Homework due 11/29. Please write the name of any collaborator you worked with.

1. Compression of switching sequences. Given $N$ compression algorithms $c_1, c_2, \ldots, c_N$, and an input sequence of length $T$, $x_1, x_2, \ldots, x_T \in \{0, 1\}$.

As we saw in class, we can design a compression algorithm that does as well as the best of these, with only a $\log N$ overhead. For example, if one compression algorithm was for French text and another for English, we could compress either. In this problem, imagine that you want to compress a document which has segments of English and French.

To be precise, you want to give a compression algorithm $c$ with the following guarantee, for all sequences $x_1, x_2, \ldots, x_T \in \{0, 1\}$, and for all $k > 0$ ($k$ is the number of segments), for all segmentations $1 = i_1 < i_2 \ldots < i_k < i_{k+1} = T + 1$,

$$|c(x)| \leq k(2 + \log_2 N) + \sum_{j=1}^{k} \min_{n \in \{1,2,\ldots,N\}} |c_n(x_{i_j}x_{i_j+1} \ldots x_{i_{j+1}-1})|.$$  

In words, the compression is as good as if you split the document into $k$ segments and compressed each one separately with the best code. The overhead (regret) is at most $k(2 + \log N)$.

2. Online prediction of switching sequences. Given $N$ probability distributions over binary sequences, specified online by $p_n(x_t = 1|x_1x_2\ldots x_{t-1}) \in [0, 1]$. That is, each probability distribution can be accessed as an oracle, where the input is a bit sequence and the output is the probability that the next bit is 1. (From this, one can compute $p_n(x_1x_2\ldots x_T) = \prod_{t=1}^{T} p_n(x_t|x_{t-1})$.) We want to define another probability distribution $p$, specified in the same way, with a similar guarantee as above.

In particular, we want, for all sequences $x_1, x_2, \ldots, x_T \in \{0, 1\}$, and for all $k > 0$ ($k$ is the number of segments), for all segmentations $1 = i_1 < i_2 \ldots < i_k < i_{k+1} = T + 1$,

$$p(x) \geq \frac{1}{(TN)^k} \prod_{j=1}^{k} \max_{n \in \{1,2,\ldots,N\}} |p_n(x_{i_j}x_{i_j+1} \ldots x_{i_{j+1}-1})|.$$
In words, the probability is as large as if you split the sequence into \( k \) segments and predicted each one separately with the best predictor. The overhead (competitive ratio) is at most \((TN)^k\). You’ll get more credit if you can give an efficient algorithm for computing \( p(x_t|x_1\ldots x_{t-1}) \) that calls the \( N \) component probability oracles a number of times that is polynomial in \( N \) and \( T \).

3. Lempel-Ziv. In class, we saw that the length of the encoding using Lempel-Ziv compression is at most \( N \log N \), where \( N \) is the number of phrases. It would be nice to guarantee that this is not much longer than the original sequence length \( T \). Show that this is the case, i.e., for any \( \epsilon > 0 \), for sufficiently large \( T \) (which may depend on \( \epsilon \)), Lempel-Ziv compression never encodes any input of length \( T \) by more than \((1 + \epsilon)\) factor. You may either argue directly based on the number of codewords, or using the optimality of Lempel-Ziv compression.

4. No truly universal compression algorithm. Show that there is no compression algorithm that compresses every sequence of length \( T \) to an encoded length of less than \( T \).