More Bits and Bytes
Huffman Coding

Encoding Text: How is it done?
ASCII, UTF, Huffman algorithm

ASCII

0100 0011
0100 0001
0101 0100

C
A
T

ASCII	0000 0000	0000 0001	0000 0010	0000 0011	0000 0100	0000 0101	0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111
0000	$^{N}_{U}$	$^{S}_{H}$	$^{S}_{X}$	$^{E}_{X}$	$^{E}_{T}$	$^{E}_{Q}$	$^{A}_{K}$	$^{B}_{L}$	$^{B}_{S}$	$^{H}_{T}$	$^{L}_{F}$	$^{Y}_{T}$	$^{F}_{F}$	$^{C}_{R}$	$^{S}_{0}$	$^{S}_{I}$
0001	$^{D}_{L}$	$^{D}_{1}$	$^{D}_{2}$	$^{D}_{3}$	$^{D}_{4}$	$^{N}_{K}$	$^{S}_{Y}$	$^{E}_{\Sigma}$	$^{C}_{N}$	$^{E}_{M}$	$^{S}_{B}$	$^{E}_{C}$	$^{F}_{S}$	$^{G}_{S}$	$^{R}_{S}$	$^{U}_{S}$
0010		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	$^{D}_{T}$
1000	$^{8}_{0}$	$^{8}_{1}$	$^{8}_{2}$	$^{8}_{3}$	$^{I}_{N}$	$^{N}_{L}$	$^{S}_{S}$	$^{E}_{S}$	$^{H}_{S}$	$^{H}_{J}$	$^{Y}_{S}$	$^{P}_{D}$	$^{P}_{V}$	$^{R}_{I}$	$^{S}_{2}$	$^{S}_{3}$
1001	$^{D}_{C}$	$^{P}_{1}$	$^{P}_{Z}$	$^{S}_{E}$	$^{C}_{C}$	$^{M}_{M}$	$^{S}_{P}$	$^{E}_{P}$	$^{Q}_{8}$	$^{Q}_{Q}$	$^{Q}_{A}$	$^{C}_{S}$	$^{S}_{T}$	$^{O}_{S}$	$^{P}_{M}$	$^{A}_{P}$
1010	$^{A}_{O}$	¡	¢	£	♀	¥	¦	§	¨	©	♂	«	¬	-	®	¯
1011	°	±	^{2}	^{3}	´	µ	¶	·	¸	^{1}	º	»	¼	½	¾	¿
1100	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
1101	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	β
1110	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
1111	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

0100 0111|0110 1111|0010 0000| 0101 0011|0110 1100|0111 0101|0110 0111|0111 0011|

UTF-8: All the alphabets in the world

- Uniform Transformation Format: a variable-width encoding that can represent every character in the Unicode Character set
- 1,112,064 of them!!!
- http://en.wikipedia.org/wiki/UTF-8
- UTF-8 is the dominant character encoding for the World-Wide Web, accounting for more than half of all Web pages.
- The Internet Engineering Task Force (IETF) requires all Internet protocols to identify the encoding used for character data
- The supported character encodings must include UTF-8.

UTF is a VARIABLE LENGTH ALPHABET CODING

- Remember ASCII can only represent 128 characters (7 bits)
- UTF encodes over one million
- Why would you want a variable length coding scheme?

Bits	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+007F	0xxxxxxx					
11	U+07FF	110xxxxx	10xxxxxx				
16	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
21	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
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`

UTF-8

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
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

What is the first Unicode value represented by this sequence?
11101010 10000011 10000111 00111111 11000011 10000000

A. 0000000001101010
B. 0000000011101010
C. 0000001010000111
D. 1010000011000111

UTF-8

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
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

How many Unicode characters are represented by this sequence?
11101010 10000011 10000111 00111111 11000011 10000000

A. 1
B. 2
C. 3
D. 4
E. 5

How many bits for all of Unicode?

There are 1,112,064 different Unicode characters. If a fixed bit format (like ascii with its 7 bits) where used, how many bits would you need for each character? (Hint: $2^{10} = 1024$)

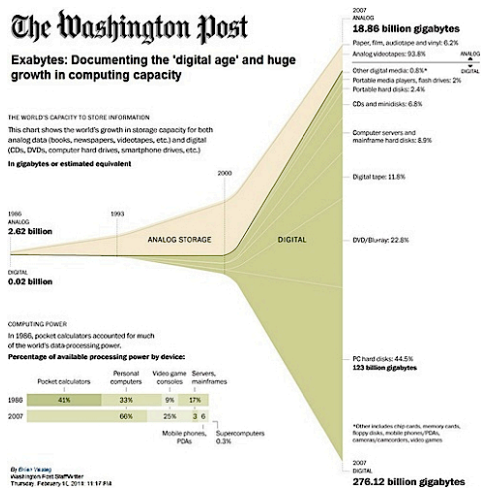
A. 10
B. 17
C. 21
D. 32
E. 40

Coding can be used to do Compression

- What is CODING?
  - The conversion of one representation into another
- What is COMPRESSION?
  - Change the representation (digitization)

  in order to **reduce** size of data

  (number of bits needed to represent data)
- Benefits
  - **Reduce storage** needed
    - Consider growth of digitized data.
  - **Reduce** transmission cost / **latency /** bandwidth
  - When you have a 56K dialup modem, every savings in BITS counts, **SPEED**
    - Also consider telephone lines, texting



What makes it possible to do Compression?

- IN OTHER WORDS: When is Coding USEFUL?
- When there is **Redundancy**
  - Recognize repeating patterns
  - Exploit using
    - Dictionary
    - Variable length encoding
- When **Human perception is less sensitive** to some information
  - Can discard less important data

How easy is it to do it?

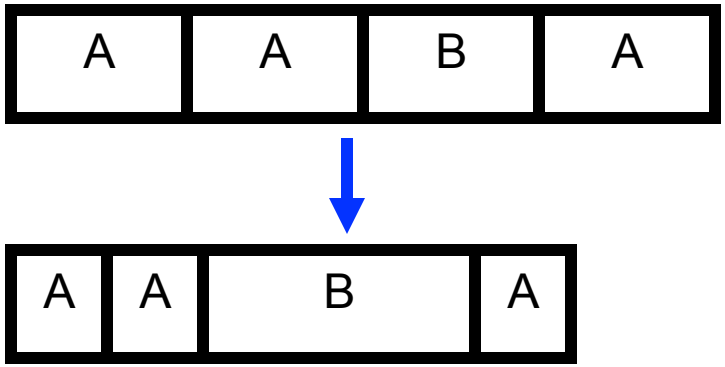
- Depends on data
  - Random data ⇒ hard
    - Example: 10011110100 ⇒ ?
  - Organized data ⇒ easy
    - Example: 11111111111 ⇒ 1×10
- WHAT DOES THAT MEAN?
  - **There is NO universally best compression algorithm**
  - **It depends on how tuned the coding is to the data you have**

Can you lose information with Compression?

- **Lossless** Compression is not guaranteed
  - Pigeonhole principle
    - Reduce size 1 bit ⇒ can only store ½ of data
    - Example
      - 000, 001, 010, 011, 100, 101, 110, 111 ⇒ 00, 01, 10, 11
  - CONSIDER THE ALTERNATIVE
  - IF LOSSLESS COMPRESSION WERE GUARANTEED THEN
    - Compress file (reduce size by 1 bit)
    - Recompress output
    - Repeat (until we can store data with 0 bits)
  - OBVIOUS CONTRADICTION => IT IS NOT GUARANTEED.

Huffman Code: A Lossless Compression

- Use Variable Length codes based on frequency (like UTF does)
- Approach
  - Exploit statistical frequency of symbols
  - What do I MEAN by that? WE COUNT!!!
- HELPS when the frequency for different symbols varies widely
- Principle
  - Use fewer bits to represent frequent symbols
  - Use more bits to represent infrequent symbols



Huffman Code Example

- "dog cat cat bird bird bird bird fish"

Symbol	Dog	Cat	Bird	Fish
Frequency	1/8	1/4	1/2	1/8
Original Encoding	00	01	10	11
	2 bits	2 bits	2 bits	2 bits
Huffman Encoding	110	10	0	111
	3 bits	2 bits	1 bit	3 bits

- Expected size
  - Original $\Rightarrow$ 1/8×2 + 1/4×2 + 1/2×2 + 1/8×2 = 2 bits / symbol
  - Huffman $\Rightarrow$ 1/8×3 + 1/4×2 + 1/2×1 + 1/8×3 = 1.75 bits / symbol

Huffman Code Example

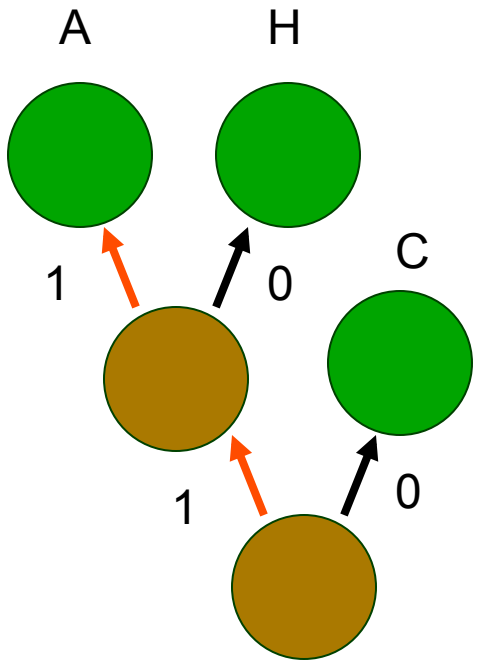
Symbol	Dog	Cat	Bird	Fish
Frequency	1/8	1/4	1/2	1/8
Original Encoding	00	01	10	11
	2 bits	2 bits	2 bits	2 bits
Huffman Encoding	110	10	0	111
	3 bits	2 bits	1 bit	3 bits

How many bits are saved using the above Huffman coding for the sequence Dog Cat Bird Bird Bird?

A. 0 B. 1 C. 2 D. 3 E. 4

Huffman Code Algorithm: Data Structures

- Binary (Huffman) tree
  - Represents Huffman code
  - Edge ⇒ code (0 or 1)
  - Leaf ⇒ symbol
  - Path to leaf ⇒ encoding
  - Example
    - A = "11", H = "10", C = "0"
    - Good when ???
      - A, H less frequent than C in messages



- Want to efficiently build a binary tree

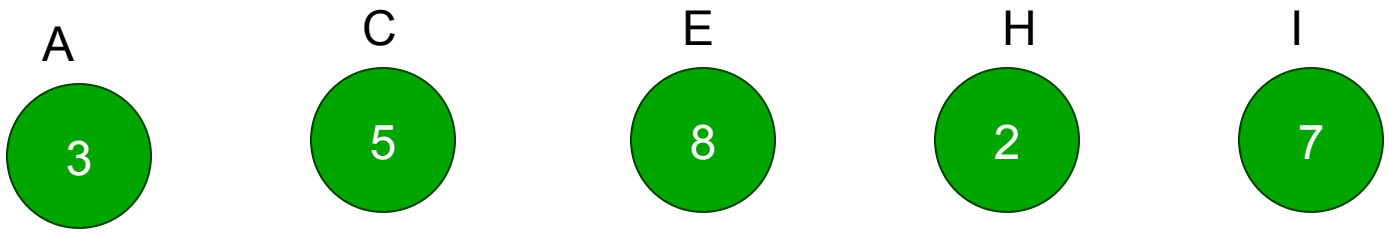
Huffman Code Algorithm Overview

- Order the symbols with least frequent first (will explain)
- Build a tree piece by piece…
- Encoding
  - Calculate frequency of symbols in the message, language
  - JUST COUNT AND DIVIDE BY TOTAL NUMBER OF SYMBOLS
  - Create binary tree representing "best" encoding
  - Use binary tree to encode compressed file
    - For each symbol, output path from root to leaf
    - Size of encoding = length of path
  - Save binary tree

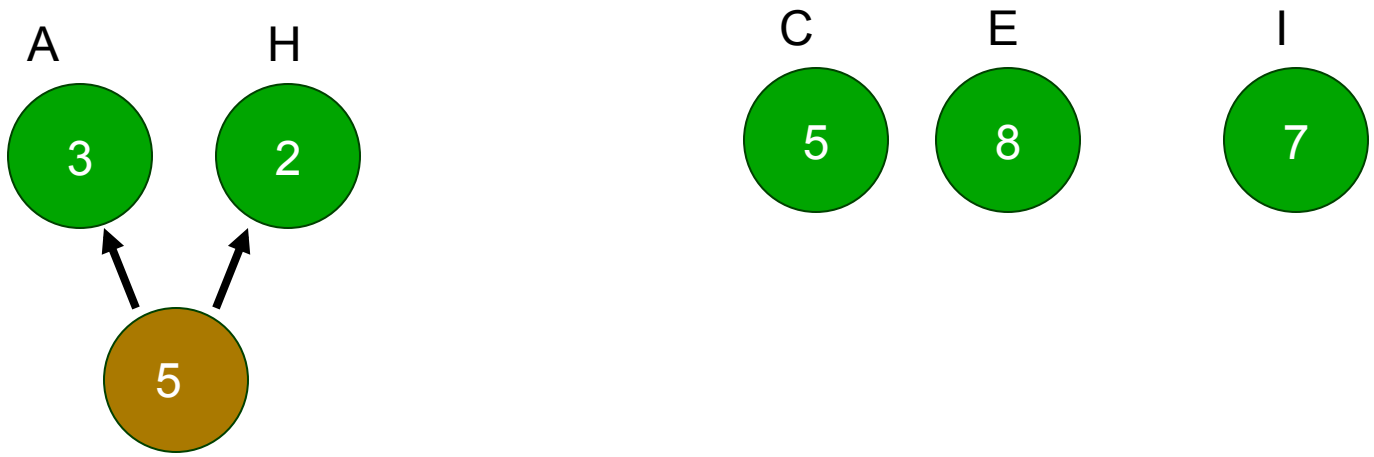
Huffman Code – Creating Tree

- Algorithm (Recipe)
  - Place each symbol in leaf
    - Weight of leaf = symbol frequency
  - Select two trees L and R (initially leafs)
    - Such that L, R have lowest frequencies among all tree
    - Which L, R have the lowest number of occurrences in the message?
  - Create new (internal) node
    - Left child ⇒ L
    - Right child ⇒ R
    - New frequency ⇒ frequency(L) + frequency(R)
  - Repeat until all nodes merged into one tree

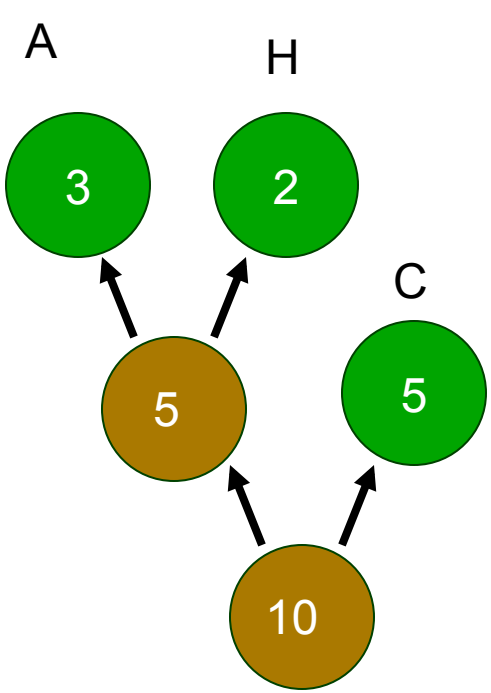
Huffman Tree Construction 1

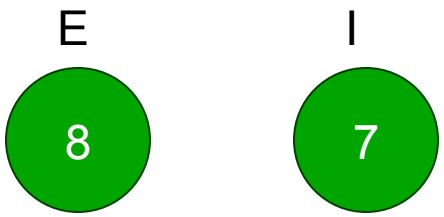


Huffman Tree Step 2: can first re-order by frequency

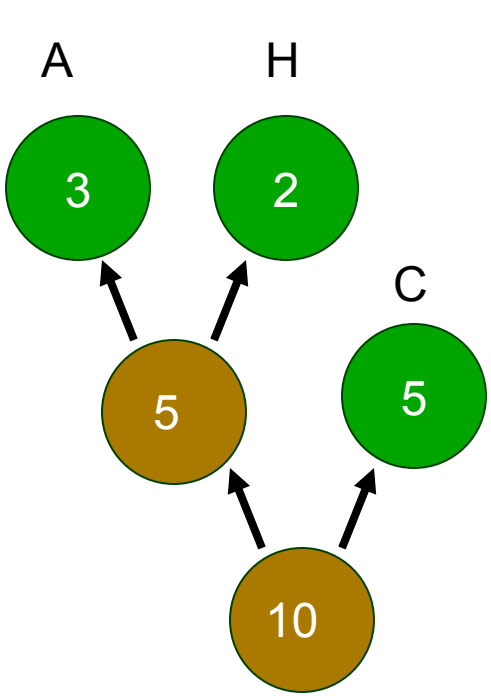


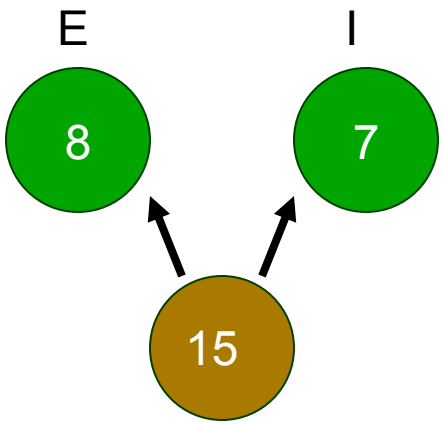
Huffman Tree Construction 3



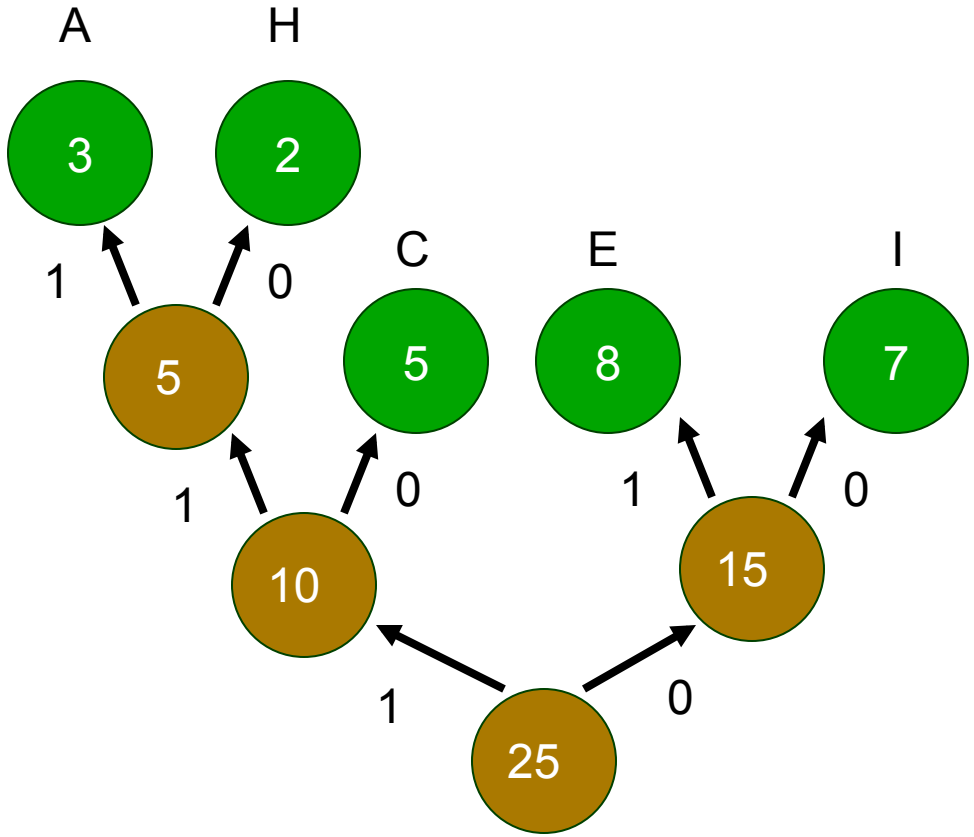


Huffman Tree Construction 4





Huffman Tree Construction 5



E = 01
I = 00
C = 10
A = 111
H = 110

Huffman Coding Example

- Huffman code

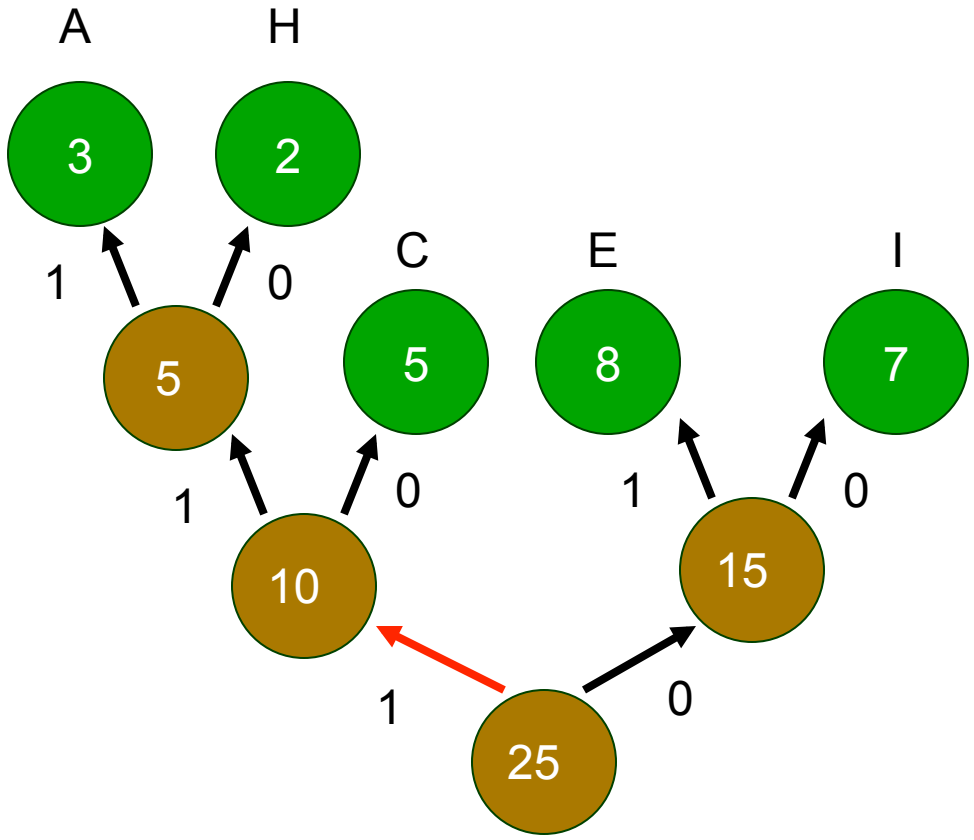
E	=	01
I	=	00
C	=	10
A	=	111
H	=	110

- Input
  - ACE
- Output
  - (111)(10)(01) = 1111001

Huffman Code Algorithm Overview

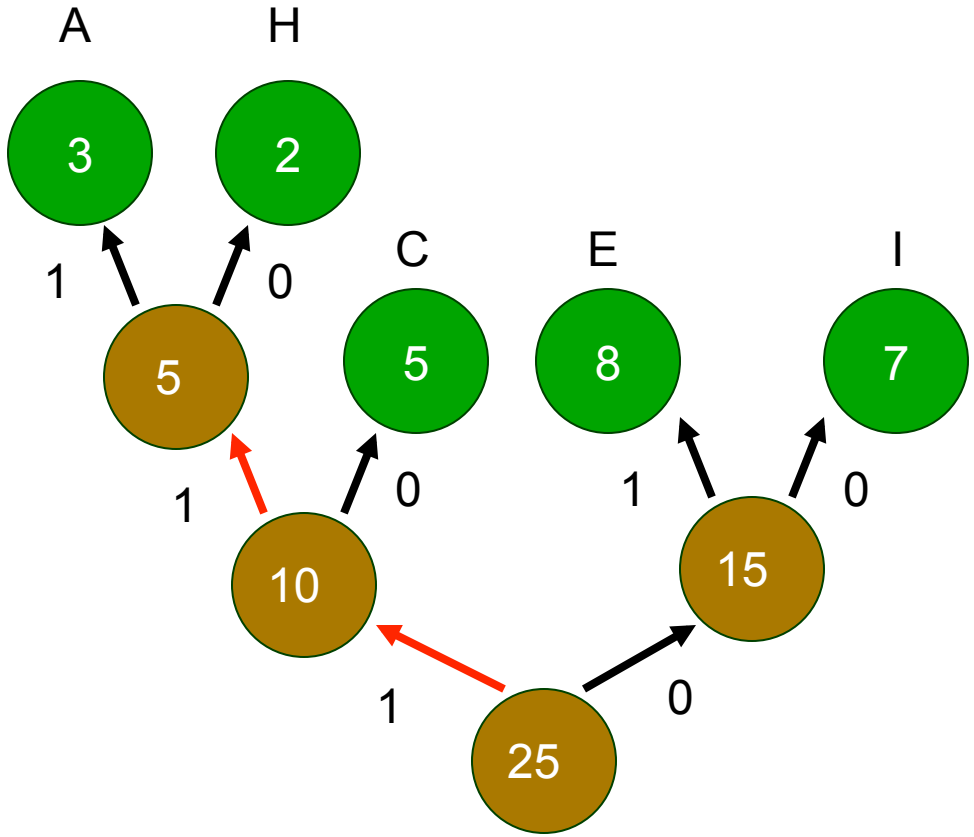
- Decoding
  - Read compressed file & binary tree
  - Use binary tree to decode file
    - Follow path from root to leaf

Huffman Decoding I



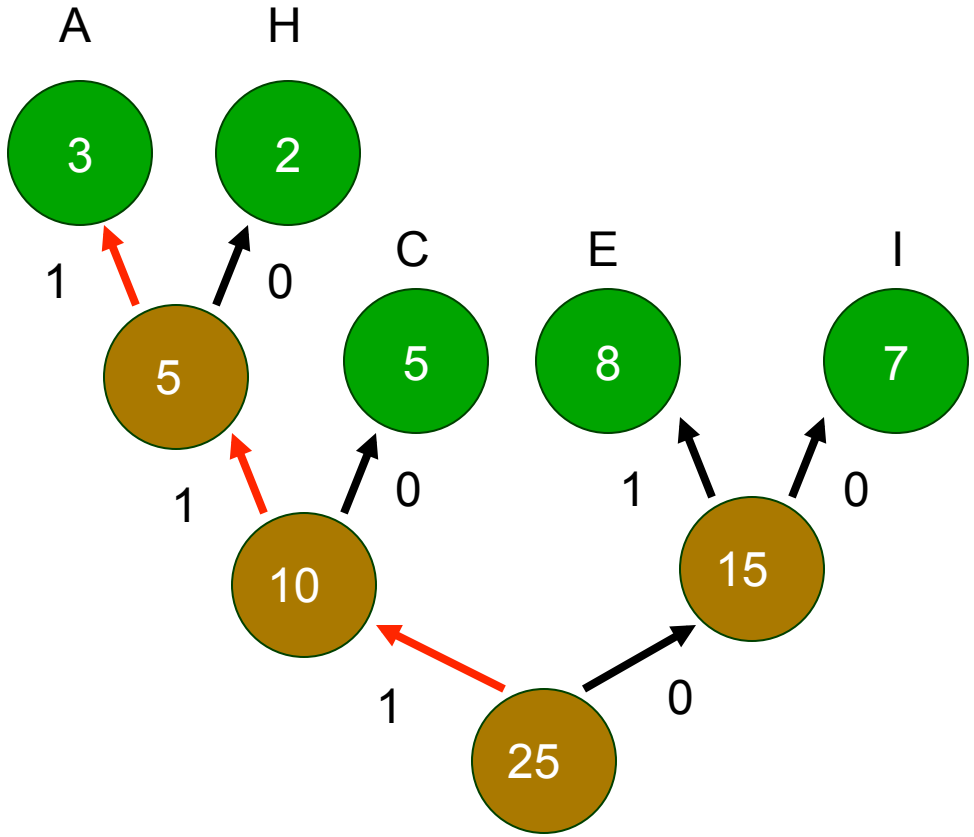
1111001

Huffman Decoding 2



1111001

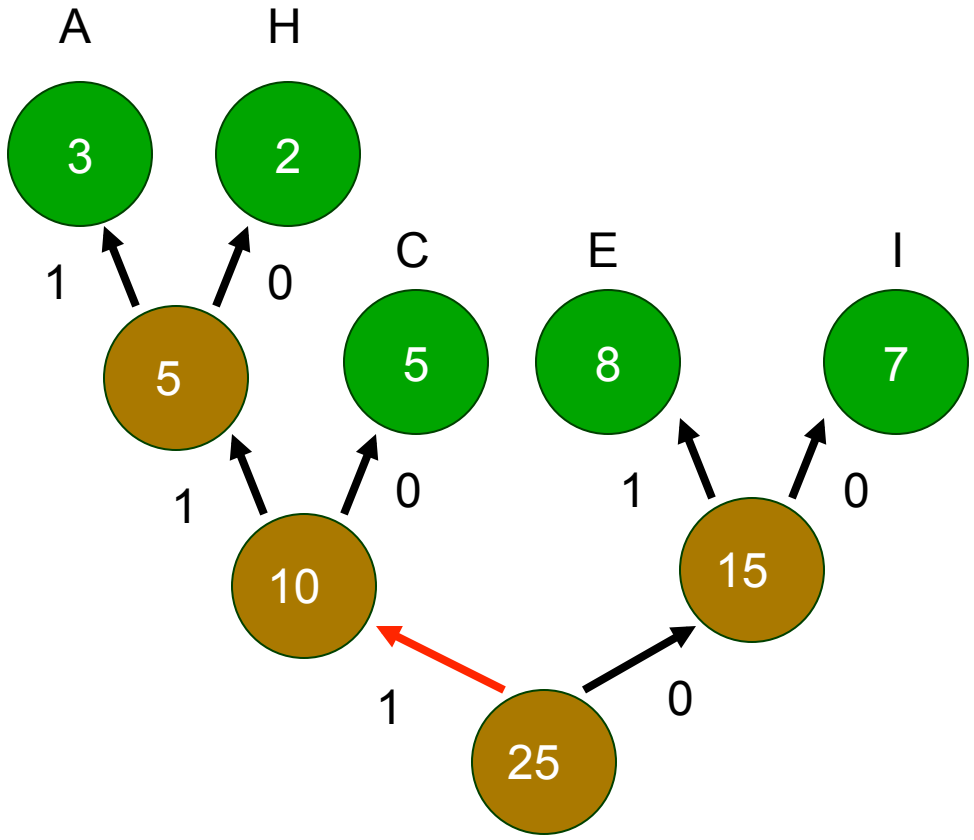
Huffman Decoding 3



1111001

A

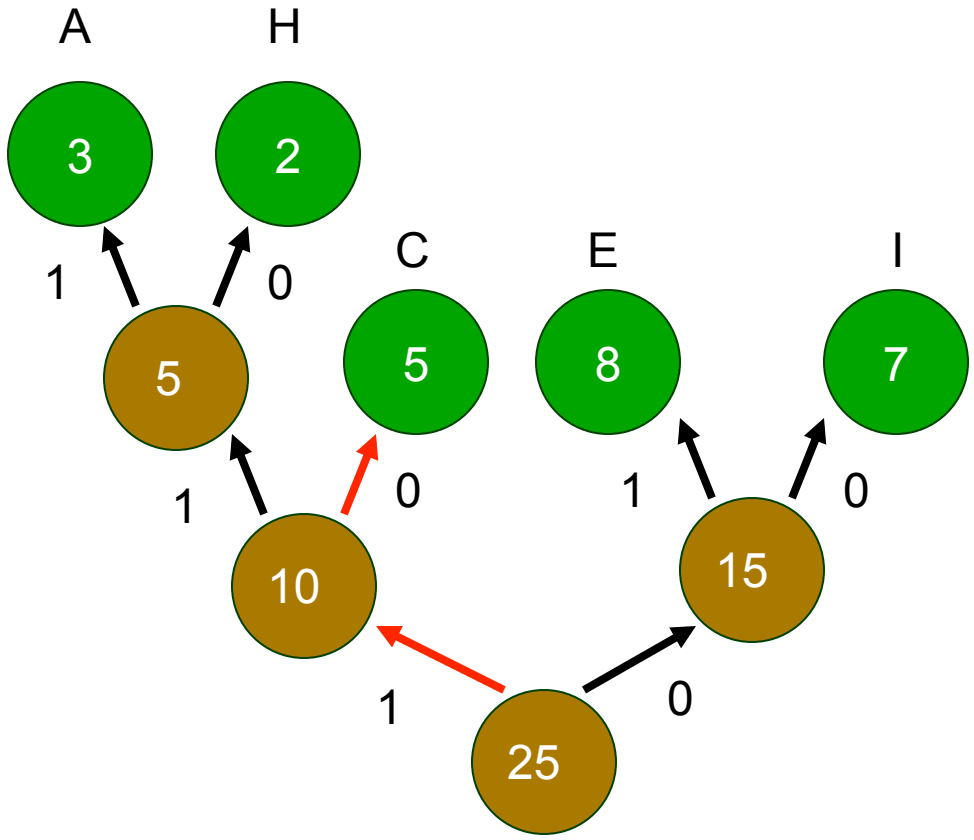
Huffman Decoding 4



1111001

A

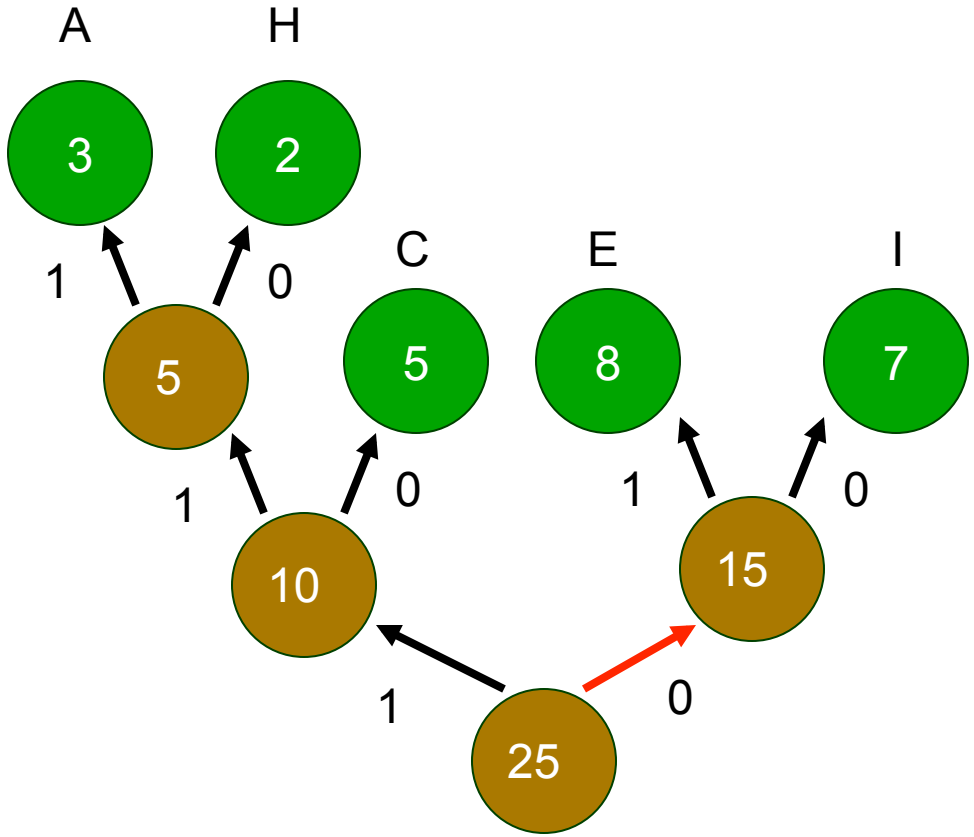
Huffman Decoding 5



1111001

AC

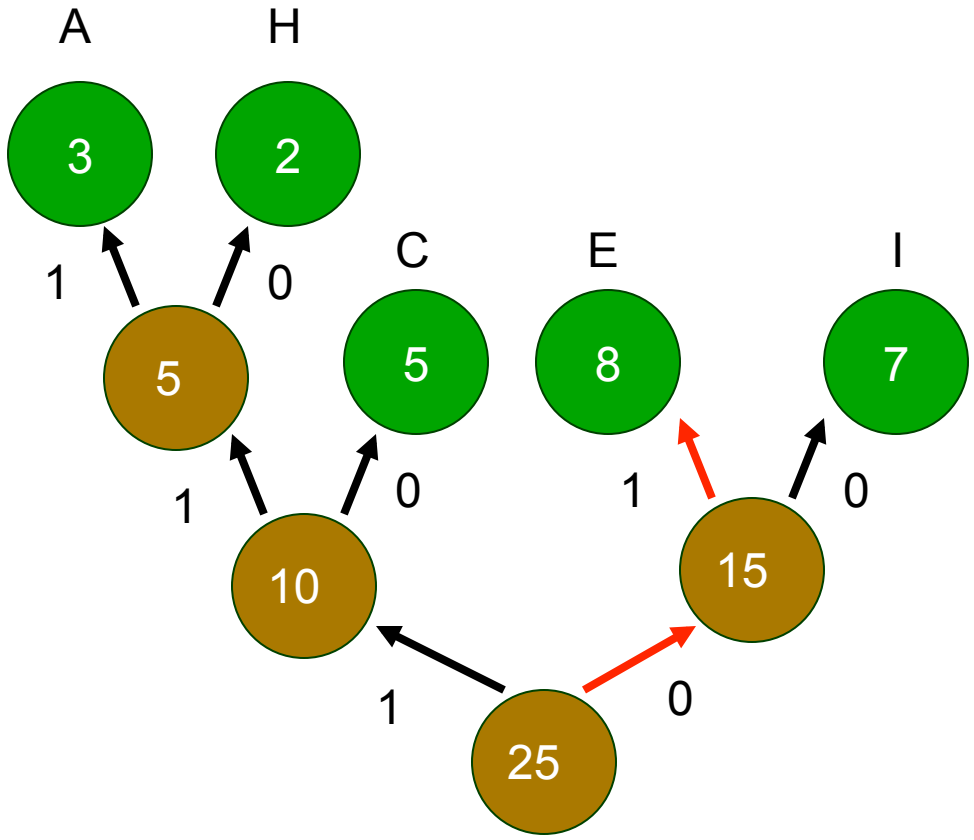
Huffman Decoding 6



1111001

AC

Huffman Decoding 7



1111001

ACE

Huffman Code Properties

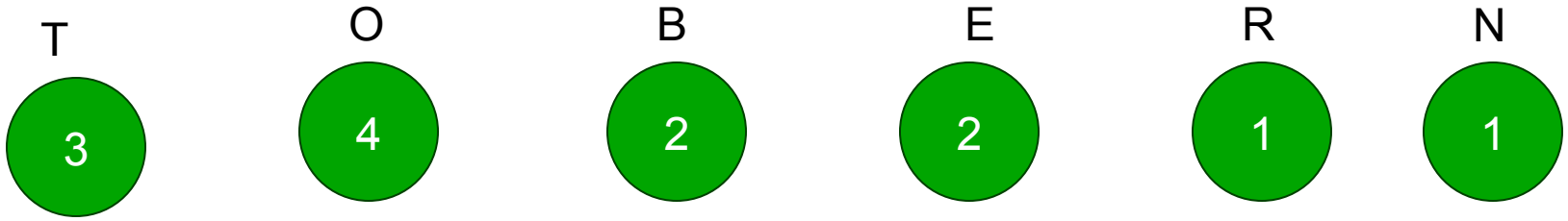
- Prefix code
  - No code is a prefix of another code
  - Example
    - Huffman("dog") ⇒ 01
    - Huffman("cat") ⇒ 011 // not legal prefix code
  - Can stop as soon as complete code found
  - No need for end-of-code marker
- Nondeterministic
  - Multiple Huffman coding possible for same input
  - If more than two trees with same minimal weight

Huffman Code Properties

- Greedy algorithm
  - Chooses best local solution at each step
  - Combines 2 trees with lowest frequency
- Still yields overall best solution
  - Optimal prefix code
  - Based on statistical frequency
- Better compression possible (depends on data)
  - Using other approaches (e.g., pattern dictionary)

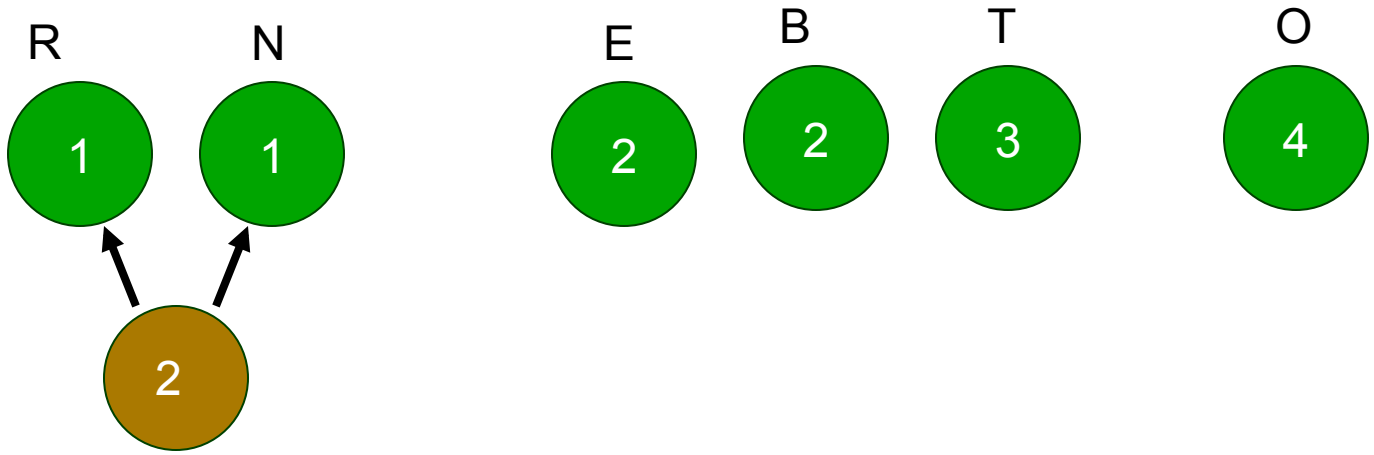
Huffman Coding. Another example.

Huffman Tree Example 2. Step 1

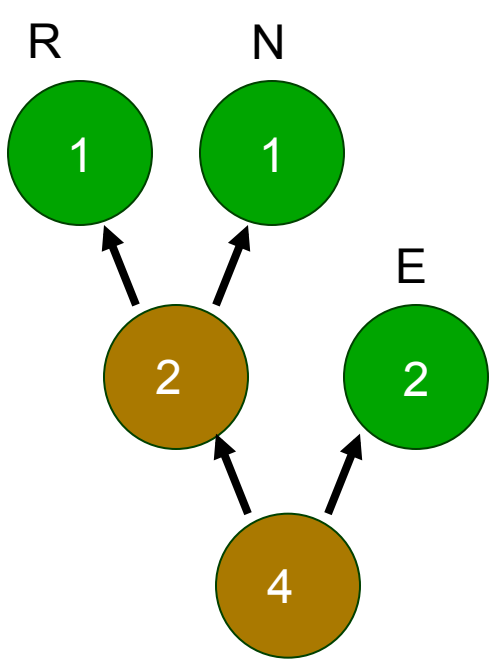


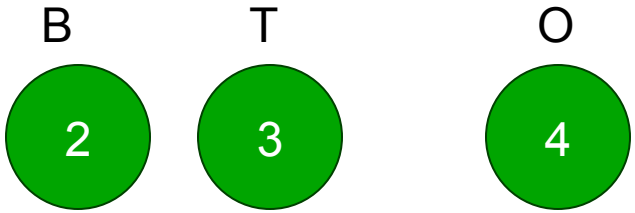
- "TO BE OR NOT TO BE"
  - T = 3
  - O = 4
  - B = 2
  - E = 2
  - R = 1
  - N = 1

Huffman Tree: TO BE OR NOT TO BE

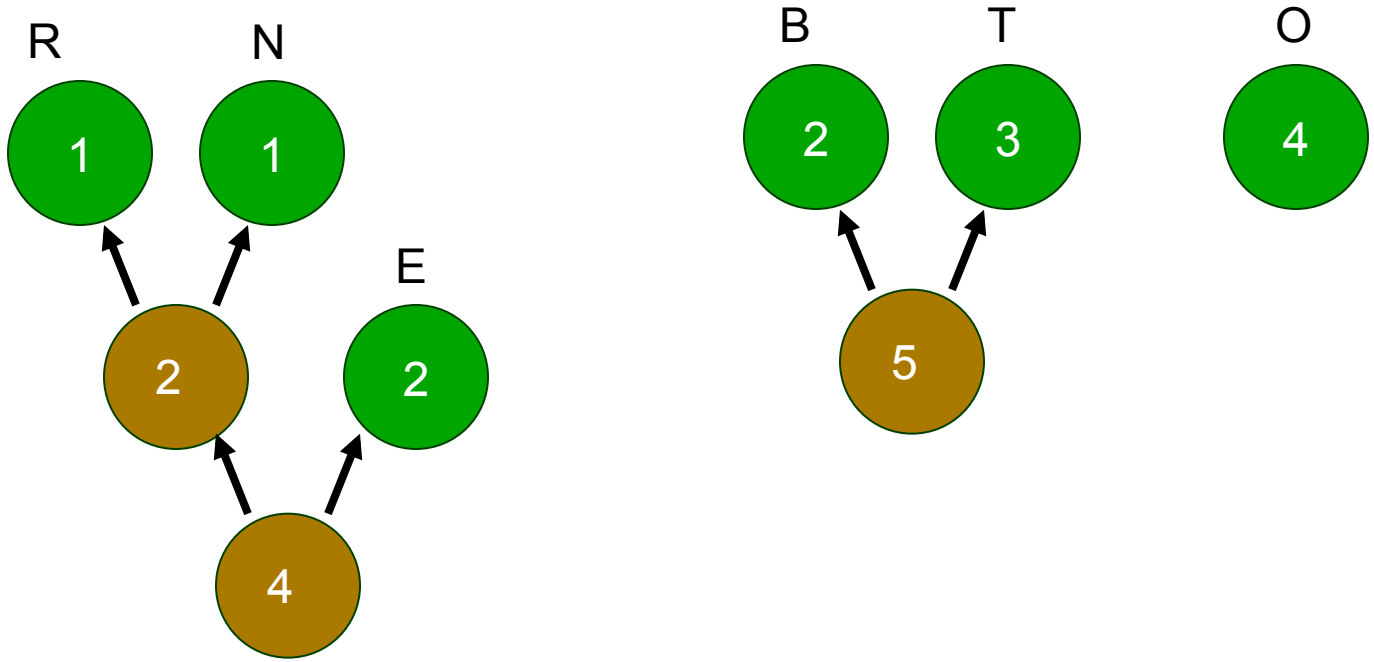


Huffman Tree: TO BE OR NOT TO BE

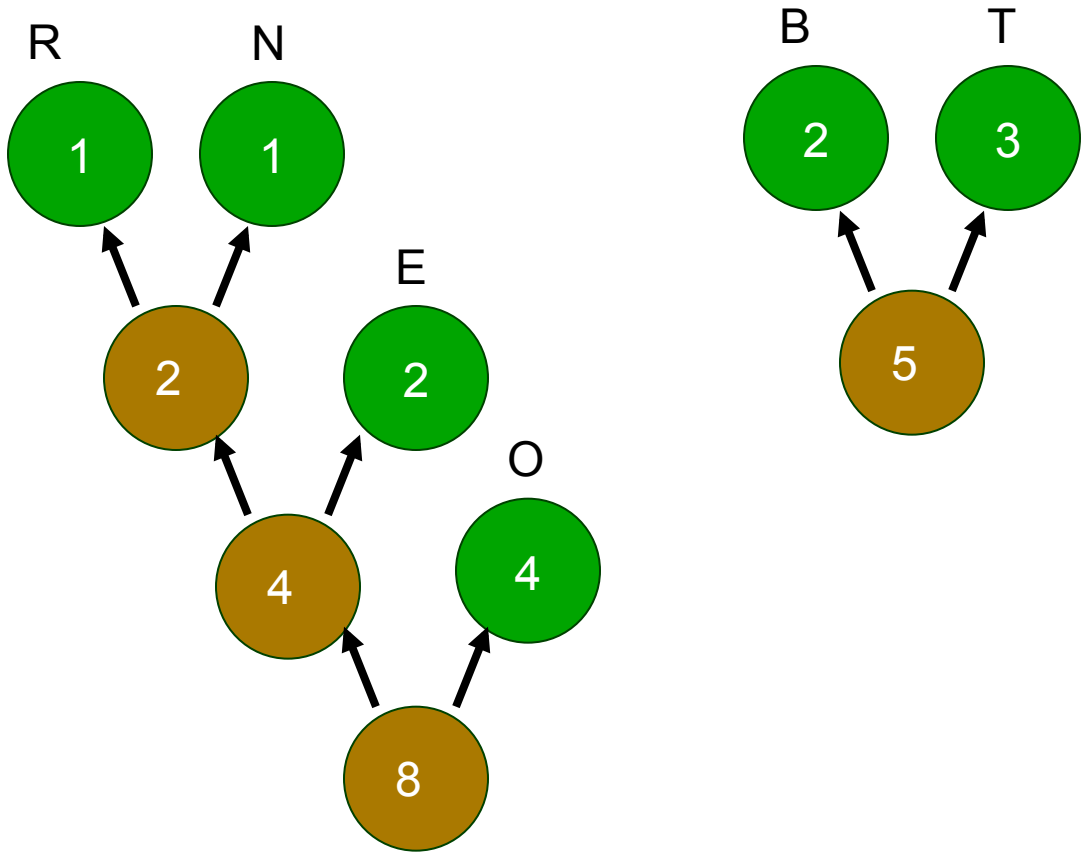




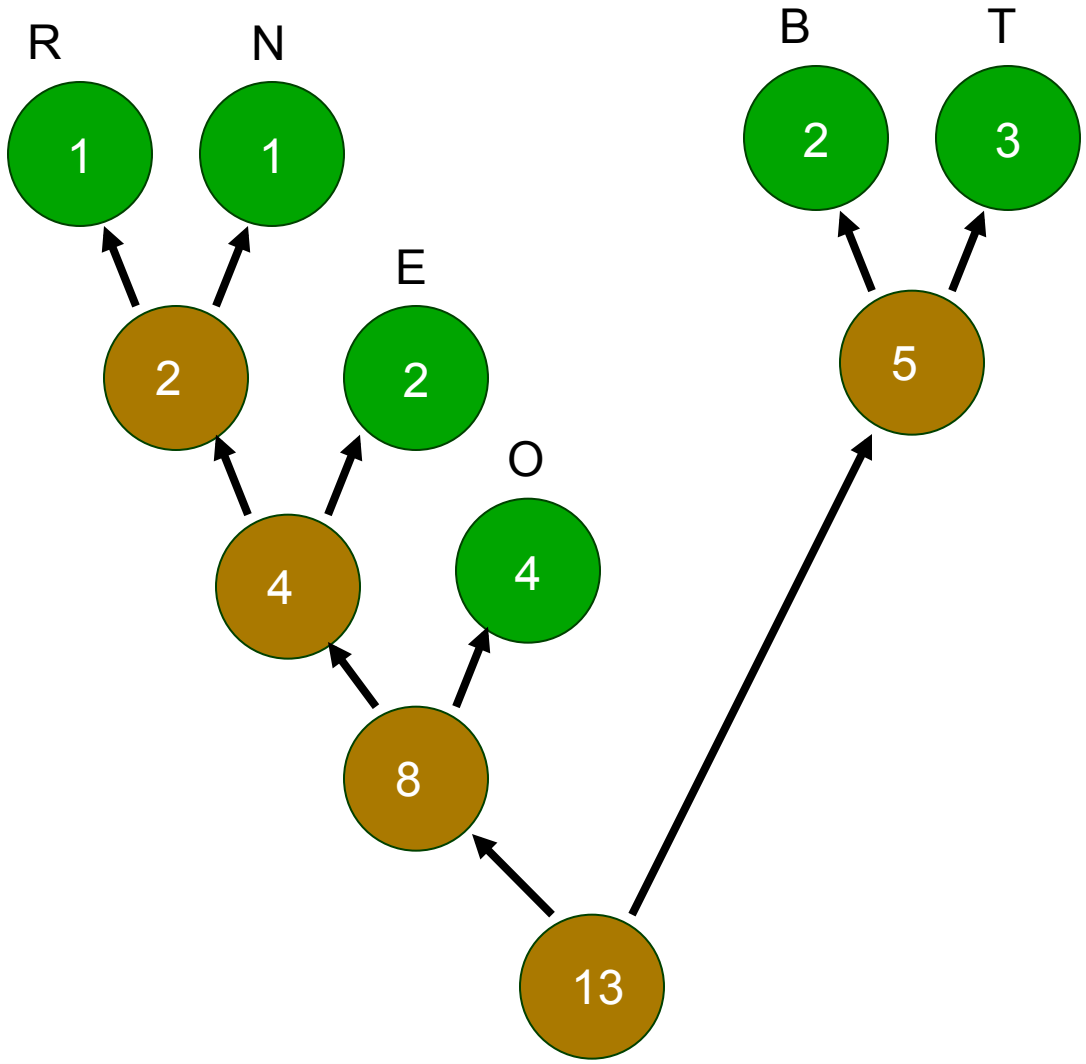
Huffman Tree: TO BE OR NOT TO BE



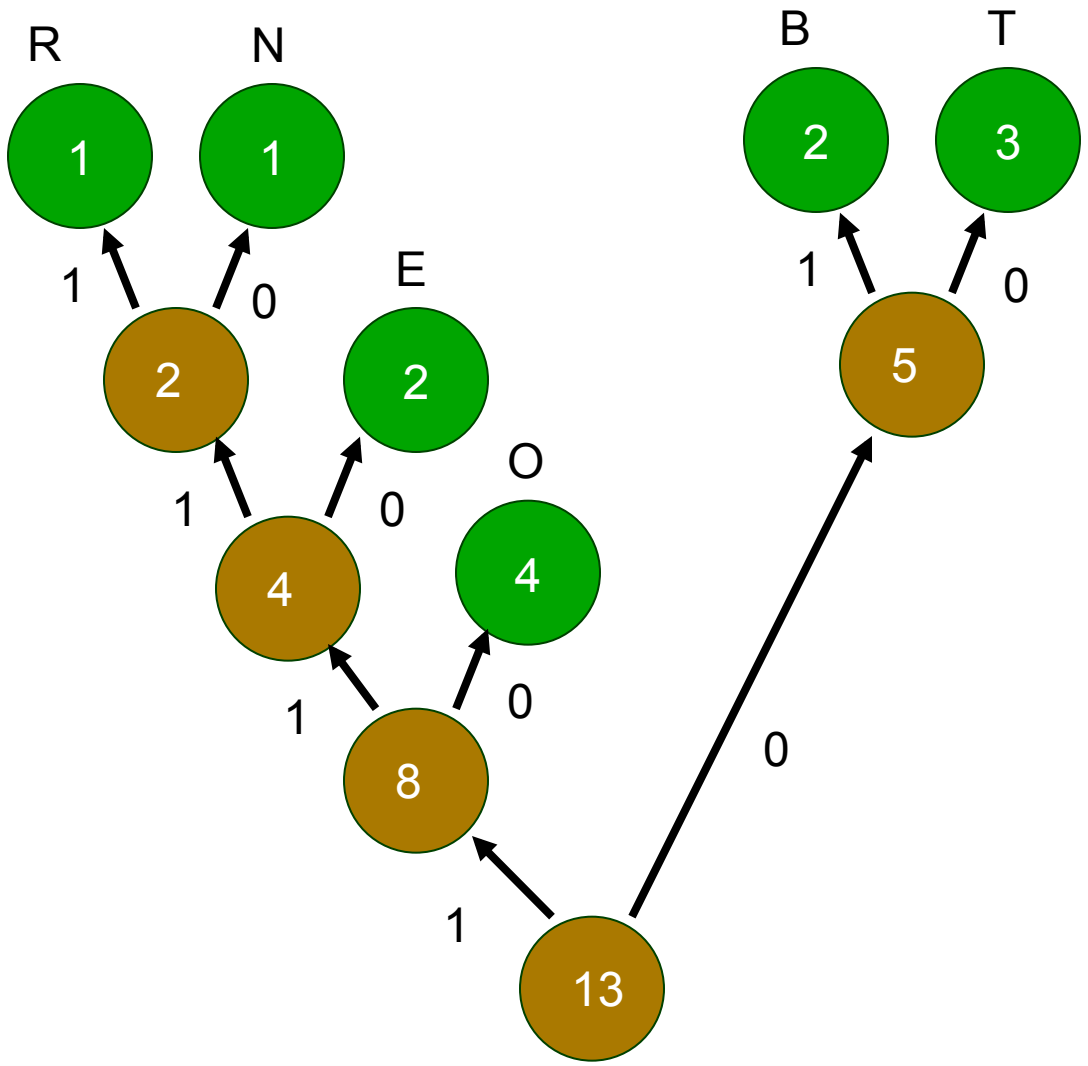
Huffman Tree: TO BE OR NOT TO BE



Huffman Tree: TO BE OR NOT TO BE

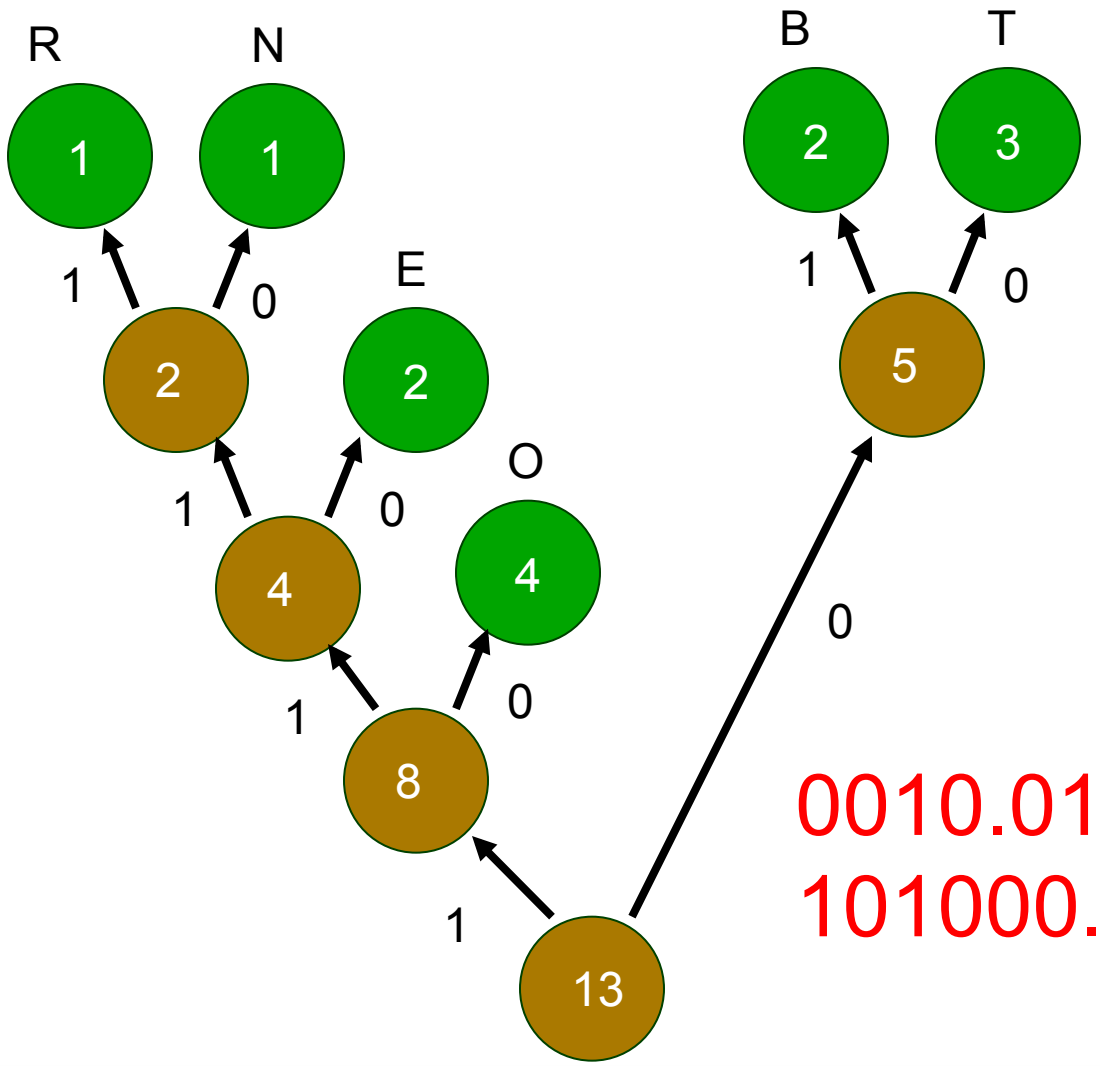


Huffman Tree: TO BE OR NOT TO BE



N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

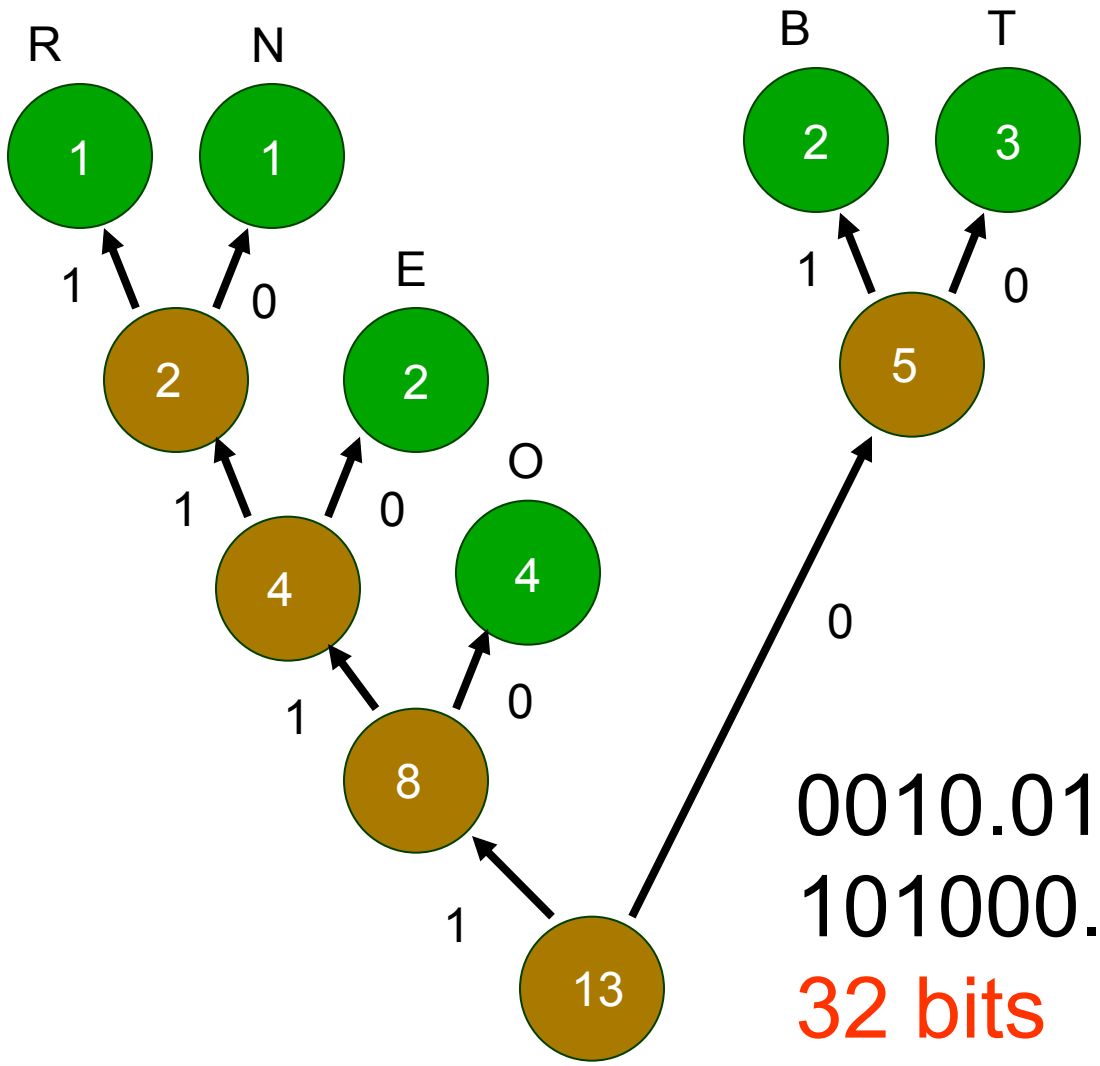
Huffman Tree: TO BE OR NOT TO BE



N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

0010.01110.101111.11
101000.0010.01110

Huffman Tree: TO BE OR NOT TO BE



N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

0010.01110.101111.11
101000.0010.01110
32 bits

TO BE OR NOT TO BE

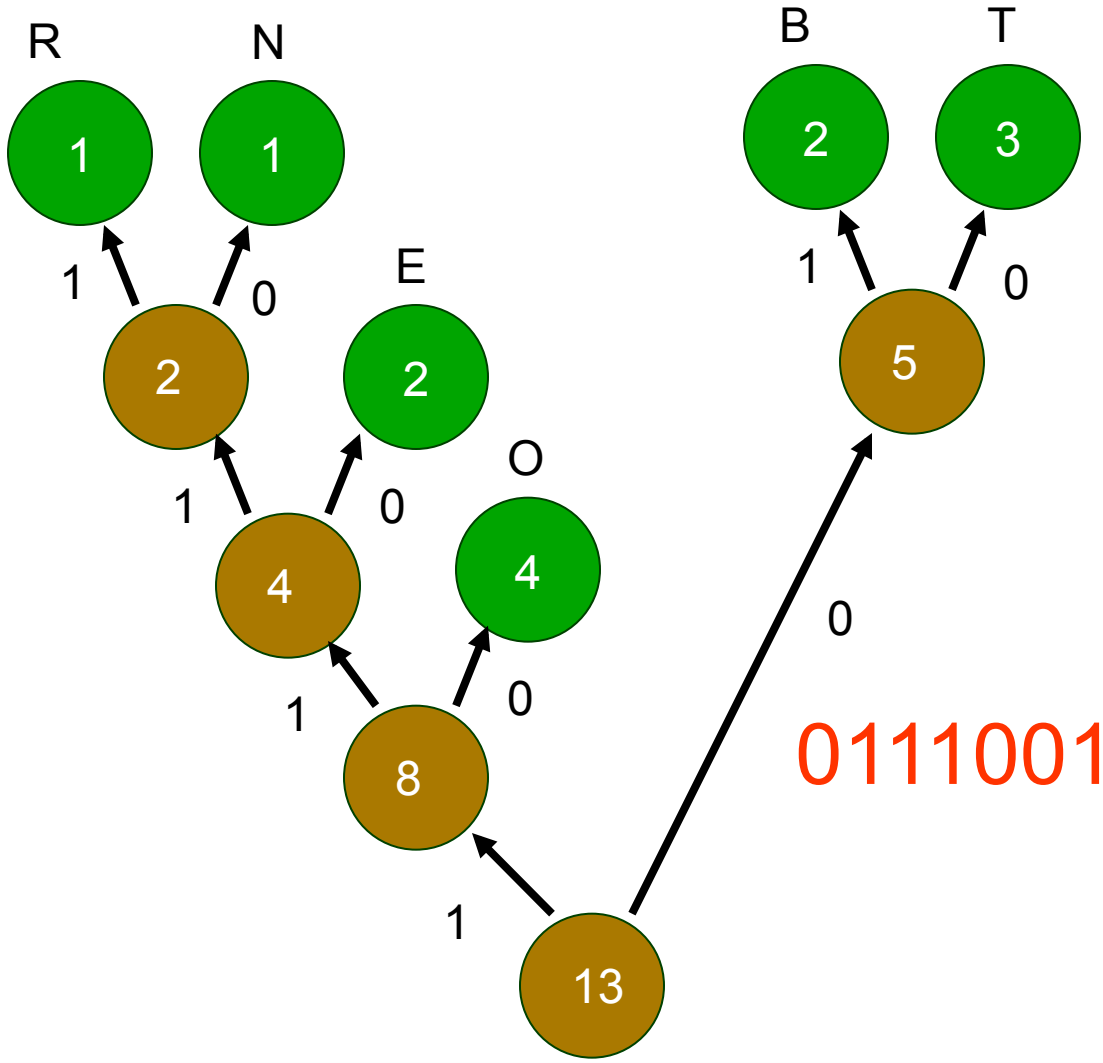
How many bits would it take to store this message if every letter was represented with the same number of bits? You should first figure out how many bits it takes to represent 6 different values/ letters.

N	=	1110
R	=	1111
E	=	110
B	=	01
O	=	10
T	=	00

A. 26
B. 32
C. 39
D. 48
E. 52

0010.01110.101111.11
101000.0010.01110
32 bits

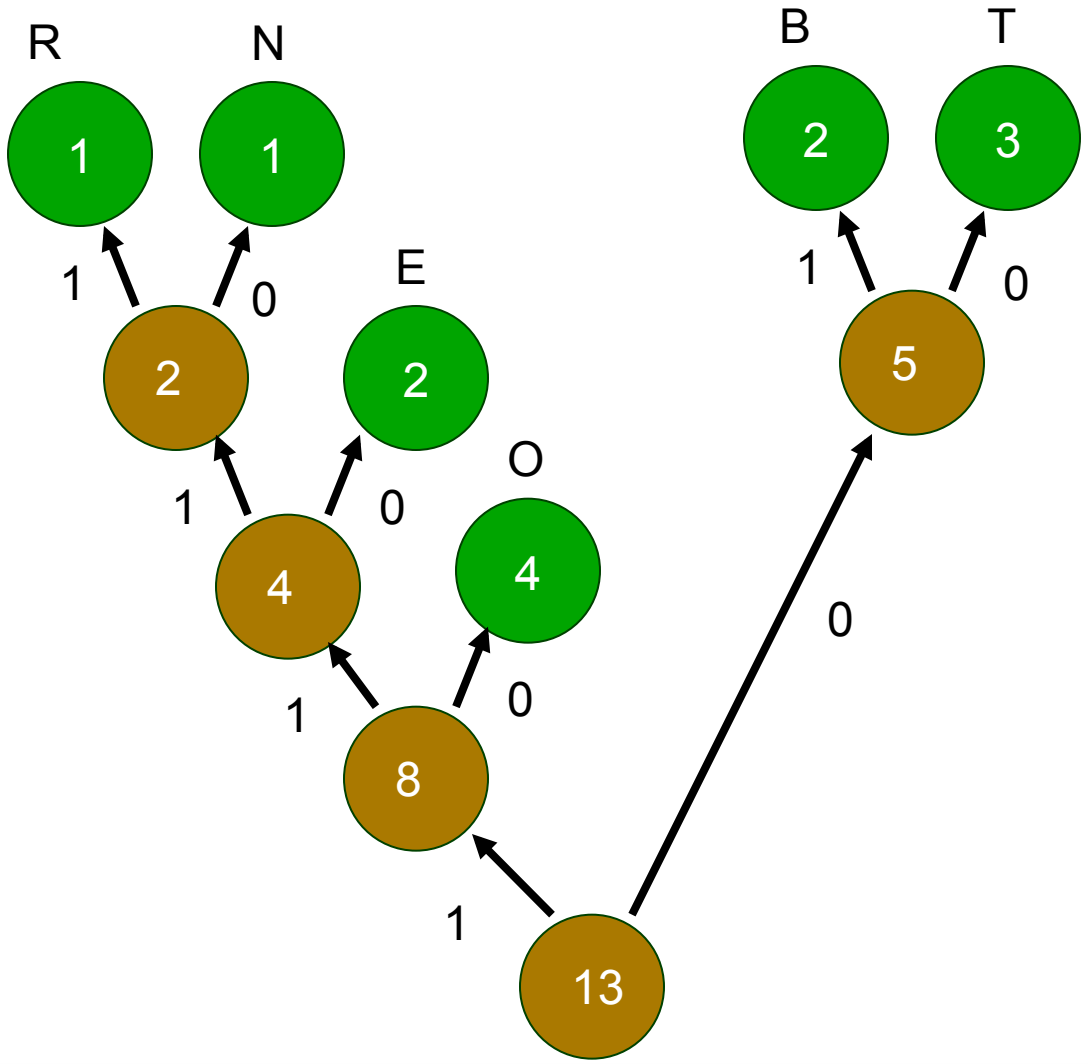
DECODING: 2nd example "BEBE"



N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

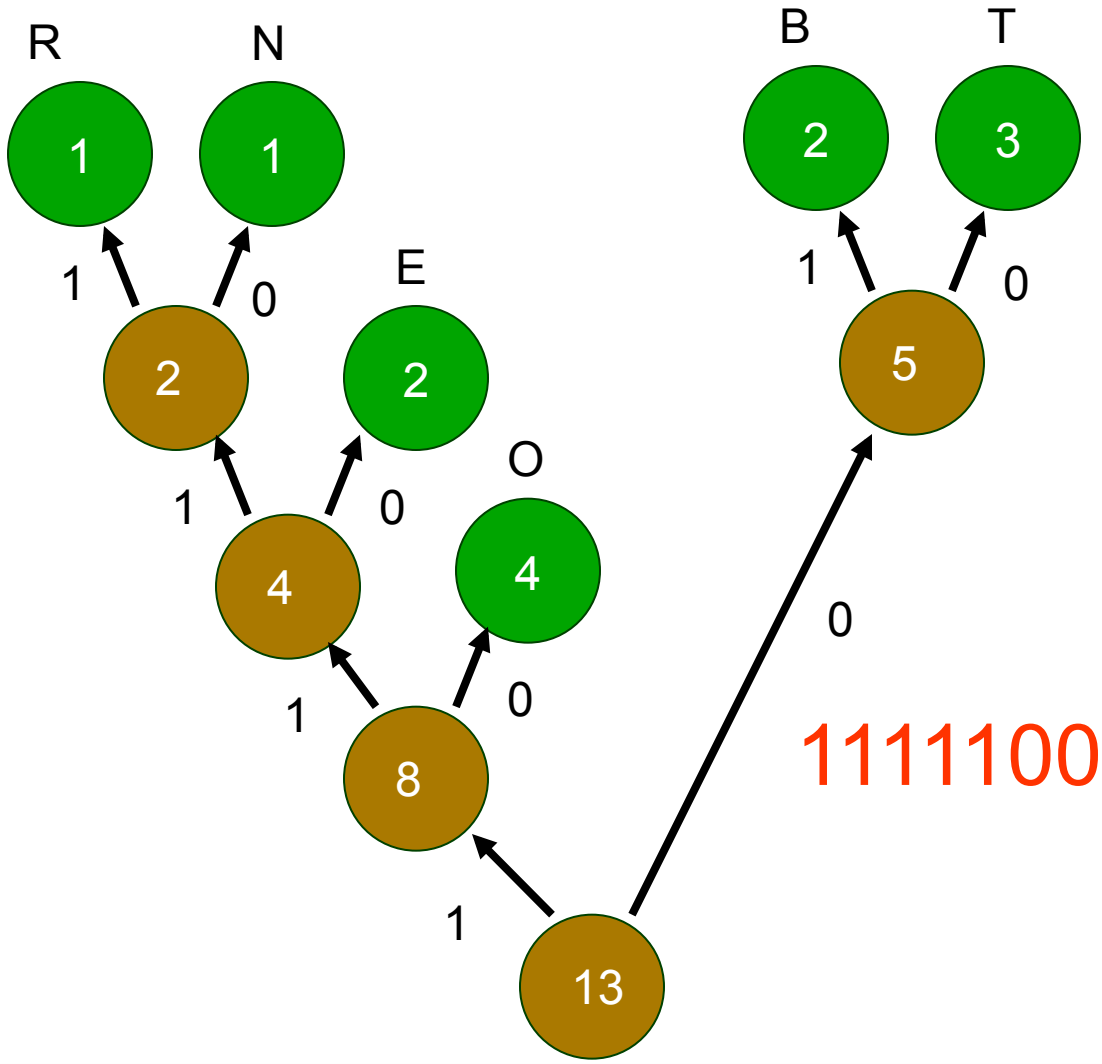
0111001110 = ?

No code is prefix of another



N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

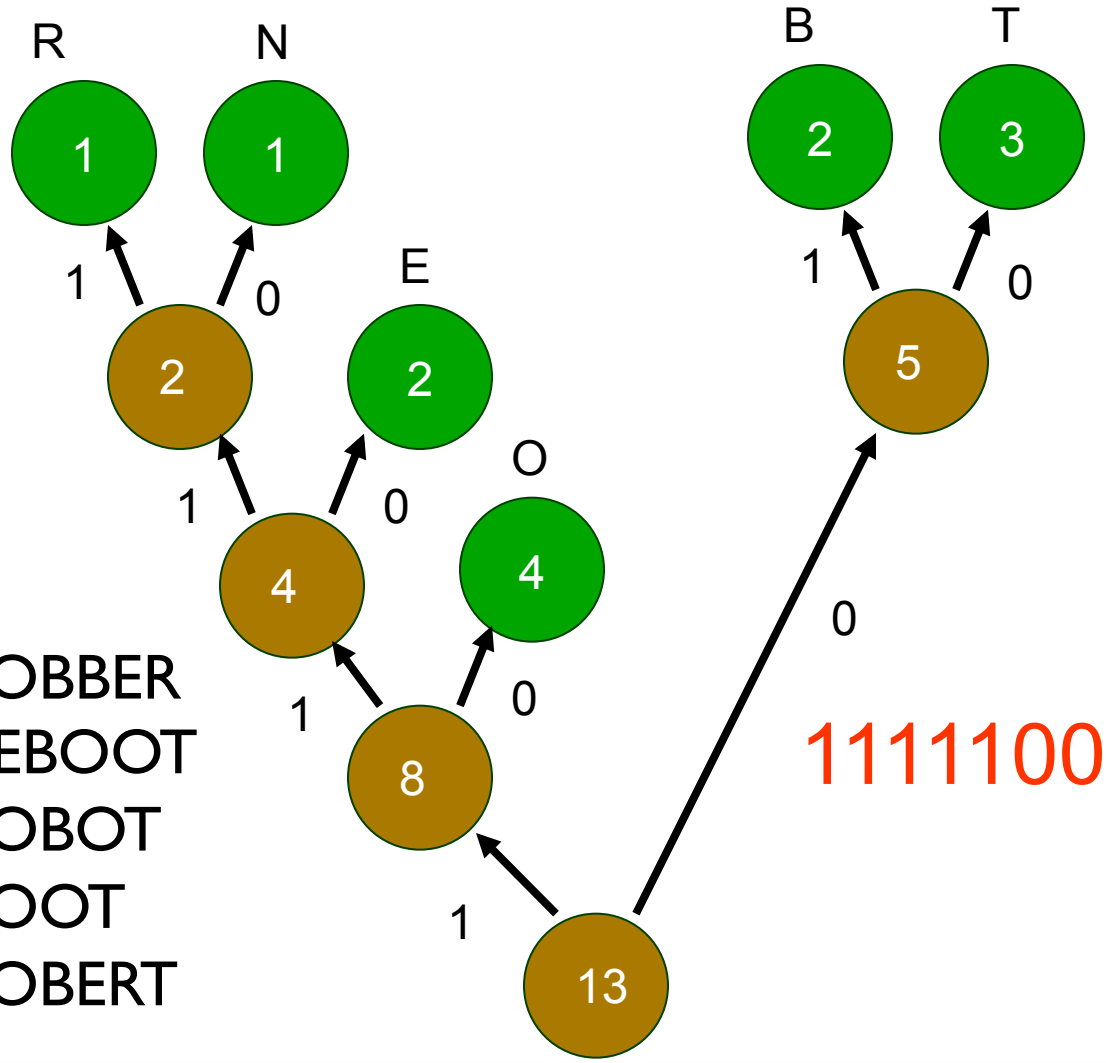
DECODING: Your turn



N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

111110011000 = ?

DECODING: Your turn



N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

A. ROBBER
B. REBOOT
C. ROBOT
D. ROOT
E. ROBERT

111110011000 = ?