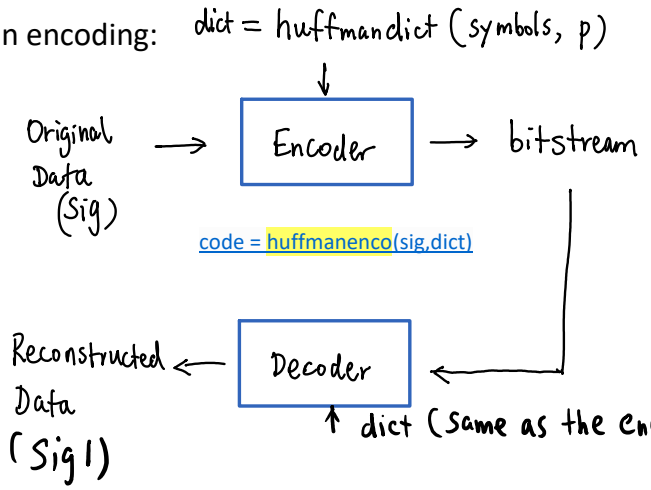
Lecture 13

Huffman codec (cont'd):

Huffman encoding:




```
>> symbols = 1:6;
p = [.5 .125 .125 .125 .0625 .0625];
>>[dict,avglen] = huffmandict(symbols,p);
>> Sig = randsrc(100,1,[symbols;p]);

>> code = huffmanenco(Sig,dict);
>> 224/100
ans =
    2.2400

>> avglen
avglen =
    2.1250
```

Sig1 = huffmandeco(code,dict);

huffmandeco

Decode binary code by Huffman decoding

Source entropy: (= avglen)

```
p = [.5 .125 .125 .125 .0625 .0625];

>> -sum(p.*log2(p))
ans =
    2.1250
```

Lossless check:

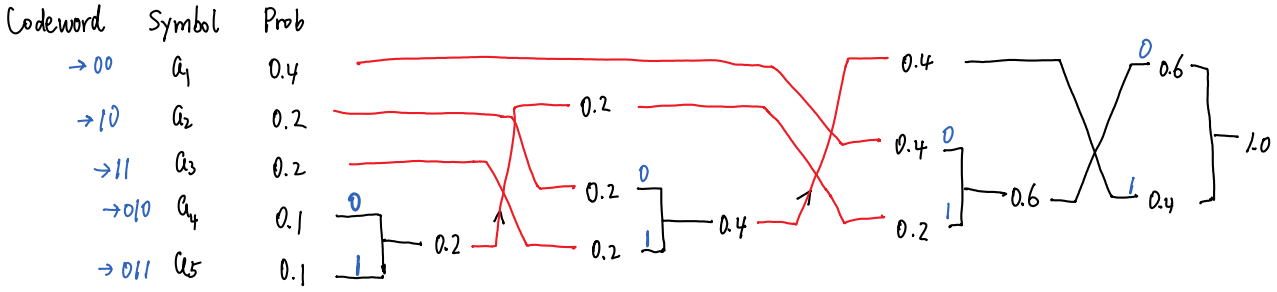
```
>> isequal(Sig, sig1)
ans =
  logical
   1
```

Asymmetry between Huffman Encoding and Decoding (in terms of computational complexity)

```
>> Sig = randsrc(100000,1,[symbols;p]);
>> tic; code = huffmanenco (Sig,dict); toc
Elapsed time is 0.032314 seconds.

>> tic; Sig1 = huffmandeco (code,dict); toc
Elapsed time is 0.349683 seconds.
```

Comparison of encoding and decoding using the following example:



Input : $a_1, a_4, a_5, a_2, a_3, a_1$

Coded bitstream: 00 010 011 10 11 00 (table lookup)

Decoded sequence: a_1 a_4 a_5

Table

a_1 :	00
a_2 :	10
a_3 :	11
a_4 :	010
a_5 :	011

Decoding involves multiple table entries -- more costly computationally

- Huffman coding of symbol blocks

Codeword	Symbol	Prob
→ 00	a_1	0.4
→ 10	a_2	0.2
→ 11	a_3	0.2
→ 010	a_4	0.1
→ 011	a_5	0.1

$\{a_1a_1, a_1a_2, a_1a_3, \cdots, a_2a_1, a_2a_2, \cdots, a_5a_5\}$

$5 \times 5 = 25$ block symbols

If we assume that the symbols are independent:

$P(a_1a_1) = P(a_1) \cdot P(a_1)$

$\vdots \qquad \vdots$

$P(a_5a_5) = P(a_5) \cdot P(a_5)$

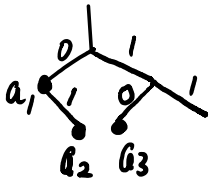
- Huffman codes are uniquely decodable!

Not all the codes are uniquely decodable:

e.g., Code : { 0, 01, 10 }

↓ ↓ ↓

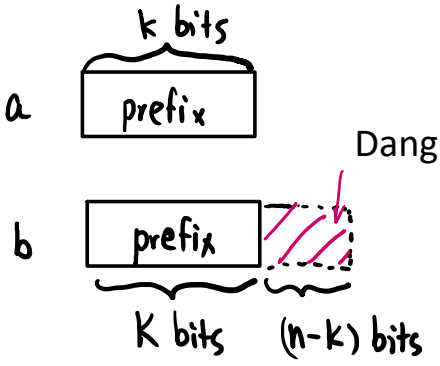
a_1 a_2 a_3



Input data: a_1 a_3 —Encoding→ 0 10 —Decoding→ { a_2 a_1 ; a_1 a_3 }

- Test for uniquely decodable code:

Based on Dangling Suffix:



For example:

a_1: 0

a_2: 01 ⇒ a_1 is a prefix of b.

↑ dangling suffix

Another example:

a : 0/0

b : 0/0 11 ← dangling suffix