Characters

CS240

ASCII

- **American Standard Code for Information Interchange**
- based on the English alphabet
- Originally developed for telegraph/teleprinters.
- encodes 128 specified characters into 7 bits
  - Letters, numbers, control characters
- 8th bit was used by computer vendors to “extend” ASCII by adding their own characters (i.e. line drawing characters)

ASCII Table

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Unicode

- Assigns a number or “code point” to each character.
- 1,114,112 code points from 0 to 0x10FFFF.
- Latest version of Unicode standard contains more than 110,000 characters.
- First 128 code points are same as ASCII.
- A code point is written as “U+” followed by a hexadecimal number. i.e. A is U+0041

Unicode Encoding

- Can be implemented by different character encodings:
  - UCS-2 – uses 2 bytes. Obsolete. It cannot encode every character in the current Unicode standard.
  - UTF-8 – uses 1 to 6 bytes. Used by Unix and HTML.
  - UTF-16 – uses 2 bytes or 4 bytes. Extends UCS-2. Used by Windows XP, Windows Vista, Windows 7

UTF-8

- **UCS Transformation Format—8-bit**
- a variable-width encoding
- designed for backward compatibility with ASCII
- The first 128 characters of Unicode are encoded with the same binary value as ASCII
- encodes each of the 1,112,064 code points in the Unicode character set using one to six bytes

UTF-8 used in HTML

```
<!DOCTYPE html>
<html lang="en" dir="ltr" class="client-nojs">
<head>
<meta charset="UTF-8" />
```

UTF-8 Encoding Scheme

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

- Code Points 0 to 127 - Same value as the ASCII code. It is represented with one byte, with the high order bit set to 0.
- Code points 128 and higher are represented by 2 to 6 bytes, with a *leading byte* and one or more *continuation bytes*.
  - The number of high-order 1s in the leading byte of a multi-byte sequence indicates the number of bytes in the sequence.
  - All continuation bytes contain ‘10’ in the high order bits.
- The code point’s binary value is encoded in the remaining bits, with 0’s padded to the left if necessary.

UTF-8 Encoding Scheme

- Example:

$ = U+0024

U+0024 is within the range of U+0000 to U+007F, so it is stored in 1 byte.

$(24)_{16}$ is $(36)_{10}$ or $(100100)_2$

According to the specifications, the format for storing into one byte is:

0xxxxxxx where 0 is placed in the highest order bit and the binary is stored into the remaining 7 bits.

100100 is only 6 bits, so we need to pad it with a 0 to make it seven bits:

100100 -> 0100100

The resulting UTF-8 encoded byte will be: 00100100

UTF-8 Encoding Scheme

- Example:

¢ = U+00A2

U+00A2 is within the range of U+0080 to U+07FF, so it is stored in 2 bytes.
$(A2)_{16}$ is $(162)_{10}$ or $(10100010)_2$

According to the specifications, the format for storing into two bytes is:

1st byte: 110xxxxx where 110 is placed in the higher order bits, indicating that this is a **leading byte** of 2 bytes.

2nd byte: 10xxxxxx where 10 is placed in the higher order bits, indicating that this is a **continuation byte**.

10100010 is only 8 bits, so we need to pad it with a 0 to make it 11 bits:

10100010 -> 00010100010

The resulting UTF-8 encoded bytes will be: 11000010 10100010

UTF-8 Encoding Scheme

- Exercise:
- Encode:

  U+10012

  U+A1A

  U+ABC123
- Decode:

  11010010 10100011

  11100101 10101010 10001001

Compression

- encoding information using fewer bits than the original representation

- Lossless data compression – allows original data to be reconstructed from compressed data without loss of information.

- Lossy data compression – allows for some loss. Reconstructed data is an approximation of original data.

Huffman Encoding

- Assigns binary codes to characters to reduce the overall number of bits that would otherwise be needed to store a string of characters.
- Uses a binary tree to generate a code table for lossless data compression.
- A variable length binary code is assigned to each character
- Most frequent characters represented by the smallest binary numbers
- Least frequent characters represented by the largest binary numbers

Huffman Encoding

- ABRACADABRA
- 5 distinct characters.
- In fixed width encoding, would require 3 bits to represent the 5 distinct characters.

i.e. A = 000
B = 001
R = 010
C = 011
D = 100

If each character is 3 bits, and ABRACADABRA has 11 characters, we would need 33 bits to store that word.

Generate a Huffman Table

- List all the characters of the word: ABRACADABRA
- List the frequency of each character

  A: 5

  B: 2

  R: 2

  C: 1

  D: 1

Generate a Huffman Table

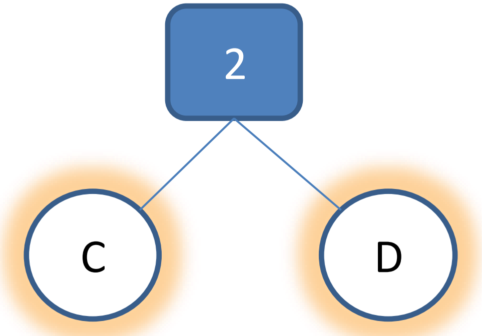
A: 5

B: 2

R: 2

C: 1

D: 1



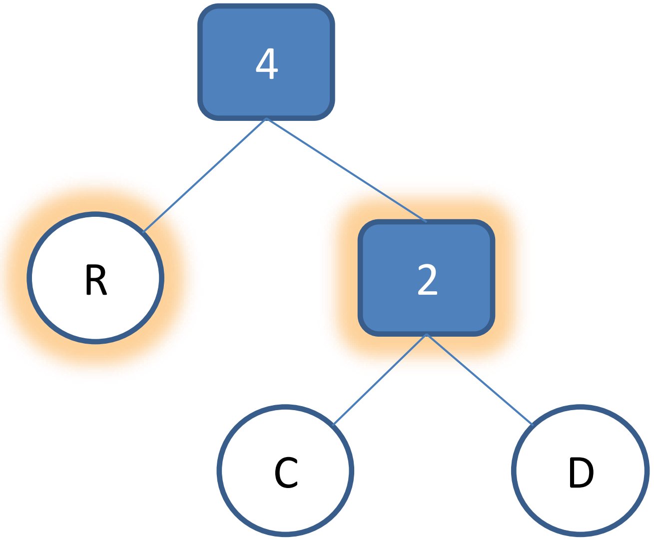
Generate a Huffman Table

A: 5

B: 2

R: 2

2

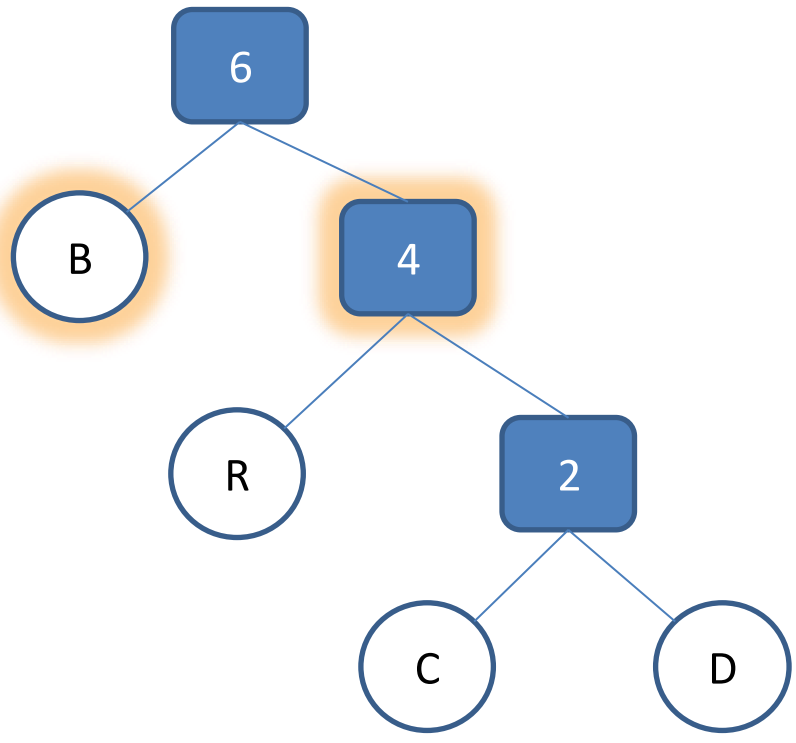


Generate a Huffman Table

A: 5

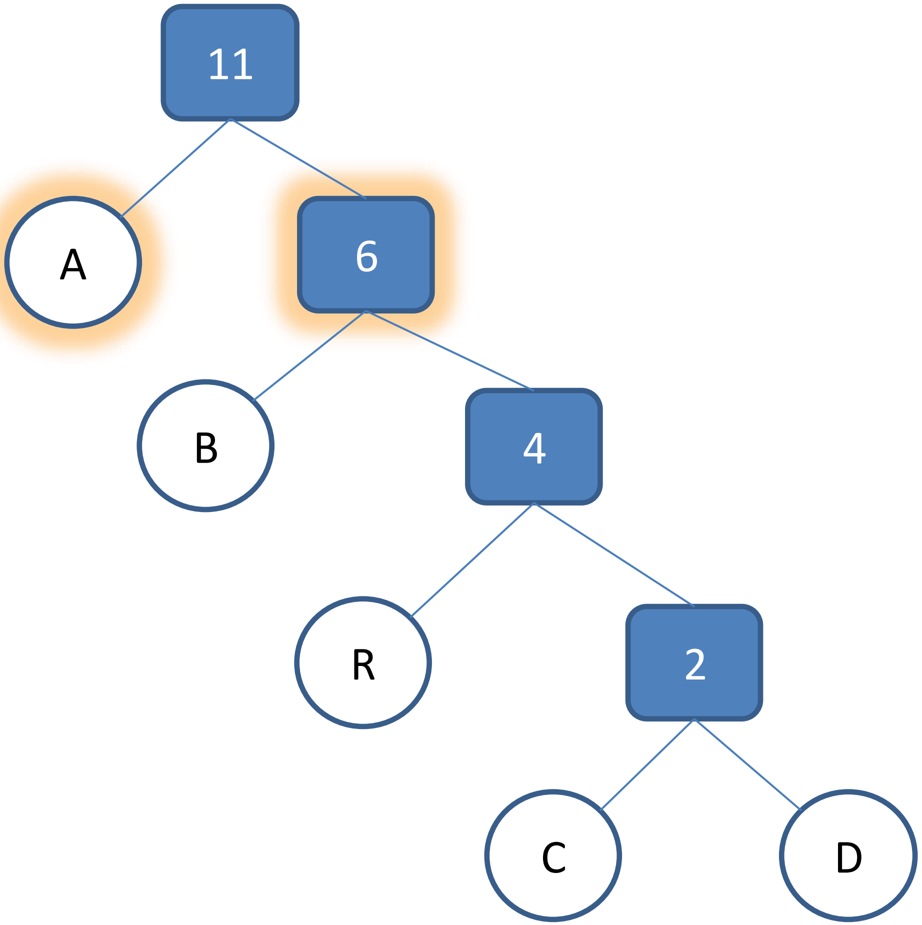
B: 2

4



Generate a Huffman Table

A: 5
6

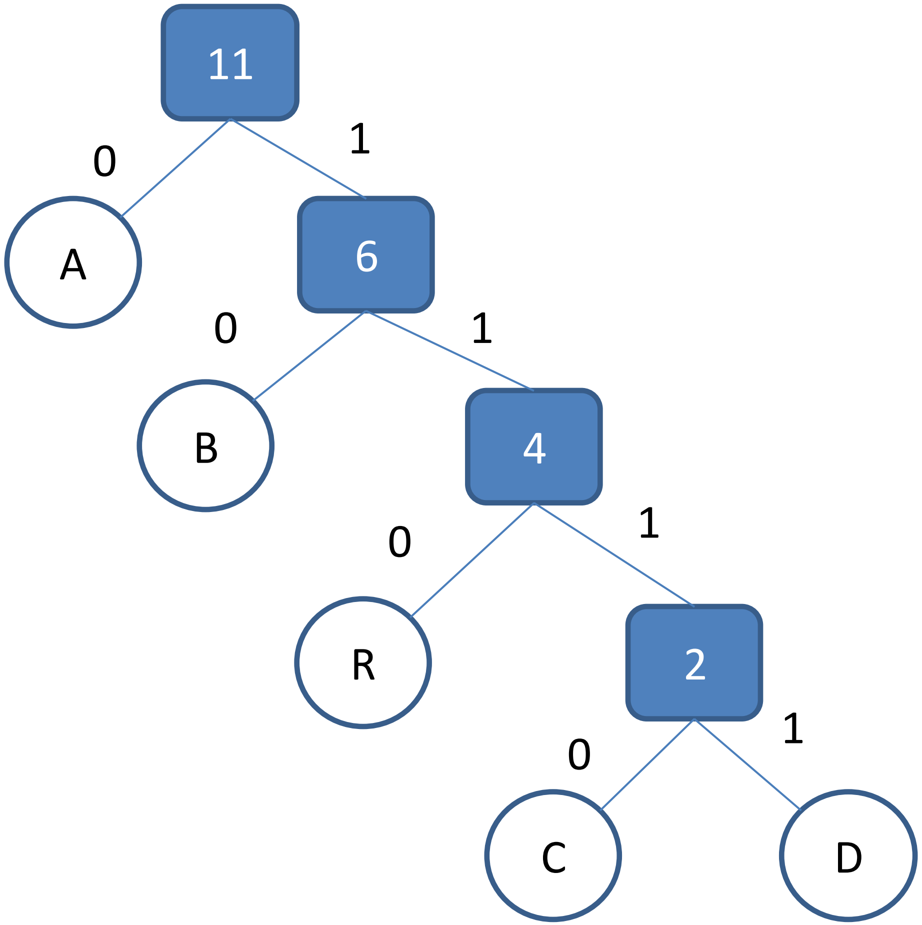


Generate a Huffman Table

Generate Code:

Assign 0 to left branches

Assign 1 to right branches



A = 0

B = 10

R = 110

C = 1110

D = 1111

Encode data

Using: A = 0

B = 10

R = 110

C = 1110

D = 1111

A	B	R	A	C	A	D	A	B	R	A
0	10	110	0	1110	0	1111	0	10	110	0

23 bits to encode ‘ABRACADABRA’ vs 33 bits.