6.896 Sublinear Time Algorithms	March 1, 2007
Lecture 8	
Lecturer: Ronitt Rubinfeld	*Scribe: Jacob Scott*

1 Huffman Coding and Entropy

Consider a string $w = w_1 w_2 \ldots w_m$ on an alphabet $A = a_1 a_2 \ldots a_n$. We will now be considering our data as fixed, as opposed to being generated from a probability distribution as in previous lectures. Thus, we can consider the frequency of each letter in the alphabet, $p = \{p_1, p_2, \ldots, p_n\}$. We can now define a code $C = \{c_1, c_2, \ldots, c_n\}$ such that c_i is the "code word" for a_i. The following coding algorithm encodes w:

Coding Algorithm
scan left to right
$\rightarrow$ if $w_i = a_j$ write c_j

Choice of code
We would like to pick variable lengths from the c_i's to minimize

$$L(C) = \sum_i p(i)|c_i|$$

Which can be considered the expected length of a letter a_i drawn from p and written as c_i. *Shannon's Source Coding Theorem* relates this quantity to entropy as follows:

$$L(C) \geq H(p)$$

Huffman codes achieve this bound when for all i there is an integral j_i such that $p_i = 2^{-j_i}$.
Some examples of distributions and their entropies are:

1. $H(U_n) = \log n$
2. $H(p_1 = 1, p_{i>1} = 0) = 0$
3. If $p_1 = 1/2, p_2 = p_3 = \ldots = p_n$:

$$\begin{aligned} L(C) &\geq H(p) \\ &= -1/2 \log 1/2 + (n-1)\frac{1}{2(n-1)} \log \frac{1}{2(n-1)} \\ &= \log 2 + 1/2 \log \frac{1}{n-1} \end{aligned}$$

This is approximately half of the entropy of the uniform distribution.

4. If $p_i = 1/2^i$:

$$H(p) = 2$$

5. If $p_1 = p_2 = \ldots = \frac{1}{l}, p_{l+1} = \ldots = p_n = 0$:

$$H(p) = \log l$$

2 Distinct Colors

Before moving to talk about Lempel-Ziv compression, we will explore the following questions: how many distinct letters are there in a string? This problem arises in many areas, for example in the study of databases and statistics.

Formally, the **Colors Problem** is:

Given "colors" input $x_1 \dots x_n$, how many distinct colors (values) do the x_i's take on? We care about query complexity, and assume that accessing any x_i takes one query.

Considering unique integers rather than colors, there are two related settings with which to model the problem. The first is as described above, where the input is a string such as $s = 1111211311544123\dots$, and the output counts the number of different values we see. In the second, the input is a a probability distribution on the domain $[1, 2, \dots, |A|]$ with the following constraint: $p_i = \frac{l_i}{n}$ where l_i is an integer ≥ 0.

Here the number of colors is the size of its support $c = |\{i|p_i \neq 0\}|$. In this second model, we can sample from the distribution and try to estimate c. We now present one algorithm that gives a (weak) approximation of the answer for the first setting, though the algorithm can easily be adapted to the second setting by taking samples of the distribution whenever samples from the input string are called for.

A-multiplicative Approximation
Take $10n/A^2$ uniform samples (with replacement) from $x_1, ..., x_n$
Let $\hat{c}$ be the number of distinct colors/values in the samples.
Output $\hat{c} \cdot A$.

<u>Analysis</u>
If c is the actual number of distinct colors, then $\hat{c} \leq c$.

Claim 1 *Taking s samples from c colors where each color has probability $\geq \frac{1}{n}$ givesn $\geq \frac{cs}{10n}$ distinct colors with probability $\geq \frac{3}{4}$.*

This claim gives us a $n^{1/4}$ approximation in $O(\sqrt{n})$, a $n^{1/8}$ approximation in $O(n^{3/4})$, and a $\log n$ approximation in $O(n/\log^2 n)$. A $n^{1/2}$ approximation requires no queries, just output $\sqrt{n}$ as the number of unique colors is between 1 and n.

Proof Let X_i be a Bernoulli RV that is one iff color i is selected in s samples and 0 otherwise. Then $X = \sum X_i$ = number of distinct colors in sample.

$$\begin{aligned} E[X] = \sum_i E[X_i] &\geq c(1 - (1 - \frac{1}{n})^s) \\ &\geq c(1 - e^{-s/n}) \\ &\geq (1 - e^{-1})\frac{cs}{n} \end{aligned}$$

The third step comes from the inequality $(1 - e^{-x}) \geq (1 - e^{-1})x$ for $x \in [0, 1]$.

We will use the Chebyshev bound to show that X is close to its mean.

$$\begin{aligned} Var[X] &= E[X^2] - E[X]^2 \leq E[X^2] \\ &\leq E[X] \\ Pr[X \leq \delta E[X]] &\leq Pr[|X - E[X]| \geq (1 - \delta)E[X] \\ &\leq \frac{Var[X]}{(1-\delta)^2 E[X]^2} \\ &\leq \frac{1}{(1-\delta)^2 E[X]} \end{aligned}$$

We will set $\delta = 3 - \sqrt{8}$ and prove $Pr[X \geq \frac{cs}{10n}] \geq \frac{3}{4}$.

- Case 1: $E[X] \geq \frac{4}{(1-\delta)^2}$

$$\begin{aligned} Pr[X \geq \delta E[X]] &\geq \frac{3}{4} \\ X \geq \delta E[X] &\Rightarrow X \geq \delta(1-e^{-1})\frac{cs}{n} \\ &\Rightarrow X \geq \frac{cs}{10n} \end{aligned}$$

  The above steps come from substituting $E[X]$ and δ and simple algebra. Clearly, $Pr[X \geq \frac{cs}{10n}] \geq \frac{3}{4}$.

- Case 2: $E[X] < \frac{4}{(1-\delta)^2}$

$$\begin{aligned} (1-e^{-1})\frac{cs}{n} &\leq \frac{4}{(1-\delta)^2} \\ \frac{cs}{10n} &\leq \frac{4}{10(1-\delta)^2(1-e^{-1})} \leq 1 \end{aligned}$$

  Since $\frac{cs}{10n} \leq 1$, and we must have found at least one color, $Pr[X \geq \frac{cs}{10n}] = 1$.

■

When $s = \frac{10n}{A^2}$, $\frac{cs}{10n} = \frac{c}{A^2}$. Thus, with probability $\geq \frac{3}{4}$, $\frac{c}{A^2} \leq \hat{c} \leq c$. Thus, $\frac{c}{A} \leq \hat{c}A \leq cA$, and returning $\hat{c}A$ gives us an A-approximation.

Theorem 2 *A-approximating the number of distinct colors takes $\Theta(\frac{n}{A^2})$ queries.*

Proof (outline) We have just proved that this takes $O(\frac{n}{A^2})$ queries. To prove the lower bound, we take the string 0^n and insert A^2 unique colors (new numbers) at random to the string. A full proof would show that any deterministic algorithm using $o(\frac{n}{A^2})$ queries would be unable to distinguish between the two, and then use Yao's Min-Max principle to prove a lower bound for randomized algorithms. ■

3 Lempel Ziv

We now turn to the Lempel Ziv compression algorithm from LZ'77.

In this algorithm, we encode $w \rightarrow LZ(w)$, where each LZ codeword is either a alphabet character or a (pointer to previous location in input, length)-pair

The LZ compression algorithm works as follows:

$t \leftarrow 1$
repeat:
  find the longest substring $w_t w_{t+1} \dots w_{t+l-1}$ where
  $\exists$ index $p < t$ such that $w_p w_{p+1} \dots w_{p+l-1]} = w_t w_{t+1} \dots w_{t+l-1}$
  If none, next symbol $= w_t$
    $t \leftarrow t + 1$
  else next symbol $= (p, l)$ (the compressed segment
    $t \leftarrow t + l$

Some examples:

- $aaaaaaa \rightarrow a, (1, 6)$
- $abcd \rightarrow a, b, c, d$
- $abaabaaabaaa \leftarrow a, b, (1, 1), (1, 4), (4, 5)$

Note that we can't expect to compress most strings very well. This is because there are $2^{n+1} - 1$ strings of length $\leq n$, but only $2^n - 1$ strings of length $\leq n - 1$. What we are interested in is how compressible a given string is. One interesting parameter that gives a good approximation of this is the number of distinct substrings of length l, which is upper bounded by both 2^l and $n - l$.