

CS 6351 DATA COMPRESSION

THIS LECTURE: LOSSLESS COMPRESSION PART II

Instructor: Abdou Youssef

OBJECTIVES OF THIS LECTURE

By the end of this lecture, you will be able to:

- Explain simple Markov models, apply them to model and describe binary streams of data with inter-symbol correlation, and use them for block coding and arithmetic coding
- Describe and apply Arithmetic Coding (AC) for lossless compression, and determine where it is applicable
- Explain how AC exploits the inter-symbol correlation and their conditional probability distribution
- Argue that AC achieves the true entropy of the source with memory (i.e., the entropy that accounts for all the modeled inter-symbol correlations)
- Describe and apply Lempel-Ziv (LZ) lossless compression, and determine where it is applicable
- Explain how it exploits fully the inter-symbol correlation and their conditional probability distribution without actually needing or using the probabilities (magic!)

OUTLINE

- Review of probability, especially conditional probability
- Markov Models of order 1
- Arithmetic Coding
- LZ compression

CONDITIONAL PROBABILITIES (1/4)

- Suppose that we have a binary input stream $x = x_1x_2x_3 \dots$ where the value of each bit x_i is highly correlated to (or dependent on) the values of the previous bits
- That dependence is quantified with conditional probabilities
- For example, suppose that every bit is probably = the previous bit
- Let's quantify that by saying that there is 90% chance that the current bit is equal to the previous bit (and thus there is a 10% it is different)
 - So, if previous bit=0, the current bit is 0 with 90% probability
 - if previous bit=1, the current bit is 1 with 90% probability
- We can write that as $\Pr[x_i = 0/x_{i-1} = 0]=0.9$, $\Pr[x_i = 1/x_{i-1} = 1]=0.9$

CONDITIONAL PROBABILITIES (2/4)

- We can write that as $\Pr[x_i = 0/x_{i-1} = 0]=0.9$, $\Pr[x_i = 1/x_{i-1} = 1]=0.9$
- We read “ $[x_i = 0/x_{i-1} = 0]$ ” as “ $x_i = 0$ given that $x_{i-1} = 0$ ”
- We read “ $\Pr[x_i = 0/x_{i-1} = 0]=0.9$ ” as “probability that $x_i = 0$ given that $x_{i-1} = 0$ is 0.9”
- By probability theory, the sum of probabilities of opposite events is 1
- Therefore, if $\Pr[x_i = 0/x_{i-1} = 0]=0.9$, then $\Pr[x_i = 1/x_{i-1} = 0]=0.1$
- Generally speaking, if probabilistic (i.e., random) event A can depend on another even B, then we can talk about the conditional probability of A given B, denoted $\Pr[A/B]$ or $\Pr[A|B]$, which stands for the prob. of A knowing that B occurred
- $\Pr[A/B]$ can be \neq from the probability of A ($\Pr[A]$) by itself (irrespective of B)
- Actually, we say that events **A and B are independent** if **$\Pr[A/B]=\Pr[A]$**

CONDITIONAL PROBABILITIES (3/4)

1. Probability of two events A and B occurring simultaneously is denoted

$$\Pr[A \text{ and } B] = \Pr[A, B] = \Pr[A \cap B]$$

2. **Formal definition of conditional probability:** $\Pr[A/B] = \frac{\Pr[A \cap B]}{\Pr[B]}$

3. Hence, $\Pr[B \cap A] = \Pr[A \cap B] = \Pr[B] \times \Pr[A/B]$ (and also $= \Pr[A] \times \Pr[B/A]$)

4. If A and B are independent, then $\Pr[A \cap B] = \Pr[A] \times \Pr[B]$

5. Note that event A can depend on multiple events, say B and C. We can then write $\Pr[A/(B \text{ and } C)] = \Pr[A/(B, C)] = \Pr[A/(B \cap C)] = \Pr[A/BC]$ to represent the probability of A given B and C (i.e., given that B and C both occurred)

6. Generalizing (4):

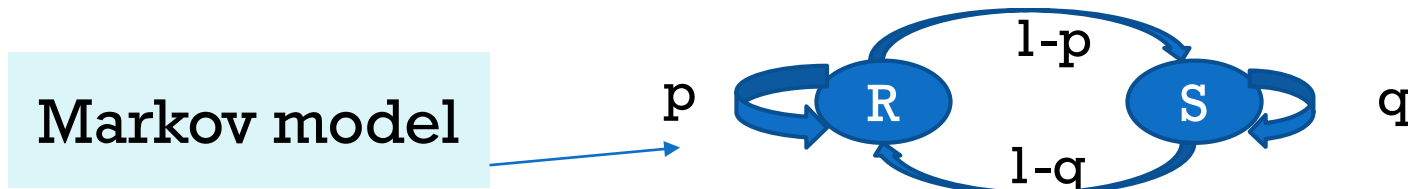
$$\Pr[A_1 \cap A_2 \cap \dots \cap A_n] = \Pr[A_1] \times \Pr[A_2/A_1] \times \Pr[A_3/A_1 A_2] \times \dots \times \Pr[A_n/A_1 A_2 \dots A_{n-1}]$$

CONDITIONAL PROBABILITIES (4/4)

- Going back to binary streams, we note that a bit can depend on several previous bits, quantified as $\Pr[x_i/x_{i-1}x_{i-2} \dots]$
- For example, if each bit depends on the previous two bits, then we need to know $\Pr[x_i=0/x_{i-2}x_{i-1} = 00]$, $\Pr[x_i=0/x_{i-2}x_{i-1} = 01]$, $\Pr[x_i=0/x_{i-2}x_{i-1} = 10]$, and $\Pr[x_i=0/x_{i-2}x_{i-1} = 11]$
- Note that if we know $\Pr[x_i=0/x_{i-2}x_{i-1}]$, then we can compute $\Pr[x_i=1/x_{i-2}x_{i-1}] = 1 - \Pr[x_i=0/x_{i-2}x_{i-1}]$
- If the data correlation is such that **every x_i depends on just the previous bit**, that is, it does not depend on the bits before the previous bit, we express those facts as: $\Pr[x_i/x_{i-1}x_{i-2} \dots] = \Pr[x_i/x_{i-1}]$. In such case, we use line 6 of last slide:
 - **$\Pr[x_1x_2 \dots x_n] = \Pr[x_1] \Pr[x_2/x_1] \Pr[x_3/x_2] \dots \Pr[x_n/x_{n-1}]$**

MARKOV MODELS (OF ORDER 1)

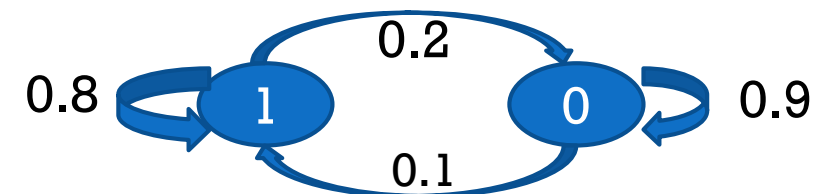
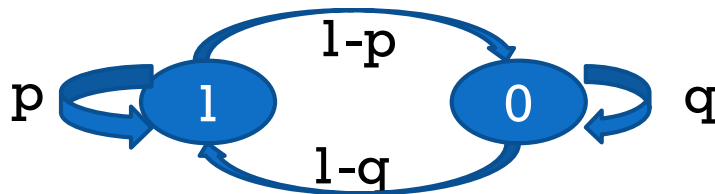
- Many probabilistic phenomena can be modeled by Markov models
- As an example, consider the weather and, keeping things simple, assume that the weather can be in one of just two states: sunny (S) or rainy (R)
- As we all know, if it is raining now, there is a high chance it will be raining next minute, and if it is sunny now, there is a high chance it is sunny next minute
- We can *model* that situation, i.e., represent and quantify it, with a state diagram:



- Which means the system (weather) moves from state R to state S with probability $1-p$, from state R to itself (i.e., remain in state R) with probability p , from state S to state R with probability $1-q$, and from S to itself with prob. q

MARKOV MODEL APPLIED TO BINARY STREAMS

- In binary streams where each bit depends on just the previous bit, we can model the situation with the same model (Markov model or order 1), where “sunny” is “0” and rainy is “1”:



- Take a binary source where $p=0.8$, $q=0.9$, which means that $\Pr[1/1]=0.8$, $\Pr[0/1]=0.2$, $\Pr[0/0]=0.9$, $\Pr[1/0]=0.1$
- Regardless of the values of p and q , the non-conditional probabilities of 0 and 1 will be assumed $\frac{1}{2}$, i.e., $\Pr[0]=\Pr[1]=1/2$
 - Thus, $\Pr[000]=\Pr[0]\Pr[0/0]\Pr[0/0]=\frac{1}{2}q \cdot q = \frac{1}{2}q^2$, $\Pr[110]=\Pr[1]\Pr[1/1]\Pr[0/1]=\frac{1}{2}p(1-p)$,
 - $\Pr[010]=\Pr[0]\Pr[1/0]\Pr[0/1]=\frac{1}{2}(1-q)(1-p)$, etc.

APPLICATION OF MARKOV MODELS TO BLOCK CODING OF BINARY STREAMS

- Take the same binary source of the previous slide $p=0.8$, $q=0.9$
- How can we use the Markov model to develop Block Huffman coder (for blocks of size 3 say)?
 - Take all 3-bit blocks (000, 001, 010, 011, 100, 101, 110, 111)
 - Compute the probability of each block as we saw in the previous slide: $\Pr[000] = \frac{1}{2}q^2 = \frac{.81}{2} = 0.405$, $\Pr[110] = \frac{1}{2}p(1 - p) = 0.08$, etc.
 - Build a Huffman tree for the 8 3-bit blocks
 - Use that tree for coding any binary string: divide the string into consecutive 3-bit blocks, and replace each block by its Huffman codeword
- Note: a typical stream generated by that source: 000000001111100000001111...
- So, for such streams, not only RLE, Golomb, and diff. Golomb, but also Block Huffman can apply. Which is best?

ARITHMETIC CODING

-- PRELIMINARIES --

- Arithmetic Coding achieves a bitrate equal to the entropy of the source with memory
- It codes the whole input sequence at once, rather than individual symbols
- It is used mostly on binary streams, although it can be generalized to non-binary data
- In this course, we'll develop it only for binary streams
- Consider binary streams of n bits (for a fixed n) generated by a probabilistic source
- There are 2^n binary streams of n bits, each with its own probability of occurrence
 - Example: the previous Markov source where $p=0.8$ and $q=0.9$, and take $n=4$
 - There are $2^4 = 16$ binary streams (of 4 bits), each occurring with a different probability
 - $\Pr[0000] = \frac{1}{2} q^3 = 0.3645$, $\Pr[1111] = \frac{1}{2} p^3 = 0.2560$, $\Pr[1010] = (1/2)(0.2)(0.1)(0.2) = 0.002$, etc.

ARITHMETIC CODING (AC)

-- MAIN CONCEPTUAL IDEA--

- The Conceptual Main Idea
 - We will subdivide the unit interval $[0, 1]$ into 2^n subintervals
 - Each n -bit binary stream will correspond to one unique subinterval $[L, R)$ in a one-to-one fashion, where
 - $\Pr[\text{the stream}] = \text{the length of the subinterval} = R - L$
 - When an input stream x is to be coded, AC computes the corresponding interval $[L, R)$, and then codes it as $r_1 r_2 \dots r_t$
 - where $0.r_1 r_2 \dots r_t \dots$ is the binary representation of $(L+R)/2$
 - and $t = \lceil -\log \Pr[x] \rceil$
- Note that by Shannon's information theory, the amount of information in stream x is $-\log \Pr[x]$, which is a lower bound on coding x , and so AC achieves that lower bound

So, in essence, what AC does is compute the interval $[L, R)$ for a given input stream x

ARITHMETIC CODING (AC)

-- METHOD --

Input: a binary stream $x = x_1x_2 \dots x_n$, and a probabilistic model of x

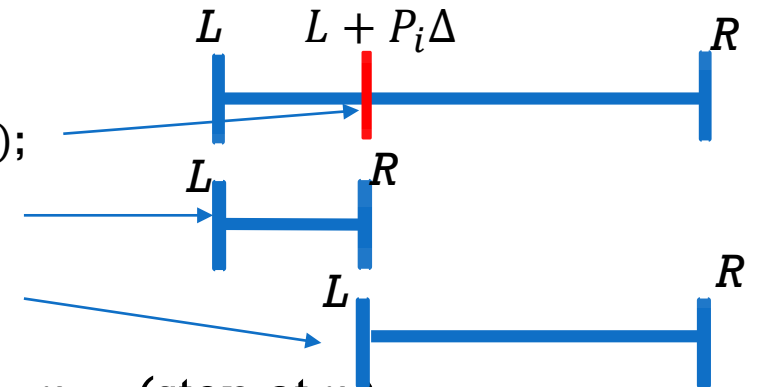
Output: a coded bitstream

Method:

There is a patent on AC by IBM called the Q-Coder

1. Let $I = [L, R)$ where initially $L = 0, R = 1$;
2. For $i=1$ to n do
 - a. Let $P_i = \Pr[0/x_1x_2 \dots x_{(i-1)}]$; // regardless of what x_i is
 - b. Let $\Delta = R - L$; // length of the interval now
 - c. **Split** interval I into 2 subintervals: $[L, L + P_i\Delta)$ and $[L + P_i\Delta, R)$;
 - d. **Choose**: If $(x_i==0)$, reduce I to $[L, L + P_i\Delta)$, i.e., $R := L + P_i\Delta$;
If $(x_i==1)$, reduce I to $[L + P_i\Delta, R)$, i.e., $L := L + P_i\Delta$;
3. Let $t = \lceil -\log \Delta \rceil$, and $r = \frac{L+R}{2}$ expressed in binary as $0.r_1r_2 \dots r_t \dots$ (stop at r_t)
4. Output:= $r_1r_2 \dots r_t$, i.e., code the input $x_1x_2 \dots x_n$ as $r_1r_2 \dots r_t$

Observation 1: Every subinterval is nested inside the earlier intervals



Observation 2: r is in every subinterval that was chosen. Why?

$t = \lceil -\log(\text{length of last subinterval}) \rceil$

ARITHMETIC CODING (AC)

-- EXAMPLE (1/2) --

- Binary Markov Source $\Pr[0/0] = \Pr[1/1] = \frac{3}{4}$, $\Pr[0/1] = \Pr[1/0] = \frac{1}{4}$, and $\Pr[0] = \Pr[1] = \frac{1}{2}$

- Code input $x = 110$

- $i = 1, x_1 = 1, \Delta = 1, P_1 = \Pr[0] = \frac{1}{2}$:

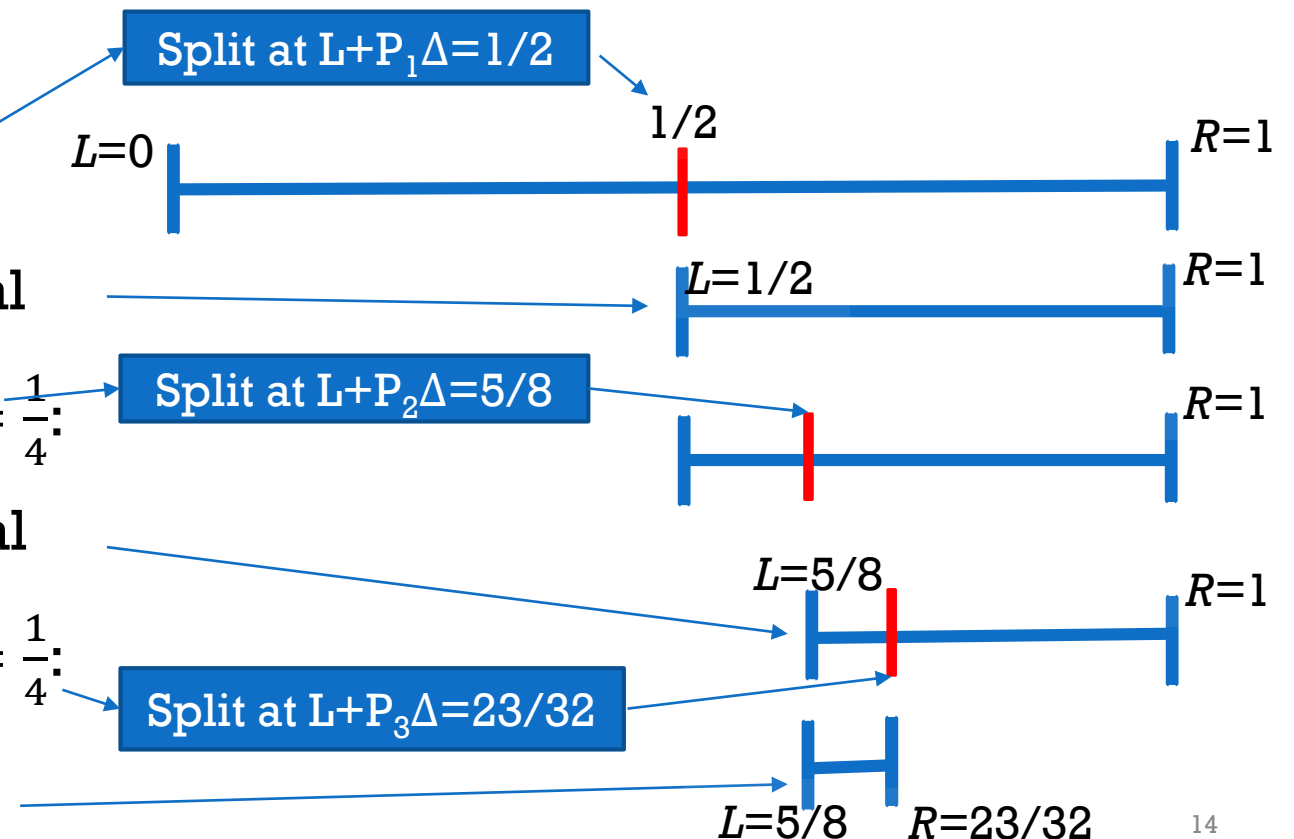
- Since $x_1 = 1$, choose right interval

- $i = 2, x_2 = 1, \Delta = \frac{1}{2}, P_2 = \Pr[0/1] = \frac{1}{4}$:

- Since $x_2 = 1$, choose right interval

- $i = 3, x_3 = 0, \Delta = \frac{3}{8}, P_3 = \Pr[0/1] = \frac{1}{4}$:

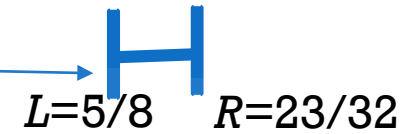
- Since $x_3 = 0$, choose left interval



ARITHMETIC CODING (AC)

-- EXAMPLE (2/2) --

- The final interval was:



- Therefore, $L = \frac{5}{8}, R = \frac{23}{32}, \Delta = R - L = \frac{3}{32}$
- $t = \lceil -\log \Delta \rceil = \lceil -\log \frac{3}{32} \rceil = \lceil 3.4150 \rceil = 4$
- So the coded bitstream will have 4 bits
- $\frac{L+R}{2} = \frac{43}{64}$, convert to binary, we get $\frac{L+R}{2} = 0.101011$
- Take the first 4 bits after the point: 1010
- Therefore, the coded bitstream r is: **1010**
- Done with the example

- This example led to expansion, not compression.
- Not surprising because the input is small
- For realistic input, there will be compression, usually $CR \approx 2$

ARITHMETIC CODING (AC)

-- EXERCISES --

- **Exercise:** We saw in the AC coder algorithm that, for the final values of R, L , we have $t = \lceil -\log(R - L) \rceil$ and $\frac{L+R}{2} = 0.r_1r_2 \dots r_tr_{t+1} \dots$.

Let $r = dec(0.r_1r_2 \dots r_t) = \frac{r_1}{2} + \frac{r_2}{2^2} + \dots + \frac{r_t}{2^t}$. Prove that $L \leq r < R$.

- **Exercise:** Again, taking the final values of R, L in the AC coder where the input is $x = x_1x_2 \dots x_n$. Prove that $\Pr[x] = R - L$.

ARITHMETIC CODING (AC)

-- IMPLEMENTATION ISSUES--

- This algorithm quickly runs into precision (underflow) problems
 - For long input streams, the probabilities P_i could get too small to represent accurately on most computers, i.e., the computer will make them 0
 - Also, R and L become so close to each other that the computer makes them equal
 - That will break the algorithm
- Different implementation methods for dealing with this precision problem have been devised
- Also, interesting ways for deriving the probabilities have been developed but will not be presented here
- Potential term project: Addressing the implementation issues of AC

AC Implementation methods are not covered in the course

ARITHMETIC DECODING

-- METHOD --

Input: coded bitstream $r_1 r_2 \dots r_t$, and the same probabilistic model used by the AC coder

Output: the original data $x = x_1 x_2 \dots x_n$

Method:

1. Set $r = \text{dec}(0.r_1 r_2 \dots r_t) = \frac{r_1}{2} + \frac{r_2}{2^2} + \dots + \frac{r_t}{2^t}$;

2. Set $L = 0, R = 1, \Delta = R - L, I = [L, R), i = 1, T = 0, P_i = \Pr[1^{st} \text{ bit} = 0]; // T = \lceil -\log \Delta \rceil$

3. While($T < t$) do

a. **Split** interval I into 2 subintervals: $[L, L + P_i \Delta)$ and $[L + P_i \Delta, R)$

b. **Choose:** If(r in left subinterval, i.e., $L \leq r < L + P_i \Delta$) // it means the AC coder must have found that $x_i = 0$

$x_i = 0$; reduce I to $[L, L + P_i \Delta)$, i.e., $R := L + P_i \Delta; \Delta = R - L$;

Else // r is the right subinterval, which means that the AC coder must have found that $x_i = 1$

$x_i = 1$; reduce I to $[L + P_i \Delta, R)$, i.e., $L := L + P_i \Delta; \Delta = R - L$;

c. Set $T = \lceil -\log \Delta \rceil; i++$;

4. Output:= $x_1 x_2 \dots x_n$

- **Interval-Splitting:** The AC decoder imitates the coder in the splitting of the unit interval, using $P_i: [L, L + P_i \Delta)$ and $[L + P_i \Delta, R)$;
- **Subinterval choosing:** After each split, the decoder chooses the subinterval that contains r , and updates L, R and Δ accordingly
- **Decoding:** If it chooses the left subinterval, it means x_i must have been 0, otherwise it must have been 1



ARITHMETIC DECODING

-- EXAMPLE (1/2) --

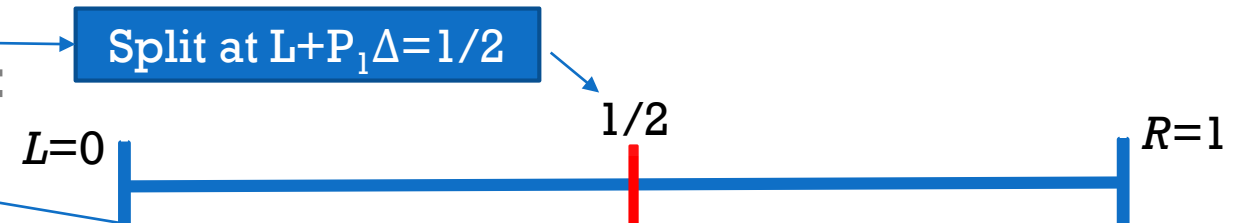
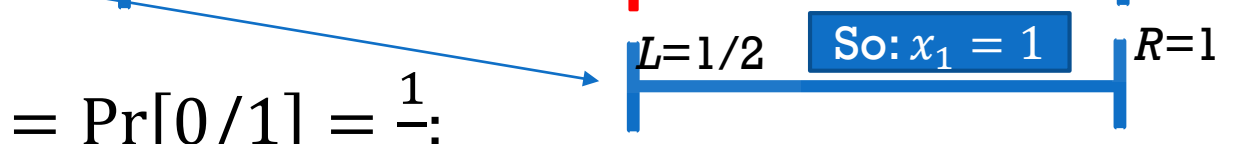

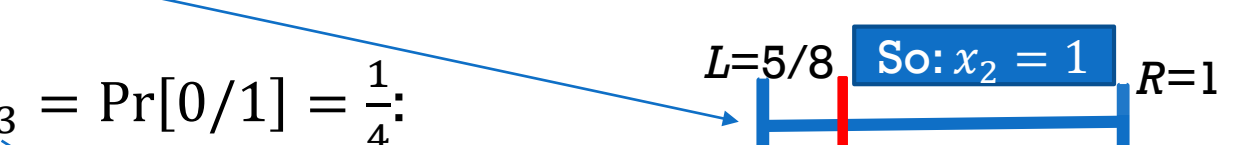
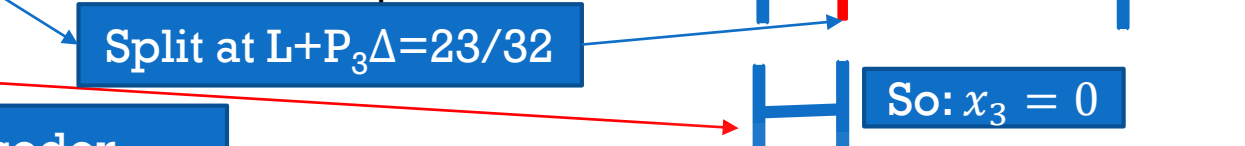

- Take the coding example earlier, where $\Pr[0/0] = \Pr[1/1] = \frac{3}{4}$,
 $\Pr[0/1] = \Pr[1/0] = \frac{1}{4}$, and $\Pr[0] = \Pr[1] = \frac{1}{2}$;
- And take the coded bitstream 1010.
- We will decode it next (and see if we get back 110)
- $r = \text{dec}(0.r_1r_2 \dots r_t) = \text{dec}(0.1010) = \frac{1}{2} + \frac{1}{8} = \frac{5}{8}$;
- $t = \text{length}(r) = 4$.
- $L = 0, R = 1, \Delta = R - L, I = [L, R), i = 1, T = 0, P_i = \Pr[1^{\text{st}} \text{ bit} = 0] = \frac{1}{2}$

ARITHMETIC DECODING

-- EXAMPLE (2/2) --

- Markov Source $\Pr[0/0] = \Pr[1/1] = \frac{3}{4}$, $\Pr[0/1] = \Pr[1/0] = \frac{1}{4}$, and $\Pr[0] = \Pr[1] = \frac{1}{2}$

- Coded bitstream 1010, $r=5/8$

- $i = 1, L = 0, R = 1, \Delta = 1, P_1 = \Pr[0] = \frac{1}{2}$:
 Split at $L + P_1\Delta = 1/2$

- Choose right interval b/c r is in it:
 $L = 1/2$ So: $x_1 = 1$ $R = 1$

- $i = 2, \Delta = \frac{1}{2}, T = \lceil -\log \Delta \rceil = 1, P_2 = \Pr[0/1] = \frac{1}{4}$:
 Split at $L + P_2\Delta = 5/8$

- Choose right interval b/c r is in it:
 $L = 5/8$ So: $x_2 = 1$ $R = 1$

- $i = 3, \Delta = \frac{3}{8}, T = \lceil -\log \Delta \rceil = 2 > t, P_3 = \Pr[0/1] = \frac{1}{4}$:
 Split at $L + P_3\Delta = 23/32$

- Choose left interval b/c r is in it:
 $L = 5/8$ So: $x_3 = 0$ $R = 23/32$


Now, $\Delta = \frac{3}{32}, T = \lceil -\log \Delta \rceil = 4 = t$, so the decoder stops. Decoded data: $x_1x_2x_3 = 110$. Correct.

ARITHMETIC CODING

-- EXERCISES --

- **Exercise:** The AC presented is focused on binary input streams. Generalize the algorithm to non-binary alphabets
- **Exercise** (potential term project): Review, evaluate and compare some of the different ways that addressed the implementation issues of Arithmetic Coding in the literature. Can you think of other ways? If so, what, and how well do they work?

LEMPERL-ZIV (LZ) COMPRESSION

--PRELIMINARIES --

- Unlike Huffman and AC, LZ does not need a probabilistic model about the data
- Yet, it achieves the entropy in the limit (i.e., for large input data)!
- It works by
 - Scanning the input stream left to right
 - Keeping a dictionary DICT of substrings seen previously in the input
 - Coding each later substring (in the input) by recording the location where that substring occurred earlier
 - The dictionary grows by adding to it newly encountered input-substrings of the form Wa , where W is already in the dictionary, a is a symbol, but Wa is not (yet) in DICT
 - In other terms, LZ divides the input stream into a sequence of substrings of the form Wa , and records for each Wa the pair (J, a) where J is the location of W in DICT
 - The coded bitstream will consist of the list of those (J, a) pairs

LEMPERL-ZIV (LZ) COMPRESSION

--CODER METHOD--

Input: a binary stream $x = x_1x_2 \dots x_n$ // no need for a probabilistic model

Output: a coded bitstream

Method:

- Assume $DICT[0] = \text{empty} = - = \epsilon$
- So, if $W_i = \text{empty}$, then $j=0$

1. Maintain a dictionary (DICT) of patterns seen so far
2. Set $i=1$, $J_i = \text{empty}$, $W = \text{empty}$, $a = x_1$, $DICT[1] = Wa$ (i.e., x_1), output = x_1 ;
3. Set $i=2$;
4. While (there are still input symbols) do
 - a. Read from the remaining input until the substring scanned (call it S_i) is no longer in DICT
 - b. Write S_i as $S_i = W_i a_i$, where W_i is in DICT and a_i is the last symbol in S_i (the symbol after W_i)
 - c. Let j be the index where $W_i = DICT[j]$; // $j < i$
 - d. Let $J_i = d2b(j)$ using $\lceil \log i \rceil$ bits // d2b is decimal to binary converter
 - e. Code $W_i a_i$ as (J_i, a_i) and append that code to the output
 - f. Store $W_i a_i$ in $DICT[i]$, i.e., $DICT[i] = W_i a_i$;
 - g. $i++$;

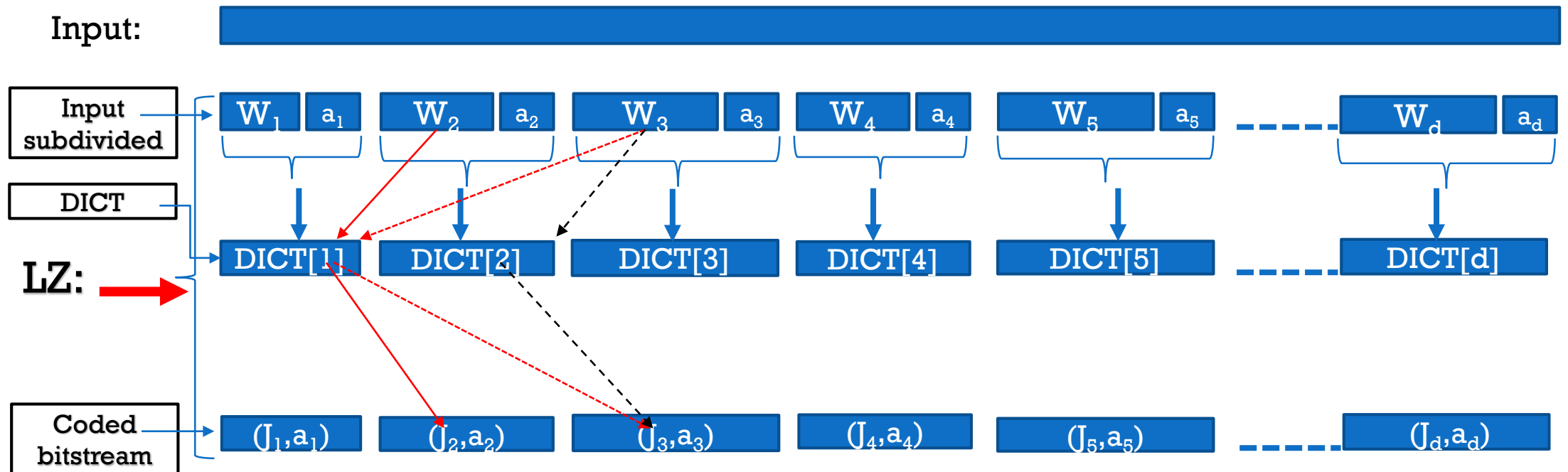
Observe that:

- $x = DICT[1] DICT[2] DICT[3] \dots$
- Output = sequence of (J_i, a_i) 's generated

a_i could be non-existent in the very last
 $W_i a_i$: Code $W_i a_i$ as $(J_i, -)$, i.e., as J_i alone

LEMPLEL-ZIV (LZ) COMPRESSION

--OVERVIEW--



Notes:

- Every W_i is in $DICT[1 : (i-1)]$
- J_i is the index where W_i is stored in $DICT[1 : (i-1)]$, i.e., $W_i = DICT[J_i]$
- $W_i a_i$ is stored in location i of DICT, i.e., $DICT[i] := W_i a_i$
- W_1 is non-existent, i.e., empty (we denote it as $-$ or ϵ)
- So J_1 is non-existent, i.e., empty (we denote it as $-$)

LEMPERL-ZIV (LZ) COMPRESSION

-- REMARKS --

- The dictionary is not stored/transmitted in the coded bitstream
- Rather, it is used by the coder to carry out the coding, and then discarded
- As will be seen, the decoder will be able to reconstruct the dictionary from the coded bitstream alone
- No probabilities were needed or used
- The LZ bitrate is asymptotically optimal (i.e., approaches the entropy of the source with memory) **without** the need to know or compute the underlying probability model of the input data.
- The proof will not be provided. See [paper](#) if interested

LEMPERL-ZIV (LZ) COMPRESSION

-- EXAMPLE --

- Input: $x=0010100100010010100110101$

Notation: When we find $S_i = W_i a_i$ in x , we color W in green and a in red.
 $DICT[i] = W_i a_i$

i	$\lceil \log i \rceil$	$J_i = (j)_2$	W_i	a_i	DICT[i]
1	0	empty	empty	0	0
2	1	$(1)_2 = 1$	0	1	01
3	2	$(10)_2 = 2$	01	0	010
4	2	$(11)_2 = 3$	010	0	0100
5	3	$(100)_2 = 4$	0100	1	01001
6	3	$(101)_2 = 5$	01001	1	010011
7	3	$(011)_2 = 3$	010	1	0101

0010100100010010100110101
 0010100100010010100110101
 0010100100010010100110101
 0010100100010010100110101
 0010100100010010100110101
 0010100100010010100110101
 0010100100010010100110101

Output: Sequence of (J_i, a_i) 's for $i=1,2,\dots$. Therefore:
 Output: $(-,0) (1,1) (10,0) (11,0) (100,1) (101,1) (011,1)$. Finally:
 Output: 011100110100110110111

Recall:
 $DICT[0] = \text{empty} = - = \epsilon$

LZ DECODER

-- METHOD --

Input: a coded bitstream $y = y_1 y_2 \dots$; **Output:** the original data x

Method:

1. $i=1$, $W_1=\text{empty}$, $a_1 = y_1$, $\text{DICT}[1]=W_1 a_1$ (i.e., y_1), $x = W_1 a_1$ (i.e., y_1);
 2. $i=2$;
 3. While (the coded bitstream is not fully scanned) do
 - a. $j=\text{decimal}(\text{the next } \lceil \log i \rceil \text{ bits from } y)$;
 - b. $W_i=\text{DICT}[j]$;
 - c. $a_i=\text{the next symbol from } y$;
 - d. append $W_i a_i$ to the right of x ;
 - e. $\text{DICT}[i]=W_i a_i$
 - f. $i++$;
- // if $a_i=\text{empty}$, at end of decoding, simply append W_i to the right of x

Observe that:

- $x=\text{DICT}[1] \text{ DICT}[2] \text{ DICT}[3] \dots$
- Which is what we want based on what we observed in the LZ coder

LZ DECODER

-- EXAMPLE --

Decoding $y=01110\ 0110100110110111$ from the previous example

i	$\lceil \log i \rceil$	$j = \text{next } \lceil \log i \rceil$ bits of y	$W_i = \text{DICT}[j]$	a_i	$\text{DICT}[i] = W_i a_i$	$x = (\text{previous}(x))(W_i a_i)$
1	0	empty	empty	0	0	0
2	1	$(1)_2 = 1$	0	1	01	001
3	2	$(10)_2 = 2$	01	0	010	001010
4	2	$(11)_2 = 3$	010	0	0100	0010100100
5	3	$(100)_2 = 4$	0100	1	01001	001010010001001
6	3	$(101)_2 = 5$	01001	1	010011	001010010001001010011
7	3	$(011)_2 = 3$	010	1	0101	0010100100010010100110101

Decoded data $x = 0010100100010010100110101$

NEXT LECTURE

- Differential pulse-code modulation (dpcm)
- Bitplane coding
 - Graycodes
- Limitations of lossless compression
- Lossless compression in standards, graphic file formats, and utilities
- That will be the end of lossless compression; afterwards, we turn our attention to lossy compression