CSE 326: Data Structures

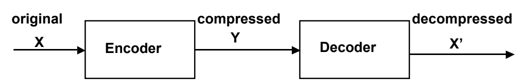
Dictionaries for Data Compression

Neva Cherniavsky
Summer 2006

Why compress files?

What is a file?

Data Compression



- **Lossless** compression X = X'
- **Lossy** compression X != X'
- **Compression Ratio** |X|/|Y|
  - Where |X| is the # of bits in X.

Dictionary Coding

- Does not use statistical knowledge of data.
- Encoder: As the input is processed develop a dictionary and transmit the index of strings found in the dictionary.
- Decoder: As the code is processed reconstruct the dictionary to invert the process of encoding.
- Examples: LZW, LZ77, Sequitur
- Applications: Unix Compress, gzip, GIF

LZW Encoding Algorithm

```
Repeat
  find the longest match w in the dictionary
  output the index of w
  put wa in the dictionary where a was the
          unmatched symbol
```

LZW Encoding Example (1)

Dictionary

0 a
1 b

a b a b a b a b a

LZW Encoding Example (2)

Dictionary

0 a
1 b
2 ab

a b a b a b a b a
0

LZW Encoding Example (3)

Dictionary

0 a
1 b
2 ab
3 ba

a b a b a b a b a
0 1

LZW Encoding Example (4)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba

a b a b a b a b a
0 1 2

LZW Encoding Example (5)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab

a b a b a b a b a
0 1 2 4

LZW Encoding Example (6)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab

a b a b a b a b a
0 1 2 4 3

LZW Decoding Algorithm

- Emulate the encoder in building the dictionary. Decoder is slightly behind the encoder.

```
initialize dictionary;
decode first index to w;
put w? in dictionary;
repeat
   decode the first symbol s of the index;
   complete the previous dictionary entry with s;
   finish decoding the remainder of the index;
   put w? in the dictionary where w was just decoded;
```

13

LZW Decoding Example (1)

Dictionary

0 a
1 b
2 a?

0 1 2 4 3 6
a

14

LZW Decoding Example (2a)

Dictionary

0 a
1 b
2 ab

0 1 2 4 3 6
a b

15

LZW Decoding Example (2b)

Dictionary

0 a
1 b
2 ab
3 b?

0 1 2 4 3 6
a b

16

LZW Decoding Example (3a)

Dictionary

0 a
1 b
2 ab
3 ba

0 1 2 4 3 6
a b a

17

LZW Decoding Example (3b)

Dictionary

0 a
1 b
2 ab
3 ba
4 ab?

0 1 2 4 3 6
a b ab

18

LZW Decoding Example (4a)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba

0 1 2 4 3 6
a b ab a

LZW Decoding Example (4b)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 aba?

0 1 2 4 3 6
a b ab aba

LZW Decoding Example (5a)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab

0 1 2 4 3 6
a b ab aba b

LZW Decoding Example (5b)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 ba?

0 1 2 4 3 6
a b ab aba ba

LZW Decoding Example (6a)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 bab

0 1 2 4 3 6
a b ab aba ba b

LZW Decoding Example (6b)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 bab
7 bab?

0 1 2 4 3 6
a b ab aba ba bab

Decoding Exercise

Base Dictionary

0 1 4 0 2 0 3 5 7

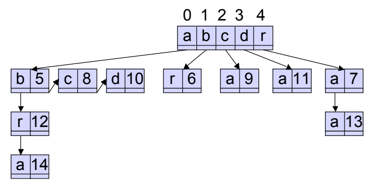
0 a
1 b
2 c
3 d
4 r

25

Trie Data Structure for Encoder's Dictionary

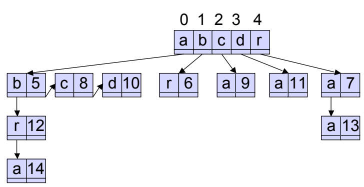
- Fredkin (1960)

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	abr
4	r	13	raa
5	ab	14	abra
6	br		
7	ra		
8	ac		



26

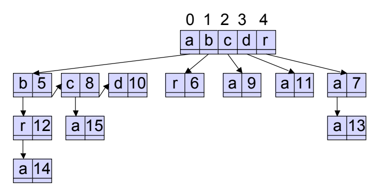
Encoder Uses a Trie (1)



a b r a c a d a b r a a b r a c a d a b r a
0 1 4 0 2 0 3 5 7 12

27

Encoder Uses a Trie (2)



a b r a c a d a b r a a b r a c a d a b r a
0 1 4 0 2 0 3 5 7 12 8

28

Decoder's Data Structure

- Simply an array of strings

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	abr
4	r	13	raa
5	ab	14	abr?
6	br		
7	ra		
8	ac		

0 1 4 0 2 0 3 5 7 12 8 ...
a b r a c a d ab ra abr

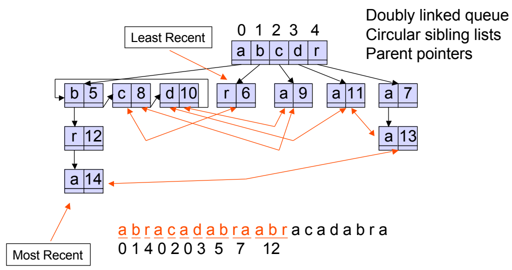
29

Bounded Size Dictionary

- Bounded Size Dictionary
  - n bits of index allows a dictionary of size 2^n
  - Doubtful that long entries in the dictionary will be useful.
- Strategies when the dictionary reaches its limit.
  1. Don't add more, just use what is there.
  2. Throw it away and start a new dictionary.
  3. Double the dictionary, adding one more bit to indices.
  4. Throw out the least recently visited entry to make room for the new entry.

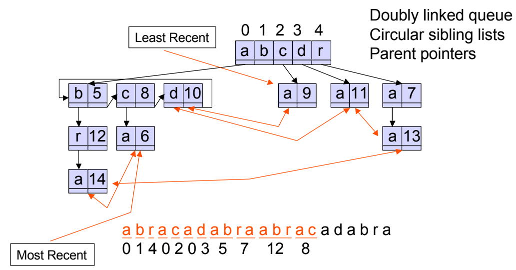
30

Implementing the LRV Strategy



31

Implementing the LRV Strategy



32

Notes on LZW

- Extremely effective when there are repeated patterns in the data that are widely spread.
- Negative: Creates entries in the dictionary that may never be used.
- Applications:
  - Unix compress, GIF, V.42 bis modem standard

33

LZ77

- Ziv and Lempel, 1977
- Dictionary is implicit
- Use the string coded so far as a dictionary.
- Given that $x_1x_2...x_n$ has been coded we want to code $x_{n+1}x_{n+2}...x_{n+k}$ for the largest k possible.

34

Solution A

- If $x_{n+1}x_{n+2}...x_{n+k}$ is a substring of $x_1x_2...x_n$ then $x_{n+1}x_{n+2}...x_{n+k}$ can be coded by <j,k> where j is the beginning of the match.
- Example

ababababa babababababababab....
coded

ababababa bababab babababab....
<2,8>

35

Solution A Problem

- What if there is no match at all in the dictionary?

ababababa cabababababababab....
coded

- Solution B. Send tuples <j,k,x> where
  - If k = 0 then x is the unmatched symbol
  - If k > 0 then the match starts at j and is k long and the unmatched symbol is x.

36

Solution B

- If $x_{n+1}x_{n+2}...x_{n+k}$ is a substring of $x_1x_2...x_n$ and $x_{n+1}x_{n+2}... x_{n+k}x_{n+k+1}$ is not then $x_{n+1}x_{n+2}...x_{n+k}$ x_{n+k+1} can be coded by
  $<j,k, x_{n+k+1}>$
  where j is the beginning of the match.
- Examples

ababababa cababababababab....

ababababa c abababab ababab....
<0,0,c> <1,9,b>

37

Solution B Example

a babababababababababab.....
<0,0,a>

a b ababababababababababab.....
<0,0,b>

a b aba bababababababababab.....
<1,2,a>

a b aba babab abababababab.....
<2,4,b>

a b aba babab ababababababa bab.....
<1,10,a>

38

Surprise Code!

a bababababababababababab$
<0,0,a>

a b ababababababababababab$
<0,0,b>

a b ababababababababababab$
<1,22,$>

39

Surprise Decoding

<0,0,a><0,0,b><1,22,$>

<0,0,a>	a
<0,0,b>	b
<1,22,$>	a
<2,21,$>	b
<3,20,$>	a
<4,19,$>	b
...	
<22,1,$>	b
<23,0,$>	$

40

Surprise Decoding

<0,0,a><0,0,b><1,22,$>

<0,0,a>	a
<0,0,b>	b
<1,22,$>	a
<2,21,$>	b
<3,20,$>	a
<4,19,$>	b
...	
<22,1,$>	b
<23,0,$>	$

41

Solution C

- The matching string can include part of itself!
- If $x_{n+1}x_{n+2}...x_{n+k}$ is a substring of
  $x_1x_2...x_n\ x_{n+1}x_{n+2}...x_{n+k}$
  that begins at $j \le n$ and $x_{n+1}x_{n+2}... x_{n+k}x_{n+k+1}$ is not then $x_{n+1}x_{n+2}...x_{n+k}\ x_{n+k+1}$ can be coded by
  $<j,k, x_{n+k+1}>$

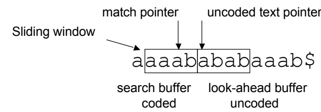
42

In Class Exercise

- Use Solution C to code the string
  - aaaabaaabaabab$

Bounded Buffer – Sliding Window

- We want the triples <j,k,x> to be of bounded size. To achieve this we use bounded buffers.
  - Search buffer of size s is the symbols $x_{n-s+1}...x_n$ j is then the offset into the buffer.
  - Look-ahead buffer of size t is the symbols $x_{n+1}...x_{n+t}$
- Match pointer can start in search buffer and go into the look-ahead buffer but no farther.



tuple
<2,5,a>

Search in the Sliding Window

	offset	length
aaaababaabaaab$	1	0
aaaababaabaaab$	2	1
aaaababaabaaab$	2	2
aaaababaabaaab$	2	3
aaaababaabaaab$	2	4
aaaababaabaaab$	2	5

tuple
<2,5,a>

Coding Example

s = 4, t = 4, a = 3

	tuple
aaaababaabaaab$	<0,0,a>
aaaababaabaaab$	<1,3,b>
aaaababaabaaab$	<2,5,a>
aaaababaabaaab$	<4,2,$>

Coding the Tuples

- Simple fixed length code

$$\lceil \log_2(s+1) \rceil + \lceil \log_2(s+t+1) \rceil + \lceil \log_2 a \rceil$$

s = 4, t = 4, a = 3	tuple	fixed code
	<2,5,a>	010 0101 00

- Variable length code using adaptive Huffman or arithmetic code on Tuples
  - Two passes, first to create the tuples, second to code the tuples
  - One pass, by pipelining tuples into a variable length coder

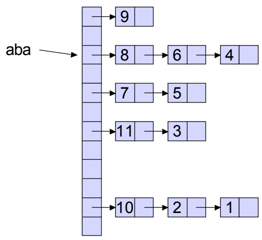
Zip and Gzip

- Search Window
  - Search buffer 32KB
  - Look-ahead buffer 258 Bytes
- How to store such a large dictionary
  - Hash table that stores the starting positions for all three byte sequences.
  - Hash table uses chaining with newest entries at the beginning of the chain. Stale entries can be ignored.
- Second pass for Huffman coding of tuples.
- Coding done in blocks to avoid disk accesses.

Example

12

aaaabababaaabaaaababababaaabba$



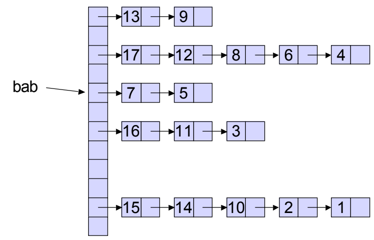
Offset =12 – 8 = 4
Length = 5
Tuple = <4,5,a>

49

Example

18

aaaababababaaabaaaababababaaabba$



No match
Tuple = <0,0,b>

50

Notes on LZ77

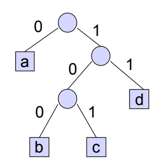
- Very popular especially in unix world
- Many variants and implementations
  - Zip, Gzip, PNG, PKZip,Lharc, ARJ
- Tends to work better than LZW
  - LZW has dictionary entries that are never used
  - LZW has past strings that are not in the dictionary
  - LZ77 has an implicit dictionary. Common tuples are coded with few bits.

51

Huffman Coding

- Huffman (1951)
- Uses frequencies of symbols in a string to build a variable rate prefix code.
  - Each symbol is mapped to a binary string.
  - More frequent symbols have shorter codes.
  - No code is a prefix of another.
- Example:

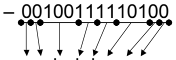
a 0
b 100
c 101
d 11



52

Variable Rate Code Example

- Example: a 0, b 100, c 101, d 11
- Coding:
  - aabddcaa = 16 bits
  - 0 0 100 11 11 101 0 0= 14 bits
- Prefix code ensures unique decodability.
  - 00100111110100



  - a a b d d c a a

53

Cost of a Huffman Tree

- Let $p_1, p_2, \ldots, p_m$ be the probabilities for the symbols $a_1, a_2, \ldots, a_m$, respectively.
- Define the cost of the Huffman tree T to be

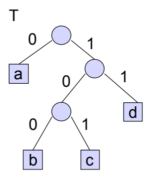
$$C(T) = \sum_{i=1}^{m} p_i r_i$$

where r_i is the length of the path from the root to a_i.
- C(T) is the expected length of the code of a symbol coded by the tree T. C(T) is the bit rate of the code.

54

Example of Cost

- Example: a 1/2, b 1/8, c 1/8, d 1/4



C(T) = 1 x 1/2 + 3 x 1/8 + 3 x 1/8 + 2 x 1/4 = 1.75
a b c d

55

Huffman Tree

- Input: Probabilities $p_1, p_2, \ldots, p_m$ for symbols $a_1, a_2, \ldots, a_m$, respectively.
- Output: A tree that minimizes the average number of bits (bit rate) to code a symbol. That is, minimizes

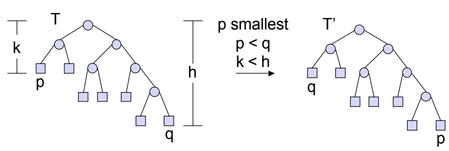
$$HC(T) = \sum_{i=1}^{m} p_i r_i \quad \text{bit rate}$$

where r_i is the length of the path from the root to a_i. This is the Huffman tree or Huffman code

56

Optimality Principle 1

- In a Huffman tree a lowest probability symbol has maximum distance from the root.
  - If not exchanging a lowest probability symbol with one at maximum distance will lower the cost.

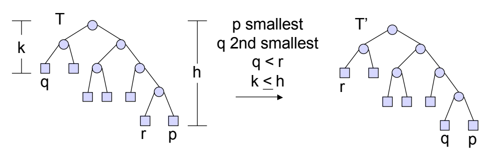


C(T') = C(T) + hp - hq + kq - kp = C(T) - (h-k)(q-p) < C(T)

57

Optimality Principle 2

- The second lowest probability is a sibling of the the smallest in some Huffman tree.
  - If not, we can move it there not raising the cost.

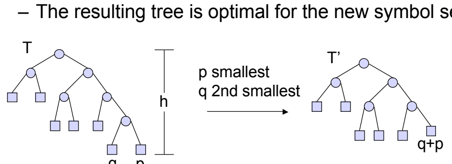


C(T') = C(T) + hq - hr + kr - kq = C(T) - (h-k)(r-q) $\leq$ C(T)

58

Optimality Principle 3

- Assuming we have a Huffman tree T whose two lowest probability symbols are siblings at maximum depth, they can be replaced by a new symbol whose probability is the sum of their probabilities.
  - The resulting tree is optimal for the new symbol set.

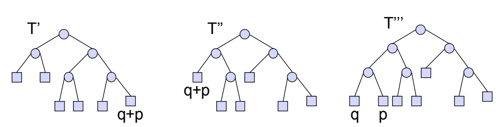


C(T') = C(T) + (h-1)(p+q) - hp -hq = C(T) - (p+q)

59

Optimality Principle 3 (cont')

- If T' were not optimal then we could find a lower cost tree T''. This will lead to a lower cost tree T''' for the original alphabet.



C(T''') = C(T'') + p + q < C(T') + p + q = C(T) which is a contradiction

60

Recursive Huffman Tree Algorithm

1. If there is just one symbol, a tree with one node is optimal. Otherwise
2. Find the two lowest probability symbols with probabilities p and q respectively.
3. Replace these with a new symbol with probability p + q.
4. Solve the problem recursively for new symbols.
5. Replace the leaf with the new symbol with an internal node with two children with the old symbols.

61

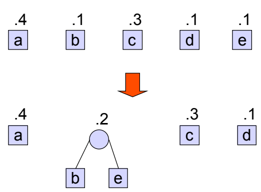
Iterative Huffman Tree Algorithm

```
form a node for each symbol a_i with weight p_i;
insert the nodes in a min priority queue ordered by probability;
while the priority queue has more than one element do
   min1 := delete-min;
   min2 := delete-min;
   create a new node n;
   n.weight := min1.weight + min2.weight;
   n.left := min1;
   n.right := min2;
   insert(n)
return the last node in the priority queue.
```

62

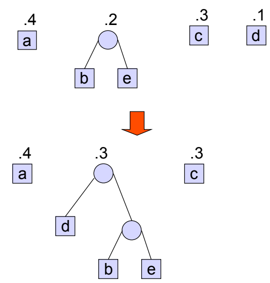
Example of Huffman Tree Algorithm (1)

- P(a) =.4, P(b)=.1, P(c)=.3, P(d)=.1, P(e)=.1



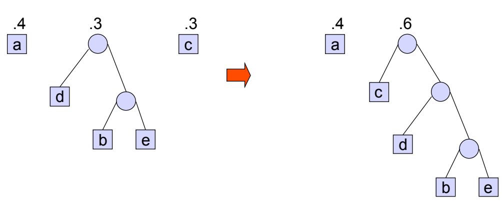
63

Example of Huffman Tree Algorithm (2)



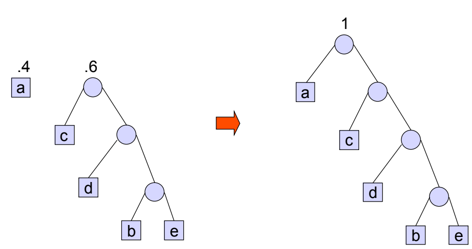
64

Example of Huffman Tree Algorithm (3)



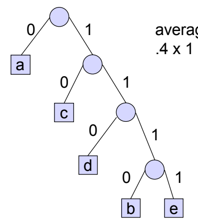
65

Example of Huffman Tree Algorithm (4)



66

Huffman Code



average number of bits per symbol is
.4 x 1 + .1 x 4 + .3 x 2 + .1 x 3 + .1 x 4 = 2.1

a	0
b	1110
c	10
d	110
e	1111

Optimal Huffman Code vs. Entropy

- P(a) =.4, P(b)=.1, P(c)=.3, P(d)=.1, P(e)=.1

Entropy

$H = -(.4 \times \log_2(.4) + .1 \times \log_2(.1) + .3 \times \log_2(.3) + .1 \times \log_2(.1) + .1 \times \log_2(.1))$
= 2.05 bits per symbol

Huffman Code

$HC = .4 \times 1 + .1 \times 4 + .3 \times 2 + .1 \times 3 + .1 \times 4$
= 2.1 bits per symbol
pretty good!

In Class Exercise

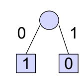
- P(a) = 1/2, P(b) = 1/4, P(c) = 1/8, P(d) = 1/16, P(e) = 1/16
- Compute the Huffman tree and its bit rate.
- Compute the Entropy
- Compare
- Hint: For the tree change probabilities to be integers: a:8, b:4, c:2, d:1, e:1. Normalize at the end.

Quality of the Huffman Code

- The Huffman code is within one bit of the entropy lower bound.

$$H \le HC \le H + 1$$

- Huffman code does not work well with a two symbol alphabet.
  - Example: P(0) = 1/100, P(1) = 99/100
  - HC = 1 bits/symbol



  - $H = -((1/100)^*\log_2(1/100) + (99/100)\log_2(99/100))$
    = .08 bits/symbol

Powers of Two

- If all the probabilities are powers of two then

$$HC = H$$

- Proof by induction on the number of symbols.

Let $p_1 \le p_2 \le \ldots \le p_n$ be the probabilities that add up to 1

If n = 1 then HC = H (both are zero).

If n > 1 then $p_1 = p_2 = 2^{-k}$ for some k, otherwise the sum cannot add up to 1.

Combine the first two symbols into a new symbol of probability $2^{-k} + 2^{-k} = 2^{-k+1}$.

Powers of Two (Cont.)

By the induction hypothesis

$$HC(p_1 + p_2, p_3, \ldots, p_n) = H(p_1 + p_2, p_3, \ldots, p_n)$$
$$= -(p_1 + p_2)\log_2(p_1 + p_2) - \sum_{i=3}^{n} p_i \log_2(p_i)$$
$$= -2^{-k+1}\log_2(2^{-k+1}) - \sum_{i=3}^{n} p_i \log_2(p_i)$$
$$= -2^{-k+1}(\log_2(2^{-k}) + 1) - \sum_{i=3}^{n} p_i \log_2(p_i)$$
$$= -2^{-k}\log_2(2^{-k}) - 2^{-k}\log_2(2^{-k}) - \sum_{i=3}^{n} p_i \log_2(p_i) - 2^{-k} - 2^{-k}$$
$$= -\sum_{i=1}^{n} p_i \log_2(p_i) - (p_1 + p_2)$$
$$= H(p_1, p_2, \ldots, p_n) - (p_1 + p_2)$$

Powers of Two (Cont.)

By the previous page,

$$HC(p_1+p_2,p_3,\ldots,p_n) = H(p_1,p_2,\ldots,p_n) - (p_1+p_2)$$

By the properties of Huffman trees (principle 3),

$$HC(p_1,p_2,\ldots,p_n) = HC(p_1+p_2,p_3,\ldots,p_n) + (p_1+p_2)$$

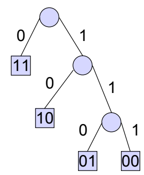
Hence,

$$HC(p_1,p_2,\ldots,p_n) = H(p_1,p_2,\ldots,p_n)$$

73

Extending the Alphabet

- Assuming independence P(ab) = P(a)P(b), so we can lump symbols together.
- Example: P(0) = 1/100, P(1) = 99/100
  - P(00) = 1/10000, P(01) = P(10) = 99/10000, P(11) = 9801/10000.



HC = 1.03 bits/symbol (2 bit symbol)
= .515 bits/bit

Still not that close to H = .08 bits/bit

74

Quality of Extended Alphabet

- Suppose we extend the alphabet to symbols of length k then

$$H \le HC \le H + 1/k$$

- Pros and Cons of Extending the alphabet
  - \+ Better compression
  - \- 2^k symbols
  - \- padding needed to make the length of the input divisible by k

75

Huffman Codes with Context

- Suppose we add a one symbol context. That is in compressing a string $x_1x_2\ldots x_n$ we want to take into account x_{k-1} when encoding x_k.
  - New model, so entropy based on just independent probabilities of the symbols doesn't hold. The new entropy model (2nd order entropy) has for each symbol a probability for each other symbol following it.
  - Example: {a,b,c}

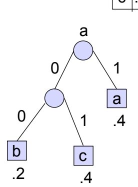
prev \ next	a	b	c
a	.4	.2	.4
b	.1	.9	0
c	.1	.1	.8

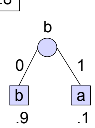
76

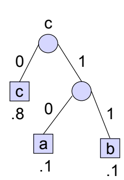
Multiple Codes

prev \ next	a	b	c
a	.4	.2	.4
b	.1	.9	0
c	.1	.1	.8

Code for first symbol
a 00
b 01
c 10







a b b a c c

00 00 0 1 01 0

77

Complexity of Huffman Code Design

- Time to design Huffman Code is O(n log n) where n is the number of symbols.
  - Each step consists of a constant number of priority queue operations (2 deletemin's and 1 insert)

78

Approaches to Huffman Codes

1. Frequencies computed for each input
   - Must transmit the Huffman code or frequencies as well as the compressed input
   - Requires two passes
2. Fixed Huffman tree designed from training data
   - Do not have to transmit the Huffman tree because it is known to the decoder.
   - H.263 video coder
3. Adaptive Huffman code
   - One pass
   - Huffman tree changes as frequencies change

79