{d \log(H(z))}{dz} = \frac{1}{z} + \sum_{m=1}^{\infty} \frac{H_m m z^{m-1}}{1 - z^m}$$

We now make use of the following identity, which follows from the chain-rule. Below, a $'$ indicates that we are taking a derivative with respect to z:

$$\frac{H'(z)}{H(z)} = \log(H(z))'$$

Plugging in our previous result gives us:

$$\begin{aligned}
\frac{H'(z)}{H(z)} &= \frac{1}{z} + \sum_{m=1}^{\infty} \frac{H_m m z^{m-1}}{1 - z^m} \\
\Rightarrow \frac{zH'(z)}{H(z)} &= 1 + \sum_{m=1}^{\infty} H_m m \frac{z^m}{1 - z^m} && \text{(Multiply by } z\text{)} \\
&= 1 + \sum_{m=1}^{\infty} H_m m \sum_{n=1}^{\infty} z^{mn} && \text{(Geometric expansion [1])}
\end{aligned}$$

Now, we would like to extract coefficients on each z^k term. That is, we would like to rewrite the sum above as $\sum_{k=1}^{\infty} f(k) z^k$ for some function $f(k)$. We can do this by making a simple observation. Suppose we are looking for the coefficient of z^k for some k. This coefficient will

come about as a result of the product of m and n, from the outer and inner sum respectively, equaling k: $mn = k$. The values of m, in the outer sum, that correspond to z^k will therefore be such that $m = \frac{k}{n}$ where $\frac{k}{n}$ is integral for some n or, more concisely, such that $m|k$, where this notation means m "divides" k. Thus, to extract coefficients of z^k, all we need to do is sum over the values of m that divide k. We do this below; in what follows, we define $H_n = 0$ if $n < 0$:

$$\sum_{m=1}^{\infty} f(m) \sum_{n=1}^{\infty} z^{mn} = \sum_{m=1}^{\infty} \left(\sum_{k|m} f(k) \right) z^m \qquad \text{By above reasoning [2]}$$

$$\frac{zH'(z)}{H(z)} = 1 + \sum_{m=1}^{\infty} H_m m \sum_{n=1}^{\infty} z^{mn} \qquad ([1])$$

$$= 1 + \sum_{m=1}^{\infty} \left(\sum_{k|m} H_k k \right) z^m \qquad (\text{Applying } [2])$$

$$\Rightarrow zH'(z) = H(z) + \sum_{m=1}^{\infty} \left(\sum_{k|m} H_k k \right) z^m H(z)$$

$$= \sum_{n=0}^{\infty} H_n z^n + \sum_{m=1}^{\infty} \left(\sum_{k|m} H_k k \right) z^m \left(\sum_{n=0}^{\infty} H_{n-m} z^{n-m} \right) \qquad (\text{Expanding } H(z))$$

$$= \sum_{n=0}^{\infty} H_n z^n + \sum_{n=0}^{\infty} \left(\sum_{m=1}^{\infty} H_{n-m} \sum_{k|m} H_k k \right) z^n \qquad (\text{Rearranging})$$

$$= \sum_{n=0}^{\infty} \left(H_n + \sum_{m=1}^{n} H_{n-m} \sum_{k|m} H_k k \right) z^n \qquad (\text{Change index})$$

$$= \sum_{n=0}^{\infty} \left(H_n + \left(\sum_{m=1}^{n-1} H_{n-m} \sum_{k|m} H_k k \right) + H_{n-n} \sum_{k|n} H_k k \right) z^n \qquad (\text{Rewrite sum})$$

$$= \sum_{n=0}^{\infty} \left(H_n + \sum_{m=1}^{n-1} H_{n-m} \sum_{k|m} H_k k \right) z^n \qquad (\text{Because } H_0 = 0 \ [3])$$

Now, we have $zH'(z) = \sum_{n=0}^{\infty} f(n) z^n$. The last thing we need to do, is convert the left side

into a sum as well. We do this below:

$$\begin{aligned} H(z) &= \sum_{n=0}^{\infty} H_n z^n \\ \Rightarrow H'(z) &= \sum_{n=0}^{\infty} n H_n z^{n-1} \\ \Rightarrow zH'(z) &= \sum_{n=0}^{\infty} n H_n z^n && ([4]) \end{aligned}$$

Now, putting everything together, we have:

$$\begin{aligned} zH'(z) &= \sum_{n=0}^{\infty} \left(H_n + \sum_{m=1}^{n-1} H_{n-m} \sum_{k|m} H_k k \right) z^n && \text{(From [3])} \\ \sum_{n=0}^{\infty} (nH_n) z^n &= \sum_{n=0}^{\infty} \left(H_n + \sum_{m=1}^{n-1} H_{n-m} \sum_{k|m} H_k k \right) z^n && \text{(Applying [4])} \\ \Rightarrow nH_n &= H_n + \sum_{m=1}^{n-1} H_{n-m} \sum_{k|m} H_k k && \text{(Matching coefficients)} \\ \Rightarrow H_n &= \frac{\sum_{m=1}^{n-1} H_{n-m} \sum_{k|m} H_k k}{n-1} && \text{(Solving for } H_n\text{)} \\ H_n &= \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \frac{\sum_{m=1}^{n-1} H_{n-m} \sum_{k|m} H_k k}{n-1} & \text{otherwise} \end{cases} && \text{Final answer} \end{aligned}$$

This recurrence allows one to compute H_n in plolynomial time with the use of dynamic programming. To see this, start by considering the innermost sum $\sum_{k|m}$. If we have all of the $H_0, ..., H_{n-1}$ precomputed, it is clearly $O(n)$ since $m < n$. The outer sum then calls this inner sum $O(n)$ times, making the total number of operations $O(n^2)$ if we have all of the $H_0, ..., H_{n-1}$ precomputed. If we don't have the previous $H_0, ..., H_{n-1}$ precomputed, we must compute each value for H_k, which takes $O(k^2)$ time by the same reasoning. Thus, if we want to compute H_n from scratch and we have $O(n)$ space, we can do it in $O(n^3)$ time using this recurrence.

COS 488	**Collaborator 1** : last name, first name
Spring 2013	**Collaborator 2** : last name, first name

Solution to HW 6, Problem 1 (Programming)

Al-Naji, Nader

Determine the choice of four coins that maximizes the number of ways to change a dollar.

Solution:
The choice of coins is clearly $1, 2, 3, 4$. To see this, consider the following simple proof. First, 1 has to be in the set. If 1 isn't in the set and we instead have some coin values $k_2 > k_1 > 1$ in the set, then we can replace k_2 with 1 and every way to change a dollar that involves k_2 will yield *multiple* more ways using coins k_1 and 1, a contradiction. Now, let's say we have some coin value $k > 4$ in the set. Since we know 1 must be in the set, we can replace k with whatever value in the range $\{2, 3, 4\}$ that's missing and we will have strictly more ways to make change using that coin and 1 together. Thus, the choice of four coins that maximizes the number of ways to change a dollar is $1, 2, 3, 4$. This proof generalizes trivially, showing that the choice of k coins that maximizes the number of ways to make a dollar is $\{1, ..., k\}$.

We can also use the symbolic method to arrive at the same result. If we choose four coins of values k_1, k_2, k_3, k_4 respectively, then we can characterize the class of "change" as being composed of sequences of coins of each value. Denoting the class of change by C and denoting the class containing a single coin of value k_i by K_i, with OGF z^i, and using the symbolic

method yields:

$$
\begin{aligned}
\text{Let } S &\equiv \{k_1, k_2, k_3, k_4\} && \text{(Coin values)} \\
\text{Let } B &= K_{k_1} + K_{k_2} + K_{k_3} + K_{k_4} && \text{(Single coin)} \\
\Rightarrow B(z) &= z^{k_1} + z^{k_2} + z^{k_3} + z^{k_4} \\
\Rightarrow B(z) &= \sum_{k \in S} z^k \\
\Rightarrow B_n &= 1 \text{ if } n \in S \\
\text{Let } C &= MSET(B) && \text{(Number of combinations)} \\
\Rightarrow C(z) &= \prod_{n \geq 1} (1 - z^n)^{-B_n} \\
\Rightarrow \log C(z) &= \sum_{n \geq 1} -B_n \log(1 - z^n) \\
&= \sum_{n \in S} -\log(1 - z^n) \\
\Rightarrow \frac{d \log C(z)}{dz} &= \sum_{n \in S} \frac{n z^{n-1}}{1 - z^n} \\
\Rightarrow \frac{z C'(z)}{C(z)} &= \sum_{n \in S} \frac{n z^n}{1 - z^n} \\
&= \sum_{n \in S} n \sum_{m \geq 1} z^{nm} \\
&= \sum_{m \geq 1} \sum_{j \in S} \mathbf{1}_{j|m}[j] z^m \\
\Rightarrow z C'(z) &= \sum_{m \geq 1} \sum_{j \in S} \mathbf{1}_{j|m}[j] C(z) z^m \\
&= \sum_{m \geq 1} \sum_{j \in S} \mathbf{1}_{j|m}[j] \sum_{n \geq 0} C_{n-m} z^{n-m} z^m \\
&= \sum_{n \geq 0} \left(\sum_{m=1}^{n} \sum_{j \in S} \mathbf{1}_{j|m}[j] C_{n-m} \right) z^n \\
\Rightarrow n C_n &= \sum_{m=1}^{n} \sum_{j \in S} \mathbf{1}_{j|m}[k] C_{n-m} \\
\Rightarrow C_n &= \frac{\sum_{m=1}^{n} \sum_{j \in S} \mathbf{1}_{j|m}[j] C_{n-m}}{n}
\end{aligned}
$$

This gives us a polynomial time algorithm for computing C_n, which we can use to brute-force find the settings for $k_1, ..., k_4$ that maximize the number of combinations C_n when $n = 100$. The C++ program attached computes these settings in 15 seconds for $n = 100$.

Solution to HW 6, Problem 2 (Programming)

Al-Naji, Nader

Write code that computes H_n/H_{n-1} and P_n/P_{n-1}.

Solution:

The following Mathematica code computes H_n in polynomial time using the above recurrence and dynamic programming:

```
div[m_, n_]:=If[Mod[n, m] == 0, 1, 0]
h[0]:=0;
h[1]:=1;
h[n_]:=h[n] = Sum[h[n - m]Sum[div[k, m]h[k]k, {k, 1, m}], {m, 1, n - 1}]/(n - 1)
hratio[n_]:=N[h[n]/h[n - 1], 100]
hratio[100]
h[100]
```

```
2.940995378440972909814287949717829002528251
513843283516593268803371363950542982552779 70
```

The following Mathematica code computes P_n (from page 42) using the same technique:

```
p[0]:=1
p[1]:=1
p[n_]:=p[n] = Sum[DivisorSigma[1, j]p[n - j], {j, 1, n}]/(n)
pratio[n_]:=N[p[n]/p[n - 1], 100]
pratio[100]
p[100]
```

```
1.126097221309180781466629340711857170608912 8
190569292
```

It looks as if these ratios are approaching 3 and 1 respectively.