CS310 - Advanced Data Structures and Algorithms

Spring 2014 – Class 17

April 8, 2014

Character Coding and File Compression

- Covered in section 12.2.
- File compression using reduced representation of characters.
- Given a file f with n characters (size n bytes $= 8n$ bits). Each byte is a binary representation of the ASCII code of a character.
- Represent every character using a unique **code** of m bits ($m < 8$), and write a file $f1$ with the original characters replaced by their codes.
- The new file size is $8m < 8n$ bits.
- Lossless compression – we should be able to uncompress $f1$ and retrieve the original information in f.

Character Coding and File Compression

- Rationale – we usually don't need all of 8 bits to uniquely represent a character.
- Most files contain much less than 256 different types of characters.
- Example: We have a file of decimal digits and blanks and newlines, 12 codes in all.
- Using an ASCII text file, each char takes 8 bits.
- But 12 characters can easily be represented uniquely by different 4-bit binary codes.
- Such a file could easily be compressed by a factor of 2, by substituting $30 \rightarrow 0, 31 \rightarrow 1, ...39 \rightarrow 9, 0a \rightarrow a, 20 \rightarrow b$

Example

Character	ASCII (dec)	ASCII (hex)	ASCII (bin)	Code
'0'	48	30	00110000	0000
'1'	49	31	00110001	0001
'2'	50	32	00110010	0010
'3'	51	33	00110011	0011
'4'	52	34	00110100	0100
'5'	53	35	00110101	0101
'6'	54	36	00110110	0110
'7'	55	37	00110111	0111
'8'	56	38	00111000	1000
'9'	57	39	00111001	1001
' '	10	0a	00001010	1010
nl	32	20	00100000	1011

Example

- Original file "00123 890$\backslash n$00456 098$\backslash n$" or ASCII in hex: 30 30 31 32 33 20 38 39 30 0a 30 30 34 35 36 20 30 39 38 0a
- Compressed file, hex: 00 12 3b 89 0a 00 45 6b 09 8a (half as long in bytes)
- Uncompress: Read 4 bits at a time from the compressed file: $0 \rightarrow 30, 1 \rightarrow 31, ..., a \rightarrow 0a, b \rightarrow nl$
- Hex is easier to work with in this case, since $16 = 2^4$. 4 binary digits fit exactly into 1 hexadecimal digit.

Efficient File Compression

- We would like a quantitative estimate of the effectiveness of various coding schemes, so we need a distribution of character frequencies.
- Suppose digits 1 through 9 are about equally likely, though of declining frequency with size, but 0, space and newline are much more frequent.

Efficient File Compression

Char	sp	nl	0	1	2	3	4	5	6	7	8	9
Freq.	30	20	10	7	6	5	4	3	3	3	2	2

(95 total)

- With this distribution, we would like short codes for sp, nl, and 0, and longer ones for the other digits.
- But how can we ever uncompress if we dont know the length of the codes?
- The answer is to use **prefix codes**.

Prefix Codes

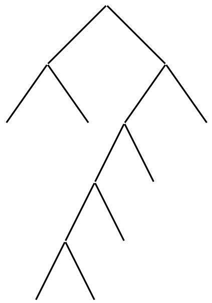
- Prefix just means some initial substring.
- For example 110 is a prefix of 11011.
- We can underline the matching bits: 11011.
- A set of prefix codes has the property that no code (a bit string) is the prefix of another code.
- With a set of prefix codes, if you match up the initial bits of the compressed data with all the bits of a certain code, it can only be that code.
- Then you move to the next bits, etc.

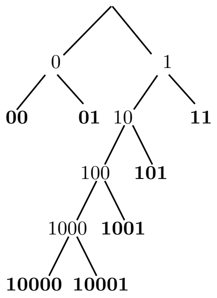
Prefix Codes

- For example $\{00, 10, \text{and } 110\}$ is a set of prefix codes, because all three pass the test:
- Testing 00: neither 10 nor 110 start with 00
- Testing 10: neither 00 nor 110 start with 10
- Testing 110: neither 00 nor 10 start with 110
- $\{0, 10, 11\}$ is also a set of prefix codes.
- The set $\{0, 01, 11\}$ is not a set of prefix codes because 0 is a prefix of 01.

Generating Prefix Codes

- How can we generate a set of prefix codes for a certain use?
- Answer: Compose a binary tree with the right number of leaves.
- Each code is determined by a path from the root to the leaf, where going left gives a 0 and going right a 1. For example:





Generating Prefix Codes

- This defines 00 and 01 from the leaves at the left, and 10000 and 10001 for the leaves at the bottom, and 10001, 1001, 101, and 11 for the leaves going up the right-hand side.
- Now we have some short codes for frequent symbols, and some longer codes for less frequent symbols.
- No bit string is a prefix of another, because each bit string specifies a different path, to a different leaf

Generating Prefix Codes

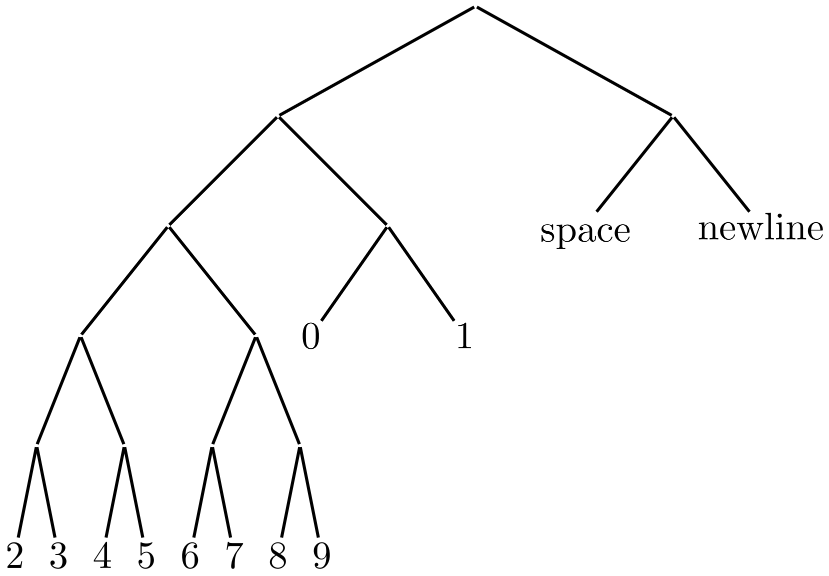
- Here is our 12-symbol example again:
- Suppose digits 1 through 9 are about equally likely, though of declining frequency with size (this is actually observed), but 0, sp, and nl are much more frequent:

Char	sp	nl	0	1	2	3	4	5	6	7	8	9
Freq.	30	20	10	7	6	5	4	3	3	3	2	2

(95 total)

- We can set up a binary tree with these symbols at the leaves, like this for our set of 12 symbols:

Generating Prefix codes



Generating Prefix codes

- Here we see that the first two levels of the binary tree just separate out 4 cases by branching 2 ways at the top and each of those branching two ways.
- Two of these 4 cases are used (the rightmost two), for the most frequent symbols, here sp and nl.
- The remaining two cases are split up into 4 cases by branching in the binary tree at the next level down.

Generating Prefix codes

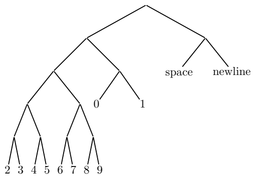
- Two of these cases are used (the rightmost two), for the most frequent digits, 0 and 1.
- The remaining two cases are split into 4 cases at the next level down and then again into 8 cases at the next level down, and then used for the remaining digits 2, 3, 4, 5, 6, 7, 8, and 9.

Generating Prefix codes

- From the binary tree, we read off the codes.
- For example, the nl is reached by traversing down the right hand side of the tree, going right 2 times, so its code is 11.
- The sp is reached by going right and then left, so its code is 10.
- 1 is reached by going left, then right, then right, so its code is 011, and so on.

Generating Prefix Codes

- From the binary tree, we read off the codes.
- nl is reached by traversing down the right hand side of the tree, going right 2 times, so its code is 11.
- The sp is reached by going right and then left, so its code is 10.
- 1 is reached by going left, then right, then right, so its code is 011, and so on.
- Total bits =291 ,much better than 4x95= 380



char	code	freq.	total bits
sp	10	30	60
nl	11	20	40
0	010	10	30
1	011	7	21
2	00000	6	30
3	00001	5	25
4	00010	4	20
5	00011	3	15
6	00100	3	15
7	00101	3	15
8	00110	2	10
9	00101	2	10

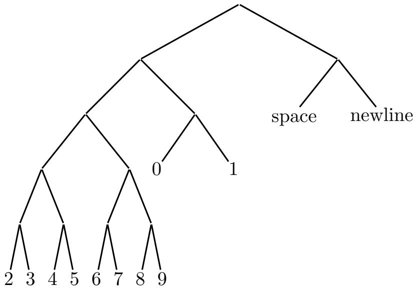
Generating Prefix Codes

- Original file “00123890\\$n$00456098\\$n$”
- In hex: 30 30 31 32 33 20 38 39 30 0a 30 30 34 35 36 20 30 39 38 0a
- Compressed file, binary: 010 010 011 00000 00001 10 00110 00111 010 11 010 010 00010 00011 00100 ...

Decompress the File

- For example: 010010001001010100011111
- 010 is the only first match. We can match down the tree until we reach a leaf, labeled by 0.

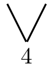
010	$\rightarrow$	0
00100	$\rightarrow$	6
10	$\rightarrow$	sp
10	$\rightarrow$	sp
00111	$\rightarrow$	9
11	$\rightarrow$	nl



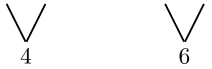
- Thus this bitstring decodes to "006 9$\backslash n$"
- Is this the optimal coding?

Huffman's Coding

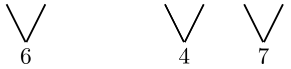
Char	sp	nl	0	1	2	3	4	5	6	7	8	9
Freq.	30	20	10	7	6	5	4	3	3	3	2	2



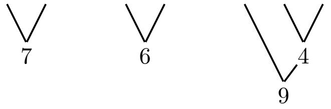
Char	sp	nl	0	1	2	3	8	9	4	5	6	7
Freq.	30	20	10	7	6	5	2	2	4	3	3	3



Char	sp	nl	0	1	6	7	2	3	8	9	4	5
Freq.	30	20	10	7	3	3	6	5	2	2	4	3

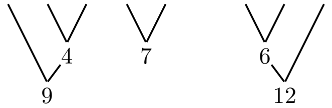


Char	sp	nl	0	4	5	1	6	7	2	3	8	9
Freq.	30	20	10	4	3	7	3	3	6	5	2	2

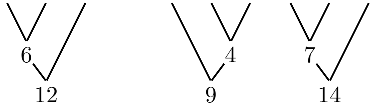


Huffman's Coding

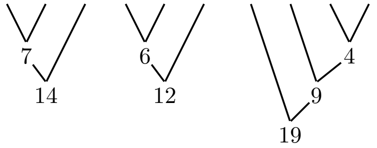
Char	sp	nl	0	3	8	9	4	5	1	6	7	2
Freq.	30	20	10	5	2	2	4	3	7	3	3	6



Char	sp	nl	6	7	2	0	3	8	9	4	5	1
Freq.	30	20	3	3	6	10	5	2	2	4	3	7

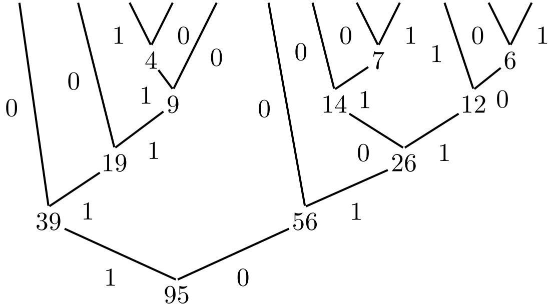


Char	sp	nl	4	5	1	6	7	2	0	3	8	9
Freq.	30	20	4	3	7	3	3	6	10	5	2	2



Huffman's Coding

Char	nl	0	8	9	3	sp	1	4	5	2	6	7
Freq.	20	10	2	2	5	30	7	4	3	6	3	3

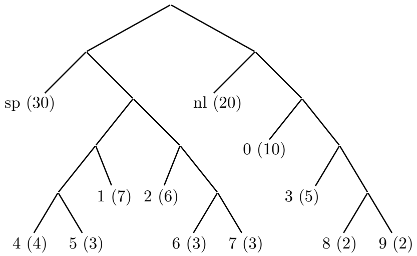


Convention to Assign code: larger weight = 0, smaller weight=1, random code for same weights

Huffman's Coding

char	code	freq.	total bits
sp	00	30	60
nl	10	20	40
0	110	10	30
1	0101	7	28
2	0110	6	24
3	1110	5	20
4	01000	4	20
5	01001	3	15
6	01110	3	15
7	01111	3	15
8	11110	2	10
9	11111	2	10

Total bits = 287


sp (30)
nl (20)
0 (10)
1 (7)
2 (6)
3 (5)
4 (4)
5 (3)
6 (3)
7 (3)
8 (2)
9 (2)