Arithmetic Coding

CS6025 Data Encoding

Yizong Cheng

2-3-15

A Problem with Huffman Coding

- The Huffman method is simple, efficient, and produces the best codes for the individual data symbols.
- The Huffman method assigns a code with an integral number of bits to each symbol in the alphabet.
- Information theory shows that a symbol with probability 0.4 should ideally be assigned a 1.32-bit code, since $-\log 2\ 0.4 \approx 1.32$.
- The Huffman method, however, normally assigns such a symbol a code of 1 or 2 bits.

Interval Narrowing

- Arithmetic coding overcomes the problem of assigning integer codes to the individual symbols by assigning one (normally long) code to the entire input file.
- The method starts with a certain interval, it reads the input file symbol by symbol, and it uses the probability of each symbol to narrow the interval.
- The output of arithmetic coding is interpreted as a number in the range [0, 1).

Dividing an Interval for Symbols

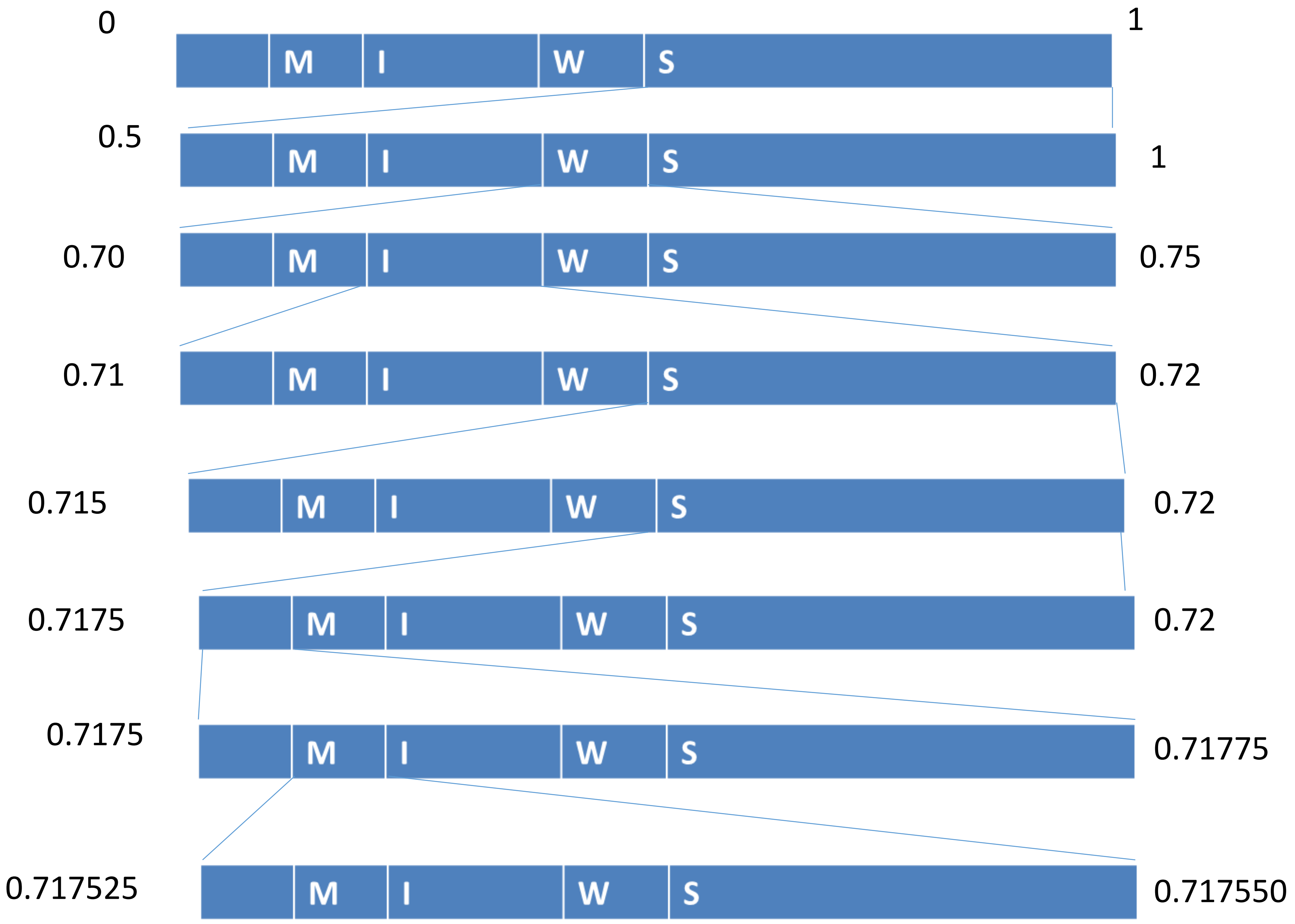
Char	Freq	Prob.	Range	CumFreq
		Total CumFreq=		10
S	5	$5/10 = 0.5$	$[0.5, 1.0)$	5
W	1	$1/10 = 0.1$	$[0.4, 0.5)$	4
I	2	$2/10 = 0.2$	$[0.2, 0.4)$	2
M	1	$1/10 = 0.1$	$[0.1, 0.2)$	1
␣	1	$1/10 = 0.1$	$[0.0, 0.1)$	0

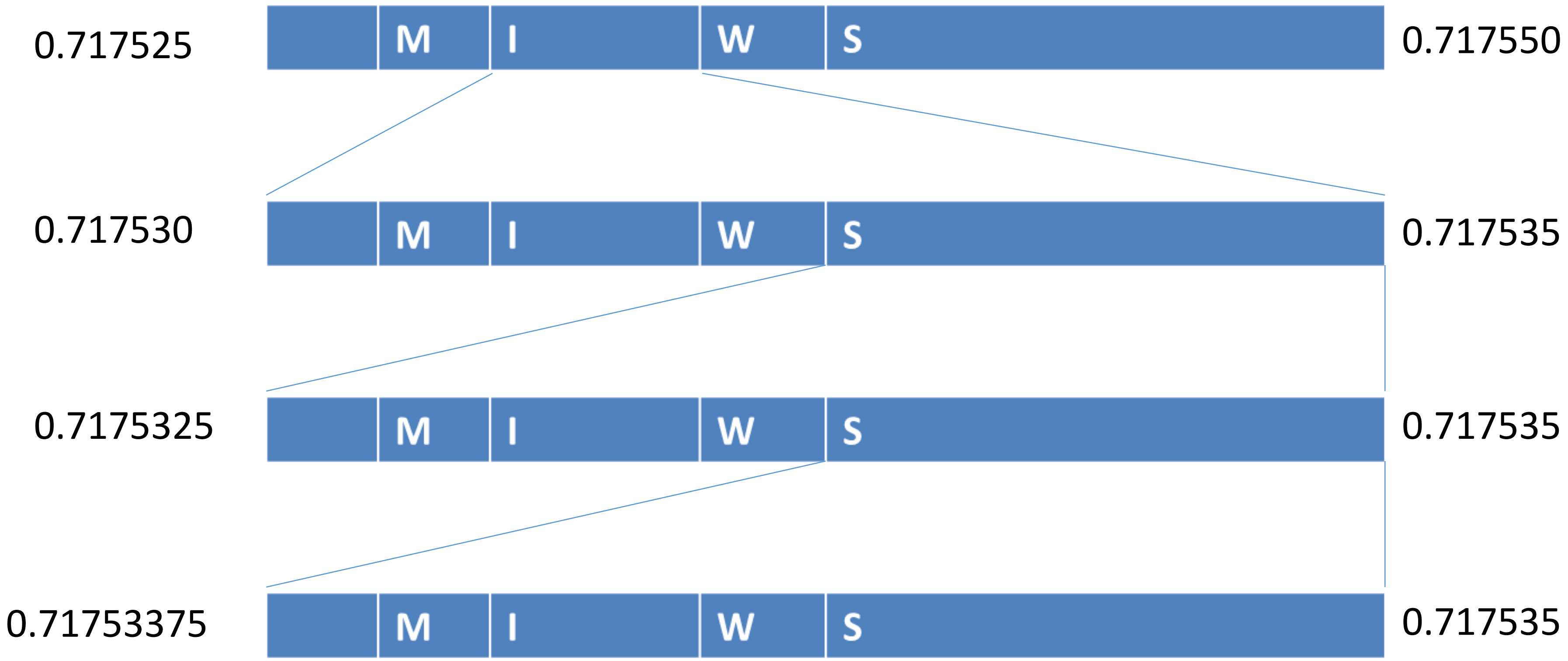
Table 5.42: Frequencies and Probabilities of Five Symbols.

```
NewHigh:=OldLow+Range*HighRange(X);
NewLow:=OldLow+Range*LowRange(X);
```

```
Code:=(Code-LowRange(X))/Range;
```

Char.		The calculation of low and high
S	L	$0.0 + (1.0 - 0.0) \times 0.5 = 0.5$
	H	$0.0 + (1.0 - 0.0) \times 1.0 = 1.0$
W	L	$0.5 + (1.0 - 0.5) \times 0.4 = 0.70$
	H	$0.5 + (1.0 - 0.5) \times 0.5 = 0.75$
I	L	$0.7 + (0.75 - 0.70) \times 0.2 = 0.71$
	H	$0.7 + (0.75 - 0.70) \times 0.4 = 0.72$
S	L	$0.71 + (0.72 - 0.71) \times 0.5 = 0.715$
	H	$0.71 + (0.72 - 0.71) \times 1.0 = 0.72$
S	L	$0.715 + (0.72 - 0.715) \times 0.5 = 0.7175$
	H	$0.715 + (0.72 - 0.715) \times 1.0 = 0.72$
␣	L	$0.7175 + (0.72 - 0.7175) \times 0.0 = 0.7175$
	H	$0.7175 + (0.72 - 0.7175) \times 0.1 = 0.71775$
M	L	$0.7175 + (0.71775 - 0.7175) \times 0.1 = 0.717525$
	H	$0.7175 + (0.71775 - 0.7175) \times 0.2 = 0.717550$
I	L	$0.717525 + (0.71755 - 0.717525) \times 0.2 = 0.717530$
	H	$0.717525 + (0.71755 - 0.717525) \times 0.4 = 0.717535$
S	L	$0.717530 + (0.717535 - 0.717530) \times 0.5 = 0.7175325$
	H	$0.717530 + (0.717535 - 0.717530) \times 1.0 = 0.717535$
S	L	$0.7175325 + (0.717535 - 0.7175325) \times 0.5 = 0.71753375$
	H	$0.7175325 + (0.717535 - 0.7175325) \times 1.0 = 0.717535$


0
1
M
I
W
S
0.5
1
0.70
0.75
0.71
0.72
0.715
0.72
0.7175
0.72
0.7175
0.71775
0.717525
0.717550

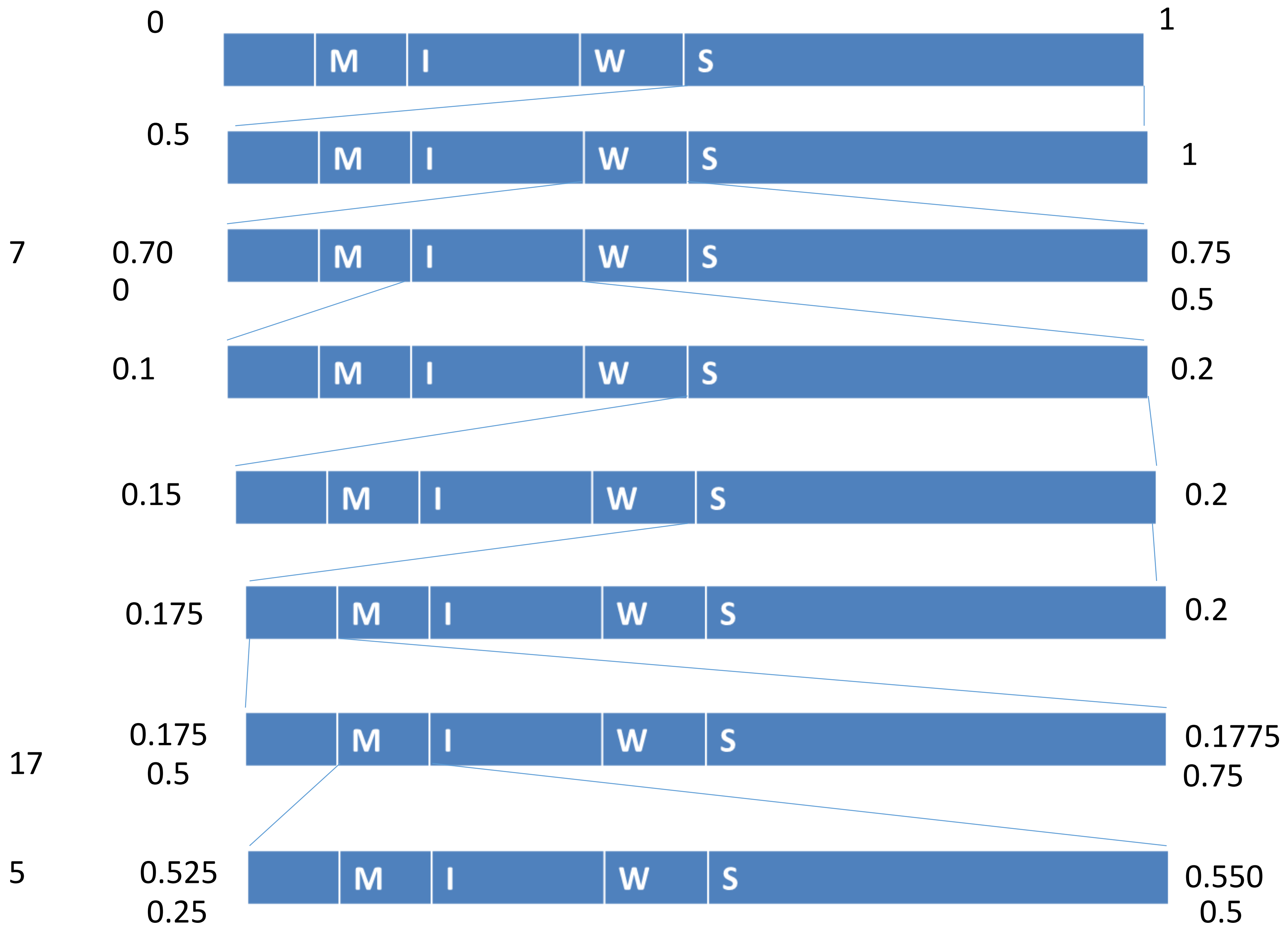

0.717525
M
I
W
S
0.717550
0.717530
M
I
W
S
0.717535
0.7175325
M
I
W
S
0.717535
0.71753375
M
I
W
S
0.717535

Char.	Code−low		Range
S	$0.71753375 - 0.5$	$= 0.21753375$	$/0.5 = 0.4350675$
W	$0.4350675 - 0.4$	$= 0.0350675$	$/0.1 = 0.350675$
I	$0.350675 - 0.2$	$= 0.150675$	$/0.2 = 0.753375$
S	$0.753375 - 0.5$	$= 0.253375$	$/0.5 = 0.50675$
S	$0.50675 - 0.5$	$= 0.00675$	$/0.5 = 0.0135$
␣	$0.0135 - 0$	$= 0.0135$	$/0.1 = 0.135$
M	$0.135 - 0.1$	$= 0.035$	$/0.1 = 0.35$
I	$0.35 - 0.2$	$= 0.15$	$/0.2 = 0.75$
S	$0.75 - 0.5$	$= 0.25$	$/0.5 = 0.5$
S	$0.5 - 0.5$	$= 0$	$/0.5 = 0$

Table 5.45: The Process of Arithmetic Decoding.

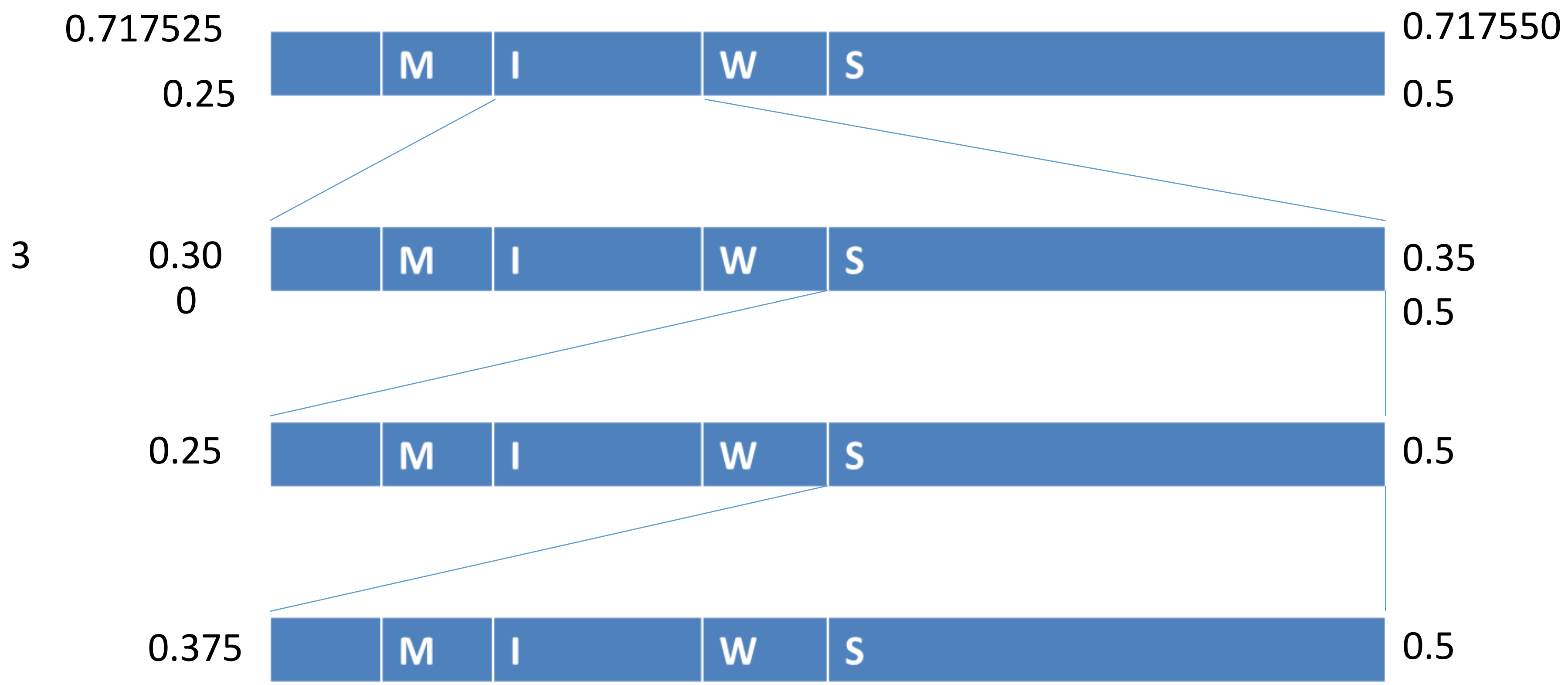
1	2	3	4	5
S	$L = 0+(1 - 0)\times 0.5 = 0.5$	5000		5000
	$H = 0+(1 - 0)\times 1.0 = 1.0$	9999		9999
W	$L = 0.5+(1 - .5)\times 0.4 = 0.7$	7000	7	0000
	$H = 0.5+(1 - .5)\times 0.5 = 0.75$	7499	7	4999
I	$L = 0+(0.5 - 0)\times 0.2 = 0.1$	1000	1	0000
	$H = 0+(0.5 - 0)\times 0.4 = 0.2$	1999	1	9999
S	$L = 0+(1 - 0)\times 0.5 = 0.5$	5000		5000
	$H = 0+(1 - 0)\times 1.0 = 1.0$	9999		9999
S	$L = 0.5+(1 - 0.5)\times 0.5 = 0.75$	7500		7500
	$H = 0.5+(1 - 0.5)\times 1.0 = 1.0$	9999		9999
␣	$L = 0.75+(1 - 0.75)\times 0.0 = 0.75$	7500	7	5000
	$H = 0.75+(1 - 0.75)\times 0.1 = 0.775$	7749	7	7499
M	$L = 0.5+(0.75 - 0.5)\times 0.1 = 0.525$	5250	5	2500
	$H = 0.5+(0.75 - 0.5)\times 0.2 = 0.55$	5499	5	4999
I	$L = 0.25+(0.5 - 0.25)\times 0.2 = 0.3$	3000	3	0000
	$H = 0.25+(0.5 - 0.25)\times 0.4 = 0.35$	3499	3	4999
S	$L = 0+(0.5 - 0)\times 0.5 = .25$	2500		2500
	$H = 0+(0.5 - 0)\times 1.0 = 0.5$	4999		4999
S	$L = 0.25+(0.5 - 0.25)\times 0.5 = 0.375$	3750		3750
	$H = 0.25+(0.5 - 0.25)\times 1.0 = 0.5$	4999		4999

Table 5.51: Encoding SWISS␣MISS by Shifting.



0
1
M I W S
0.5
1
7
0.70
0
0.75
0.5
0.1
0.2
0.15
0.2
0.175
0.2
17
0.175
0.5
0.1775
0.75
5
0.525
0.25
0.550
0.5

7175



0.71753375

0. Initialize `Low`=0000, `High`=9999, and `Code`=7175.

1. `index`= $[(7175-0+1)\times 10-1]/(9999-0+1)=7.1759 \rightarrow 7$. Symbol `S` is selected.
`Low` $=0+(9999-0+1)\times 5/10=5000$. `High` $=0+(9999-0+1)\times 10/10-1=9999$.

2. `index`= $[(7175-5000+1)\times 10-1]/(9999-5000+1)=4.3518 \rightarrow 4$. Symbol `W` is selected.
`Low` $=5000+(9999-5000+1)\times 4/10=7000$. `High` $=5000+(9999-5000+1)\times 5/10-1=7499$.
After the 7 is shifted out, we have `Low`=0000, `High`=4999, and `Code`=1753.

3. `index`= $[(1753-0+1)\times 10-1]/(4999-0+1)=3.5078 \rightarrow 3$. Symbol `I` is selected.
`Low` $=0+(4999-0+1)\times 2/10=1000$. `High` $=0+(4999-0+1)\times 4/10-1=1999$.
After the 1 is shifted out, we have `Low`=0000, `High`=9999, and `Code`=7533.

4. `index`= $[(7533-0+1)\times 10-1]/(9999-0+1)=7.5339 \rightarrow 7$. Symbol `S` is selected.
`Low` $=0+(9999-0+1)\times 5/10=5000$. `High` $=0+(9999-0+1)\times 10/10-1=9999$.

5. $\texttt{index}=[(7533-5000+1)\times 10-1]/(9999-5000+1)=5.0678\rightarrow 5$. Symbol $\texttt{S}$ is selected.
$\texttt{Low}=5000+(9999-5000+1)\times 5/10=7500$. $\texttt{High}=5000+(9999-5000+1)\times 10/10-1=9999$.

6. $\texttt{index}=[(7533-7500+1)\times 10-1]/(9999-7500+1)=0.1356\rightarrow 0$. Symbol ␣ is selected.
$\texttt{Low}=7500+(9999-7500+1)\times 0/10=7500$. $\texttt{High}=7500+(9999-7500+1)\times 1/10-1=7749$.
After the 7 is shifted out, we have $\texttt{Low}$=5000, $\texttt{High}$=7499, and $\texttt{Code}$=5337.

7. $\texttt{index}=[(5337-5000+1)\times 10-1]/(7499-5000+1)=1.3516\rightarrow 1$. Symbol $\texttt{M}$ is selected.
$\texttt{Low}=5000+(7499-5000+1)\times 1/10=5250$. $\texttt{High}=5000+(7499-5000+1)\times 2/10-1=5499$.
After the 5 is shifted out we have $\texttt{Low}$=2500, $\texttt{High}$=4999, and $\texttt{Code}$=3375.

8. `index`$= [(3375 - 2500 + 1) \times 10 - 1]/(4999 - 2500 + 1) = 3.5036 \rightarrow 3$. Symbol `I` is selected.
`Low` $= 2500+(4999-2500+1)\times 2/10 = 3000$. `High` $= 2500+(4999-2500+1)\times 4/10-1 = 3499$.
After the 3 is shifted out we have `Low`=0000, `High`=4999, and `Code`=3750.

9. `index`$= [(3750 - 0 + 1) \times 10 - 1]/(4999 - 0 + 1) = 7.5018 \rightarrow 7$. Symbol `S` is selected.
`Low` $= 0 + (4999 - 0 + 1) \times 5/10 = 2500$. `High` $= 0 + (4999 - 0 + 1) \times 10/10 - 1 = 4999$.

10. `index`$= [(3750 - 2500 + 1) \times 10 - 1]/(4999 - 2500 + 1) = 5.0036 \rightarrow 5$. Symbol `S` is selected.
`Low` $= 2500+(4999-2500+1)\times 5/10 = 3750$. `High` $= 2500+(4999-2500+1)\times 10/10-1 = 4999$.

Symbol Probability Estimation Using the Current Document

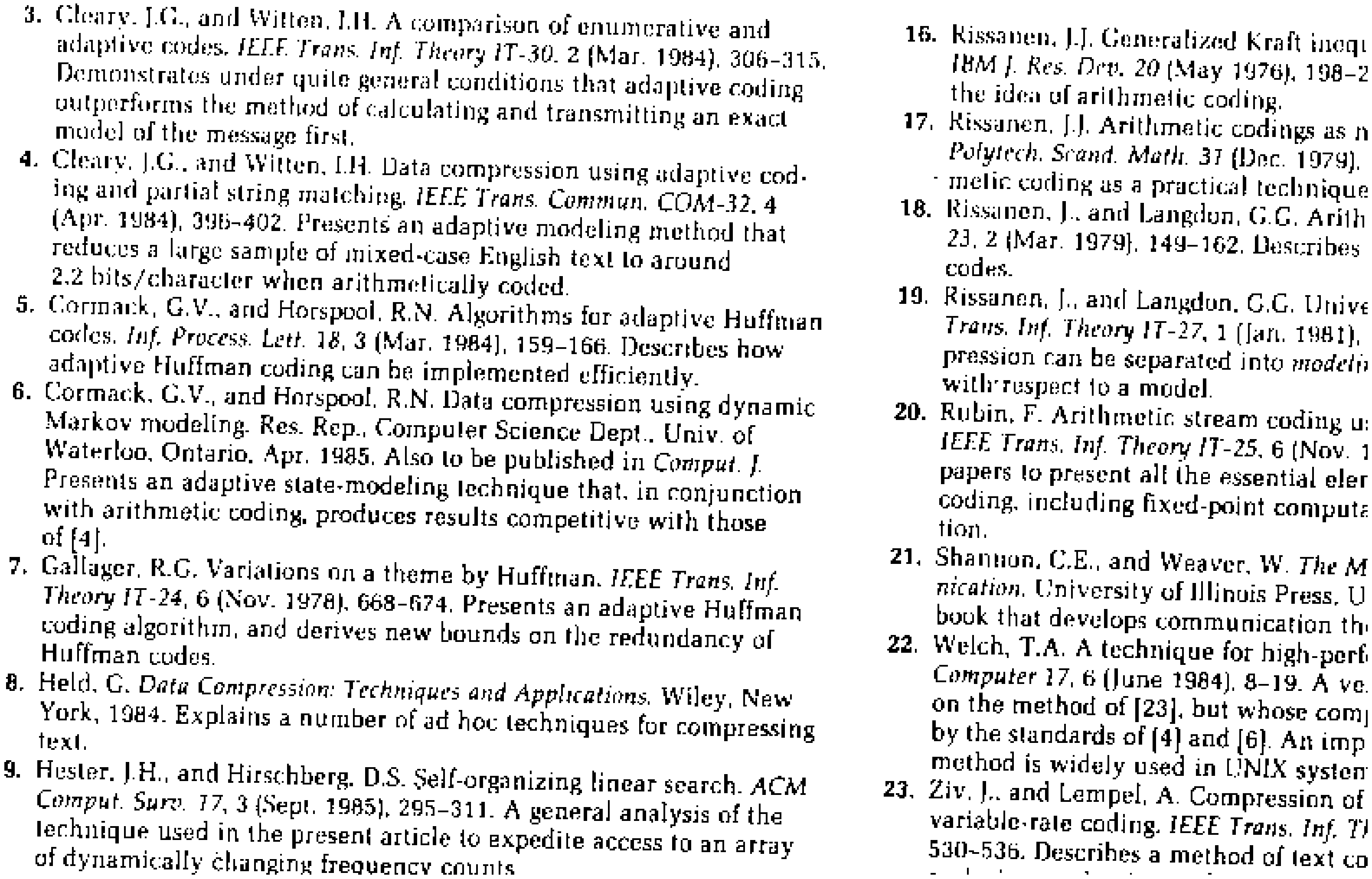
- Arithmetic coding assigns a codeword to each sequence of symbols based on probabilities of symbols.
- Each document may have its own personality.
- Use the processed part of the document to estimate the probabilities of the symbols.
  - Both encoder and decoder can do the same.
  - No probability table needs to be sent.

Symbols in Context

- Part of the processed document may be used as the context to the next symbol.
- Under different context, a symbol may have different probabilities.
- To do adaptive coding, a count for each symbol and each context.
- May further reduce the entropy of data.
- Shorter codewords for documents with distinct personalities or repeated patterns.

Bi-Leval Image

Bi-Level Image for Fax



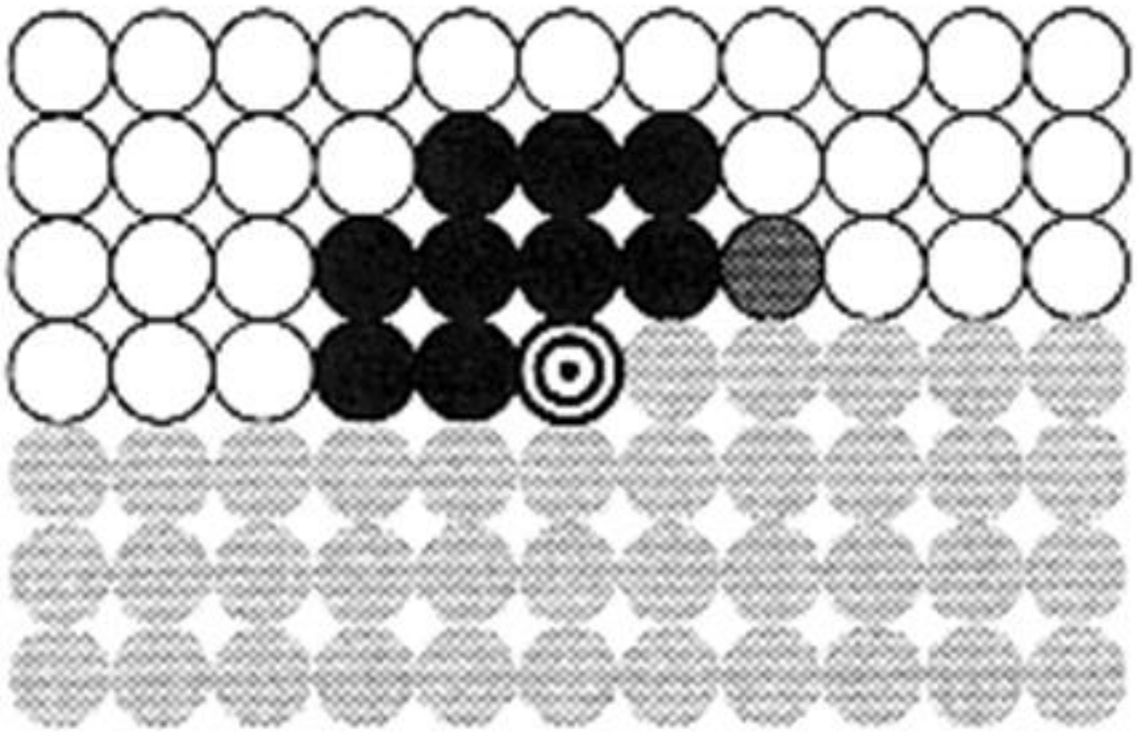
3. Cleary, J.G., and Witten, I.H. A comparison of enumerative and adaptive codes. *IEEE Trans. Inf. Theory IT-30*, 2 (Mar. 1984), 306–315. Demonstrates under quite general conditions that adaptive coding outperforms the method of calculating and transmitting an exact model of the message first.
4. Cleary, J.G., and Witten, I.H. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun. COM-32*, 4 (Apr. 1984), 396–402. Presents an adaptive modeling method that reduces a large sample of mixed-case English text to around 2.2 bits/character when arithmetically coded.
5. Cormack, G.V., and Horspool, R.N. Algorithms for adaptive Huffman codes. *Inf. Process. Lett. 18*, 3 (Mar. 1984), 159–166. Describes how adaptive Huffman coding can be implemented efficiently.
6. Cormack, G.V., and Horspool, R.N. Data compression using dynamic Markov modeling. Res. Rep., Computer Science Dept., Univ. of Waterloo, Ontario, Apr. 1985. Also to be published in *Comput. J.* Presents an adaptive state-modeling technique that, in conjunction with arithmetic coding, produces results competitive with those of [4].
7. Gallager, R.G. Variations on a theme by Huffman. *IEEE Trans. Inf. Theory IT-24*, 6 (Nov. 1978), 668–674. Presents an adaptive Huffman coding algorithm, and derives new bounds on the redundancy of Huffman codes.
8. Held, G. *Data Compression: Techniques and Applications*. Wiley, New York, 1984. Explains a number of ad hoc techniques for compressing text.
9. Hester, J.H., and Hirschberg, D.S. Self-organizing linear search. *ACM Comput. Surv. 17*, 3 (Sept. 1985), 295–311. A general analysis of the technique used in the present article to expedite access to an array of dynamically changing frequency counts.

16. Rissanen, J.J. Generalized Kraft inequ
IBM J. Res. Dev. 20 (May 1976), 198–2
the idea of arithmetic coding.
17. Rissanen, J.J. Arithmetic codings as n
Polytech. Scand. Math. 31 (Dec. 1979).
metic coding as a practical technique
18. Rissanen, J., and Langdon, G.G. Arith
23, 2 (Mar. 1979), 149–162. Describes
codes.
19. Rissanen, J., and Langdon, G.G. Unive
Trans. Inf. Theory IT-27, 1 (Jan. 1981),
pression can be separated into *modeli*
with respect to a model.
20. Rubin, F. Arithmetic stream coding u
IEEE Trans. Inf. Theory IT-25, 6 (Nov. 1
papers to present all the essential ele
coding, including fixed-point computa
tion.
21. Shannon, C.E., and Weaver, W. *The M*
nication. University of Illinois Press, U
book that develops communication th
22. Welch, T.A. A technique for high-perf
Computer 17, 6 (June 1984), 8–19. A ve
on the method of [23], but whose com
by the standards of [4] and [6]. An imp
method is widely used in UNIX system
23. Ziv, J., and Lempel, A. Compression of
variable-rate coding. *IEEE Trans. Inf. T*
530–536. Describes a method of text co

PBM Portable Bitmap Format

- A UNIX format (only deals with P4)
- P4\n   // PBM identifier for binary bi-level
- 1024 896\n // width and height
- height * (width / 8) bytes following
- One bit for each pixel
- Left to right
- Top to bottom
- Viewable with Photoshop or H7C.java

Context in JBIG



Adaptive Arithmetic Coding

```
public class H7A{

   static final int maxRange = 65536;  // 2 ** 16
   static final int half = 32768;
   static final int quarter = 16384;
   static final int threequarters = 49152;
   static final int numberOfContexts = 1024;
   int width = 0, height = 0, bytesPerRow = 0;
     // dimension of the image
   boolean[][] bitmap = null;

   int[][] count = new int[numberOfContexts][2];
   int low = 0; int high = maxRange; int follow = 0;
   int buf = 0; int position = 0;

    public static void main(String[] args){
         H7A h7 = new H7A();
         h7.readPBMHeader();
         h7.compress();
    }
```

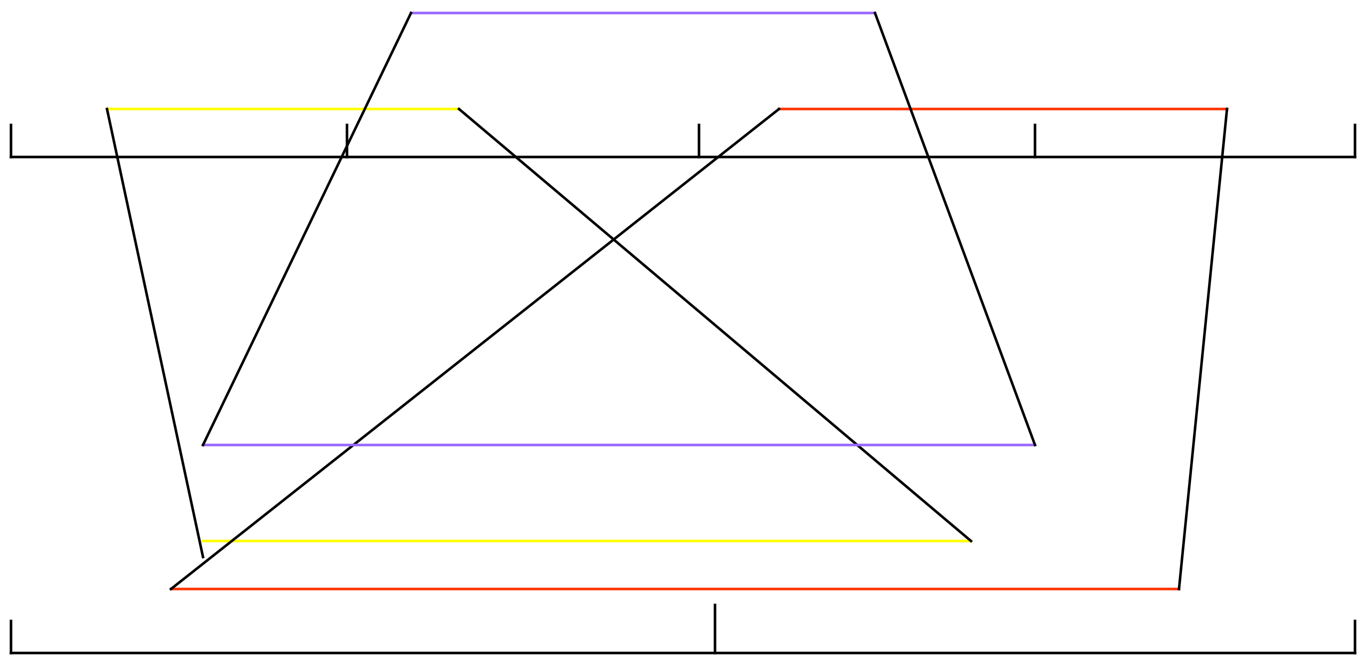
```
void compress(){
   for (int i = 0; i < height; i++){
      int column = 2;
      for (int j = 0; j < bytesPerRow; j++){
         int b = getNextByte();
         for (int test = 0x80; test > 0; test >>= 1){
           int context = getContext(column);
           boolean one = ((test & b) != 0);
           update(context, one);
           incrementCount(context, one);
           bitmap[2][column++] = one;
         }
     }
     for (int j = 2; j < width + 2; j++){
   bitmap[0][j] = bitmap[1][j];  bitmap[1][j] = bitmap[2][j]; }
    }
    if (position > 0){  buf <<= (8 - position);
      System.out.write(buf); }
    System.out.flush();
}
```

getContext

```
int getContext(int column){
  // column >= 2
  int context = 0;
  for (int k = -1; k < 2; k++){
    context <<= 1;
    if (bitmap[0][column + k]) context |= 1;
  }
  for (int k = -2; k < 3; k++){
    context <<= 1;
    if (bitmap[1][column + k]) context |= 1;
  }
  for (int k = -2; k < 0; k++){
    context <<= 1;
    if (bitmap[2][column + k]) context |= 1;
  }
  return context;
}
```

```
void update(int context, boolean one){
   int t = low + count[context][0] * (high - low) /
       (count[context][0] + count[context][1]);
   if (one) low = t; else high = t;
   for (;;){  // double until larger than quarter
      if (high < half){ // most significant bit is 0
        outputBit(0);
        for (int i = 0; i < follow; i++) outputBit(1);
        follow = 0;
        high *= 2;
        low *= 2;
      }else if (low >= half){  // most significant bit is 1
        outputBit(1);
        for (int i = 0; i < follow; i++) outputBit(0);
        follow = 0;
        high = (high * 2) - maxRange;
        low = (low * 2) - maxRange;
      }else if (low > quarter && high <= threequarters){
        follow++; // most significant bit unknown yet
        low = (low * 4 - maxRange) / 2;
        high = (high * 4 - maxRange) / 2;
      }else break;
   }
}
```

Three Cases for Doubling the Current Interval



incrementCount

```
void incrementCount(int context, boolean one){
  int v = one ? 1 : 0;
  if (++count[context][v] >= quarter){  // halve counts
    count[context][0] >>= 1; count[context][1] >>= 1;
    if (count[context][1 - v] == 0) count[context][1 - v] = 1;
  }
}

void outputBit(int bit){
   buf <<= 1;
   if (bit == 1) buf |= 1;
   position++;
   if (position == 8){
     position = 0;
     System.out.write(buf);
     buf = 0;
   }
 }
```

Decoder Program

- The same set of global variables except `follow` is replaced by `codeword.`
- Need a new procedure int inputBit().
- Codeword initialized with the first (most significant) 16 bits of the full codeword.
- Codeword is always between low and high and goes through the same doubling as them followed by adding inputBit().

Adaptive Arithmetic Decoder

```
public class H7B{

   static final int maxRange = 65536;  // 2 ** 16
   static final int half = 32768;
   static final int quarter = 16384;
   static final int threequarters = 49152;
   static final int numberOfContexts = 1024;
   int width = 0, height = 0, bytesPerRow = 0;
   boolean[][] bitmap = null;
   int[][] count = new int[numberOfContexts][2];
   int low = 0; int high = maxRange;
   int inBuf = 0; int inPosition = 0;
   int outBuf = 0; int outPosition = 0;
   int codeword = 0;


  public static void main(String[] args){
      H7B h7 = new H7B();
      h7.readPBMHeader();
      h7.uncompress();
   }
```

```
void uncompress(){
     for (int i = 0; i < 16; i++){
       codeword <<= 1;
       codeword |= inputBit();
     }
     for (int i = 0; i < height; i++){
       for (int j = 0; j < width; j++){
        int context = getContext(j + 2);
        boolean one = update(context);
        incrementCount(context, one);
        bitmap[2][j + 2] = one;
        outputBit(one);
      }
      for (int j = 2; j < width + 2; j++){
        bitmap[0][j] = bitmap[1][j];
        bitmap[1][j] = bitmap[2][j];
      }

    }
    System.out.flush();
 }
```

```
boolean update(int context){
    int t = low + count[context][0] * (high - low) /
      (count[context][0] + count[context][1]);
    boolean ret = codeword >= t;
    if ( ______ ) low = t; else high = t;
    for (;;){ // double until larger than quarter
      if (high < half){
        high *= 2;
        low *= 2;
        codeword *= 2;
        if (inputBit() > 0) codeword |= 1;
      }else if (low >= half){
        high = (high * 2) - maxRange;
        low = (low * 2) - maxRange;
        // code for codeword
      }else if (low > quarter && high <= threequarters){
        low = (low * 4 - maxRange) / 2;
        high = (high * 4 - maxRange) / 2;
        // code for codeword
      }else break;
   }
   return ret;
  }
```

Homework 7: due 2-9-15

- Complete the update() function in the adaptive arithmetic decoding program H7B.java.
- An example PBM file hand.pbm can be used to test the encoding-decoding process.
- Decode test7 into a PBM bi-level image file and.
- Submit the code you write and an image of the uncompressed test7.