CSIT113
Problem Solving

Workshop - Week 13

An Optimal Code

- Let’s say we want to encode a specific message.
- Now, what is the best code to use?
- There are several possible interpretations of the word “best”.
  - Hardest to decode.
  - Most reliable to send.
  - Shortest total length.
- In this case let us define best as shortest total length

An Optimal Code

- We can establish an upper bound on this by considering the following questions…
  - How many characters does the message contain?
  - How many different characters does the message contain?
- With the answers to these we can arrive at a good bound as follows…

An Upper Bound

- Let us say we have a message of *n* characters with *m* distinct characters.
- We can represent this in *n* x *k* bits where *k* is the smallest number of bits needed to represent *m* distinct values.
- $k = \lceil \log_2(m) \rceil$
- Can we do better than this?

First, An Example

- A useful example to consider is the representation of genetic codes.
- Genes are made up of sequences for four bases.
  - Alanine A
  - Cytosine C
  - Guanine G
  - Thymine T

Genetic Codes

- With 4 distinct characters we can use 2-bit codes to represent them.
  - $Log_2(4) = 2$
- This means that a 1120-base genetic sequence can be represented in 2240 bits using a coding scheme such as…

A	C	G	T
00	01	10	11

- Thus ACCGATCCATTA… would encode as 000101100011010100111100…

Another Example

- Take the first chapter of Alice's Adventures in Wonderland by Lewis Carroll and count…
  - The total number of characters – n.
  - The number of different characters – m.
  - Ignoring punctuation.
- The results were as follows:
  - $n = 10734$
  - $m = 27$

How Big Is Alice?

- If we calculate *k*…
  - $k = \lceil \mathrm{Log}_2(27) \rceil = 5$
- We find that we can encode the first chapter of the book in 53,670 bits.
- Is this the best possible result we can get?

Genetic Codes Again

- Consider the problem of coding genetic information in a bit more detail.
- Say that, although there are 4 codes (A, C, G, T) they do not occur with equal frequency.
- For example, let us assume that A is 10 times more likely than C which is 10 times more likely than G or T which are equally likely.
- We can express the relative frequencies as follows:
    - A : C : G : T = 100 : 10 : 1 : 1

A Better Code?

- Given these frequencies it should be clear that if we can make the code for A shorter – even if it means making other codes longer – we may be able to shorten the overall length of the coded gene sequence.
- Consider the following coding scheme…

A	C	G	T
0	10	110	111

A Better Code.

- With this scheme an A takes just 1 bit, C takes 2 bits as before and G and T each take 3 bits.
- Is this better than the fixed 2-bit per character result?
- From the given frequencies, a 1120 base sequence will have 1000 A’s, 100 C’s 10 G’s and 10 T’s
- This means we can represent the sequence using 1000x1 + 100x2 + 10x3 +10x3 = 1260 bits
- This compares very favourably with the 2240 bits we needed with a fixed length code

Huffman Coding

- Ok, we have a better coding scheme but how did we get it?
- The answer is that we used a technique developed by David Huffman in 1951 while he was a student at MIT.
- His approach started with a list of the individual characters and their counts, sorted by count.

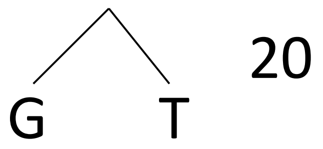
Huffman Coding

- In our genetic example this gives us…
  - A.1000
  - C.100
  - G.10
  - T.10
- We then combine the two bottom entries into a tree with a count equal to the total and re-sort the list.

Huffman Coding

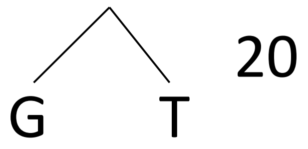
- Thus…
  - A.1000
  - C.100
  - G.10
  - T.10

- Becomes
  - A.1000
  - C.100

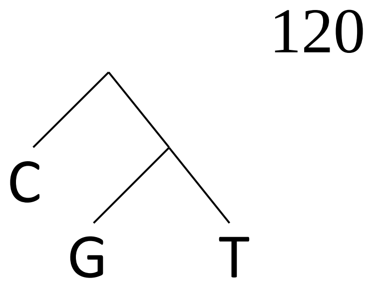


Huffman Coding

- Repeating…
  - A.1000
  - C.100



- Becomes
  - A.1000

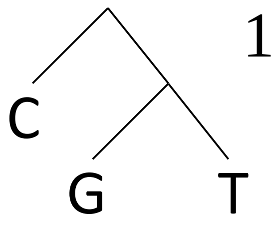


Huffman Coding

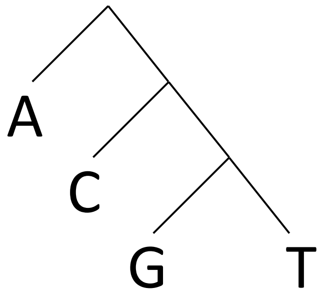
- Finally…

A.1000

120



- Becomes



1120

- If we label left branches with 0 and right branches with 1 we get the codes we saw earlier.

A	C	G	T
0	10	110	111

Strategy

1. Prepare a collection of *n* initial Huffman trees, each of which is a single leaf node. Put the *n* trees onto a **priority queue** organized by weight (frequency).
2. Remove the first two trees (the ones with lowest weight). Join these two trees to create a new tree whose root has the two trees as children, and whose weight is the sum of the weights of the two children trees.
3. Put this new tree into the priority queue.
4. Repeat steps 2-3 until all of the partial Huffman trees have been combined into one.

Another example

- Consider the following letter frequency table

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

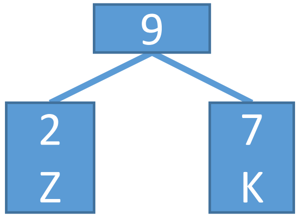
- Construct a Huffman tree

Huffman tree

- We first sort the table

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

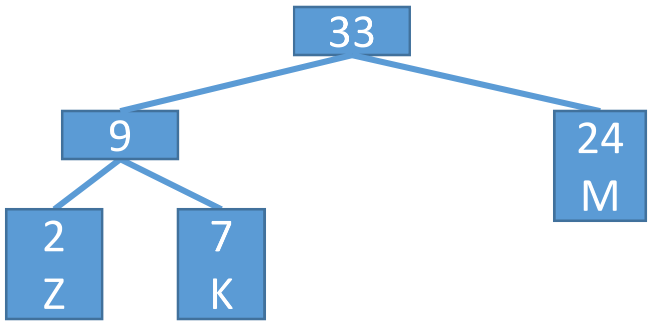
- K and Z will go first



Huffman tree

Letter	E	D	L	U	C	M		
Frequency	120	42	42	37	32	24		

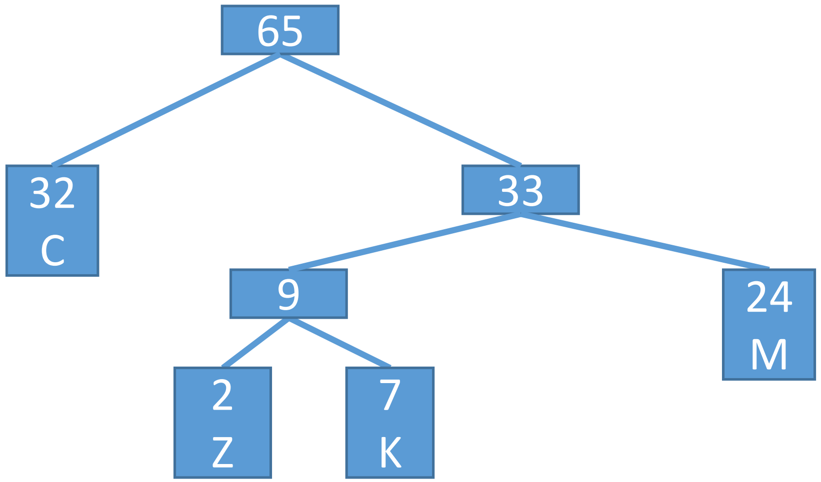
- Next is M



Huffman tree

Letter	E	D	L	U	C			
Frequency	120	42	42	37	32			

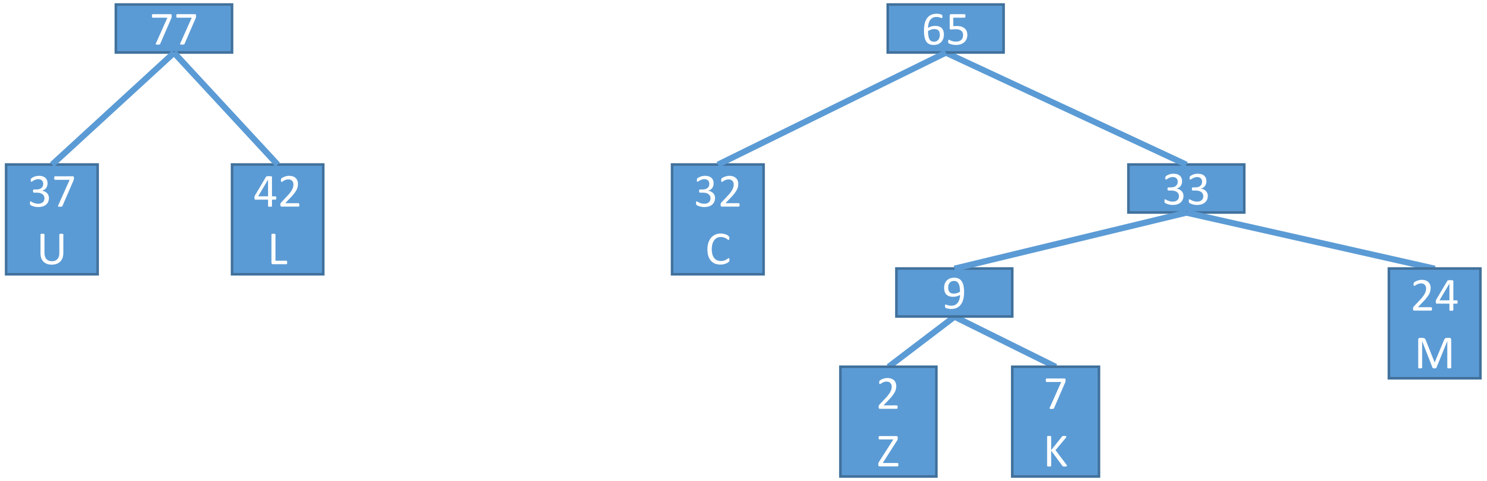
- Next is C



Huffman tree

Letter	E	D	L	U				
Frequency	120	42	42	37				

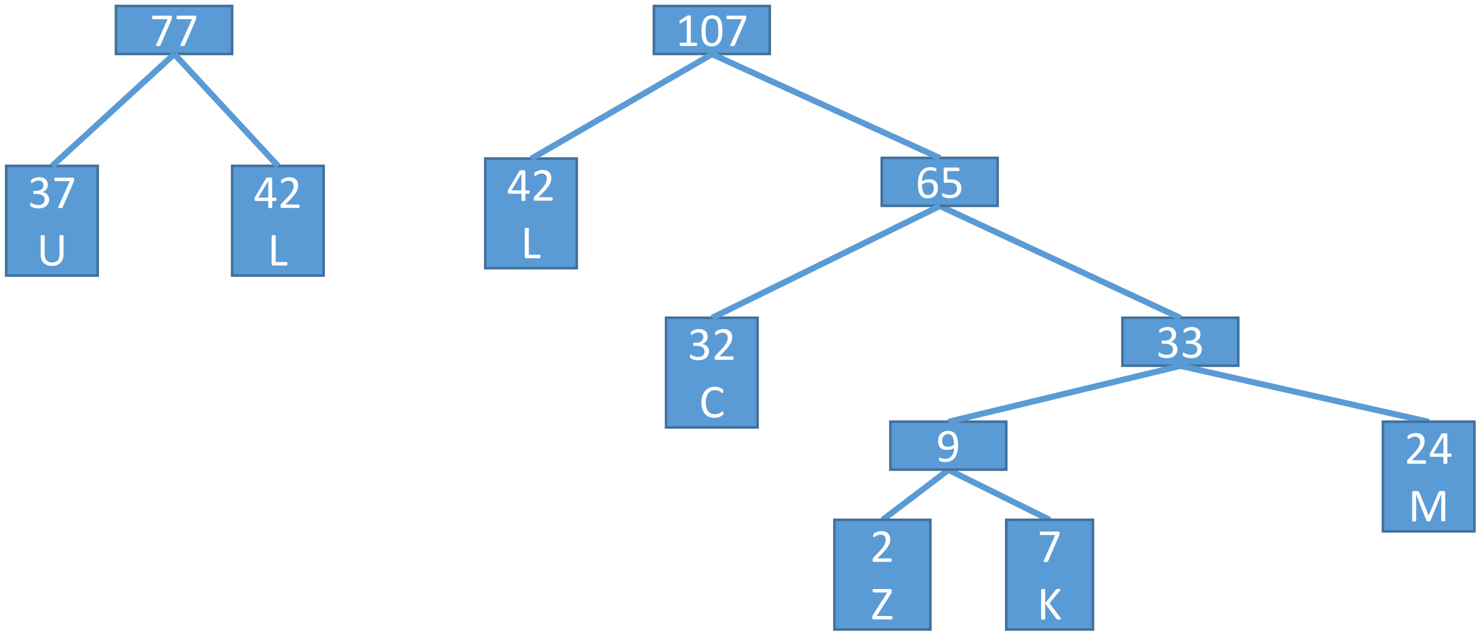
- Next is U, L



Huffman tree

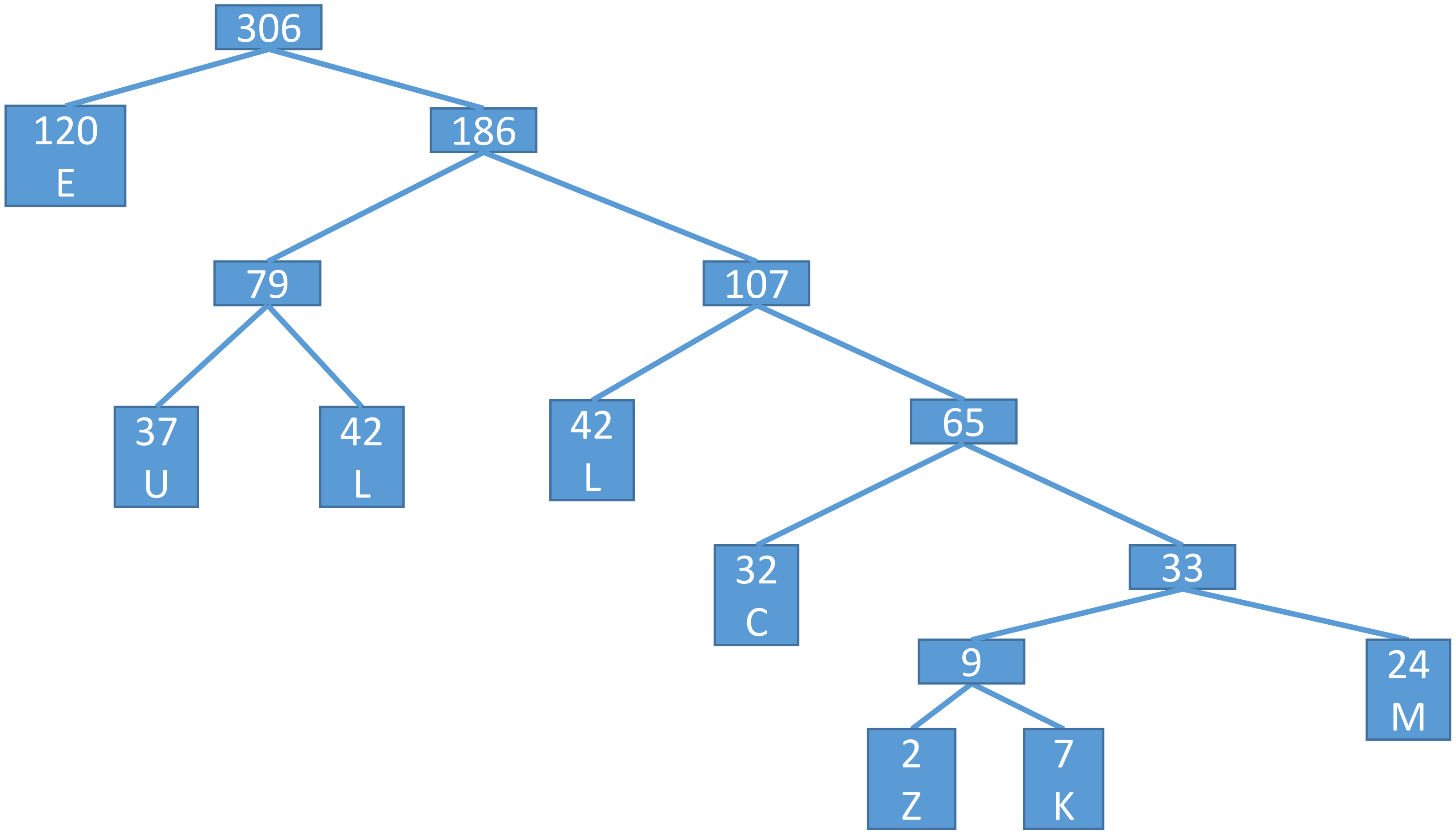
Letter	E	D						
Frequency	120	42						

- Next is D



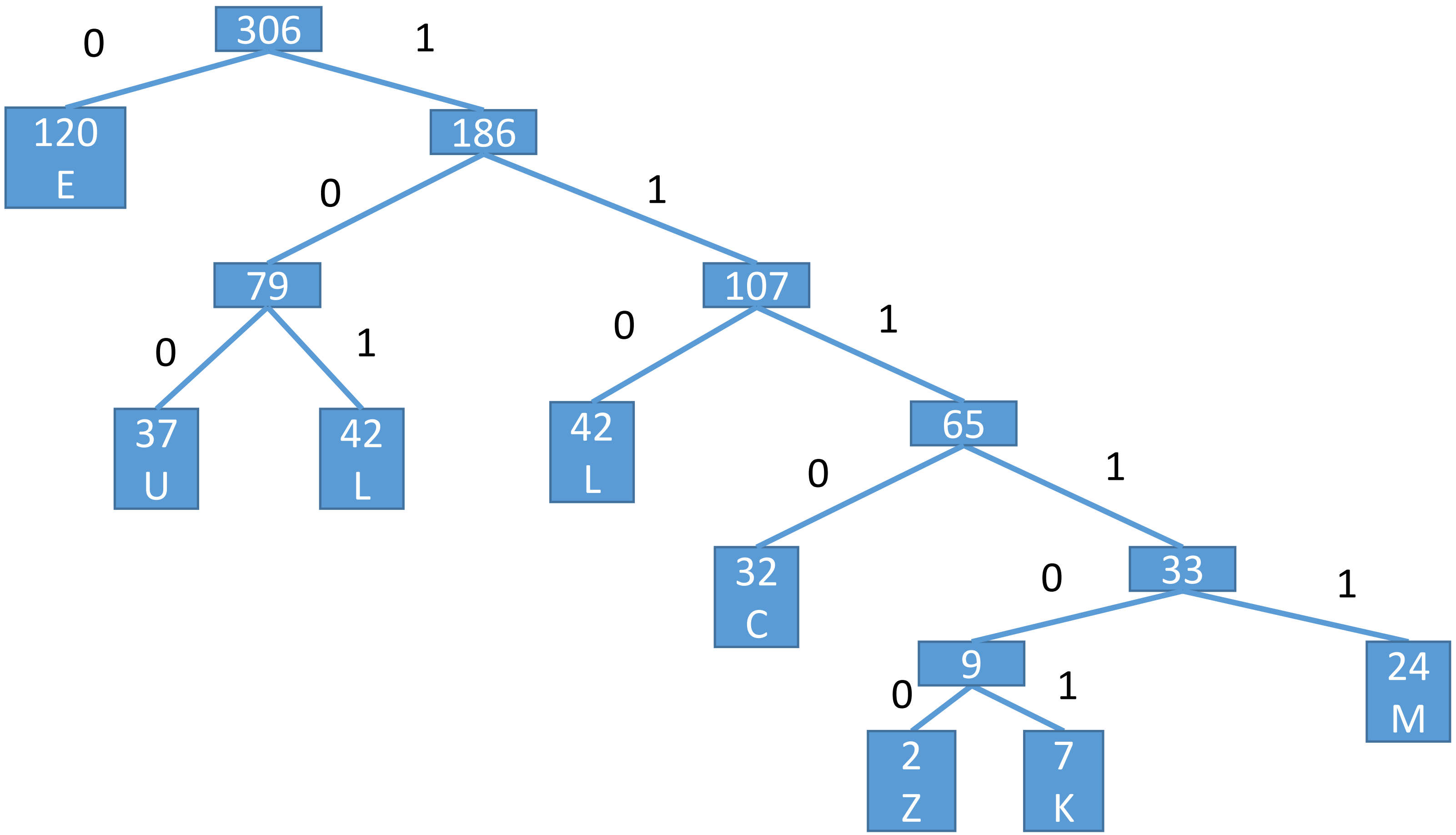
Huffman tree

- Connect two trees and then with E



Huffman tree

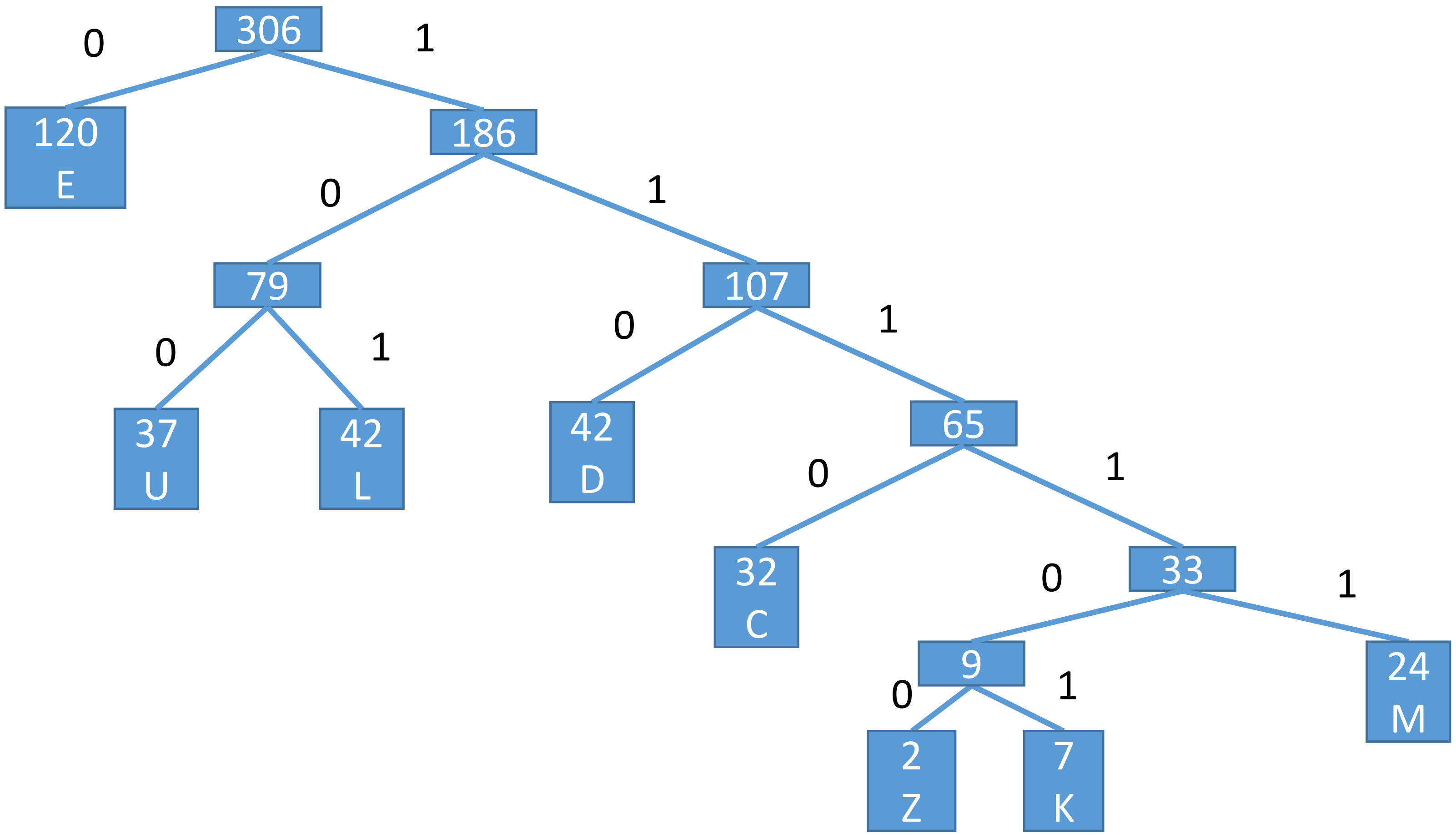
- Assign value



Huffman tree

- Hence

E	U	L	D	C	M	Z	K
0	100	101	110	1110	11111	111100	111101



A Bigger Example

- Let us return to Alice's Adventures in Wonderland.
- Chapter 1 has 10,734 characters with 27 distinct characters (A-Z, space).
- If we count the different characters we get the following table…

A	B	C	D	E	F	G	H	I	J	K	L	M
678	141	171	387	1078	186	202	592	561	6	96	418	144
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
558	696	131	6	452	502	867	249	68	252	7	163	4
space												
2119												

- We now sort this table in descending order…

A Bigger Example

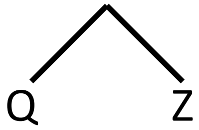
space	E	T	O	A	H	I	N	S	R	L	D	W
2119	1078	867	696	678	592	561	558	502	452	418	387	252
U	G	F	C	Y	M	B	P	K	V	X	J	Q
249	202	186	171	163	144	141	131	96	68	7	6	6
Z												
4												

- We next merge the last two entries into a tree and update the count to the sum of the values merged.

A Bigger Example

space	E	T	O	A	H	I	N	S	R	L	D	W
2119	1078	867	696	678	592	561	558	502	452	418	387	252
U	G	F	C	Y	M	B	P	K	V	X	J	(QZ)
249	202	186	171	163	144	141	131	96	68	7	6	10

- Now we sort the last entry into the correct location.
- Note: (QZ) ➔



A Bigger Example

space	E	T	O	A	H	I	N	S	R	L	D	W
2119	1078	867	696	678	592	561	558	502	452	418	387	252
U	G	F	C	Y	M	B	P	K	V	(QZ)	X	J
249	202	186	171	163	144	141	131	96	68	10	7	6

- Repeat with the new last 2 entries…

A Bigger Example

space	E	T	O	A	H	I	N	S	R	L	D	W
2119	1078	867	696	678	592	561	558	502	452	418	387	252
U	G	F	C	Y	M	B	P	K	V	(XJ)	(QZ)	
249	202	186	171	163	144	141	131	96	68	13	10	

- Again…

A Bigger Example

space	E	T	O	A	H	I	N	S	R	L	D	W
2119	1078	867	696	678	592	561	558	502	452	418	387	252
U	G	F	C	Y	M	B	P	K	V	((XJ)(QZ))		
249	202	186	171	163	144	141	131	96	68	23		

- Again…

A Bigger Example

space	E	T	O	A	H	I	N	S	R	L	D	W
2119	1078	867	696	678	592	561	558	502	452	418	387	252
U	G	F	C	Y	M	B	P	K	(V((XJ) (QZ)))			
249	202	186	171	163	144	141	131	96	91			

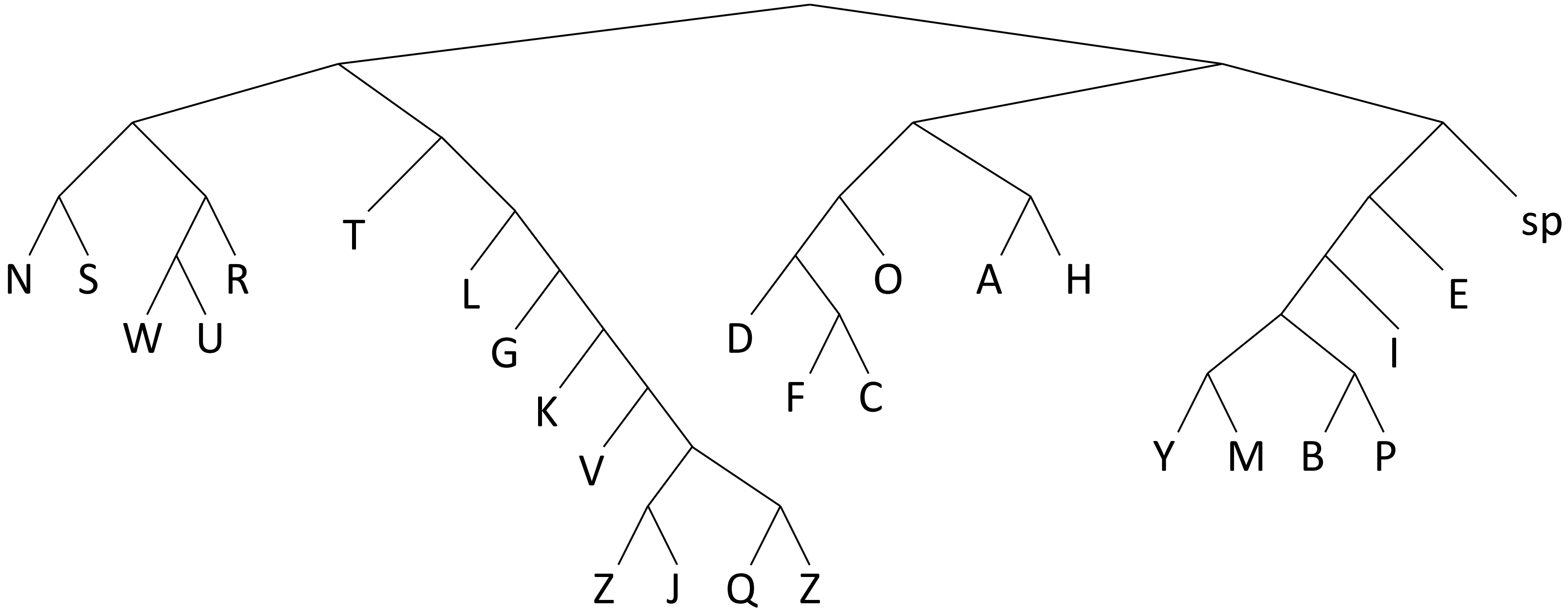
- And again…

A Bigger Example

space	E	T	O	A	H	I	N	S	R	L	D	W
2119	1078	867	696	678	592	561	558	502	452	418	387	252
U	G	(K(V((XJ)(QZ))))	F	C	Y	M	B	P				
249	202	187	186	171	163	144	141	131				

- After many more steps…

The Huffman Tree



A Bigger Example

A	B	C	D	E	F	G	H	I
1010	11000010	100011	10000	1101	100010	01110	1011	11001
J	K	L	M	N	O	P	Q	R
011111101	011110	0110	1100001	0000	1001	11000010	011111110	0011
S	T	U	V	W	X	Y	Z	space
0001	010	00101	0111110	00100	011111100	1100000	011111111	111

Shrinking Alice.

- If we multiply the number of bits for each character by the number of times it occurs we get…
- 4 x 678 + 8 x 141 + … + 4 x 9 + 3 x 2119
- This is a total of 44,756 bits.
- Compare with our previous result of 53,670 bits.
- This is a saving of around 20%.