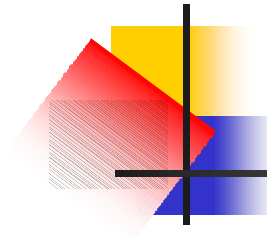
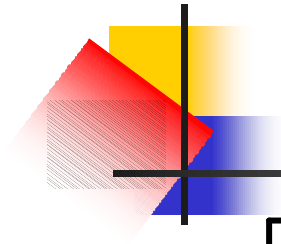


Arithmetic coding

- Let L be a set of items.
- Every item $i \in L$ has a probability $P_i \in [0,1]$, s.t. $\sum_{i \in L} P_i = 1$.
- Every item is represented by the interval: $[\sum_{j<i} P_j, \sum_{j \le i} Pj)$.
- The current interval is divided into sub-intervals according to the item's probabilities.

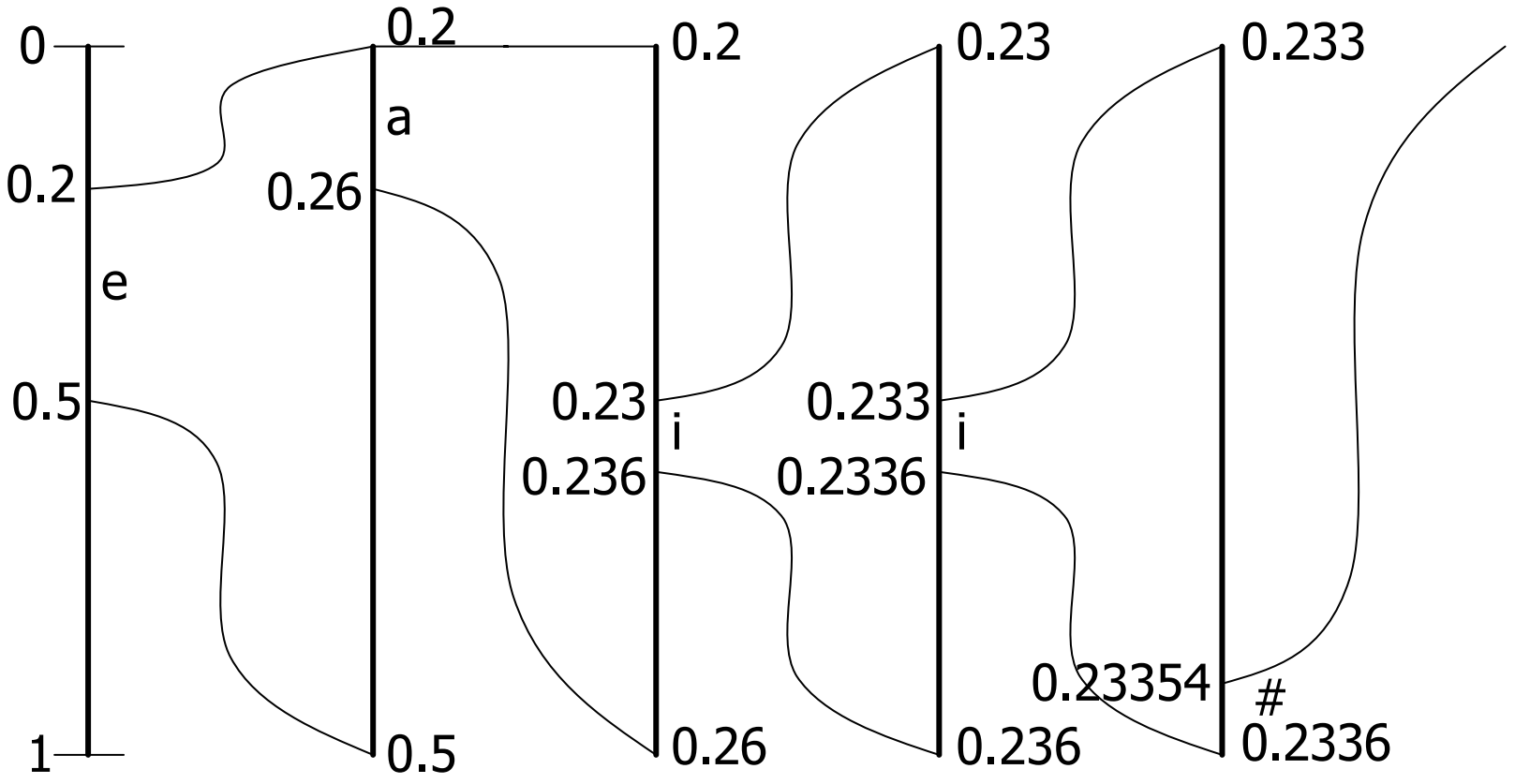


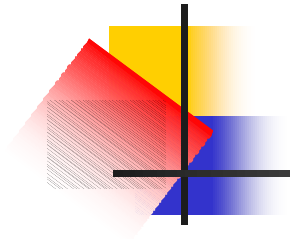
Example

Letter	Probability	Interval
a	0.2	[0,0.2)
e	0.3	[0.2,0.5)
i	0.1	[0.5,0.6)
o	0.2	[0.6,0.8)
u	0.1	[0.8,0.9)
#	0.1	[0.9,1)

Compressing String eaii#

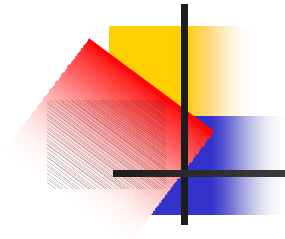
- Message can be any $x \in [0.23354, 0.2336)$





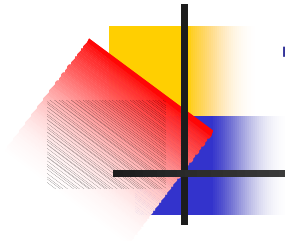
Decompression

- The shortest binary fraction in this interval is 0.0011101111110011 which is 0.23358154296875.
- This number is more than 0.2 and less than 0.5 so the first item is 'e'.
- Two methods to stop decompression:
  - A special character for EOF.
  - Attaching the number of items.



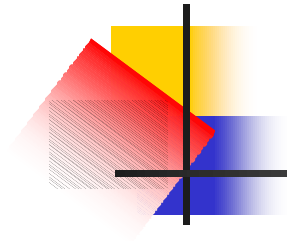
Scaling

- The intervals in arithmetic coding are shrinking very fast.
- Many digits must be used to provide sufficient precision.
- Multiplications and divisions on very long numbers are slow and complicated.



The scaling procedure

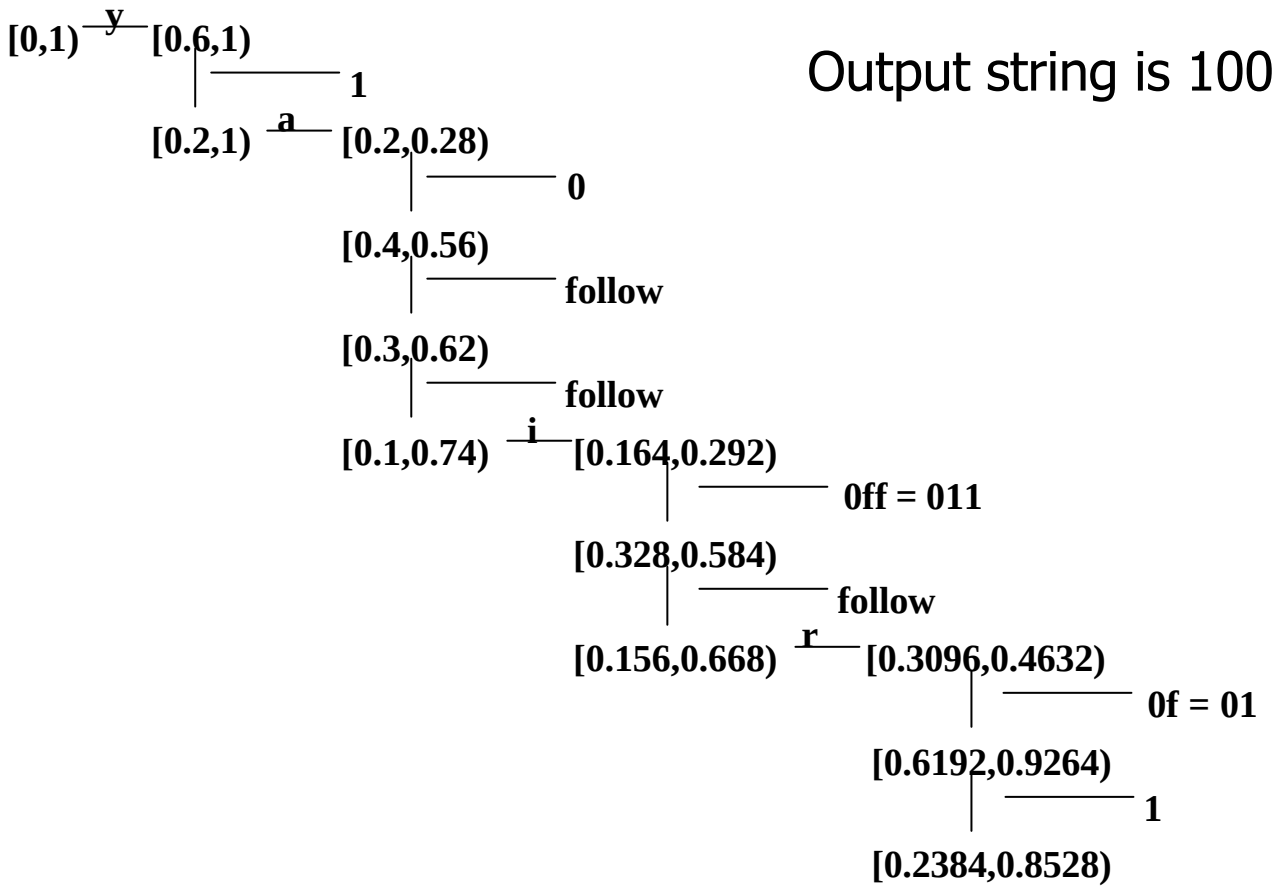
- To have a larger interval, apply **scaling**
  - If all the interval is in [0,0.5], boundaries are doubled and "0" is sent to output.
  - If all the interval is in [0.5,1], the distance from the boundaries to 1 is doubled and "1" is sent to output.
  - If all the interval is in [0.25,0.75] and the interval includes 0.5, the distance from the interval boundaries to 0.5 is doubled. Another following bit will go to output when there will be an output. The bit will be inverted from the output.



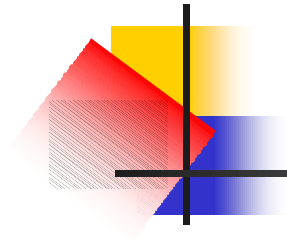
Example

- Items:
  - **a-10%**
  - **i-20%**
  - **r-30%**
  - **y-40%**
- Intervals:
  - **a-[0,0.1)**
  - **i-[0.1,0.3)**
  - **r-[0.3,0.6)**
  - **y-[0.6,1)**
- String to be encoded: "yair".

Example (Cont.)



Output string is 10011011

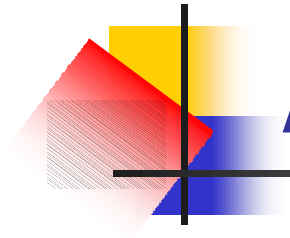


Example (cont.)

- The shortest fraction in the interval [0.2384,0.8528) is 0.5 which is 0.1 in binary. So the final output is 100110111.
- Without scaling we would get:

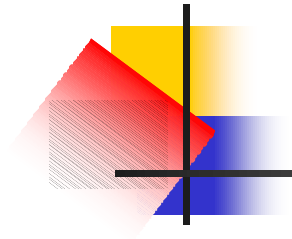
  $[0,1) \xrightarrow{y} [0.6,1) \xrightarrow{a} [0.6,0.64) \xrightarrow{i} [0.604,0.612) \xrightarrow{r} [0.6064,0.6088)$

  The shortest binary fraction in this interval is 0.100110111 which is about 0.607422.



Arithmetic code is optimal

- The information content of each item is $-\log_2 P_i$.
- Huffman gives each item an integer length so only when $\forall i$ $-\log_2 P_i$ is an integer, will Huffman be optimal.
  - Such a distribution is called a Dyadic distribution.
- If a code gives an average codeword length of $H = -\sum_{i=1}^{n} P_i \log P_i$, which is called entropy, it will be an optimal code.



Definitions

- File X: $x_1,\ldots,x_n$.
- Probabilities of items in file X: $p_1,\ldots,p_n$.
- The items' set: $a_1,\ldots,a_m$.
- $x_i \in \{a_1,\ldots,a_m\}$.
- Probabilities of items: $q_1,\ldots,q_m$.
- Frequencies of items: $f_1,\ldots,f_m$.
- $q_i=f_i/n$

Proof of optimality

- Information content in file X: $-\log \prod_{i=1}^{n} p_i$.

$$-\log \prod_{i=1}^{n} p_i = -\sum_{i=1}^{n} \log p_i = -\sum_{i=1}^{m} f_i \log q_i = -n \sum_{i=1}^{m} \frac{f_i}{n} \log q_i =$$

$$-n \sum_{i=1}^{m} q_i \log q_i = n \cdot H$$

- Holds for n items,

  so one item is encoded exactly by H bits.