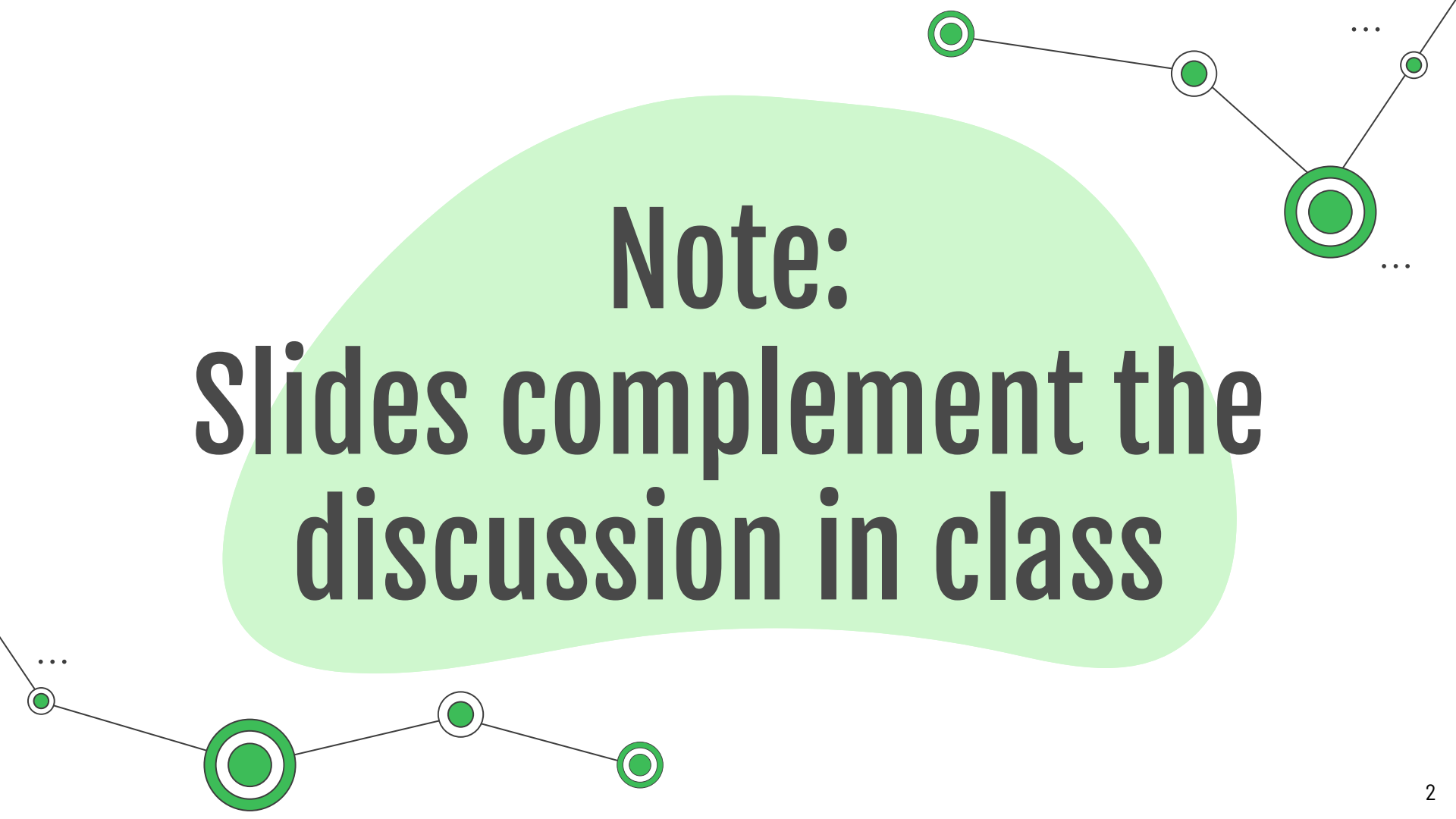




Huffman Coding

CS 251 - Data Structures and Algorithms

A decorative network diagram consisting of several green circular nodes connected by thin black lines. Some nodes are single green circles, while others are double green circles. The nodes are arranged in a non-linear fashion, with some at the top right, some at the bottom left, and one in the center. Ellipses (...) are placed near some of the nodes, suggesting a larger network. The central text is overlaid on a light green, irregularly shaped background.

Note:
**Slides complement the
discussion in class**

Table of Contents

01

Data Compression

Why it matters

02

Huffman Coding

Compressing data the greedy way






01

Data Compression

Why it matters



Why Data Compression?

01

Storage

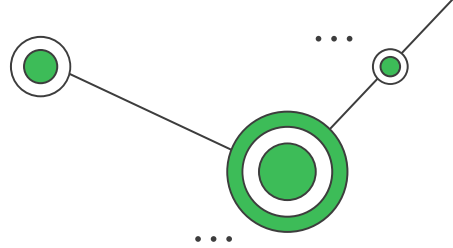
Less bits to save/store
in disk

02

Transmission

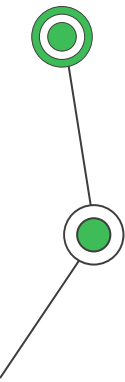
Less bits to transfer in
between devices

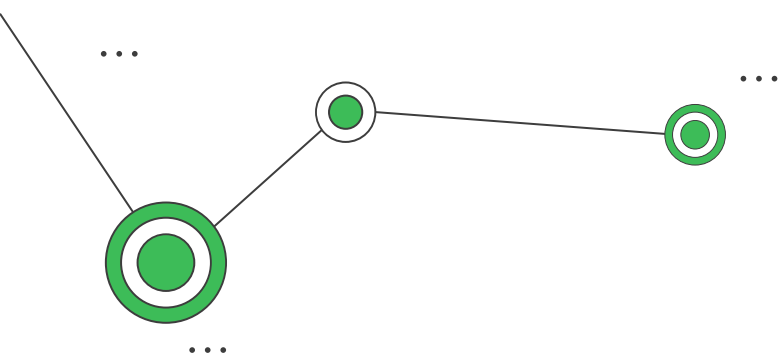
Data Compression Problem



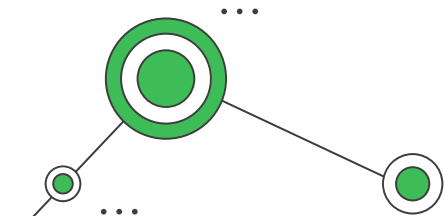
Given a string X , efficiently encode X into a shorter string Y .

How many bits does it take to encode a single character?





ASCII



- American Standard Code for Information Interchange.
- Encodes 128 specified characters into **7-bit** integers.
- **95 printable characters**: digits 0 to 9, lowercase letters a to z, uppercase letters A to Z, and punctuation symbols.
- **33 non-printing control codes** which originated with Teletype machines (obsolete).
- A unique binary string (**codeword**) per character.

Example: String Coding in C++

```
#include <iostream>
#include <bitset>

int main(int argc, char** argv)
{
    char word[] = { 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 0 };

    std::cout << word << std::endl;

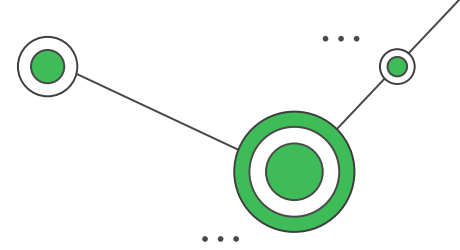
    for (char c : word)
    {
        std::cout << '\\' << c << ": " << std::bitset<8>(c) << std::endl;
    }

    return 0;
}
```

Hello World

'H':	01001000
'e':	01100101
'l':	01101100
'l':	01101100
'o':	01101111
' ':	00100000
'W':	01010111
'o':	01101111
'r':	01110010
'l':	01101100
'd':	01100100
':	00000000

Example: String Coding in Java

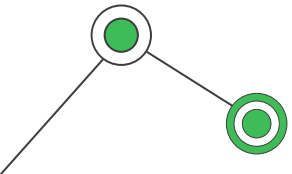


```
public class Main
{
    public static void main(String[] args)
    {
        char word[] = { 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 0 };

        System.out.println(word);

        for (char c : word)
        {
            System.out.println("'" + c + "': " + Integer.toBinaryString(c));
        }
    }
}
```

```
Hello World
'H': 1001000
'e': 1100101
'l': 1101100
'l': 1101100
'o': 1101111
' ': 100000
'W': 1010111
'o': 1101111
'r': 1110010
'l': 1101100
'd': 1100100
' ': 0
```

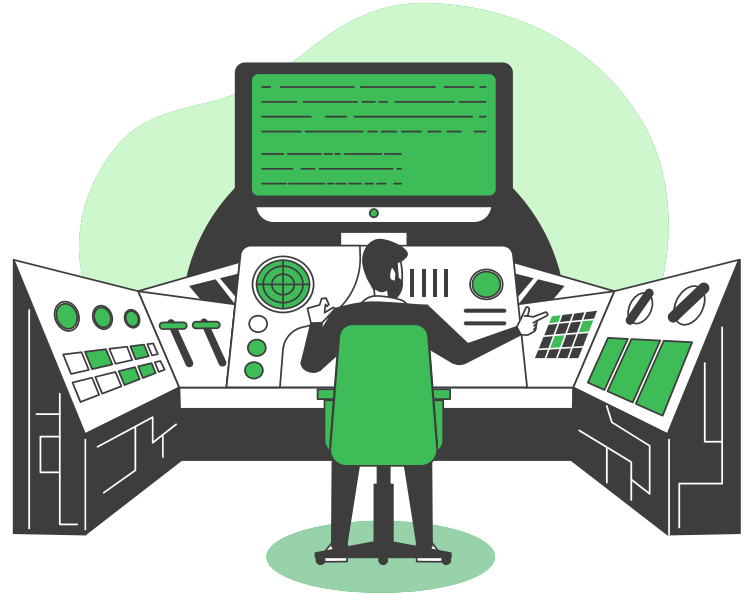


How Many Bits?

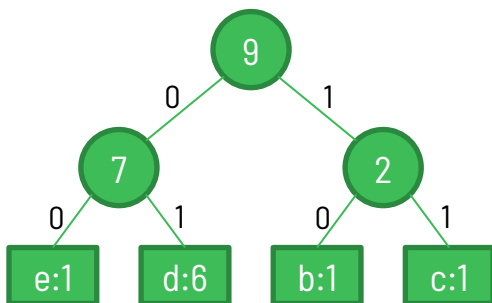
Situation: Let's encode $X = \text{"eddddddbc"}$

Options:

- Extended ASCII (8 bits per character): **72 bits**.
- Length encoding: $\text{"eddddddbc"} = \text{"e1d6b1c1"}$, then **64 bits** (still using 8 bits per character).
- Fixed-length code of 2 bits per letter: **18 bits** (Why do we need 2 bits per character?).
- Variable-length code: **14 bits** (optimal).



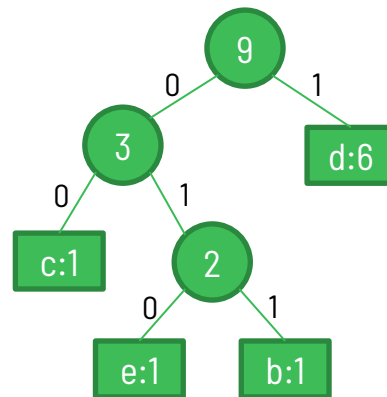
Fixed-Length Coding



Character	Codeword
e	00
d	01
b	10
c	11

Encode(eddddddbc) = 00 01 01 01 01 01 01 10 11 (18 bits)

Variable-Length Coding



Character	Codeword
e	010
d	1
b	011
c	00

Encode(eddddddbc) = 010 1 1 1 1 1 011 00 (14 bits)

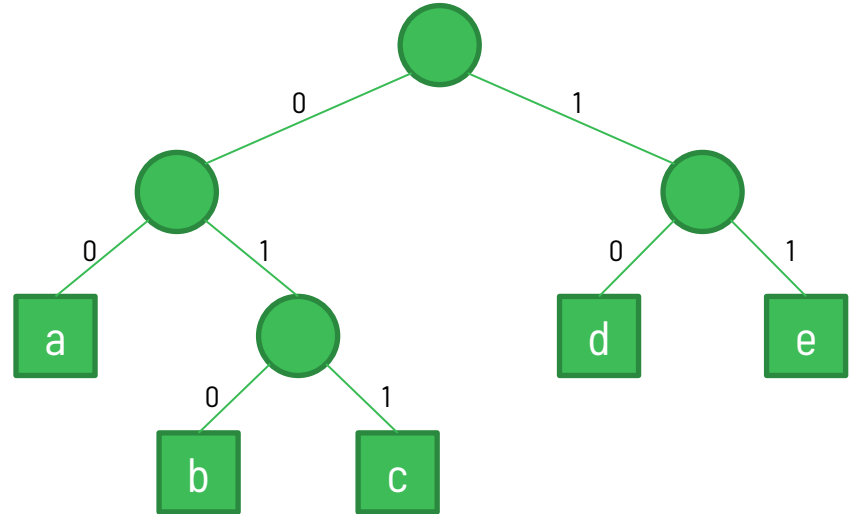
Coding Tree Definitions

A **code** is a mapping of each character of an alphabet to a binary codeword.

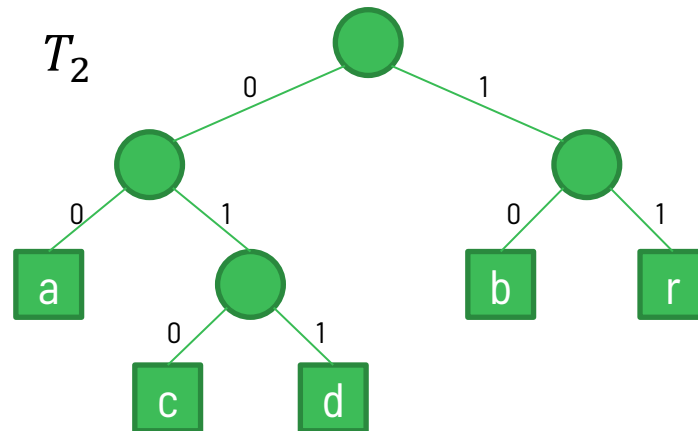
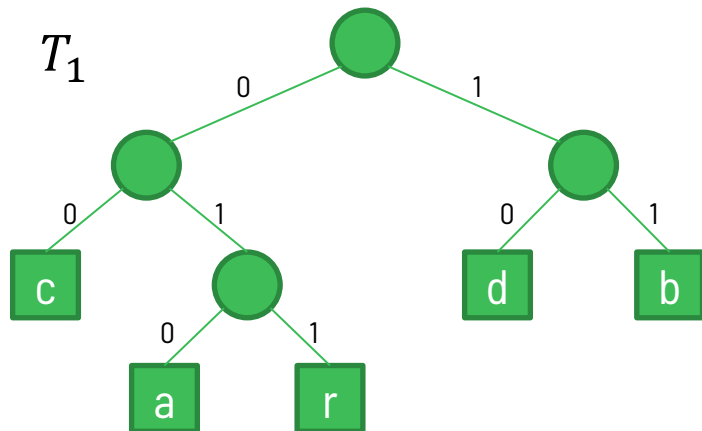
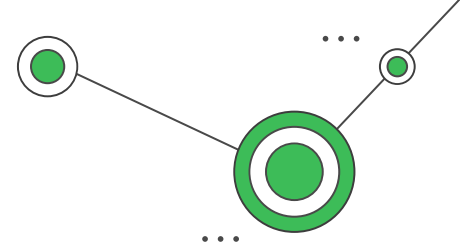
A **prefix code** is a binary code such that no code-word is the prefix of another codeword.

An **encoding tree** represents a prefix code:

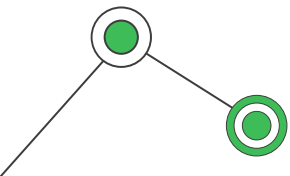
- Each external node stores a character.
- The codeword of a character is given by the path from the root to the external node storing the character (0 for a left child and 1 for a right child).



$X = \text{"abracadabra"}$



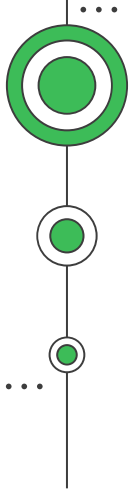
Which tree encodes X with the least number of bits?



02

Huffman Coding

Compressing data the greedy way



David A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes." Proceedings of the IRE, vol. 40, no. 9, pp. 1098-1101, 1952.

A Method for the Construction of Minimum-Redundancy Codes*

DAVID A. HUFFMAN†, ASSOCIATE, IRE

Summary—An optimum method of coding an ensemble of messages consisting of a finite number of members is developed. A minimum-redundancy code is one constructed in such a way that the average number of coding digits per message is minimized.

INTRODUCTION

ONE IMPORTANT METHOD of transmitting messages is to transmit in their place sequences of symbols. If there are more messages which might be sent than there are kinds of symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it. In this paper, the symbol or sequence of symbols associated with a given message will be called the "message code." The entire number of messages which might be transmitted will be called the "message ensemble." The mutual agreement between the transmitter and the receiver about the meaning of the code for each message of the ensemble will be called the "ensemble code."

Probably the most familiar ensemble code was stated in the phrase "one if by land and two if by sea." In this case, the message ensemble consisted of the two individual messages "by land" and "by sea," and the message codes were "one" and "two."

In order to formalize the requirements of an ensemble code, the coding symbols will be represented by numbers. Thus, if there are D different types of symbols to be used in coding, they will be represented by the digits $0, 1, 2, \dots, (D-1)$. For example, a ternary code will be constructed using the three digits $0, 1$, and 2 as coding symbols.

The number of messages in the ensemble will be called N . Let $P(i)$ be the probability of the i th message. Then

$$\sum_{i=1}^N P(i) = 1. \quad (1)$$

The length of a message, $L(i)$, is the number of coding digits assigned to it. Therefore, the average message length is

$$L_{av} = \sum_{i=1}^N P(i)L(i). \quad (2)$$

The term "redundancy" has been defined by Shannon¹ as a property of codes. A "minimum-redundancy code"

will be defined here as an ensemble code which, for a message ensemble consisting of a finite number of members, N , and for a given number of coding digits, D , yields the lowest possible average message length. In order to avoid the use of the lengthy term "minimum-redundancy," this term will be replaced here by "optimum." It will be understood then that, in this paper, "optimum code" means "minimum-redundancy code."

The following basic restrictions will be imposed on an ensemble code:

- No two messages will consist of identical arrangements of coding digits.
- The message codes will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.

Restriction (b) necessitates that no message be coded in such a way that its code appears, digit for digit, as the first part of any message code of greater length. Thus, $01, 102, 111$, and 202 are valid message codes for an ensemble of four members. For instance, a sequence of these messages 1111022020101111102 can be broken up into the individual messages $111-102-202-01-01-111-102$. All the receiver need know is the ensemble code. However, if the ensemble has individual message codes including $11, 111, 102$, and 02 , then when a message sequence starts with the digits 11 , it is not immediately certain whether the message 11 has been received or whether it is only the first two digits of the message 111 . Moreover, even if the sequence turns out to be 11102 , it is still not certain whether $111-02$ or $11-102$ was transmitted. In this example, change of one of the two message codes 111 or 11 is indicated.

C. E. Shannon¹ and R. M. Fano² have developed ensemble coding procedures for the purpose of proving that the average number of binary digits required per message approaches from above the average amount of information per message. Their coding procedures are not optimum, but approach the optimum behavior when N approaches infinity. Some work has been done by Kraft³ toward deriving a coding method which gives an average code length as close as possible to the ideal when the ensemble contains a finite number of members. However, up to the present time, no definite procedure has been suggested for the construction of such a code

* Decimal classification: R531.1. Original manuscript received by the Institute, December 6, 1951.

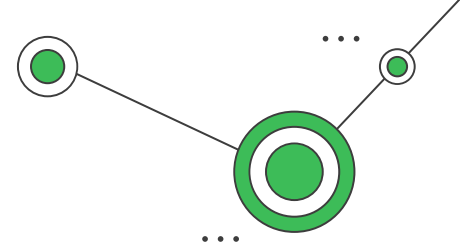
† Massachusetts Institute of Technology, Cambridge, Mass.

¹ C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. Jour.*, vol. 27, pp. 398-403; July, 1948.

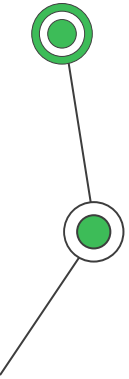
² R. M. Fano, "The Transmission of Information," Technical Report No. 65, Research Laboratory of Electronics, M.I.T., Cambridge, Mass.; 1949.

³ L. G. Kraft, "A Device for Quantizing, Grouping, and Coding Amplitude-modulated Pulses," Electrical Engineering Thesis, M.I.T., Cambridge, Mass.; 1949.

Huffman Coding



- Encode **high-frequency characters** with **short codewords**.
- No codeword is a prefix for another code word.
- Builds an optimal **coding tree** to determine the codewords.
- Huffman Coding is a **Greedy Algorithm**.





Huffman Coding Algorithm



```
algorithm HuffmanCoding( $S$ :string)
```

```
   $C \leftarrow \text{distinctCharacters}(S)$ 
```

```
   $F \leftarrow \text{computeFrequencies}(S, C)$ 
```

```
  let  $Q$  be an empty min-heap
```

```
  for each  $c \in C$  do
```

```
    let  $T$  be a new tree node
```

```
     $T.\text{char} \leftarrow c$ 
```

```
     $T.\text{freq} \leftarrow F[c]$ 
```

```
     $Q.\text{insert}(F[c], T)$ 
```

```
  end for
```

```
  while  $Q.\text{size}() > 1$  do
```

```
    let  $T$  be a new tree node
```

```
     $T.\text{left} \leftarrow Q.\text{getMin}()$ 
```

```
     $T.\text{right} \leftarrow Q.\text{getMin}()$ 
```

```
     $T.\text{freq} \leftarrow T.\text{left}.\text{freq} + T.\text{right}.\text{freq}$ 
```

```
     $Q.\text{insert}(T.\text{freq}, T)$ 
```

```
  end while
```

```
  return  $Q.\text{getMin}()$ 
```

```
end algorithm
```

algorithm HuffmanCoding(S :string)

```
 $C \leftarrow \text{distinctCharacters}(S)$   
 $F \leftarrow \text{computeFrequencies}(S, C)$   
let  $Q$  be an empty min-heap
```

```
for each  $c \in C$  do  
    let  $T$  be a new tree node  
     $T.\text{char} \leftarrow c$   
     $T.\text{freq} \leftarrow F[c]$   
     $Q.\text{insert}(F[c], T)$   
end for
```

```
while  $Q.\text{size}() > 1$  do  
    let  $T$  be a new tree node  
     $T.\text{left} \leftarrow Q.\text{getMin}()$   
     $T.\text{right} \leftarrow Q.\text{getMin}()$   
     $T.\text{freq} \leftarrow T.\text{left}.\text{freq} + T.\text{right}.\text{freq}$   
     $Q.\text{insert}(T.\text{freq}, T)$   
end while
```

```
    return  $Q.\text{getMin}()$   
end algorithm
```

Let n be the length of S . Then:

} $O(n)$

} $O(n \log(n))$

} $O(n \log(n))$

Huffman Coding is $O(n \log(n))$.

Note: Huffman Coding is $O(n \log \log(n))$ if we replace the min-heap with a van Emde Boas tree.

1) Build a Huffman Coding tree using the following frequency table:

c	E	D	C	O	A	G
$f(c)$	17	10	5	3	15	6

```
algorithm HuffmanCoding( $S$ :string)
```

```
     $C \leftarrow \text{distinctCharacters}(S)$ 
```

```
     $F \leftarrow \text{computeFrequencies}(S, C)$ 
```

```
    let  $Q$  be an empty min-heap
```

```
    for each  $c \in C$  do
```

```
        let  $T$  be a new tree node
```

```
         $T.\text{char} \leftarrow c$ 
```

```
         $T.\text{freq} \leftarrow F[c]$ 
```

```
         $Q.\text{insert}(F[c], T)$ 
```

```
    end for
```

```
    while  $Q.\text{size}() > 1$  do
```

```
        let  $T$  be a new tree node
```

```
         $T.\text{left} \leftarrow Q.\text{getMin}()$ 
```

```
         $T.\text{right} \leftarrow Q.\text{getMin}()$ 
```

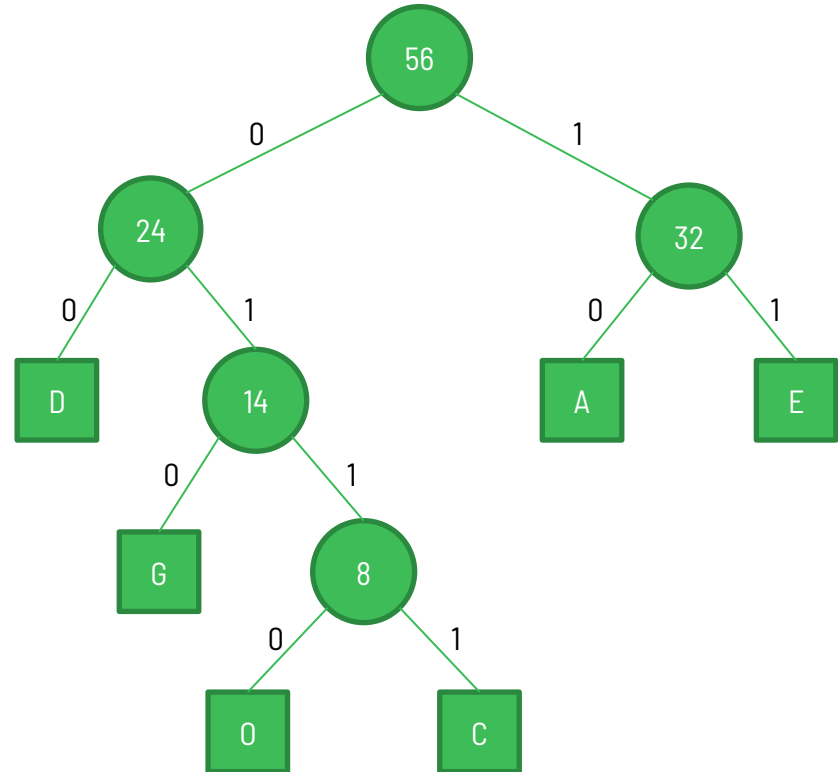
```
         $T.\text{freq} \leftarrow T.\text{left}.\text{freq} + T.\text{right}.\text{freq}$ 
```

```
         $Q.\text{insert}(T.\text{freq}, T)$ 
```

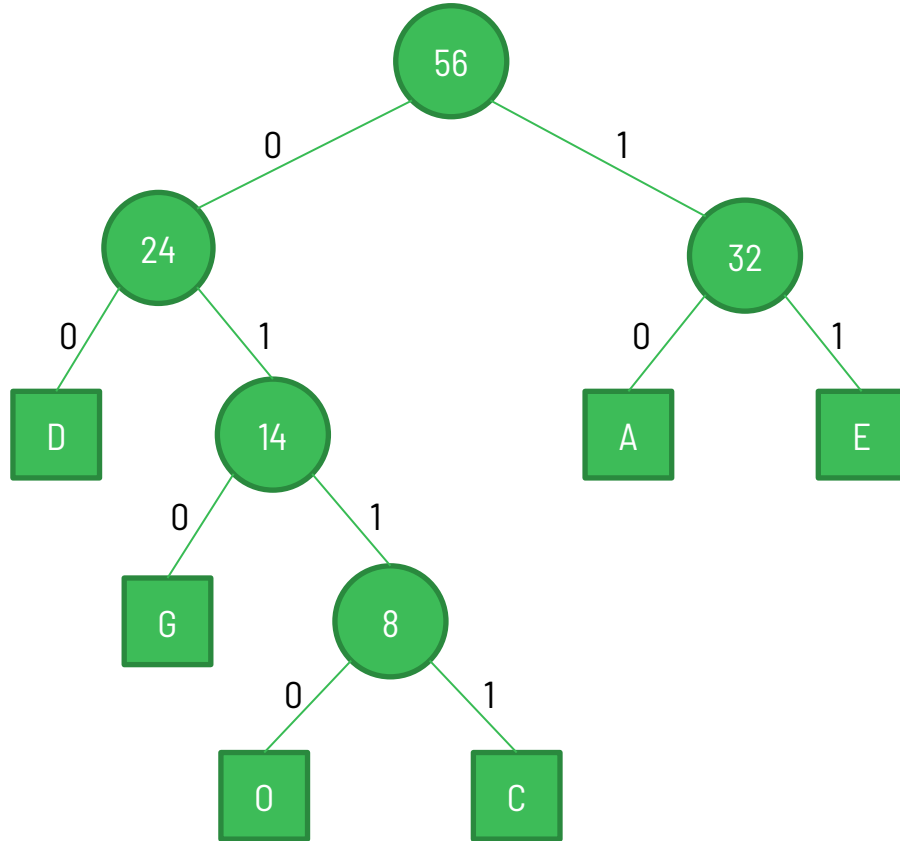
```
    end while
```

```
    return  $Q.\text{getMin}()$ 
```

```
end algorithm
```



2) Use the tree to encode the words DEED, CODE, DECADE, and GEODA.



Character	Codeword
E	11
D	00
C	0111
O	0110
A	10
G	010

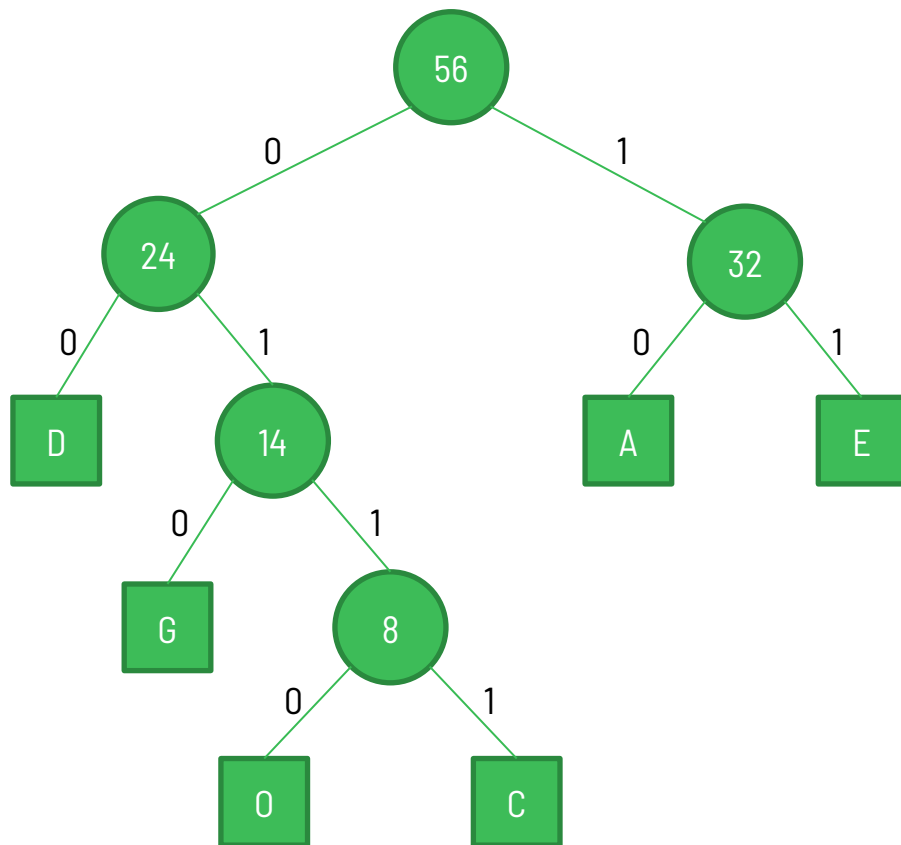
DEED = 00 11 11 00

CODE = 0111 0110 00 11

DECADE = 00 11 0111 10 00 11

GEODA = 010 11 0110 00 10

3) Use the tree to decode the bit-strings 01111001011, 0100110011000, 0011011101100011, and 11010010.



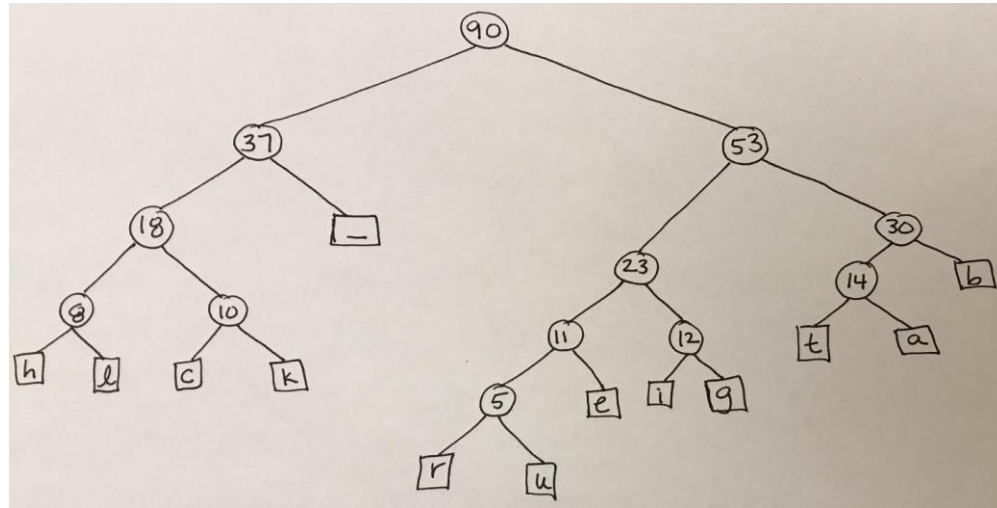
Character	Codeword
E	11
D	00
C	0111
O	0110
A	10
G	010

0111 10 010 11 = CAGE
 010 0110 0110 00 = GOOD
 00 11 0111 0110 00 11 = DECODE
 11 010 010 = EGG

For You To Practice

X = the big black bug bit the big black bear but the big black bear bit the big black bug back

c	f(c)	c	f(c)
t	7	a	7
h	4	c	5
e	6	k	5
b	16	u	3
i	6	-	19
g	6	r	2
l	4		



Disclaimer: results may vary depending on how you handle “ties” and which order you join trees.

Slides Go 111100001000010000

Do you have any questions?

CREDITS: This presentation template was created by [Slidesgo](#), including icons by [Flaticon](#), infographics & images by [Freepik](#) and illustrations by [Stories](#)