Data Structures and Algorithm

Moazzam Ali Sahi

Lecture # 23

Huffman Coding

Last Lecture

- Binary Search Tree

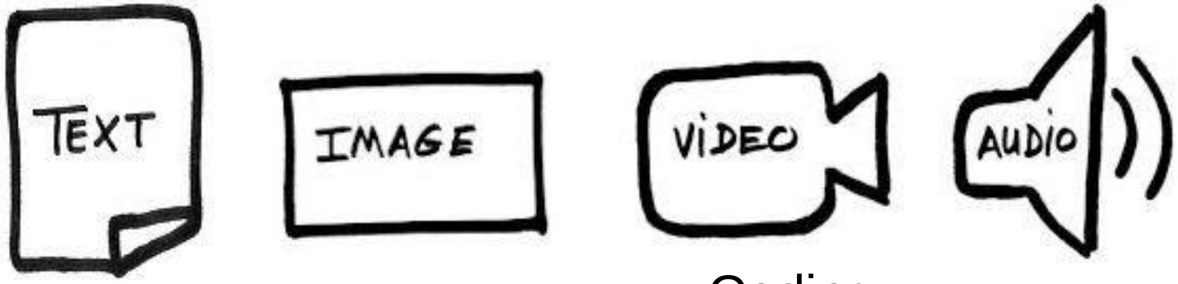
This Lecture

- What is Encoding?
- Fixed vs variable length encoding
- Huffman Coding

Data Compression

- Suppose we have 1000000000 (1G) character data file that we wish to include in an email.
- Suppose file only contains 26 letters {a,…,z}.
- Suppose each letter a in {a,…,z} occurs with frequency f_a.
- Suppose we encode each letter by a binary code
- If we use a fixed length code, we need 5 bits for each character
- The resulting message length is $5\left(f_a + f_b + \cdots f_z\right)$

Coding and Compression



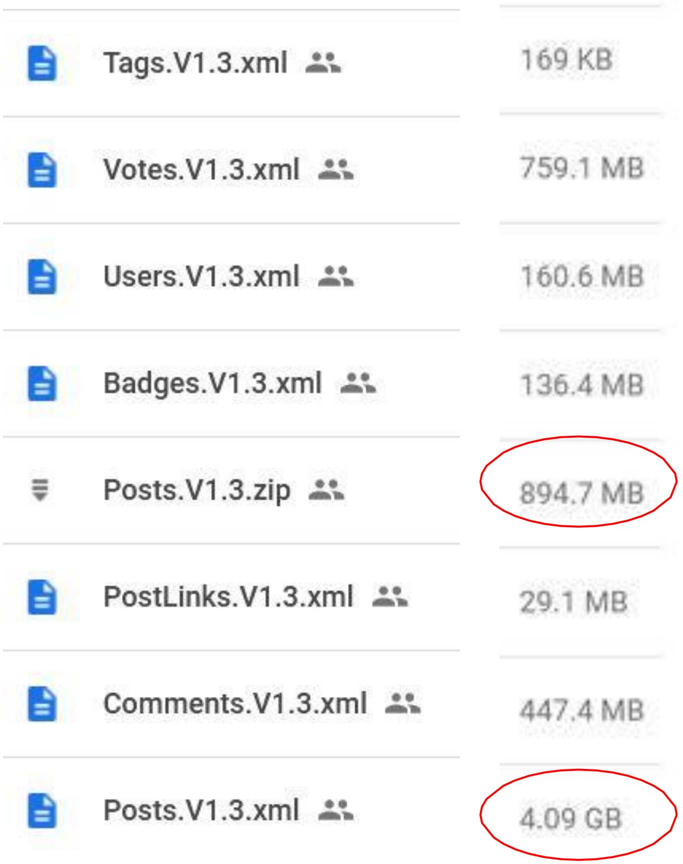
Coding

Set of Bits … 1 0 1 0 1 0 0 1 1 1 …

Compression

Compression reduces the size of a file to save space when storing and to save time when transmitting

Encoding and Compression

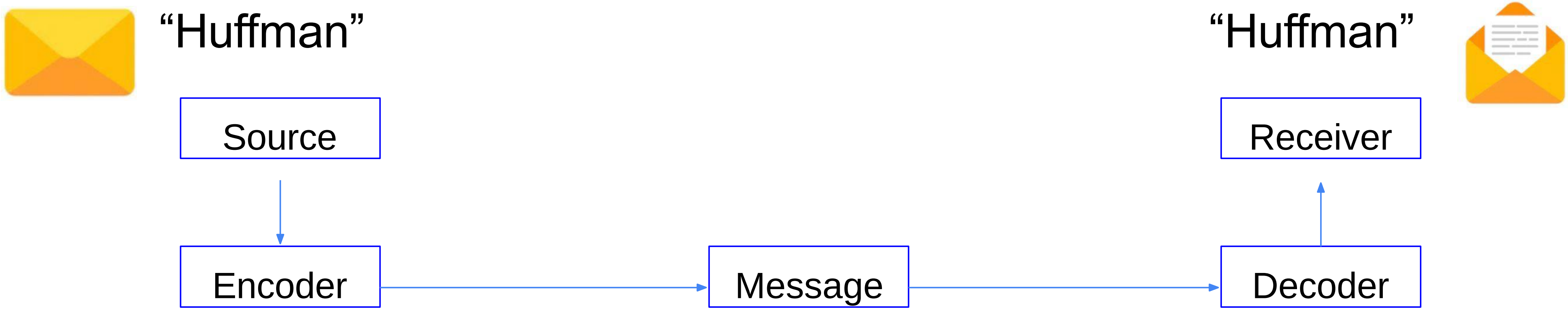


Tags.V1.3.xml	169 KB
Votes.V1.3.xml	759.1 MB
Users.V1.3.xml	160.6 MB
Badges.V1.3.xml	136.4 MB
Posts.V1.3.zip	894.7 MB
PostLinks.V1.3.xml	29.1 MB
Comments.V1.3.xml	447.4 MB
Posts.V1.3.xml	4.09 GB

Compression to reduce data size

ARQMath Lab main file posted as both XML and ZIP

Data Compression Problem



“Huffman”

Source

Encoder

Message

Decoder

Receiver

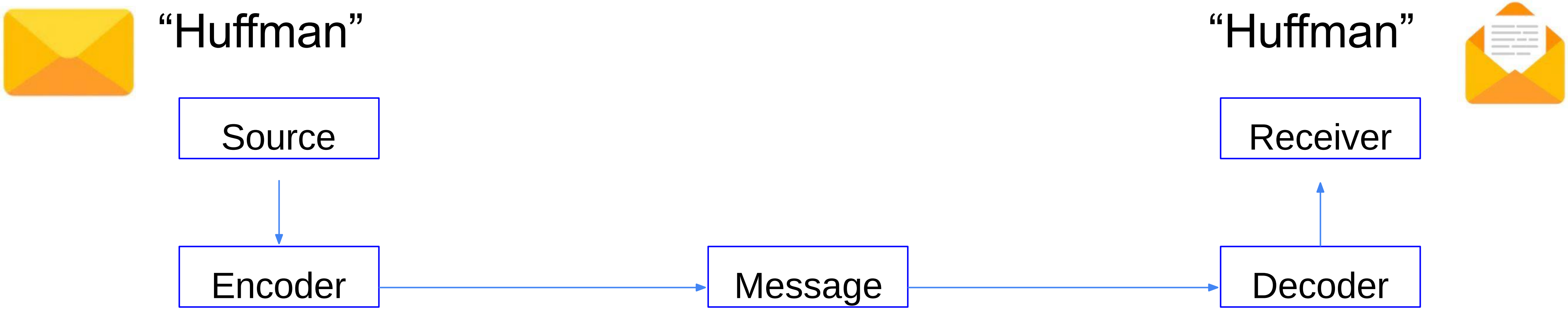
“Huffman”

H:	000
U:	001
F:	010
M:	011
A:	100
N:	101

000001010010011100101

000 001 010 010 011 100 101

Data Compression Problem



“Huffman”

Source

Encoder

Message

Decoder

Receiver

“Huffman”

H:	000
U:	001
F:	010
M:	011
A:	100
N:	101

000001010010011100101

000 001 010 010 011 100 101

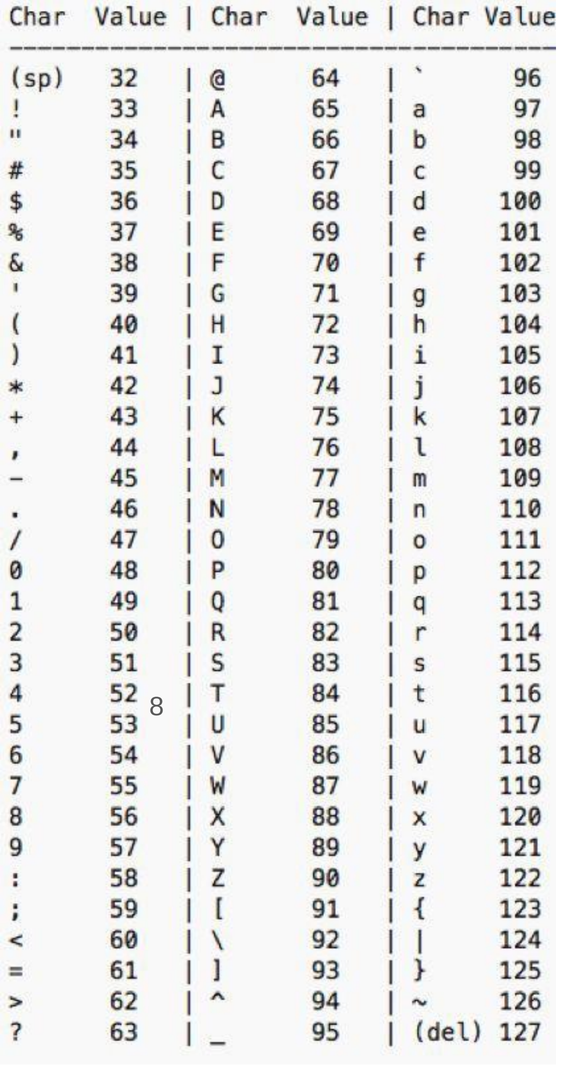
1. Single symbol should have unique binary code
2. Compression must be lossless (**lossless data compression**)
3. Unique decodability

Fixed-Length Coding

Coding Problem: Given a set of symbols, represent them as a **unique** bit string, codewords

Codes used by computer systems

- ASCII
  - uses 8 bits per character
  - can encode 256 characters



Char	Value	Char	Value	Char	Value
(sp)	32	@	64	`	96
!	33	A	65	a	97
"	34	B	66	b	98
#	35	C	67	c	99
$	36	D	68	d	100
%	37	E	69	e	101
&	38	F	70	f	102
'	39	G	71	g	103
(	40	H	72	h	104
)	41	I	73	i	105
*	42	J	74	j	106
+	43	K	75	k	107
,	44	L	76	l	108
-	45	M	77	m	109
.	46	N	78	n	110
/	47	O	79	o	111
0	48	P	80	p	112
1	49	Q	81	q	113
2	50	R	82	r	114
3	51	S	83	s	115
4	52	T	84	t	116
5	53	U	85	u	117
6	54	V	86	v	118
7	55	W	87	w	119
8	56	X	88	x	120
9	57	Y	89	y	121
:	58	Z	90	z	122
;	59	[	91	{	123
<	60	\	92	|	124
=	61	]	93	}	125
>	62	^	94	~	126
?	63	_	95	(del)	127

Fixed-Length Coding

Coding Problem: Given a set of symbols, represent them as a **unique** bit string, codewords

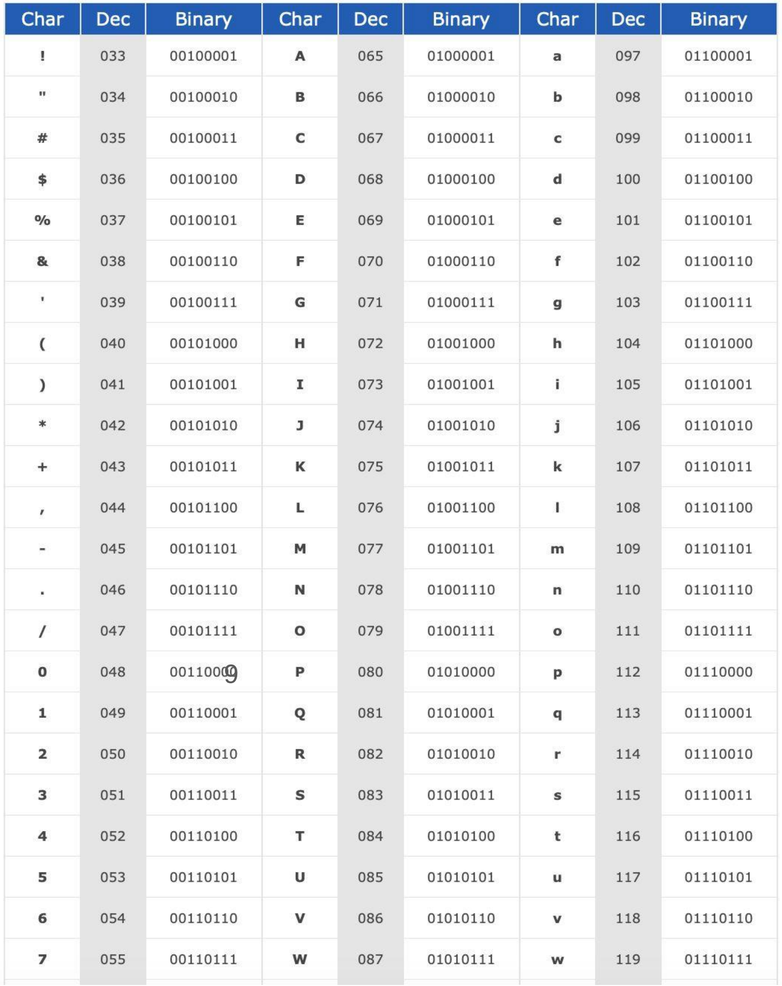
Codes used by computer systems

- ASCII
  - uses 8 bits per character
  - can encode 256 characters
- Unicode
  - 16 bits per character
  - can encode 65536 characters
  - includes all characters encoded by ASCII

ASCII and Unicode are ***fixed-length*** *codes*

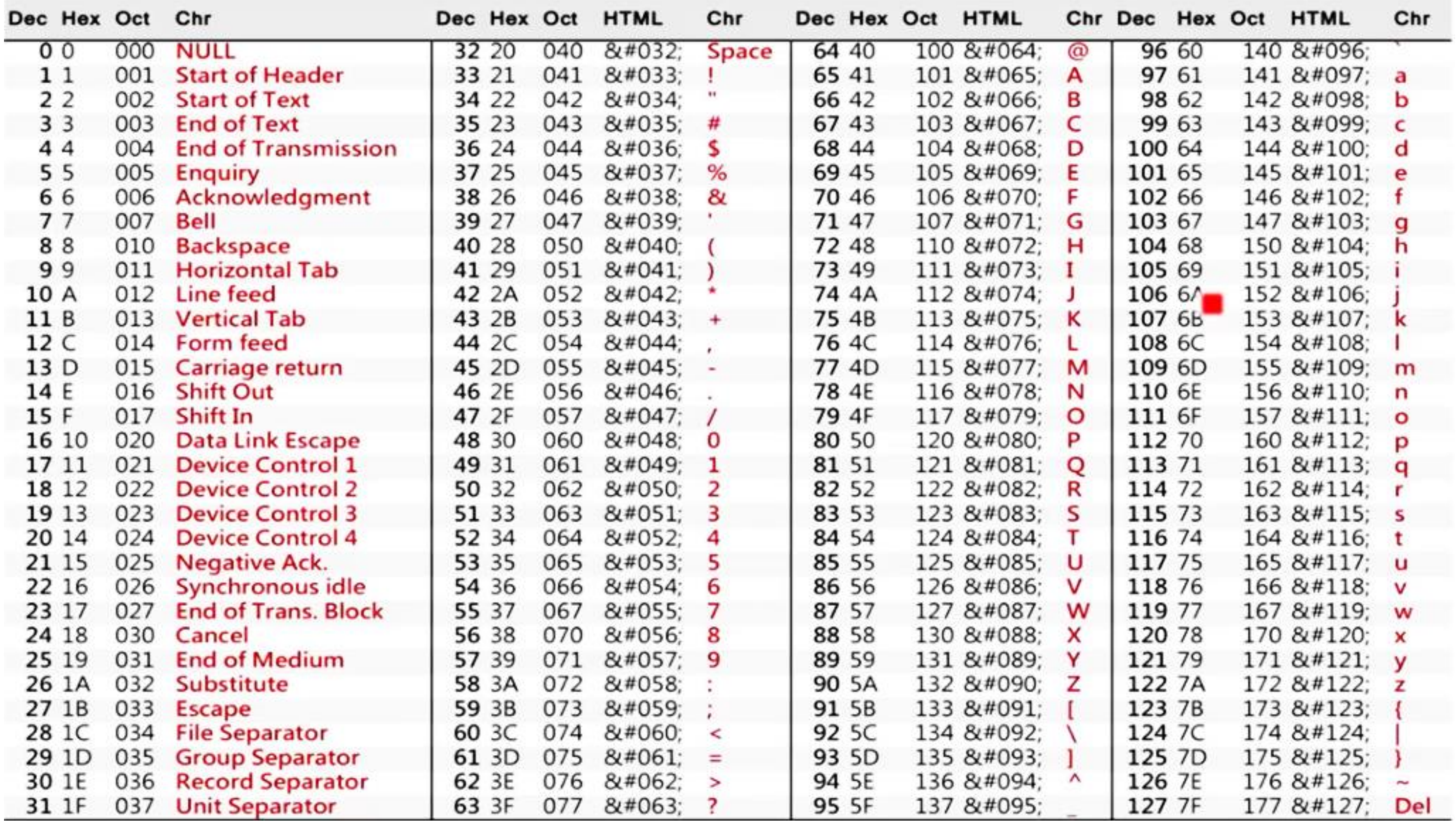
Drawbacks

- Are we using space optimally?



Char	Dec	Binary	Char	Dec	Binary	Char	Dec	Binary
!	033	00100001	A	065	01000001	a	097	01100001
"	034	00100010	B	066	01000010	b	098	01100010
#	035	00100011	C	067	01000011	c	099	01100011
$	036	00100100	D	068	01000100	d	100	01100100
%	037	00100101	E	069	01000101	e	101	01100101
&	038	00100110	F	070	01000110	f	102	01100110
'	039	00100111	G	071	01000111	g	103	01100111
(	040	00101000	H	072	01001000	h	104	01101000
)	041	00101001	I	073	01001001	i	105	01101001
*	042	00101010	J	074	01001010	j	106	01101010
+	043	00101011	K	075	01001011	k	107	01101011
,	044	00101100	L	076	01001100	l	108	01101100
-	045	00101101	M	077	01001101	m	109	01101101
.	046	00101110	N	078	01001110	n	110	01101110
/	047	00101111	O	079	01001111	o	111	01101111
0	048	00110000	P	080	01010000	p	112	01110000
1	049	00110001	Q	081	01010001	q	113	01110001
2	050	00110010	R	082	01010010	r	114	01110010
3	051	00110011	S	083	01010011	s	115	01110011
4	052	00110100	T	084	01010100	t	116	01110100
5	053	00110101	U	085	01010101	u	117	01110101
6	054	00110110	V	086	01010110	v	118	01110110
7	055	00110111	W	087	01010111	w	119	01110111

ASCII Table



Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

Variable-Length Coding

We represent different characters using different numbers of bits

Example:

a=0, b=1, c=00, d=01, e=10, f=11

Decode the following binary string: 011

0,11 → af   0,1,1 → abb   01,1 → db   **Decodability Issue**

Prefix Property

Prefix code: code that can be deciphered character by character by reading a prefix of the input binary string

Prefix-free code: no code is prefix of any other
can be visualized as a **binary tree** with the encoded characters at the leaves

Which of the following codes are prefix free?

Char\Code	**Code 1**	**Code 2**	**Code 3**	**Code 4**
A	0	0	1	1
B	100	1	01	11
C	10	00	001	10
D	11	11	0001	01

Huffman Coding

- The basic idea
  - Instead of storing each character in a file as an 8-bit ASCII value, we will instead store the more frequently occurring characters using fewer bits and less frequently occurring characters using more bits
  - On average this should decrease the file size (usually ½)
- Huffman codes can be used to compress information
  - Like WinZip – although WinZip doesn't use the Huffman algorithm
  - JPEGs do use Huffman as part of their compression process

Huffman Coding

- Huffman coding is a lossless data compression algorithm.
- In this algorithm, a variable-length code is assigned to input different characters.
- The code length is related to how frequently characters are used.
- Most frequent characters have the smallest codes and longer codes for least frequent characters.
- There are mainly two parts.
  1. First one to create a Huffman tree
  2. To traverse the tree to find codes.
- <u>**Example**</u>
  - Consider some strings “YYYZXXYYX”, the frequency of character Y is larger than X and the character Z has the least frequency.
  - So, the length of the code for Y is smaller than X, and code for X will be smaller than Z.
- Complexity for assigning the code for each character according to their frequency is O(n log n)

How to decode?

- At first it is not obvious how decoding will happen, but this is possible if we use prefix codes

Data Compression: A Smaller Example

Input:

A string with different characters, say “ACCEBFFFFAAXXBLKE”

Output:

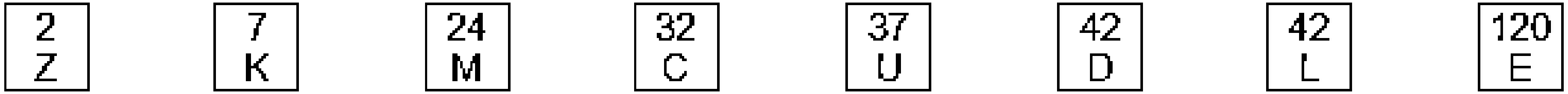
Data	Frequency	Code
K	1	0000
L	1	0001
E	2	001
F	4	01
B	2	100
C	2	101
X	2	110
A	3	111

Huffman Coding Example

Letter	C	D	E	K	L	M	U	Z
Frequency	32	42	120	7	42	24	37	2

- STEP 1:

  Sort Letters according to the frequency

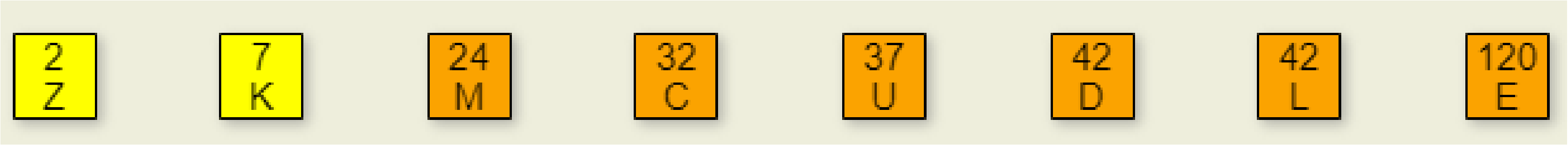


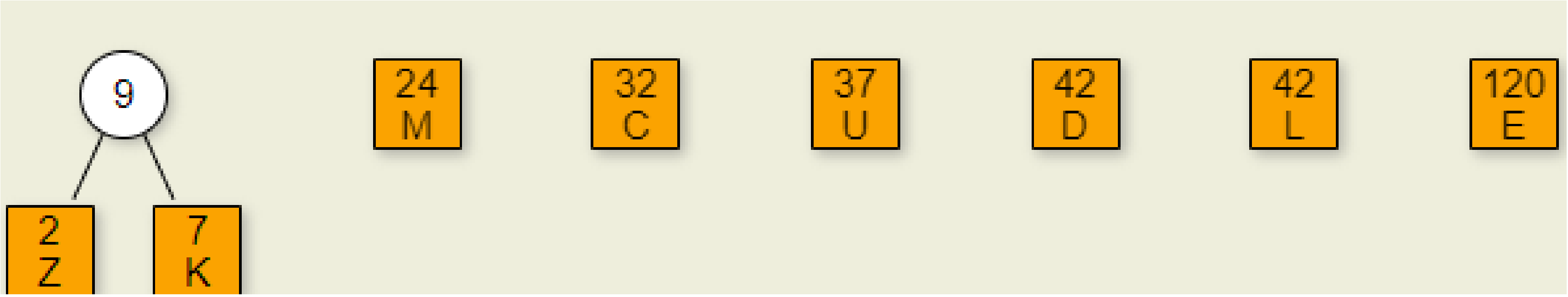
2 Z 7 K 24 M 32 C 37 U 42 D 42 L 120 E

Huffman Coding Example [Continue...]

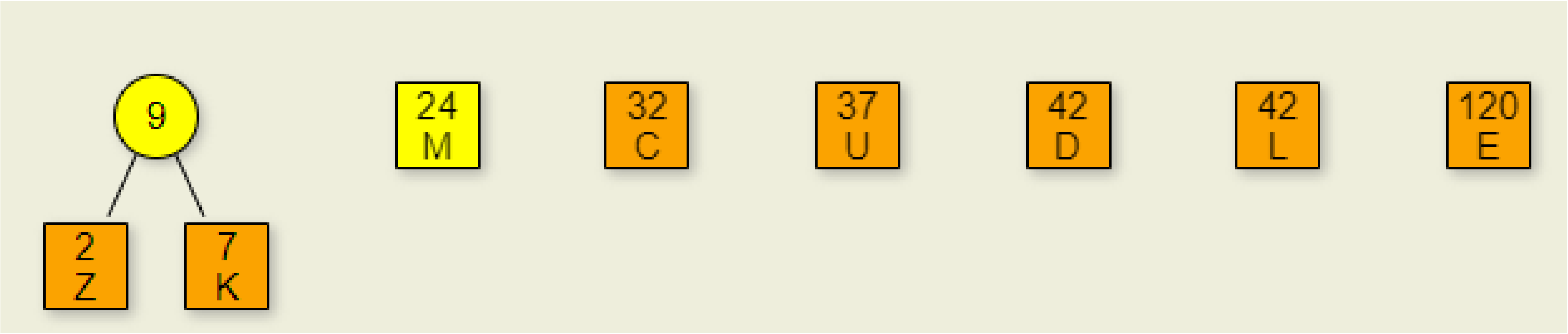
- STEP 2:

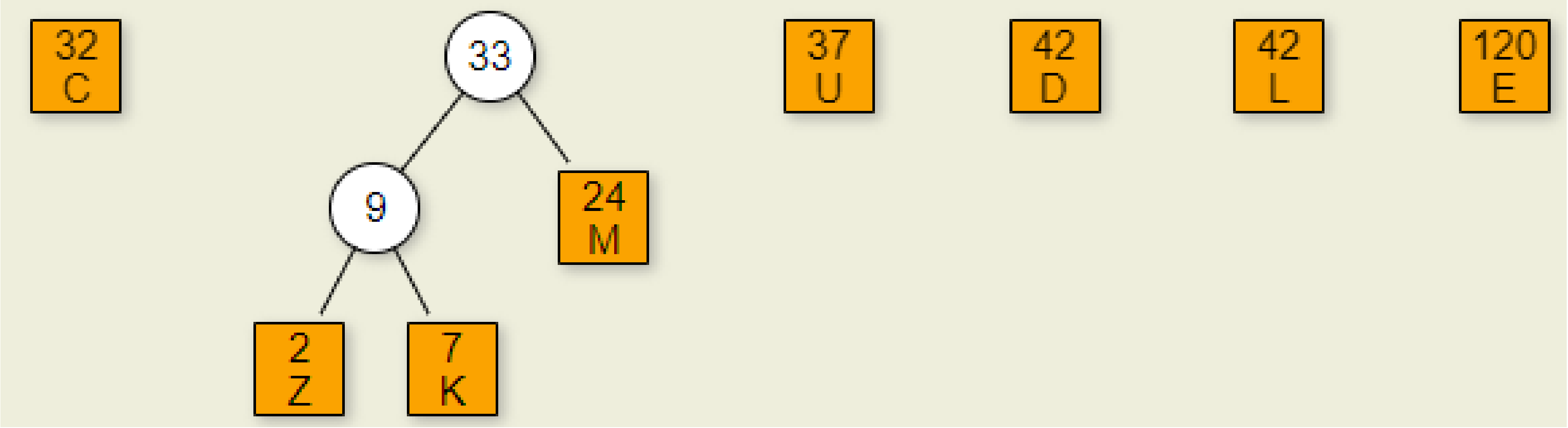
  Merge 2 lowest frequency elements



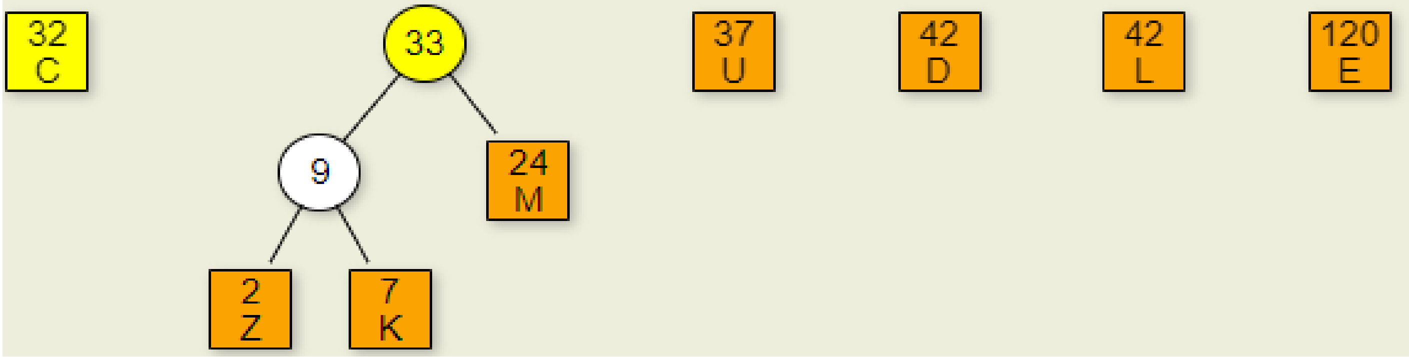


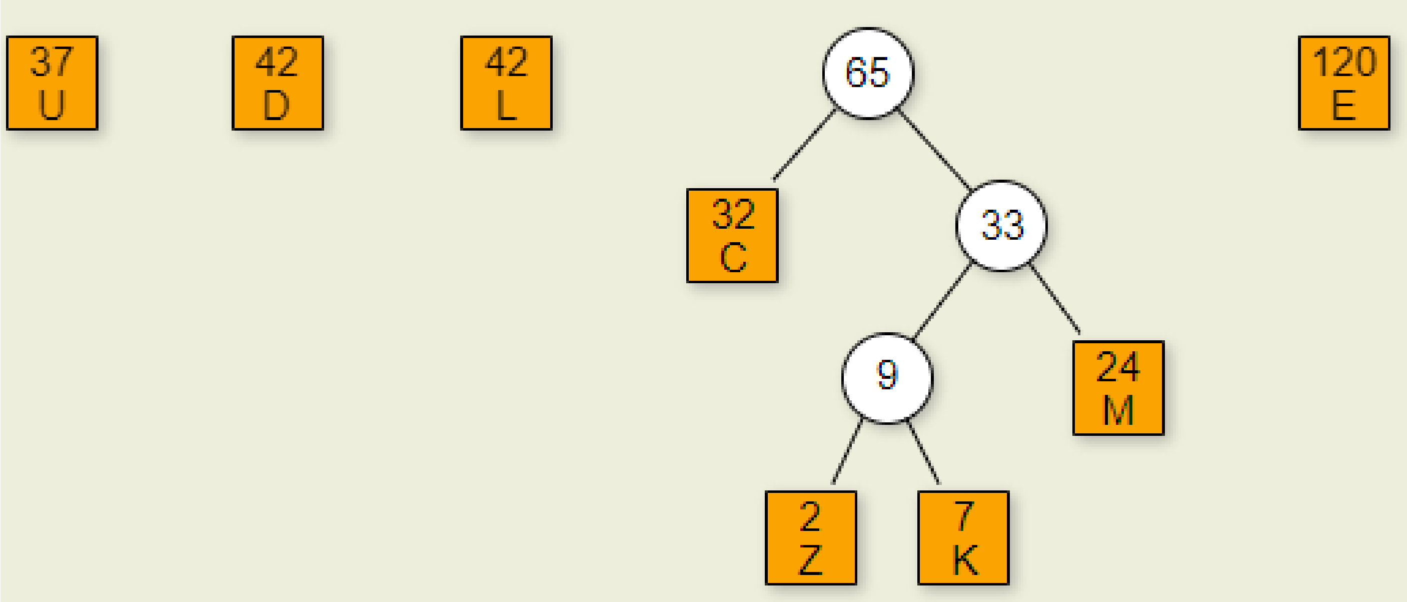
Huffman Coding Example [Continue...]



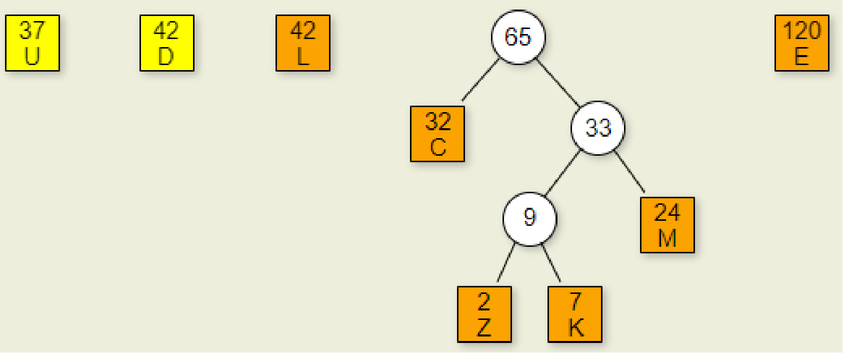


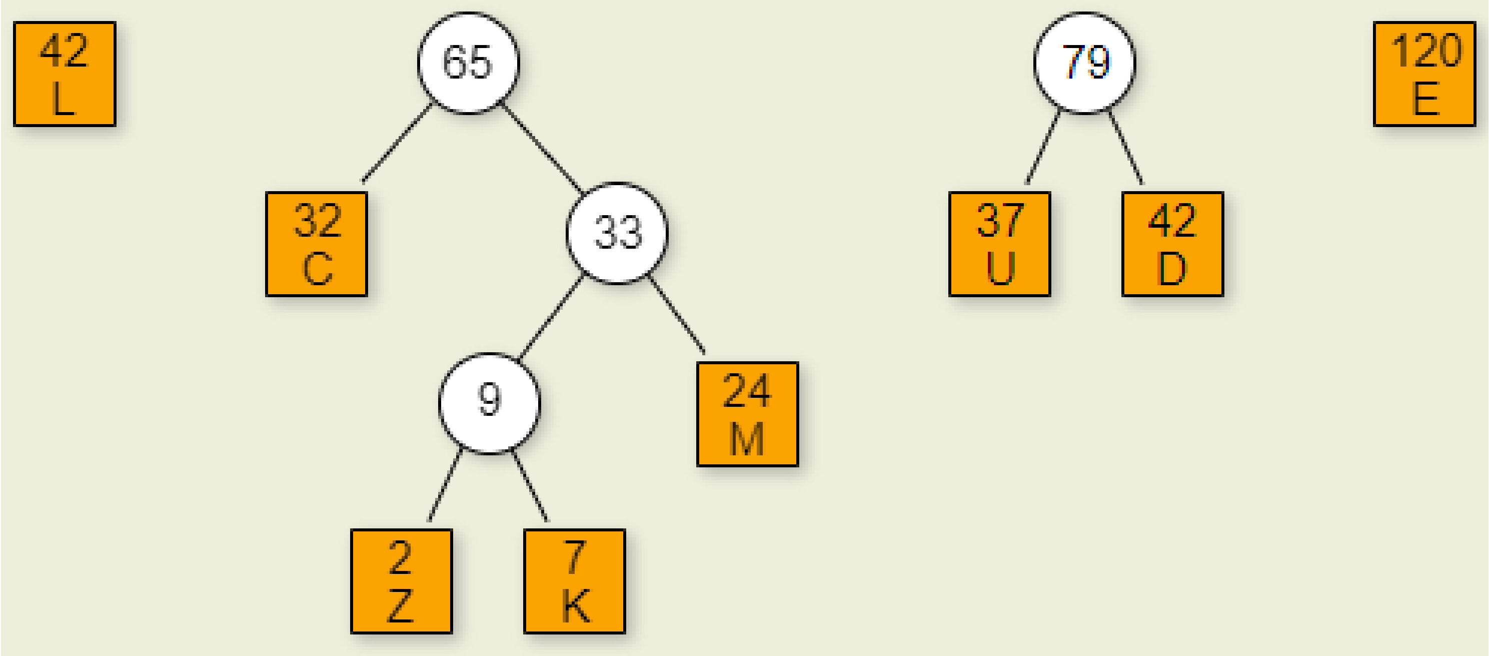
Huffman Coding Example [Continue...]



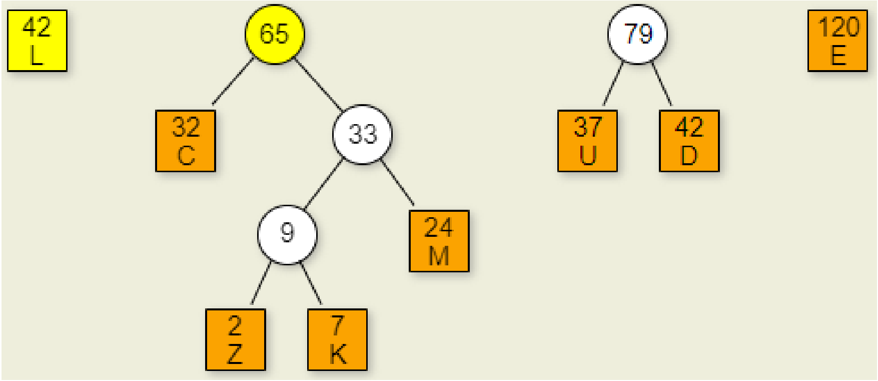


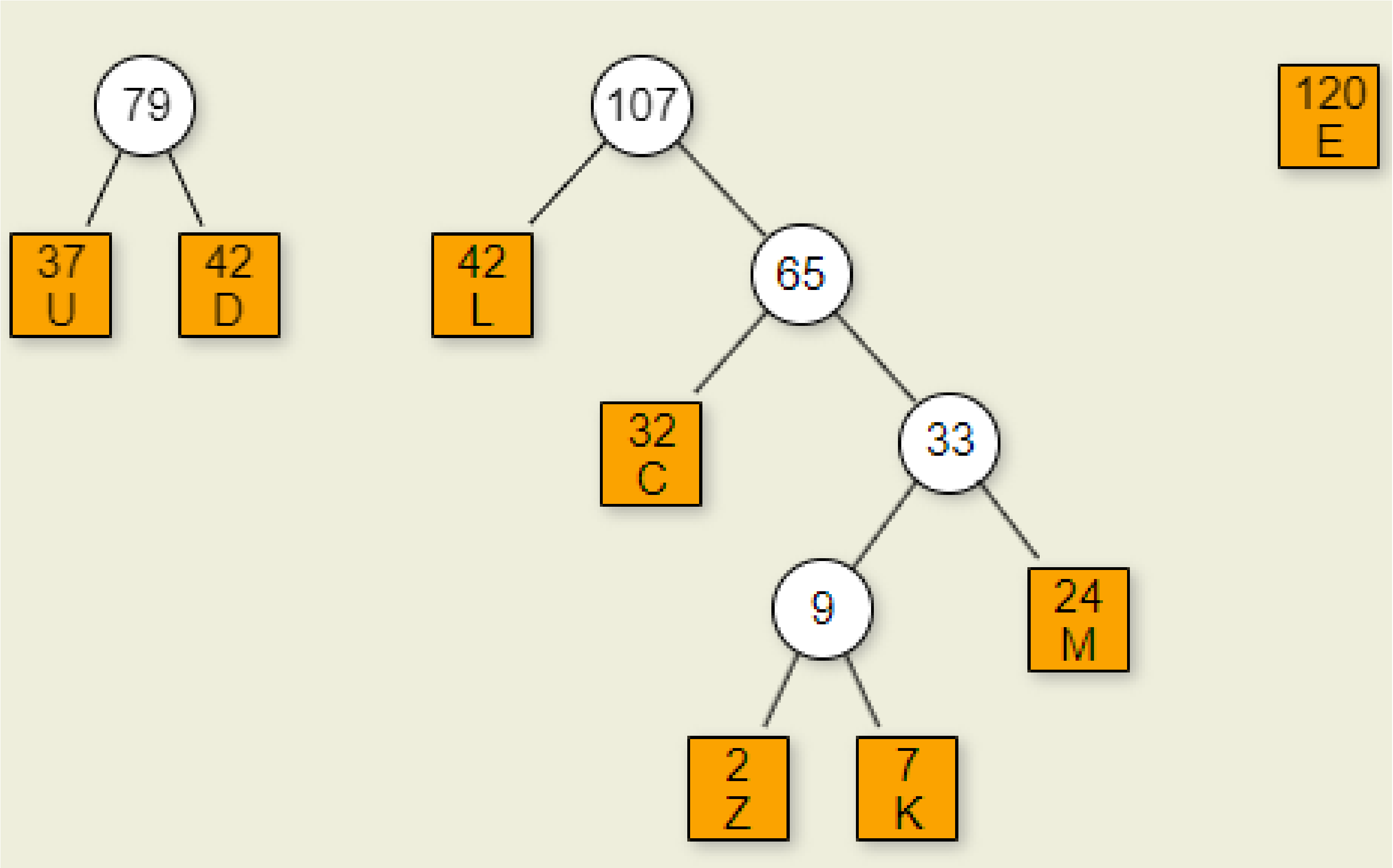
Huffman Coding Example [Continue...]



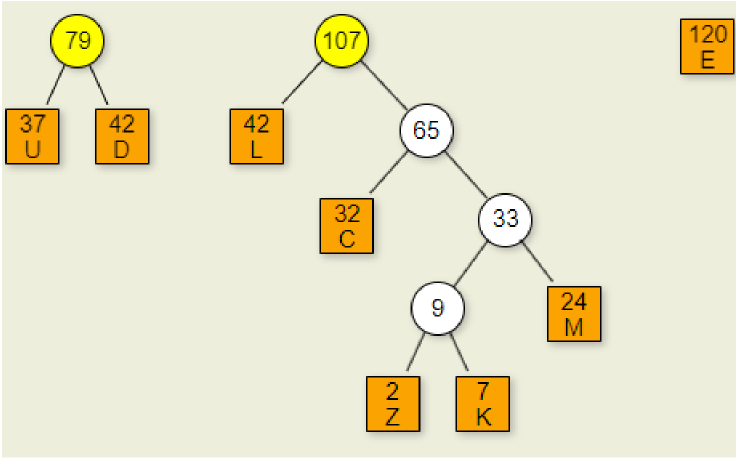


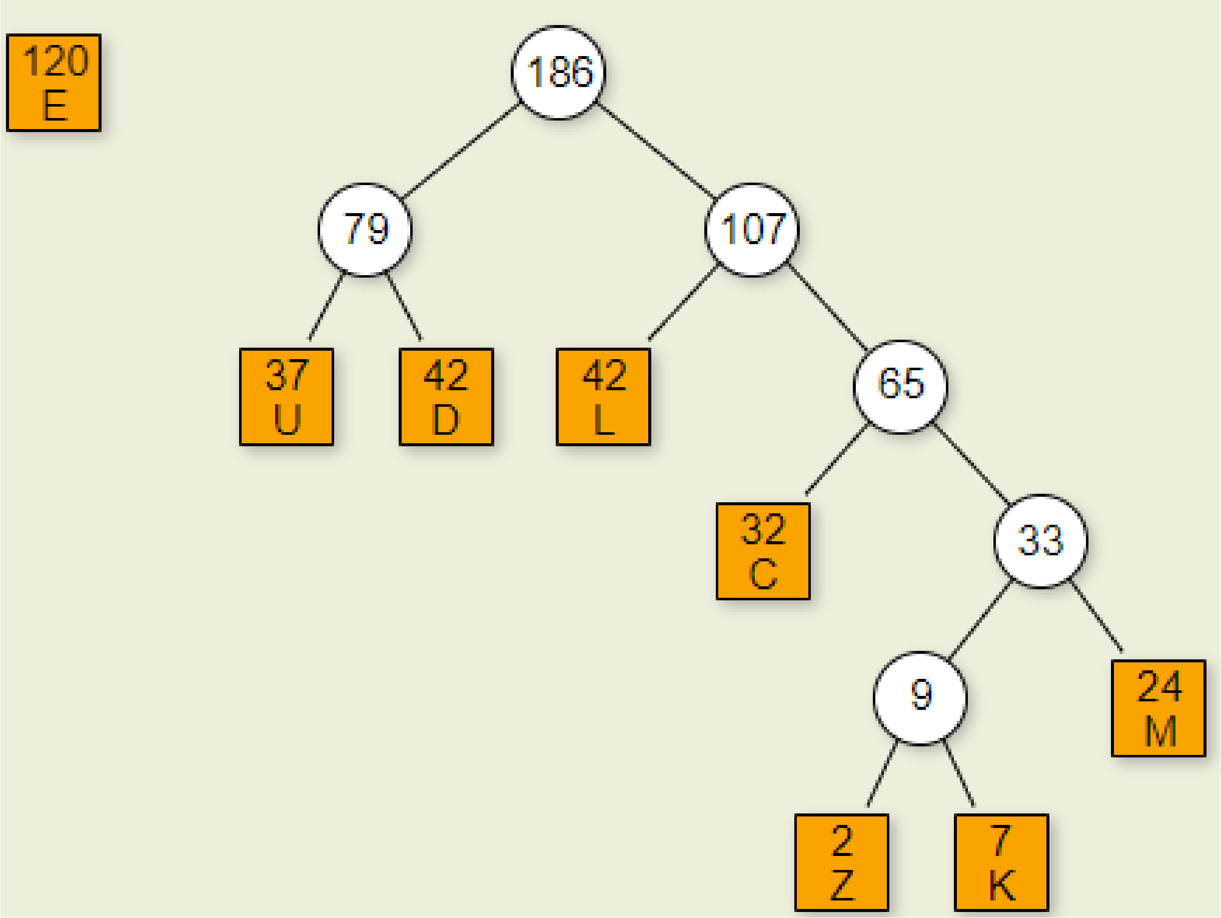
Huffman Coding Example [Continue…]



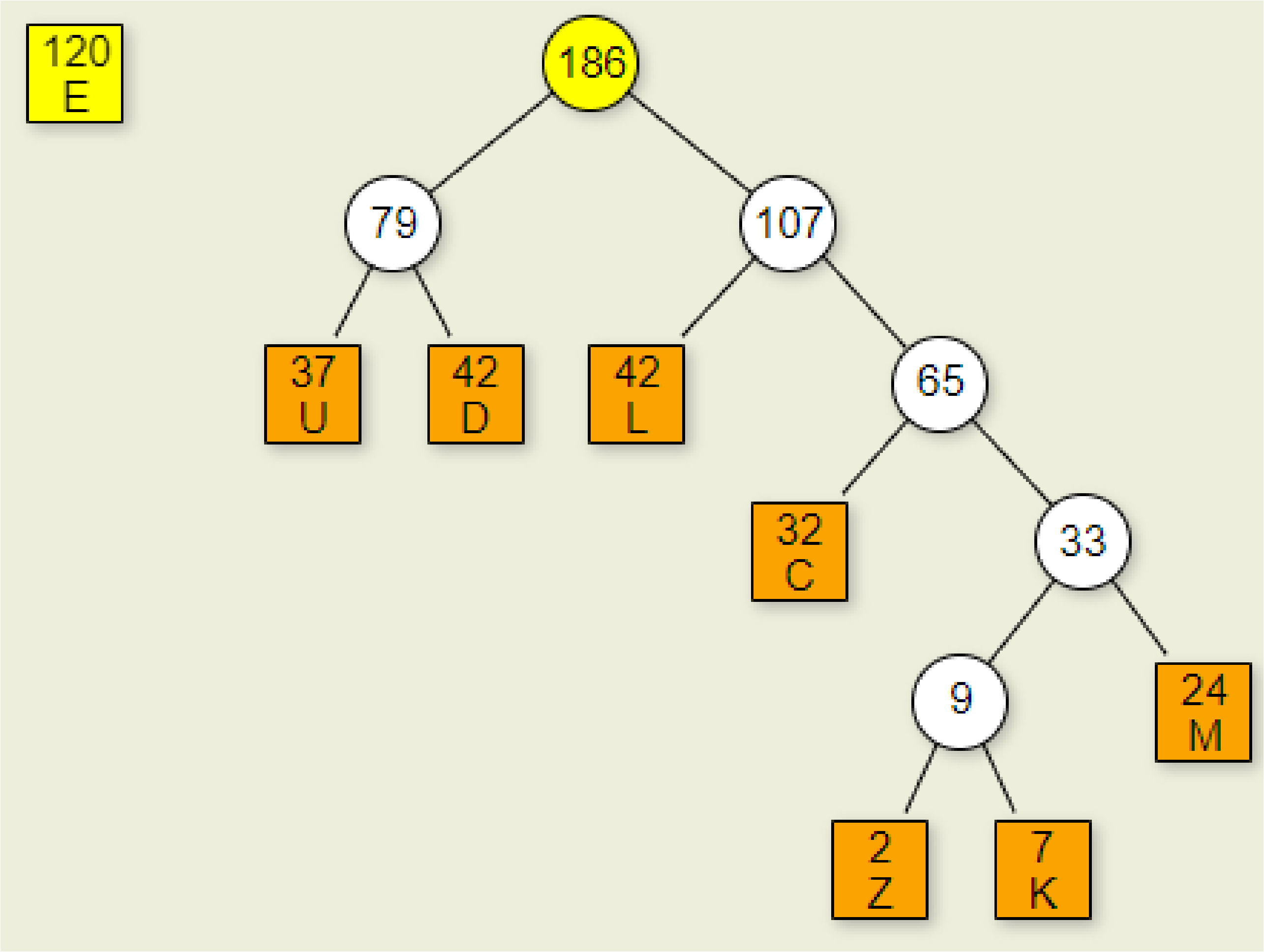


Huffman Coding Example [Continue...]

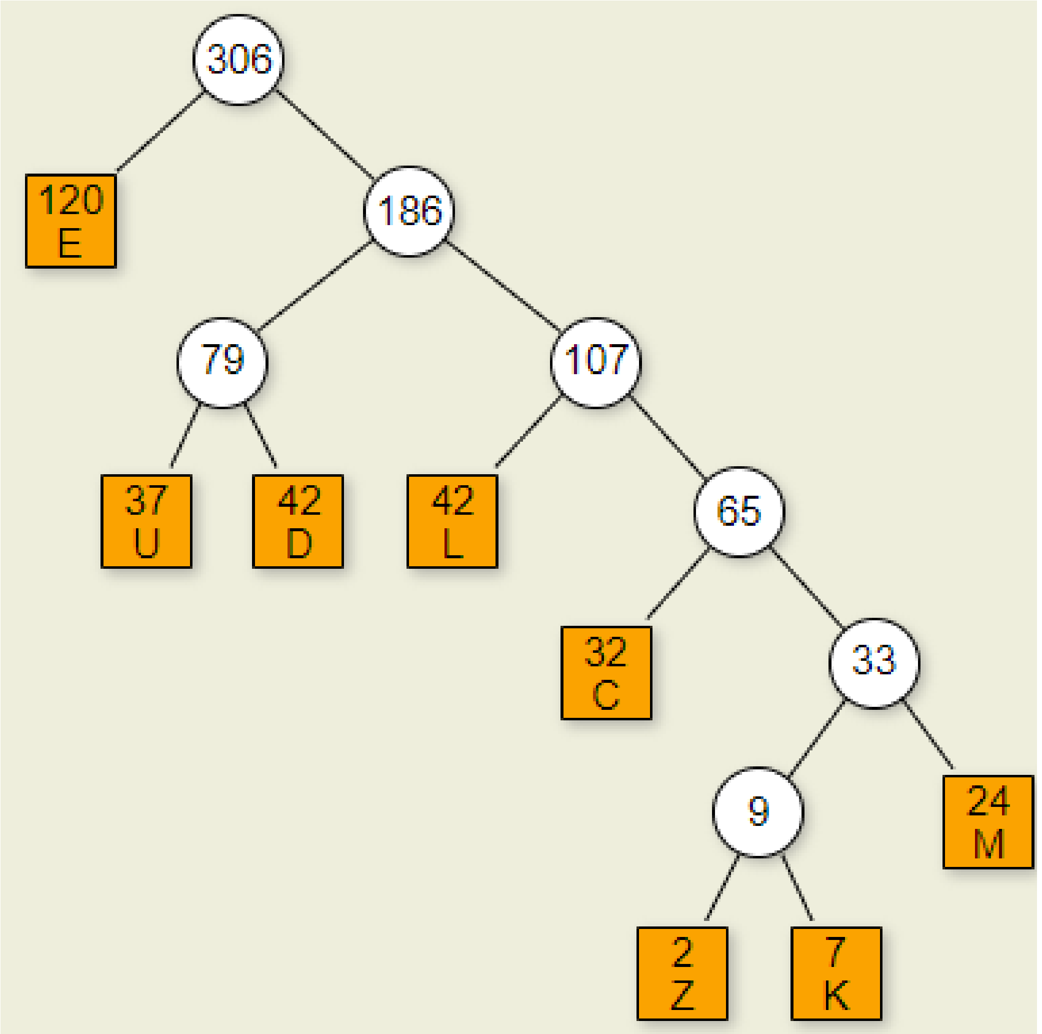




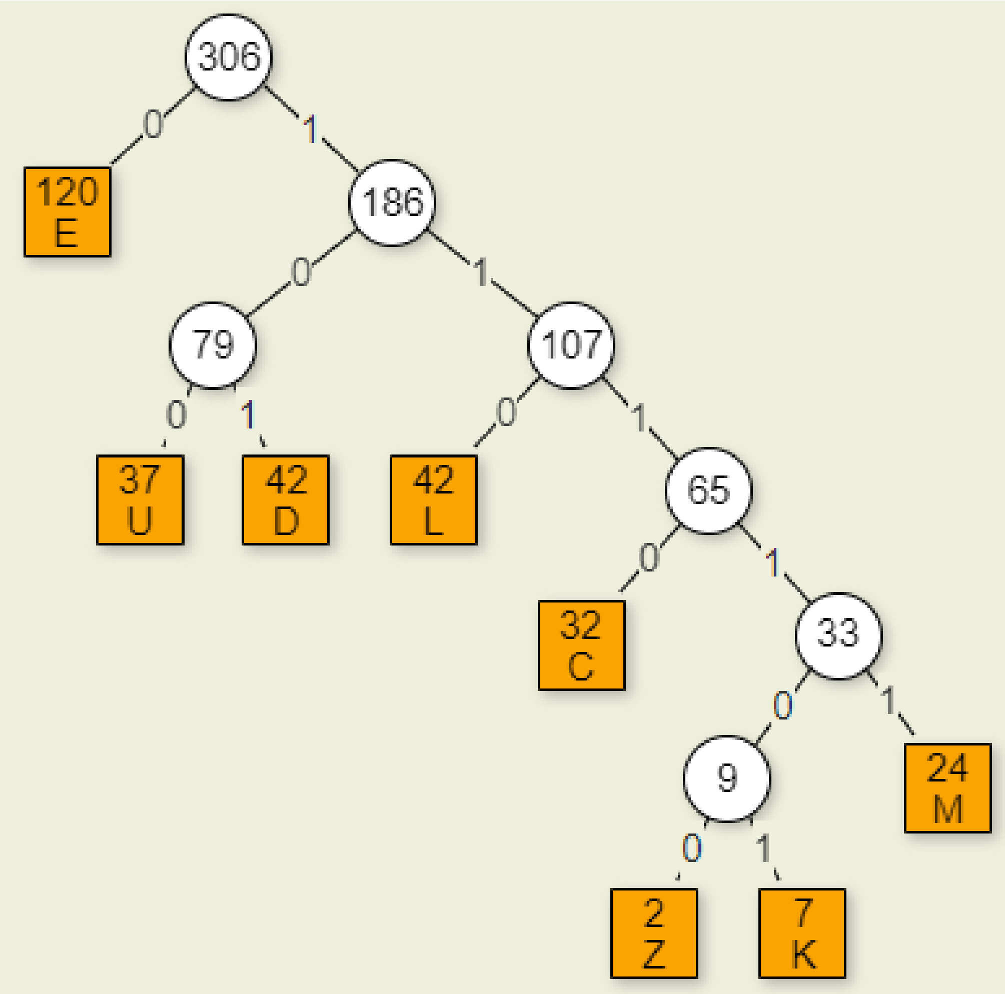
Huffman Coding Example [Continue...]



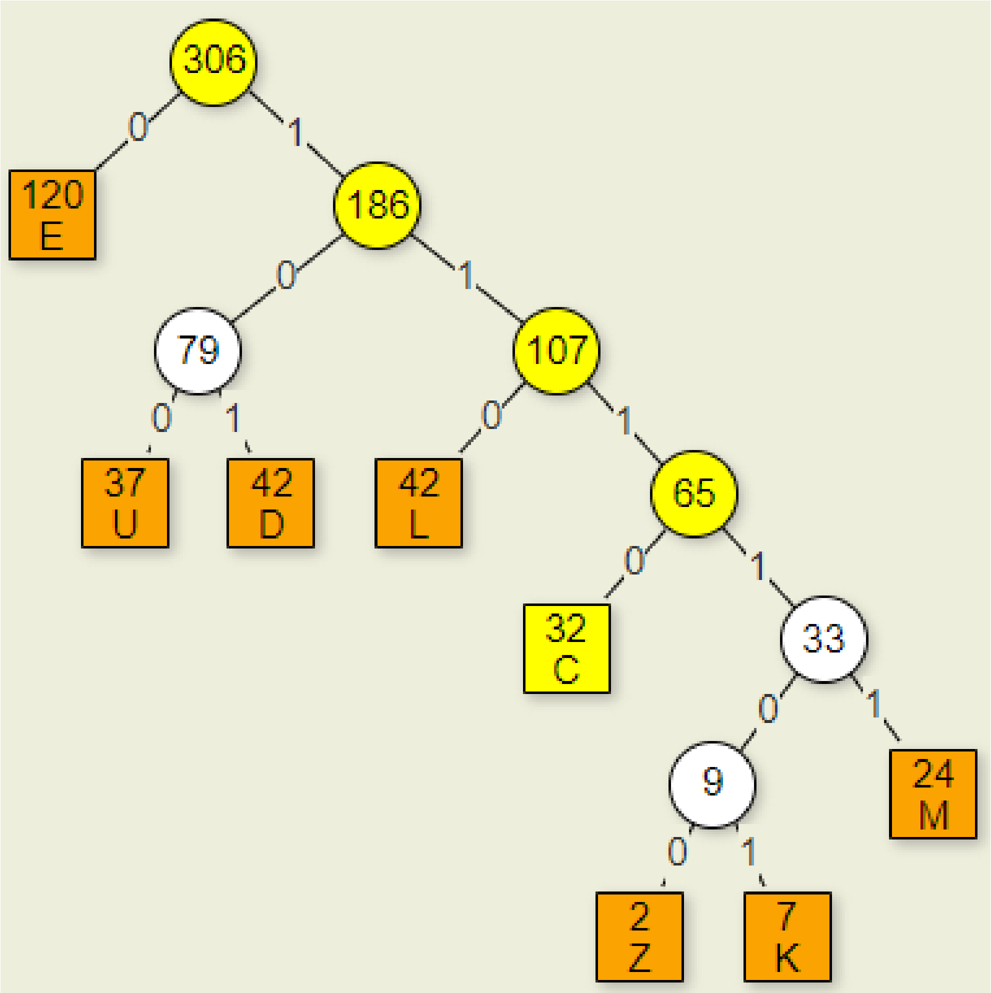
Huffman Coding Example [Final Tree]



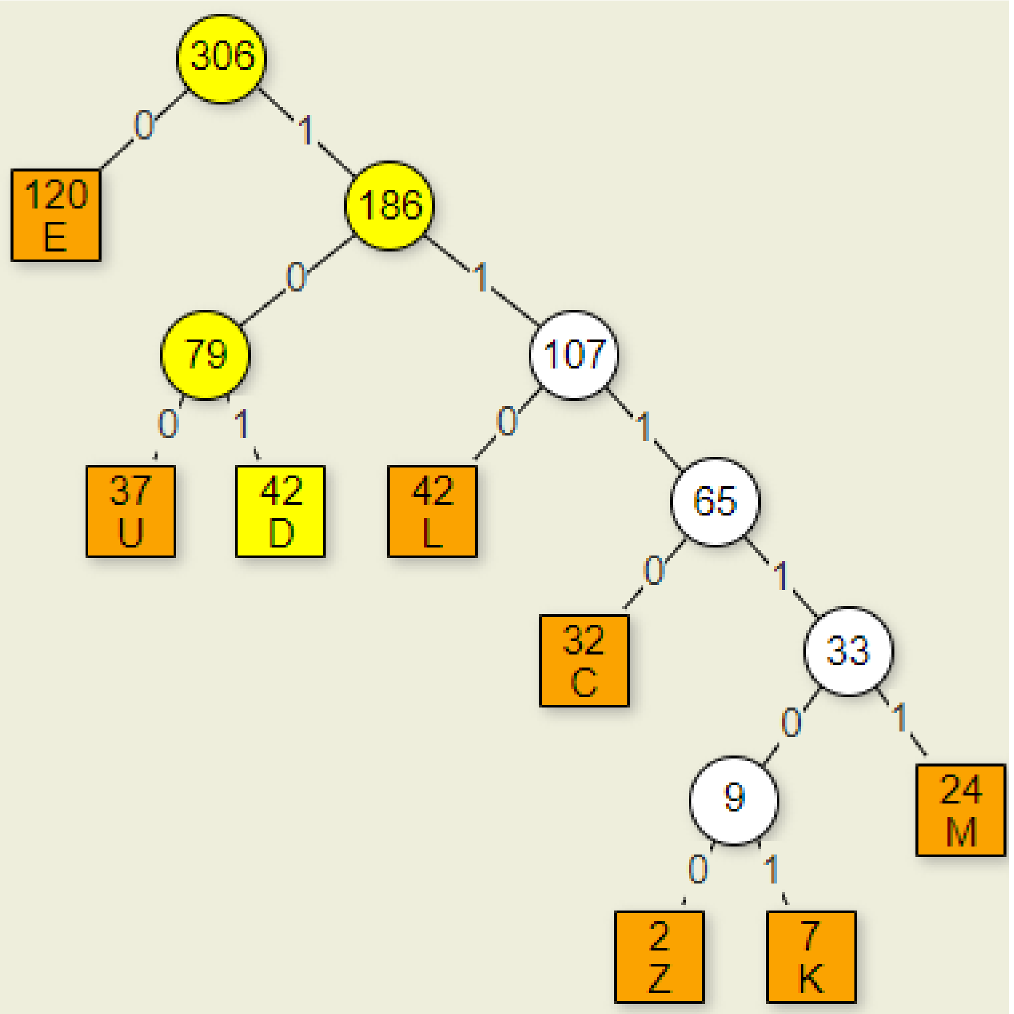
Huffman Assigning Code



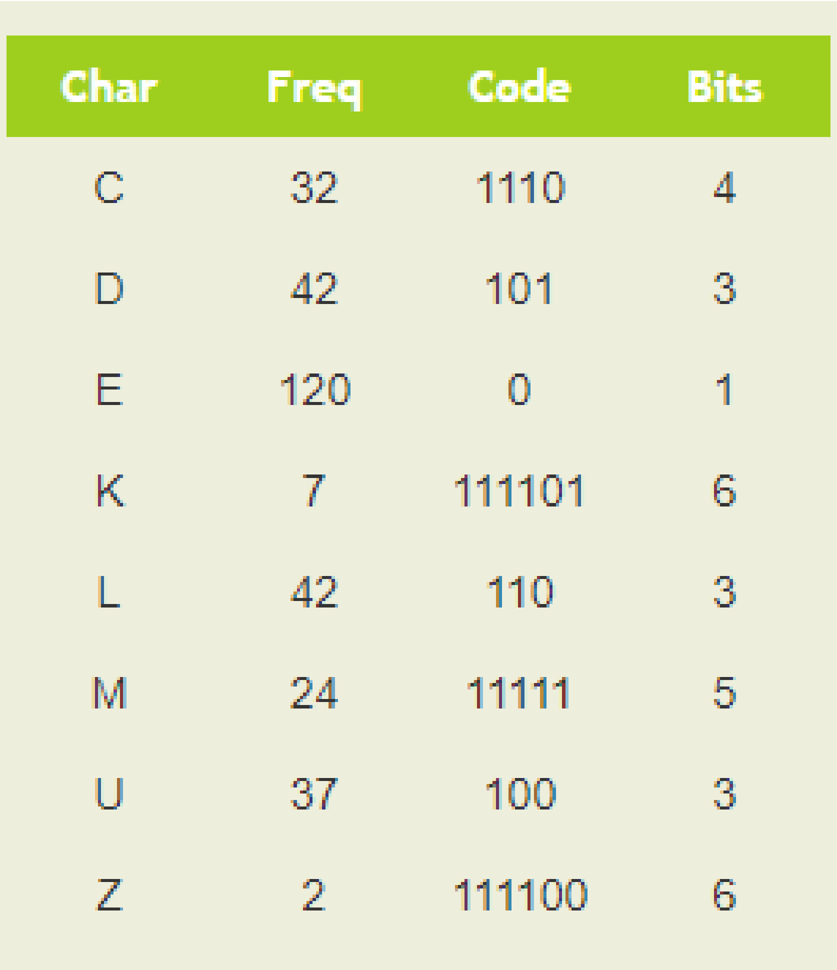
Huffman Assigning Code [Code for C]



Huffman Assigning Code [Code for D]



Huffman Assigning Code [Final]



Char	Freq	Code	Bits
C	32	1110	4
D	42	101	3
E	120	0	1
K	7	111101	6
L	42	110	3
M	24	11111	5
U	37	100	3
Z	2	111100	6

Huffman Coding –Example 2

Huffman Coding: Tree Building

1. **Put all the nodes in a priority queue by frequency**

Huffman Order

S	O	B	Y	C	A
1	1	1	1	2	2

Shannon-Fano Order

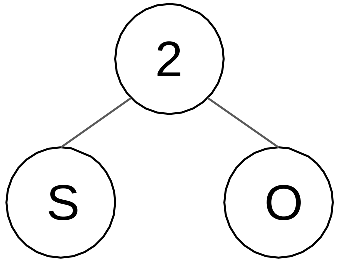
C	A	S	O	B	Y
2	2	1	1	1	1

Casco Bay

Huffman Coding: Tree Building (Casco Bay)

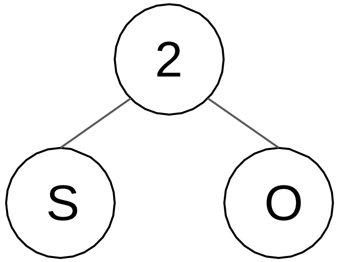
1. Put all the nodes in a priority queue by frequency
2. **While there is more than one node in the queue**:
   a. Dequeue the first two nodes
   b. Create a new node with the sum of the frequencies
   c. Reinsert the new node in the priority queue

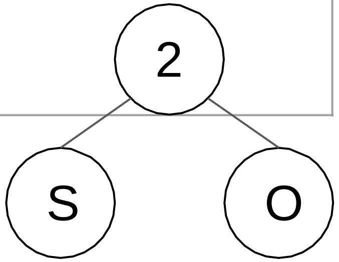
S	O	B	Y	C	A
1	1	1	1	2	2



Huffman Coding: Tree Building (Casco Bay)

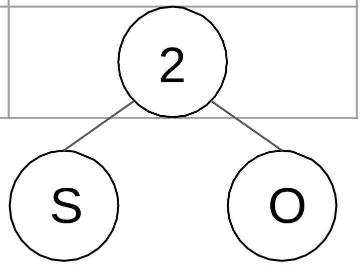
S	O	B	Y	C	A
1	1	1	1	2	2

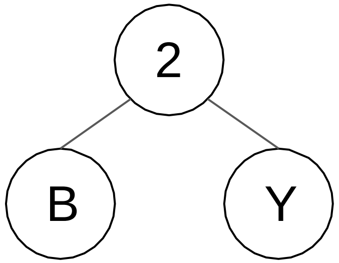


B	Y	C	A	**SO**
1	1	2	2	

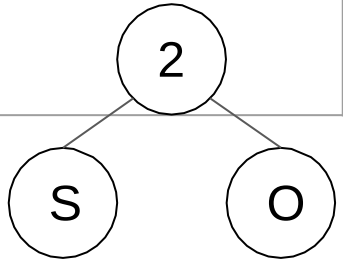
Huffman Coding: Tree Building (Casco Bay)

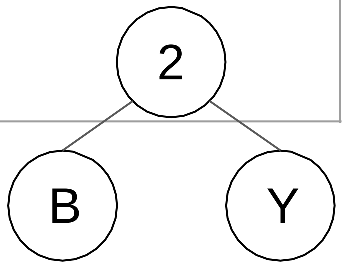
B	Y	C	A	SO
1	1	2	2	2



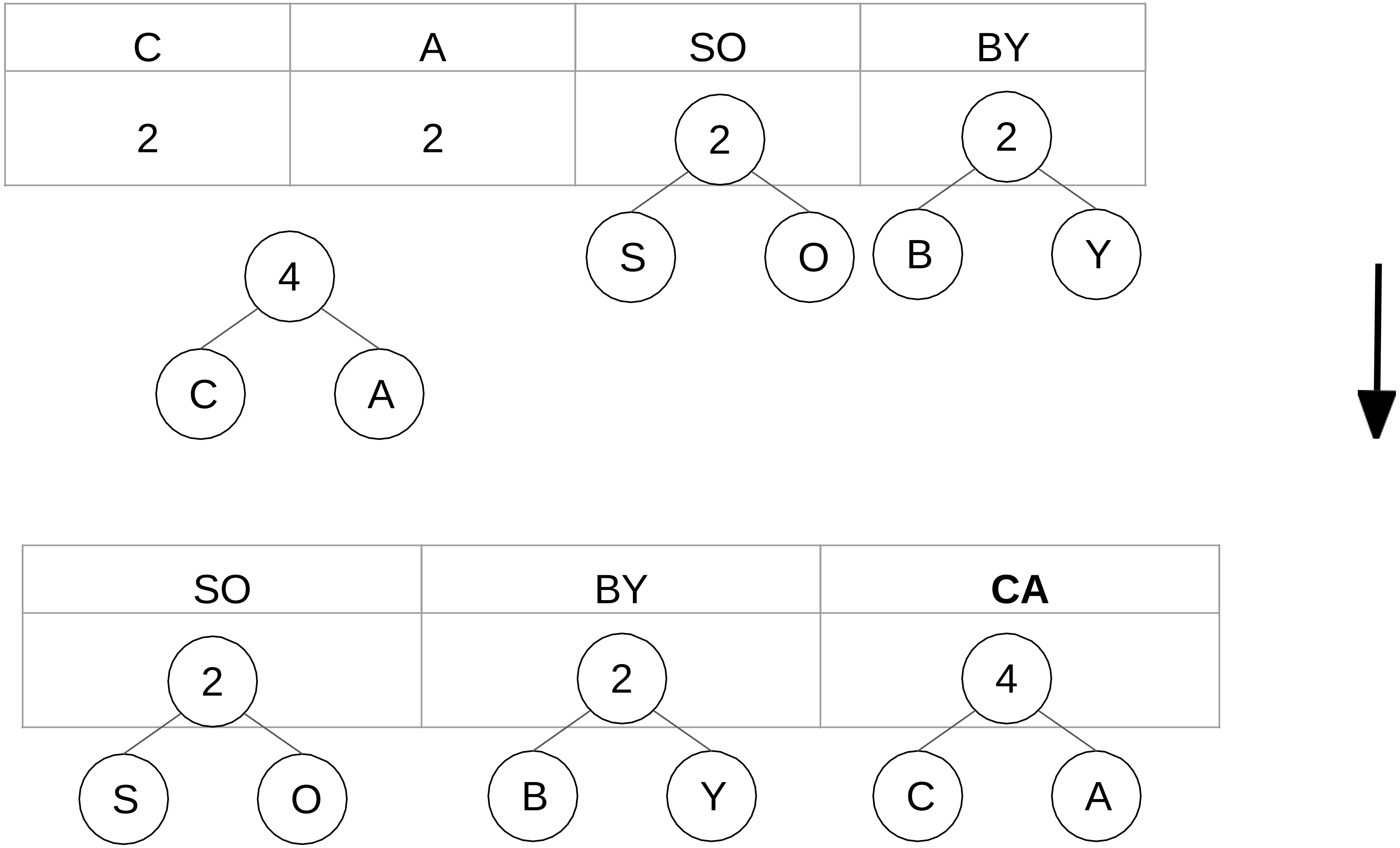


C	A	SO	**BY**
2	2	2	2

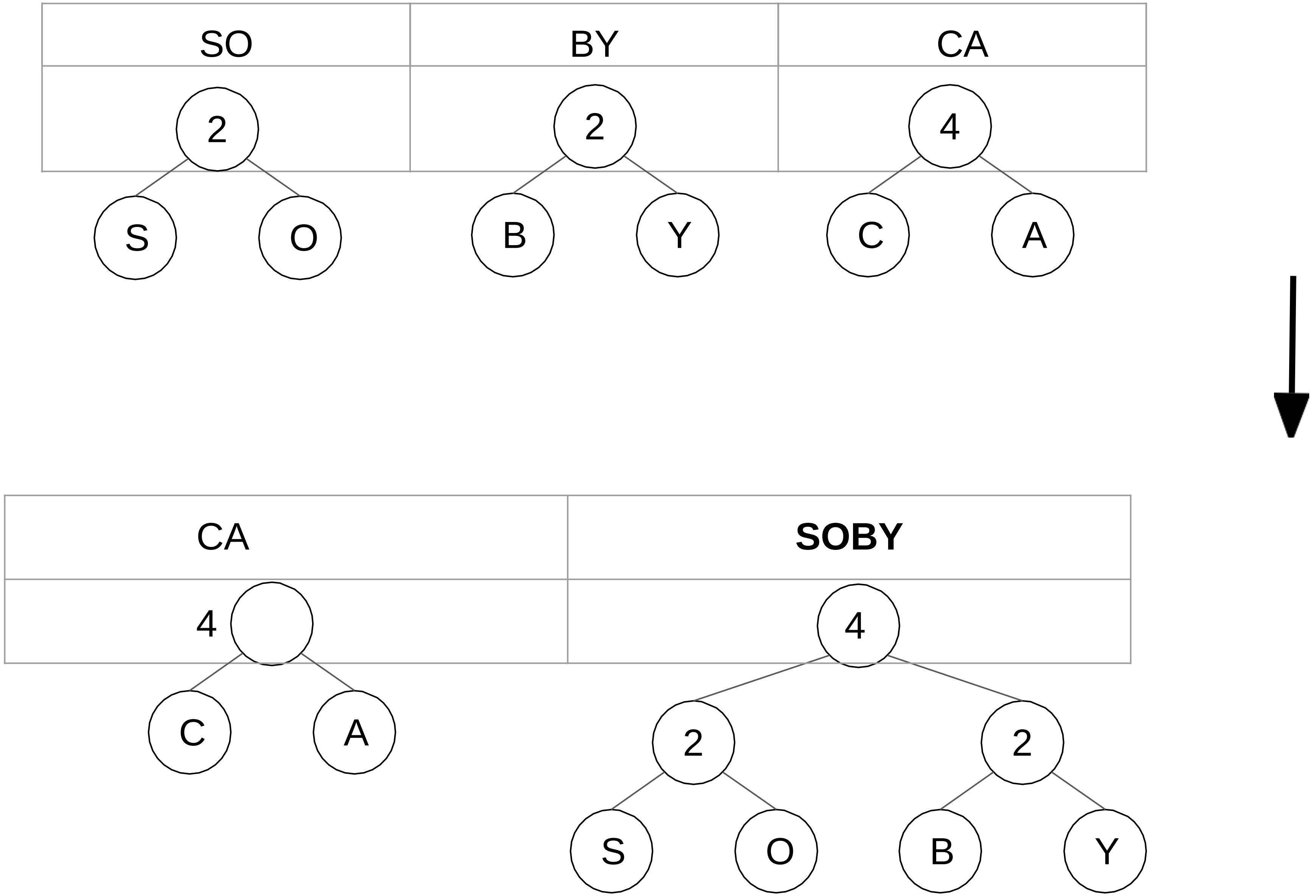




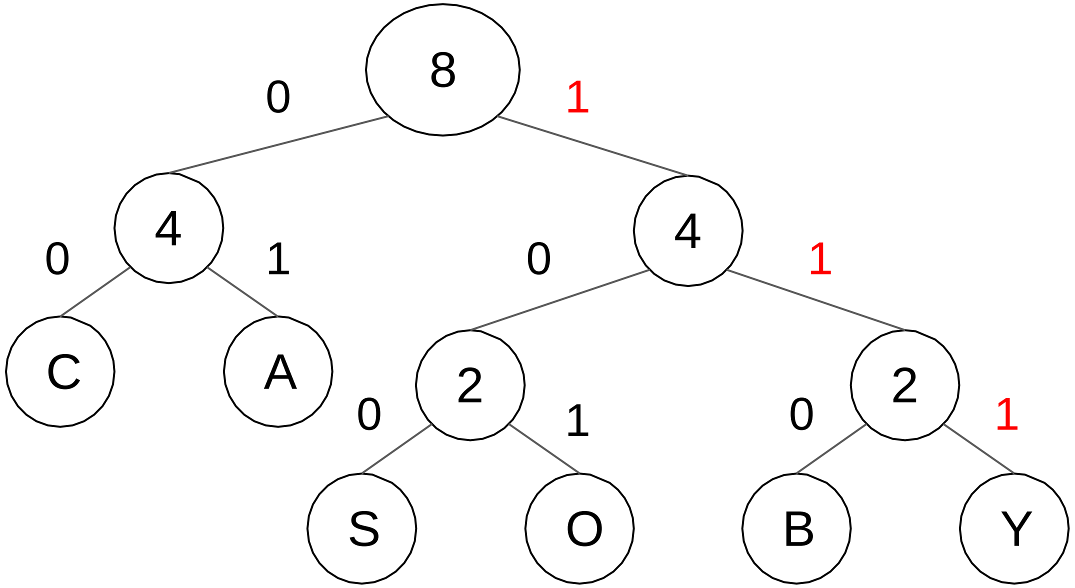
Huffman Coding: Tree Building (Casco Bay)



Huffman Coding: Tree Building (Casco Bay)



Huffman Coding: Tree Building (Casco Bay)



C	A	S	O	B	Y
00	01	100	101	110	111