UNIT-2

FUNDAMENTALS OF INFORMATION TECHNOLOGY

System software

System software is a type of computer program that is designed to run a computer's hardware and application programs. If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications. The operating system is the best-known example of system software. The OS manages all the other programs in a computer.

System software is used to manage the computer itself. It runs in the background, maintaining the computer's basic functions so users can run higher-level application software to perform certain tasks. Essentially, system software provides a platform for application software to be run on top of.

Important features of system software

Computer manufacturers usually develop the system software as an integral part of the computer. The primary responsibility of this software is to create an interface between the computer hardware they manufacture and the end user.

System software generally includes the following features:

1. **High speed.** System software must be as efficient as possible to provide an effective platform for higher-level software in the computer system.
2. **Hard to manipulate.** It often requires the use of a programming language, which is more difficult to use than a more intuitive user interface (UI).

3. **Written in a low-level computer language.** System software must be written in a computer language the central processing unit (CPU) and other computer hardware can read.
4. **Close to the system.** It connects directly to the hardware that enables the computer to run.
5. **Versatile.** System software must communicate with both the specialized hardware it runs on and the higher-level application software that is usually hardware-agnostic and often has no direct connection to the hardware it runs on. System software also must support other programs that depend on it as they evolve and change

Assembler

6. An assembler is a type of computer program that interprets software programs written in assembly language into machine language, code and instructions that can be executed by a computer.
7. An assembler enables software and application developers to access, operate and manage a computer's hardware architecture and components.
8. An assembler is sometimes referred to as the compiler of assembly language. It also provides the services of an interpreter.

Compiler and Interpreter

Compliers and interpreters are programs that help convert the high level language (Source Code) into machine codes to be understood by the computers. Computer programs are usually written on high level languages. A high level language is one that can be understood by humans. To make it clear, they contain words and phrases from the languages in common use – English or other languages for example. However, computers cannot understand high level languages as we humans do. They can only understand the programs that are developed in binary systems known as a machine code. To start with, a computer program is usually written in high level language described as a source code. These source codes must be converted into machine language and here comes the role of compilers and interpreters.

Difference between Compiler and Interpreter

Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.
An interpreter takes very less time to analyse the source code. However, the overall time to execute the process is much slower.	A compiler takes a lot of time to analyse the source code. However, the overall time taken to execute the process is much faster.
An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.
Keeps translating the program continuously till the first error is confronted. If any error is spotted,	A compiler generates the error message only after it scans the complete program and hence

it stops working and hence debugging becomes easy.	debugging is relatively harder while working with a compiler.
Interpreters are used by programming languages like Ruby and Python for example.	Compliers are used by programming languages like C and C++ for example.

Advantages and disadvantages of Interpreter and Compiler

In case of using compilers, the program codes are translated into machine code already and hence the time to execute the code is very less. On the negative side, it is not possible to change the program without going back to the source code while working with a compiler.

Interpreters make working with the source code much easier. Hence they are highly suitable especially for the beginners. On the negative side, interpreted programs can only run on the computers that have the respective interpreters.

Data

Data is a raw and unorganized fact that required to be processed to make it meaningful. Data can be simple at the same time unorganized unless it is organized. Generally, data comprises facts, observations, perceptions numbers, characters, symbols, image, etc.

Data is always interpreted, by a human or machine, to derive meaning. So, data is meaningless. Data contains numbers, statements, and characters in a raw form.

Information

Information is a set of data which is processed in a meaningful way according to the given requirement. Information is processed, structured, or presented in a given context to make it meaningful and useful.

It is processed data which includes data that possess context, relevance, and purpose. It also involves manipulation of raw data.

Information assigns meaning and improves the reliability of the data. It helps to ensure undesirability and reduces uncertainty. So, when the data is transformed into information, it never has any useless details.

KEY DIFFERENCE

- Data is a raw and unorganized fact that is required to be processed to make it meaningful whereas Information is a set of data that is processed in a meaningful way according to the given requirement.
- Data does not have any specific purpose whereas Information carries a meaning that has been assigned by interpreting data.
- Data alone has no significance while Information is significant by itself.
- Data never depends on Information while Information is dependent on Data.
- Data measured in bits and bytes, on the other hand, Information is measured in meaningful units like time, quantity, etc.
- Data can be structured, tabular data, graph, data tree whereas Information is language, ideas, and thoughts based on the given data.

Difference between Information and Data

Parameters	**Data**	**Information**
Description	Qualitative Or Quantitative Variables which helps to develop ideas or conclusions.	It is a group of data which carries news and meaning.
Etymology	Data comes from a Latin word, datum, which means “To give something.” Over a time “data” has become the plural of datum.	Information word has old French and middle English origins. It has referred to the “act of informing.”. It is mostly used for education or other known communication.
Format	Data is in the form of numbers, letters, or a set of characters.	Ideas and inferences
Represented in	It can be structured, tabular data, graph, data tree, etc.	Language, ideas, and thoughts based on the given data.
Meaning	Data does not have any specific purpose.	It carries meaning that has been assigned by interpreting data.
Interrelation	Information that is collected	Information that is processed.
Feature	Data is a single unit and is raw. It alone doesn’t have any meaning.	Information is the product and group of data which jointly carry a logical meaning.
Dependence	It never depends on Information	It depended on Data.
Measuring unit	Measured in bits and bytes.	Measured in meaningful units like time, quantity, etc.
Support for Decision making	It can’t be used for decision making	It is widely used for decision making.
Contains	Unprocessed raw factors	Processed in a meaningful way

Knowledge level	It is low-level knowledge.	It is the second level of knowledge.
Characteristic	Data is the property of an organization and is not available for sale to the public.	Information is available for sale to the public.
Dependency	Data depends upon the sources for collecting data.	Information depends upon data.
Example	Ticket sales on a band on tour.	Sales report by region and venue. It gives information which venue is profitable for that business.
Significance	Data alone has no significance.	Information is significant by itself.
Meaning	Data is based on records and observations and, which are stored in computers or remembered by a person.	Information is considered more reliable than data. It helps the researcher to conduct a proper analysis.
Usefulness	The data collected by the researcher, may or may not be useful.	Information is useful and valuable as it is readily available to the researcher for use.
Dependency	Data is never designed to the specific need of the user.	Information is always specific to the requirements and expectations because all the irrelevant facts and figures are removed, during the transformation process.

Number Systems

A number is a mathematical value used for counting and measuring objects, and for performing arithmetic calculations. Numbers have various categories like natural numbers, whole numbers, rational and irrational numbers, and so on. Similarly, there are various types of number systems that have different properties, like the binary number system, the octal number system, the decimal number system, and the hexadecimal number system.

What are Number Systems?

A number system is a system representing numbers. It is also called the system of numeration and it defines a set of values to represent a quantity. These numbers are used as digits and the most common ones are 0 and 1, that are used to represent binary numbers. Digits from 0 to 9 are used to represent other types of number systems.

Definition of Number Systems

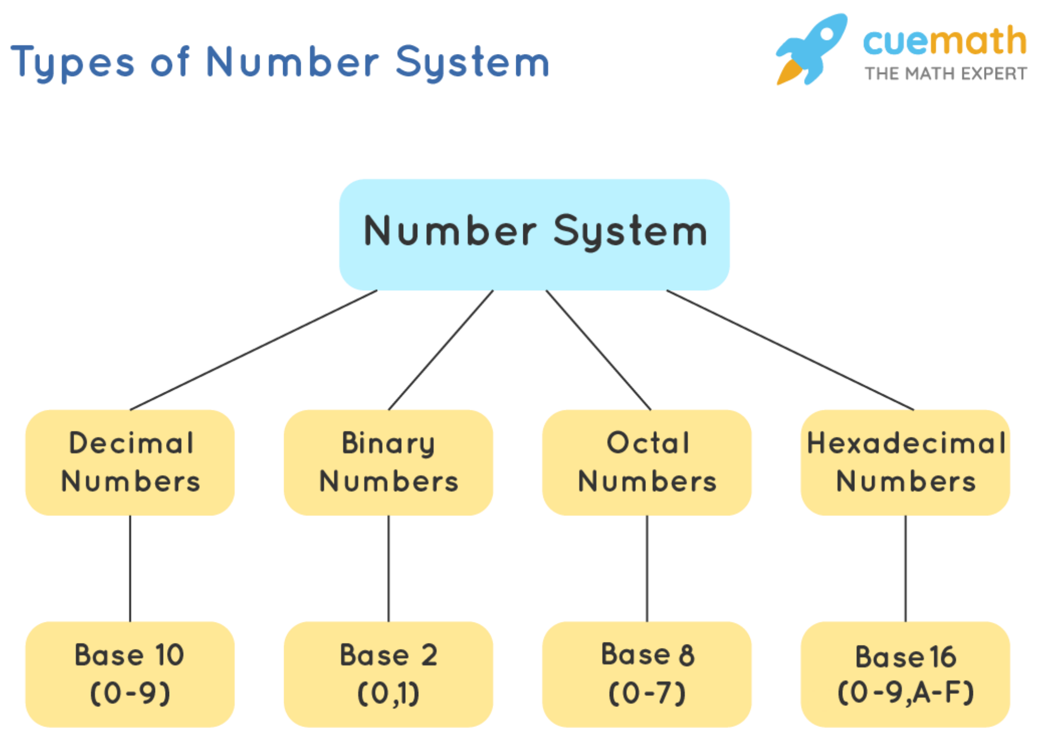
A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner. The value of any digit in a number can be determined by a digit, its position in the number, and the base of the number system. The numbers are represented in a unique manner and allow us to operate arithmetic operations like addition, subtraction, and division.

Types of Number Systems

There are different types of number systems in which the four main types are:

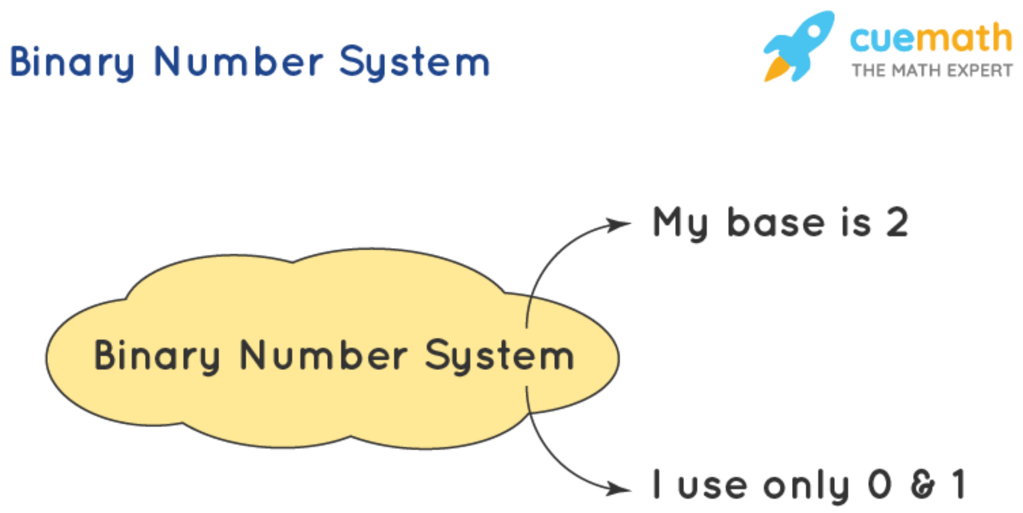
- Binary number system (Base - 2)
- Octal number system (Base - 8)
- Decimal number system (Base - 10)
- Hexadecimal number system (Base - 16)

We will study each of these systems one by one in detail.



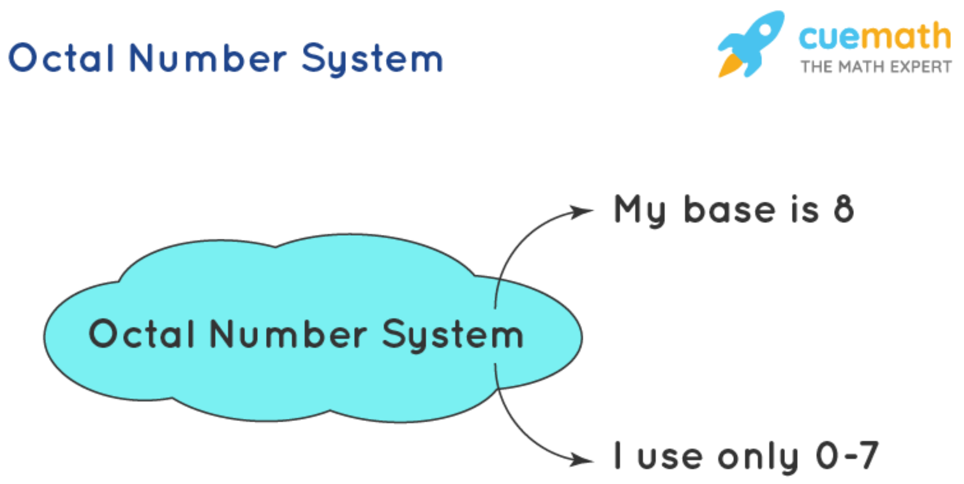
Binary Number System

The binary number system uses only two digits: 0 and 1. The numbers in this system have a base of 2. Digits 0 and 1 are called bits and 8 bits together make a byte. The data in computers is stored in terms of bits and bytes. The binary number system does not deal with other numbers such as 2,3,4,5 and so on. For example: 10001_2, 111101_2, 1010101_2 are some examples of numbers in the binary number system.



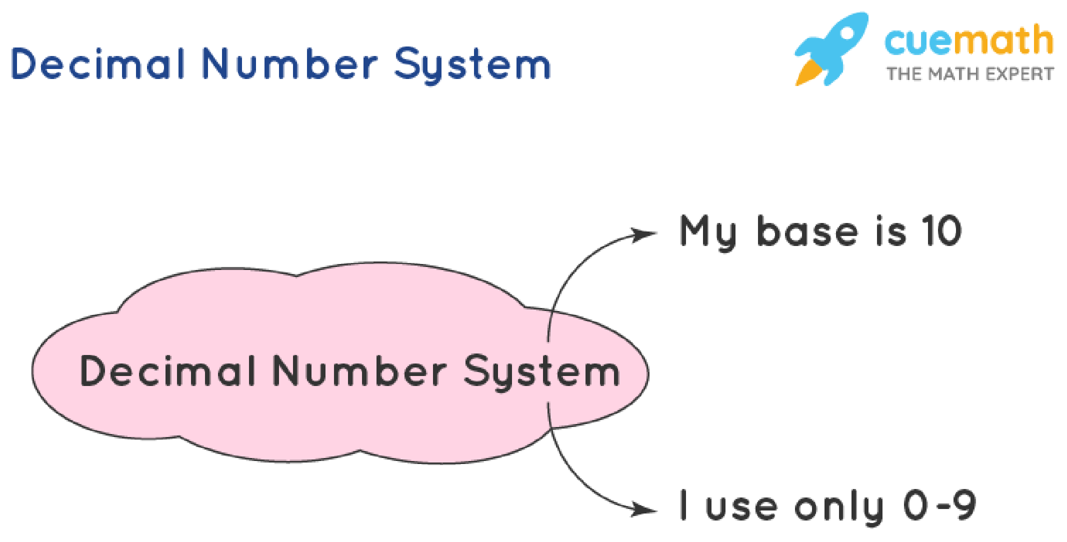
Octal Number System

The octal number system uses eight digits: 0,1,2,3,4,5,6 and 7 with the base of 8. The advantage of this system is that it has lesser digits when compared to several other systems, hence, there would be fewer computational errors. Digits like 8 and 9 are not included in the octal number system. Just as the binary, the octal number system is used in minicomputers but with digits from 0 to 7. For example: 35_8, 23_8, 141_8 are some examples of numbers in the octal number system.



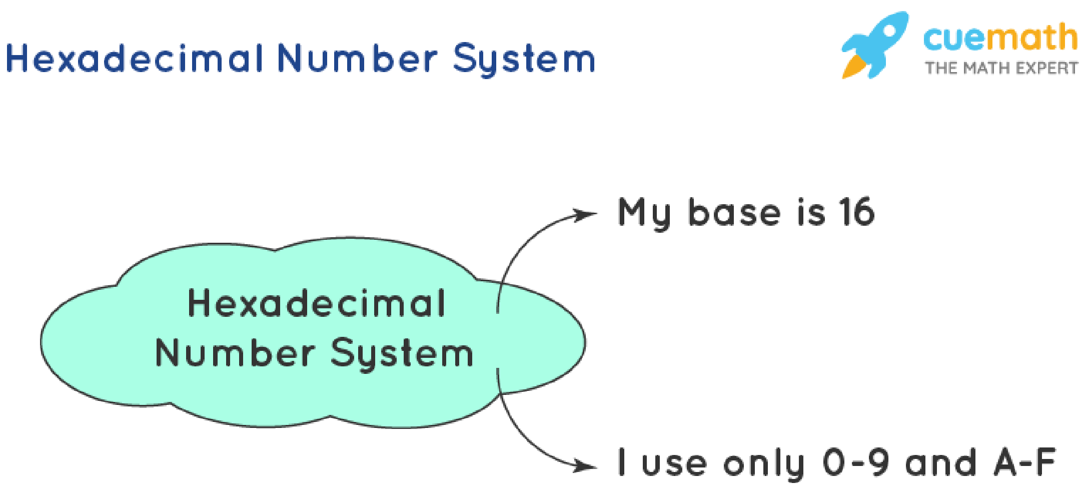
Decimal Number System

The decimal number system uses ten digits: 0,1,2,3,4,5,6,7,8 and 9 with the base number as 10. The decimal number system is the system that we generally use to represent numbers in real life. If any number is represented without a base, it means that its base is 10. For example: 723_{10}, 32_{10}, 4257_{10} are some examples of numbers in the decimal number system.



Hexadecimal Number System

The hexadecimal number system uses sixteen digits/alphabets: 0,1,2,3,4,5,6,7,8,9 and A,B,C,D,E,F with the base number as 16. Here, A-F of the hexadecimal system means the numbers 10-15 of the decimal number system respectively. This system is used in computers to reduce the large-sized strings of the binary system. For example: $7B3_{16}$, $6F_{16}$, $4B2A_{16}$ are some examples of numbers in the hexadecimal number system.



Conversion Rules of Number Systems

A number can be converted from one number system to another number system. Like binary numbers can be converted to octal numbers and vice versa, octal numbers can be converted to decimal numbers and vice versa and so on. Let us see the steps required in converting number systems.

Conversion of Binary / Octal / Hexadecimal Number Systems to Decimal Number System

To convert a number from the binary/octal/hexadecimal system to the decimal system, we use the following steps. The steps are shown by an example of a number in the binary system.

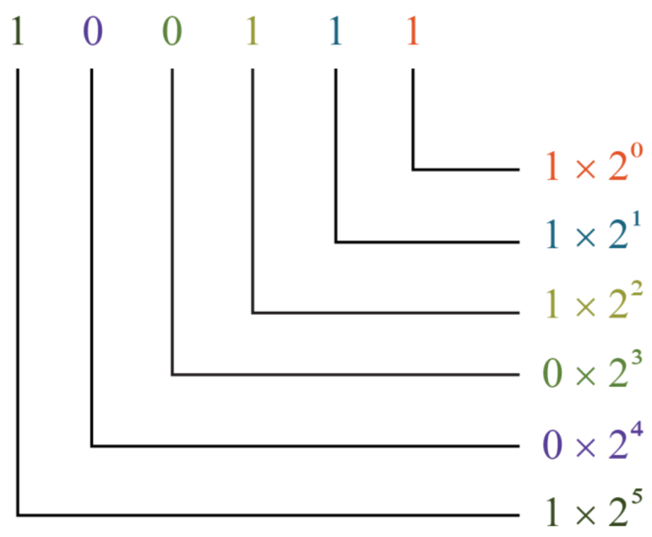
Example: Convert 100111_2 into the decimal system.

Solution:

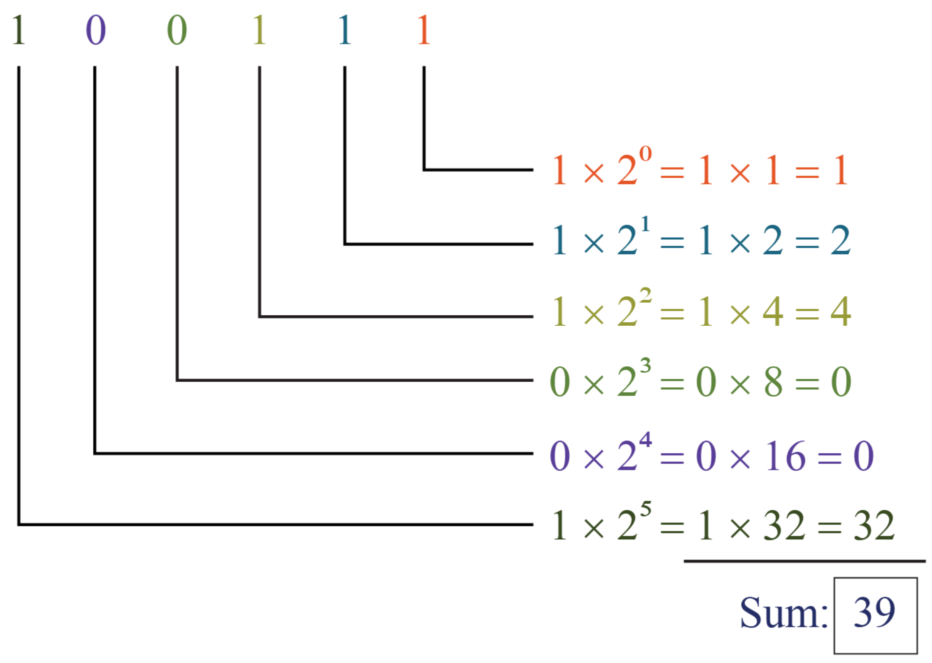
Step 1: Identify the base of the given number. Here, the base of 100111_2 is 2.

Step 2: Multiply each digit of the given number, starting from the rightmost digit, with the exponents of the base. The exponents should start with 0 and increase by

1 every time as we move from right to left. Since the base is 2 here, we multiply the digits of the given number by 2^0, 2^1, 2^2 , and so on from right to left.



Step 3: We just simplify each of the above products and add them.


1 0 0 1 1 1
$1 \times 2^0 = 1 \times 1 = 1$
$1 \times 2^1 = 1 \times 2 = 2$
$1 \times 2^2 = 1 \times 4 = 4$
$0 \times 2^3 = 0 \times 8 = 0$
$0 \times 2^4 = 0 \times 16 = 0$
$1 \times 2^5 = 1 \times 32 = 32$
Sum: $\boxed{39}$

Here, the sum is the equivalent number in the decimal number system of the given number. Or, we can use the following steps to make this process simplified.

$100111 = (1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0)$

$= (1×32) + (0×16) + (0×8) + (1×4) + (1×2) + (1×1)$

$= 32 + 0 + 0 + 4 + 2 + 1$

$= 39$

Thus, $100111_2 = 39_{10}$.

Conversion of Decimal Number System to Binary / Octal / Hexadecimal Number System

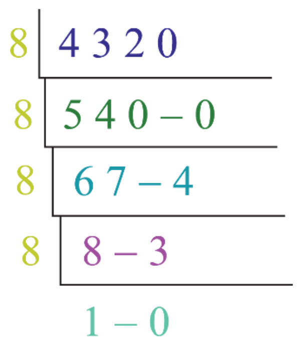
To convert a number from the decimal number system to binary/octal/hexadecimal number system, we use the following steps. The steps are shown on how to convert a number from the decimal system to the octal system.

Example: Convert 4320_{10} into the octal system.

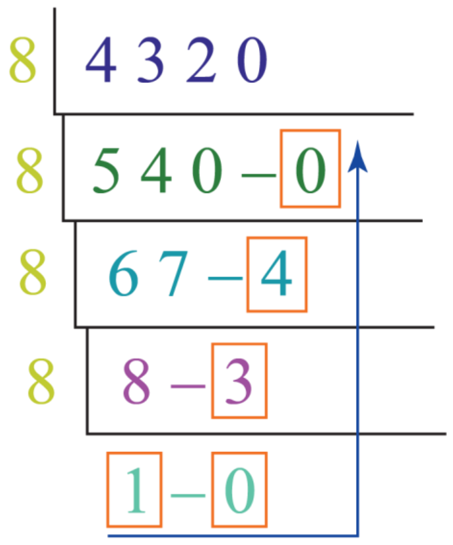
Solution:

Step 1: Identify the base of the required number. Since we have to convert the given number into the octal system, the base of the required number is 8.

Step 2: Divide the given number by the base of the required number and note down the quotient and the remainder in the quotient-remainder form. Repeat this process (dividing the quotient again by the base) until we get the quotient less than the base.



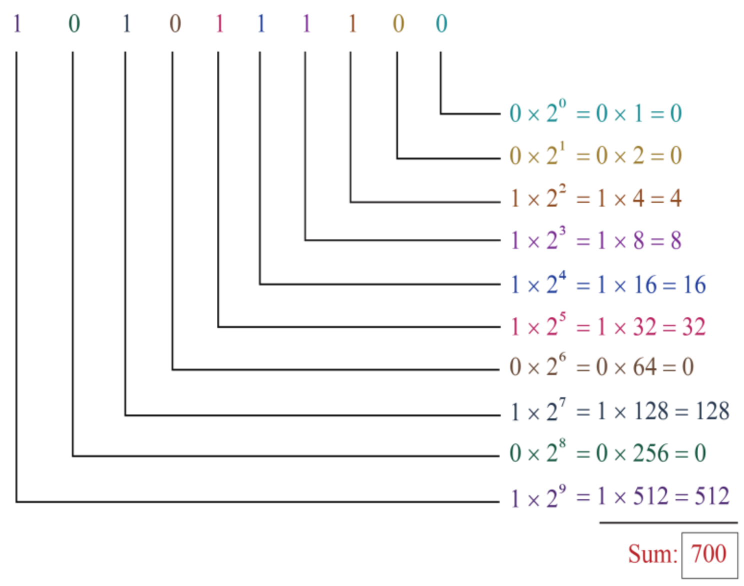
Step 3: The given number in the octal number system is obtained just by reading all the remainders and the last quotient from bottom to top.



Therefore, $4320_{10} = 10340_8$.

Conversion from One Number System to Another Number System

To convert a number from one of the binary/octal/hexadecimal systems to one of the other systems, we first convert it into the decimal system, and then we convert it to the required systems by using the above-mentioned processes.


1 0 1 0 1 1 1 1 0 0
$0 \times 2^0 = 0 \times 1 = 0$
$0 \times 2^1 = 0 \times 2 = 0$
$1 \times 2^2 = 1 \times 4 = 4$
$1 \times 2^3 = 1 \times 8 = 8$
$1 \times 2^4 = 1 \times 16 = 16$
$1 \times 2^5 = 1 \times 32 = 32$
$0 \times 2^6 = 0 \times 64 = 0$
$1 \times 2^7 = 1 \times 128 = 128$
$0 \times 2^8 = 0 \times 256 = 0$
$1 \times 2^9 = 1 \times 512 = 512$
Sum: 700

Example: Convert 1010111100_2 to the hexadecimal system.

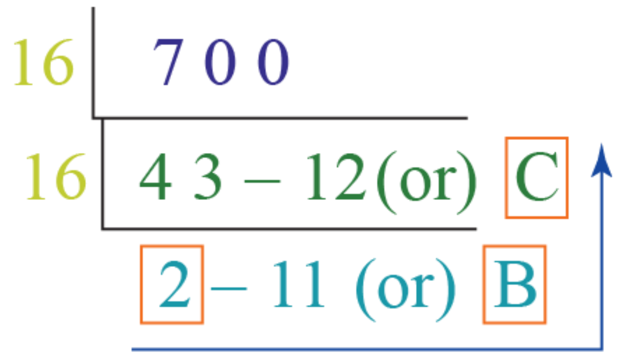
Solution:

Step 1: Convert this number to the decimal number system as explained in the above process.

Thus, $1010111100_2 = 700_{10} \rightarrow (1)$.

Step 2: Convert the above number (which is in the decimal system), into the required number system.

Here, we have to convert 700_{10} into the hexadecimal system using the above-mentioned process. It should be noted that in the hexadecimal system, the numbers 11 and 12 are written as B and C respectively.



Thus, $700_{10} = 2BC_{16} \rightarrow (2)$.

From the equations (1) and (2), $1010111100_2 = 2BC_{16}$.

Assignment-1

Solve the following question

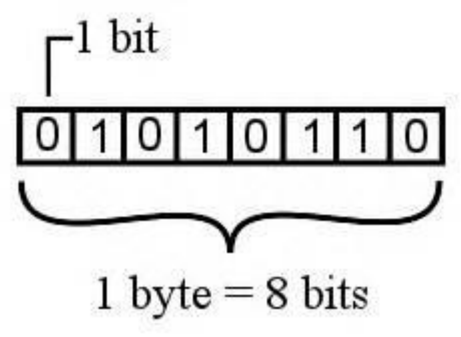
1. **Convert the binary number 11001 to decimal.**
2. **Convert the decimal number 45 to binary.**
3. **Convert the hexadecimal number B2 to binary.**
4. **Convert the binary number 11011 to hexadecimal**
5. **Convert the decimal number 20 to hexadecimal.**
6. **Convert the hexadecimal number 2C to decimal.**
7. **Convert the binary number 10101100 to its decimal equivalent**
8. **Convert the binary number 10101100 to its decimal equivalent.**
9. **Convert the hexadecimal number 0x2301 to its binary equivalent.**
10. **Covert the binary number 11010010 to a decimal number.**

TEXT REPRESENTATION

Bits and Bytes

Bits − A bit is a smallest possible unit of data that a computer can recognize or use. Computer usually uses bits in groups.

Bytes – group of eight bits is called a byte. Half a byte is called a nibble.



The following table shows conversion of Bits and Bytes –

Byte Value	**Bit Value**
1 Byte	8 Bits
1024 Bytes	1 Kilobyte
1024 Kilobytes	1 Megabyte
1024 Megabytes	1 Gigabyte

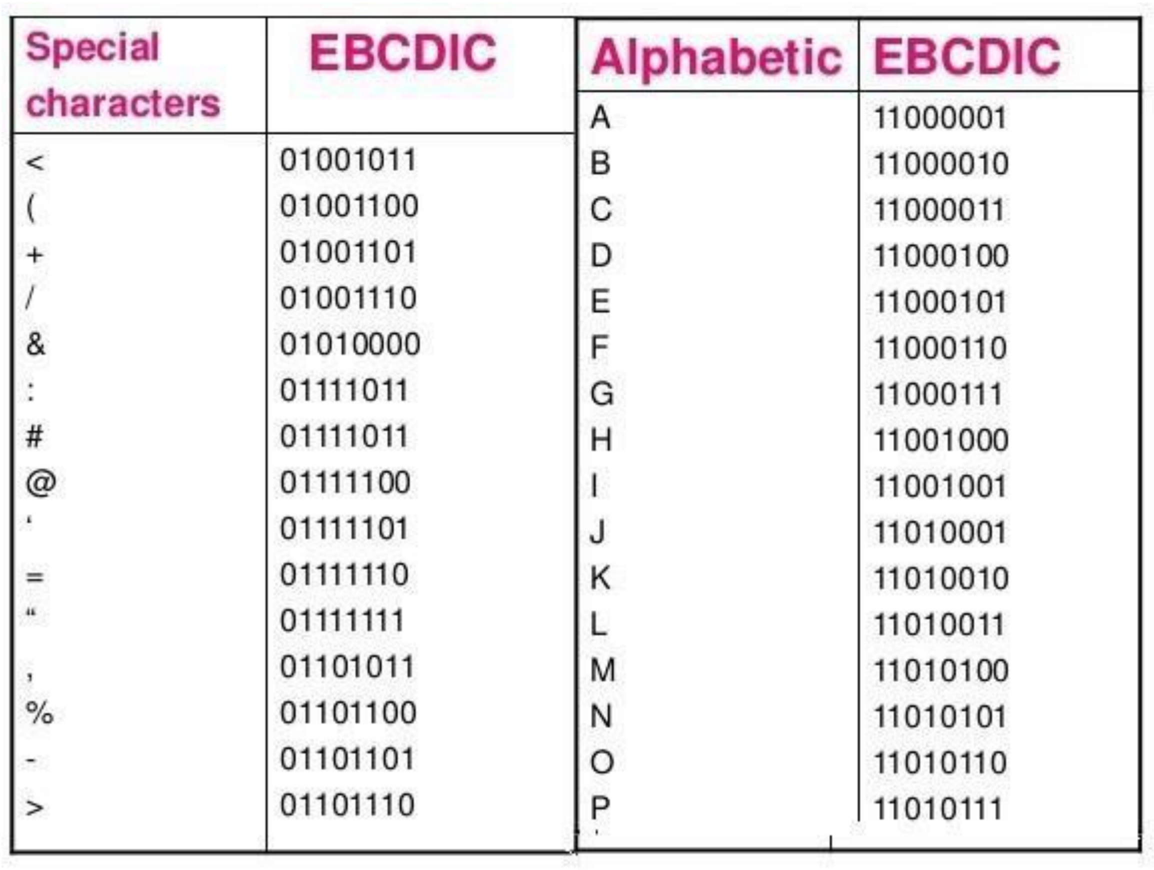
Text Code

Text code is format used commonly to represent alphabets, punctuation marks and other symbols. Four most popular text code systems are –

- EBCDIC
- ASCII
- Extended ASCII
- Unicode

EBCDIC

Extended Binary Coded Decimal Interchange Code is an 8-bit code that defines 256 symbols. Given below is the EBCDIC **Tabular column**



Special characters	EBCDIC	Alphabetic	EBCDIC
<	01001011	A	11000001
(	01001100	B	11000010
+	01001101	C	11000011
/	01001110	D	11000100
&	01010000	E	11000101
:	01111011	F	11000110
#	01111011	G	11000111
@	01111100	H	11001000
‘	01111101	I	11001001
=	01111110	J	11010001
“	01111111	K	11010010
,	01101011	L	11010011
%	01101100	M	11010100
-	01101101	N	11010101
>	01101110	O	11010110
		P	11010111

ASCII

American Standard Code for Information Interchange is an 8-bit code that specifies character values from 0 to 127.

ASCII Tabular column

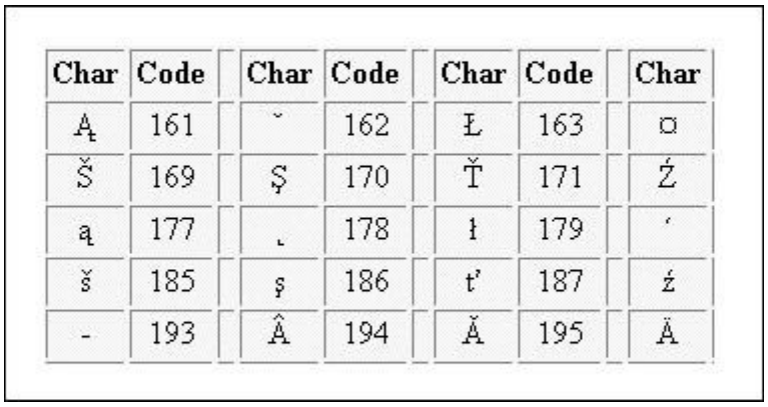
ASCII Code	Decimal Value	Character
0000 0000	0	Null prompt

0000 0001	1	Start of heading
0000 0010	2	Start of text
0000 0011	3	End of text
0000 0100	4	End of transmit
0000 0101	5	Enquiry
0000 0110	6	Acknowledge
0000 0111	7	Audible bell
0000 1000	8	Backspace
0000 1001	9	Horizontal tab
0000 1010	10	Line Feed

Extended ASCII

Extended American Standard Code for Information Interchange is an 8-bit code that specifies character values from 128 to 255.

Extended ASCII Tabular column

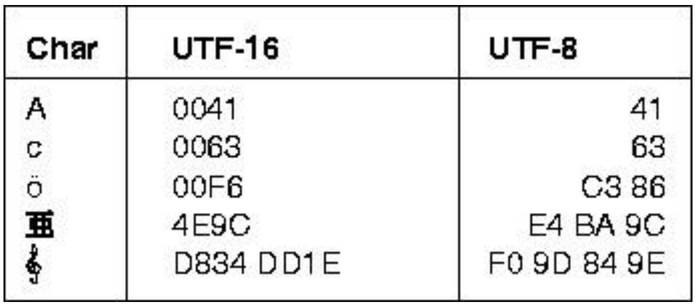


Char	Code	Char	Code	Char	Code	Char
Ą	161	˘	162	Ł	163	¤
Š	169	Ş	170	Ť	171	Ź
ą	177	˛	178	ł	179	´
š	185	ş	186	ť	187	ź
-	193	Â	194	Ă	195	Ä

Unicode

Unicode Worldwide Character Standard uses 4 to 32 bits to represent letters, numbers and symbol.

Unicode Tabular Column



Char	UTF-16	UTF-8
A	0041	41
c	0063	63
ö	00F6	C3 86
亜	4E9C	E4 BA 9C
𝄞	D834 DD1E	F0 9D 84 9E

Difference Between Unicode and ASCII

The differences between them are as follows:

Unicode **Coding Scheme**	**ASCII Coding Scheme**
• It uses variable bit encoding according to the requirement. For example, UTF-8, UTF-16, UTF-32	• It uses 7-bit encoding. As of now, the extended form uses 8-bit encoding.

• It is a standard form.	• It is not a standard all over the world.
• People use this scheme all over the world.	• It has only limited characters hence, it cannot be used all over the world.
• The Unicode characters themselves involve all the characters of the ASCII encoding. Therefore we can say that it is a superset for it.	• It has its equivalent coding characters in the Unicode.
• It has more than 128,000 characters.	• In contrast, it has only 256 characters.

Multimedia Data

Multimedia is a form of communication that combines different content forms such as text, audio, images, animations, or video into a single interactive presentation, in contrast to traditional mass media which featured little to no interaction from users, such as printed material or audio recordings. Popular examples of multimedia include video podcasts, audio slideshows and animated videos.

Data Compression

Data compression is the process of modifying, encoding or converting the bits structure of data in such a way that it consumes less space on disk.

It enables reducing the storage size of one or more data instances or elements. Data compression is also known as source coding or bit-rate reduction.

Data compression enables sending a data object or file quickly over a network or the Internet and in optimizing physical storage resources.

Data compression has wide implementation in computing services and solutions, specifically data communications. Data compression works through several compressing techniques and software solutions that utilize data compression algorithms to reduce the data size.

A common data compression technique removes and replaces repetitive data elements and symbols to reduce the data size. Data compression for graphical data can be lossless compression or lossy compression, where the former saves all replaces but save all repetitive data and the latter deletes all repetitive data.

What is Data Compression?

Data compression is used everywhere. Many different file types use compressed data. Without data compression, a 3-minute song would be over 100Mb in size, while a 10-minute video would be over 1Gb in size. Data compression shrinks big files into much smaller ones. It does this by getting rid of unnecessary data while retaining the information in the file.

Data compression can be expressed as a decrease in the number of bits required to illustrate data. Compressing data can conserve storage capacity, accelerate file transfer, and minimise costs for hardware storage and network capacity.

How Compression Works?

Compression is executed by a program that uses a procedure to identify how to reduce the data size.

Text compression can be done by eliminating unnecessary characters, embedding a repeat character to specify repeated characters, and substituting a smaller bit

string for a commonly occurring bit string. Data compression can cut a text file to 50%, or to a percentage still smaller than its original size.
For data transmission, compression can be done on the data content or on the transmission unit as a whole. When data needs to be transferred over the internet, larger files can be sent in a ZIP, GZIP or another compressed format.

What is the Purpose of Compression?
The purpose of compression is to make a file, message, or any other chunk of data smaller. Data compression can significantly decrease the amount of storage space a file takes up. If we had a 10Mb file and could shrink it down to 5Mb, we have compressed it with a compression ratio of 2, since it is half the size of the original file. If we compressed the 10Mb file to 1Mb it would have a compression ratio of 10 because the new file is a 10th the size of the original. The higher the compression ratio the better the compression. Because of compression, administrators save money and time that would otherwise be spent on storage.

Compression enhances backup storage operation and has also affected primary storage data reduction. Compression will continue to play a significant role in data reduction as data continues its own exponential growth.

Almost any type of file can be compressed, but it's imperative to follow best practices when selecting files to compress. For example, some files are already compressed, so compressing them would not have a substantial impact.

Data Compression Methods
There are two kinds of compression: Lossless and Lossy.

Lossy compression loses data, while lossless compression keeps all the data. With lossless compression, we don't get rid of any data. Instead, the technique is based on finding smarter ways to encode the data. With lossy compression, we get rid of data, which is why we need to distinguish data from information.

Lossless compression allows the potential for a file to return to its original size, without the loss of a single bit of data, when the file is uncompressed. Lossless compression is the usual approach taken with executables, as well as with text and spreadsheet files, where the loss of words or numbers would change the information. Lossless compression can compress the data whenever redundancy is present. Therefore, lossless compression takes advantage of data redundancy.

Lossy compression permanently removes bits of data that are redundant, insignificant or unnoticeable. Lossy compression is suitable with graphics, audio, video and images, where the deletion of some data bits has little or no apparent effect on the illustration of the content. In lossy compression, messages become more efficient by getting rid of unwanted data. Lossy compression lessens the size of the data while retaining more information.

Graphical image compression can be either lossy or lossless. Graphic image file formats are usually developed to compress information since the files tend to be big. JPEG is an image file format that promotes lossy image compression. Formats such as GIF and PNG use lossless compression.

Huffman Coding | Greedy Algo-3

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be

“cccd” or “ccb” or “acd” or “ab”.
See this for applications of Huffman Coding.
There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.

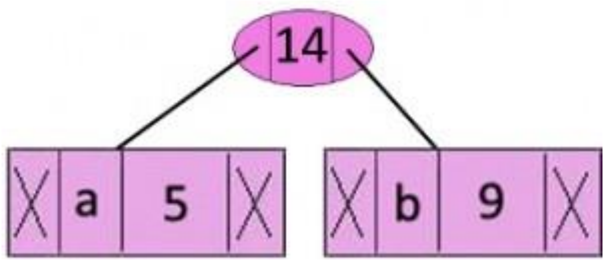
Steps to build Huffman Tree
Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.
   Let us understand the algorithm with an example:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.

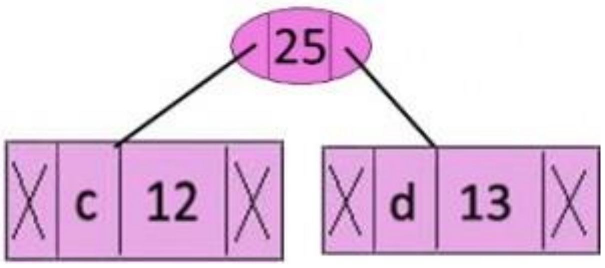


Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12

d	13
Internal Node	14
e	16
f	45

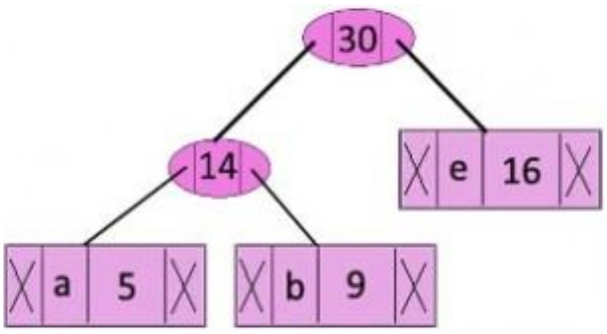
Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25



Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

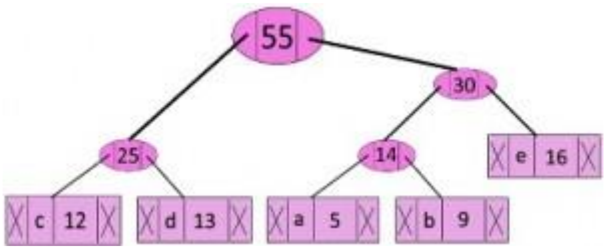
Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30



Now min heap contains 3 nodes.

character	Frequency
Internal Node	25
Internal Node	30
f	45

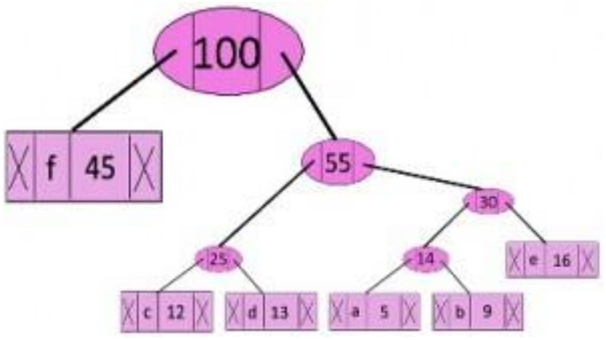
Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55



Now min heap contains 2 nodes.

```
character     Frequency
       f         45
Internal Node    55
```

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100

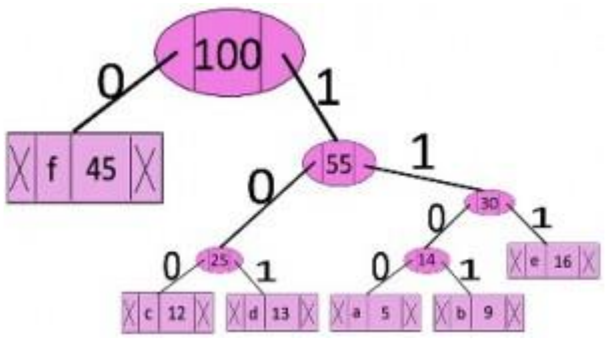


Now min heap contains only one node.

```
character      Frequency
Internal Node    100
```

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:
Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



The codes are as follows:

```
character   code-word
    f           0
    c           100
    d           101
    a           1100
    b           1101
    e           111
```

Huffman coding (also known as Huffman Encoding) is an algorithm for doing data compression, and it forms the basic idea behind file compression. This post talks about the fixed-length and variable-length encoding, uniquely decodable codes, prefix rules, and Huffman Tree construction.

Overview

We already know that every character is sequences of `0's` and `1's` and stored using 8-bits. This is known as "fixed-length encoding", as each character uses the same number of fixed-bit storage.

Given a text, how to reduce the amount of space required to store a character?

The idea is to use "variable-length encoding". We can exploit the fact that some characters occur more frequently than others in a text (refer to this) to design an algorithm that can represent the same piece of text using a lesser number of bits. In variable-length encoding, we assign a variable number of bits to characters depending upon their frequency in the given text. So, some characters might end up taking a single bit, and some might end up taking two bits, some might be encoded using three bits, and so on. The problem with variable-length encoding lies in its decoding.

Given a sequence of bits, how to decode it uniquely?

Let's consider the string `aabacdab`. It has `8` characters in it and uses 64–bits storage (using fixed-length encoding). If we note, the frequency of characters `a`, `b`, `c` and `d` are `4`, `2`, `1`, `1`, respectively. Let's try to represent `aabacdab` using a lesser number of bits by using the fact that `a` occurs more frequently than `b`, and `b` occurs more frequently than `c` and `d`. We start by randomly assigning a single bit code `0` to `a`, 2–bit code `11` to `b`, and 3–bit code `100` and `011` to characters `c` and `d`, respectively.

```
a 0
b 11
c 100
d 011
```

So, the string `aabacdab` will be encoded to `00110100011011` `(0|0|11|0|100|011|0|11)` using the above codes. But the real problem lies in decoding. If we try to decode the string `00110100011011`, it will lead to ambiguity as it can be decoded to,

```
0|011|0|100|011|0|11    adacdab
0|0|11|0|100|0|11|011   aabacabd
0|011|0|100|0|11|0|11   adacabab
…
and so on
```

To prevent ambiguities in decoding, we will ensure that our encoding satisfies the “prefix rule”, which will result in “uniquely decodable codes”. The prefix rule states that no code is a prefix of another code. By code, we mean the bits used for a particular character. In the above example, `0` is the prefix of `011`, which violates the prefix rule. If our codes satisfy the prefix rule, the decoding will be unambiguous (and vice versa).
Let's consider the above example again. This time we assign codes that satisfy the prefix rule to characters `'a'`, `'b'`, `'c'`, and `'d'`.

```
a   0
b   10
c   110
d   111
```

Using the above codes, the string `aabacdab` will be encoded to `00100110111010` `(0|0|10|0|110|111|0|10)`. Now we can uniquely decode `00100110111010` back to our original string `aabacdab`.
Now that we are clear on variable-length encoding and prefix rule, let’s talk about Huffman coding.

Huffman Coding

The technique works by creating a binary tree of nodes. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the character itself, the weight (frequency of appearance) of the character. Internal nodes contain character weight and links to two child nodes. As a common convention, bit `0` represents following the left child, and a bit `1` represents following the right child. A finished tree has `n` leaf nodes and `n-1` internal nodes. It is recommended that Huffman Tree should discard unused characters in the text to produce the most optimal code lengths.

We will use a priority queue for building Huffman Tree, where the node with the lowest frequency has the highest priority. Following are the complete steps:

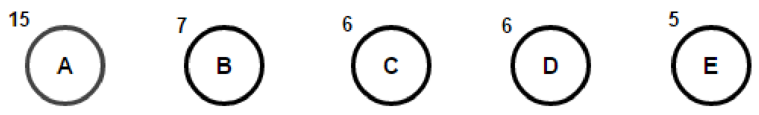
1. Create a leaf node for each character and add them to the priority queue.

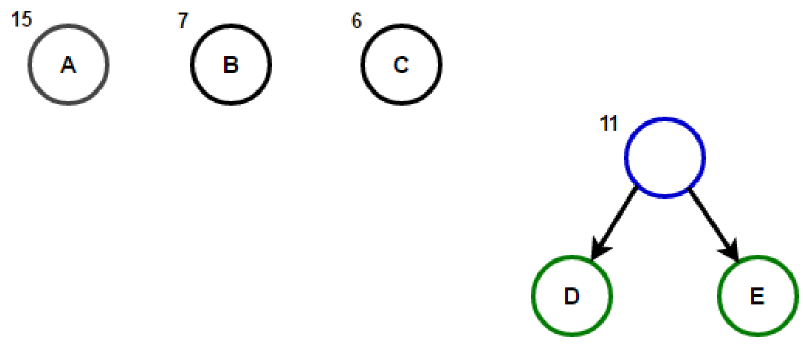
2. While there is more than one node in the queue:

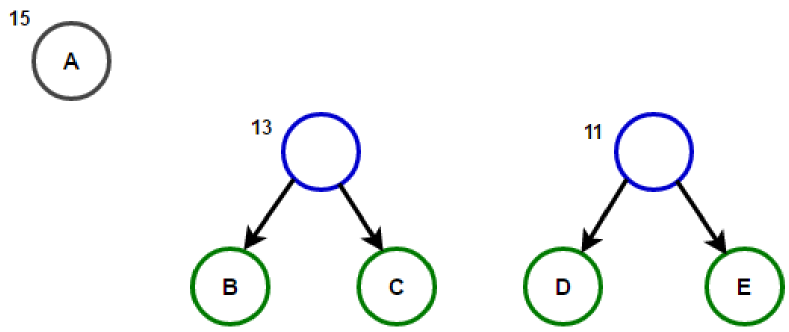
- Remove the two nodes of the highest priority (the lowest frequency) from the queue.
- Create a new internal node with these two nodes as children and a frequency equal to the sum of both nodes’ frequencies.
- Add the new node to the priority queue.

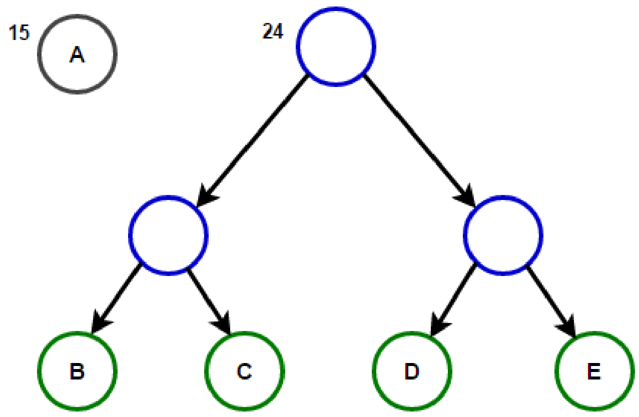
3. The remaining node is the root node and the tree is complete.

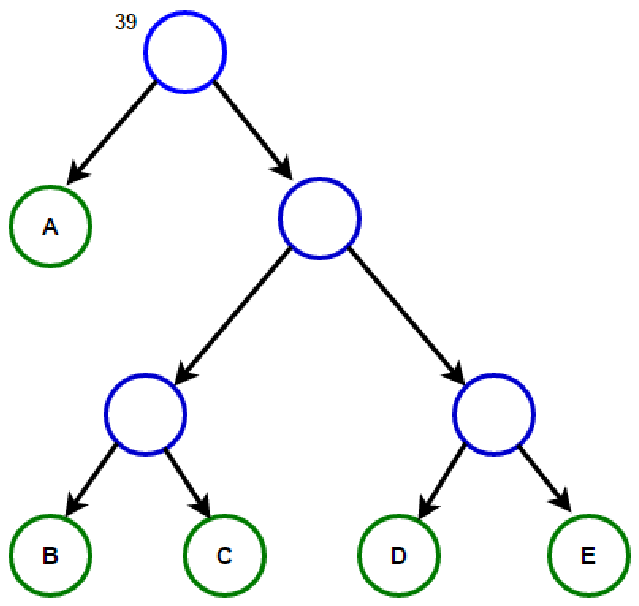
Consider some text consisting of only `'A'`, `'B'`, `'C'`, `'D'`, and `'E'` characters, and their frequencies are `15`, `7`, `6`, `6`, `5`, respectively. The following figures illustrate the steps followed by the algorithm:



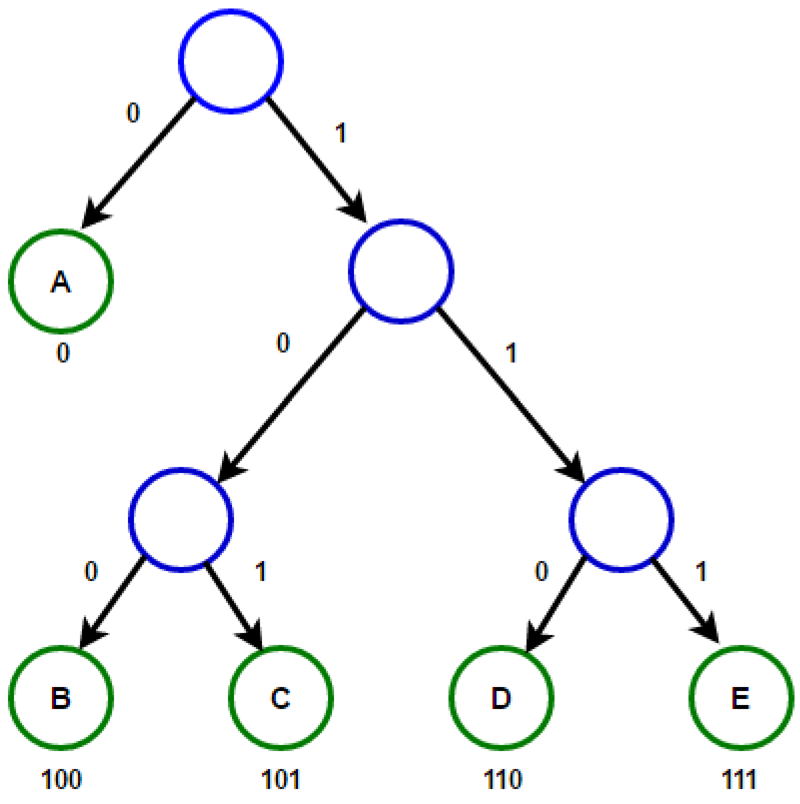

15
A
7
B
6
C
11
D
E


15
A
13
B
C
11
D
E


15
A
24
B
C
D
E



The path from the root to any leaf node stores the optimal prefix code (also called Huffman code) corresponding to the character associated with that leaf node.



Shannon-Fano Algorithm for Data Compression

DATA COMPRESSION AND ITS TYPES

Data Compression, also known as source coding, is the process of encoding or converting data in such a way that it consumes less memory space. Data compression reduces the number of resources required to store and transmit data.

It can be done in two ways- lossless compression and lossy compression. Lossy compression reduces the size of data by removing unnecessary information, while there is no data loss in lossless compression.

WHAT IS SHANNON FANO CODING?

Shannon Fano Algorithm is an entropy encoding technique for lossless data compression of multimedia. Named after Claude Shannon and Robert Fano, it assigns a code to each symbol based on their probabilities of occurrence. It is a variable-length encoding scheme, that is, the codes assigned to the symbols will be of varying length.

HOW DOES IT WORK?

The steps of the algorithm are as follows:

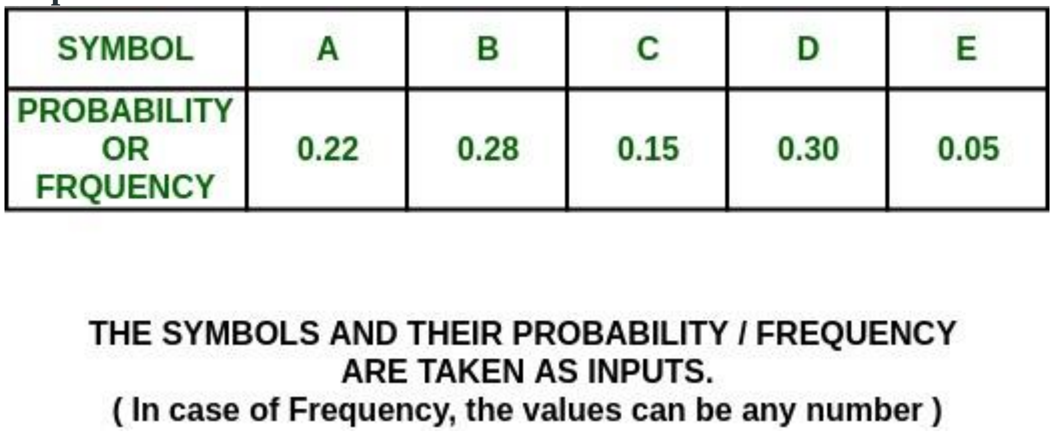
1. Create a list of probabilities or frequency counts for the given set of symbols so that the relative frequency of occurrence of each symbol is known.
2. Sort the list of symbols in decreasing order of probability, the most probable ones to the left and least probable to the right.
3. Split the list into two parts, with the total probability of both the parts being as close to each other as possible.
4. Assign the value 0 to the left part and 1 to the right part.
5. Repeat steps 3 and 4 for each part, until all the symbols are split into individual subgroups.

The Shannon codes are considered accurate if the code of each symbol is unique.

EXAMPLE:

The given task is to construct Shannon codes for the given set of symbols using the Shannon-Fano lossless compression technique.

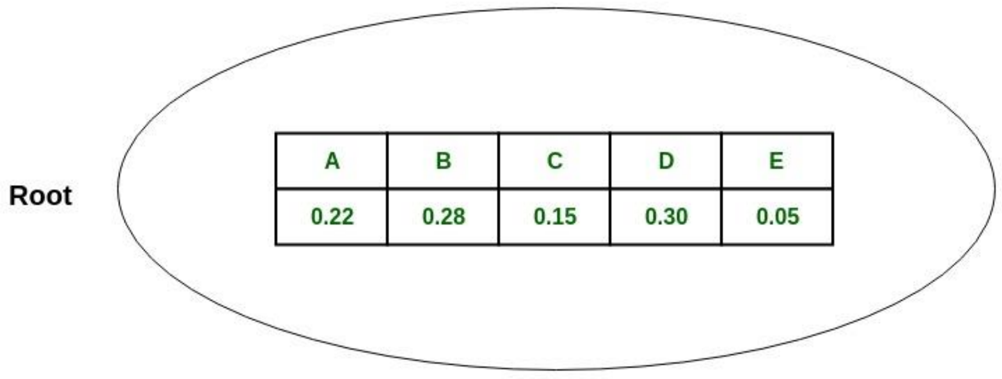
Step:



SYMBOL	A	B	C	D	E
PROBABILITY OR FRQUENCY	0.22	0.28	0.15	0.30	0.05

THE SYMBOLS AND THEIR PROBABILITY / FREQUENCY ARE TAKEN AS INPUTS.
(In case of Frequency, the values can be any number)

Tree:



A	B	C	D	E
0.22	0.28	0.15	0.30	0.05

TREE AFTER STEP 1

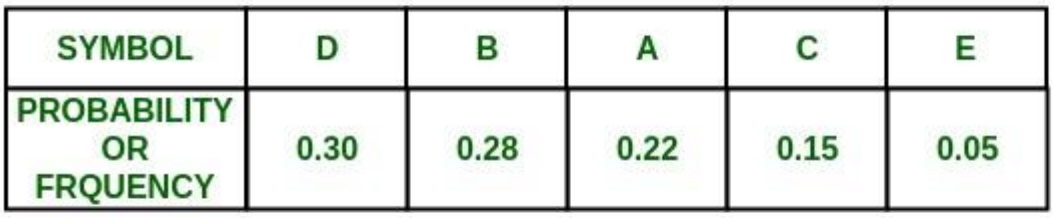
Solution:
1. Upon arranging the symbols in decreasing order of probability:

P(D) + P(B) = 0.30 + 0.2 = 0.58
and,

P(A) + P(C) + P(E) = 0.22 + 0.15 + 0.05 = 0.42
And since the almost equally split the table, the most is divided it the blockquote table isblockquotento

{D, B} *and* ***{A, C, E}***
and assign them the values 0 and 1 respectively.

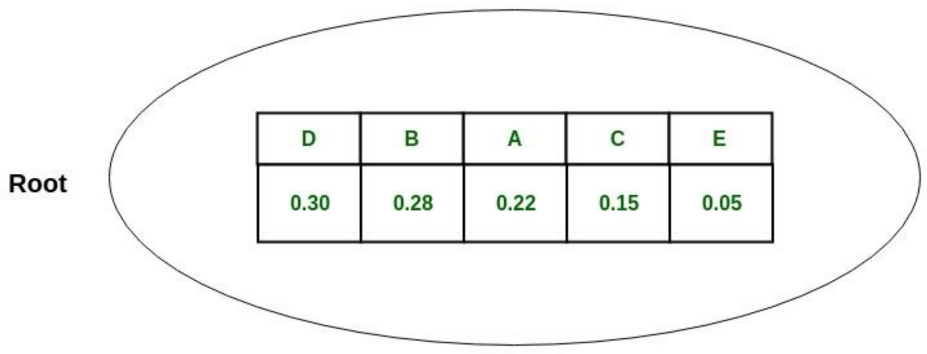
Step:



SYMBOL	D	B	A	C	E
PROBABILITY OR FRQUENCY	0.30	0.28	0.22	0.15	0.05

INPUTS ARE SORTED ACCORDING
TO THEIR PROBABILITY / FREQUENCY
(Here they are sorted according to their probability)

Tree:



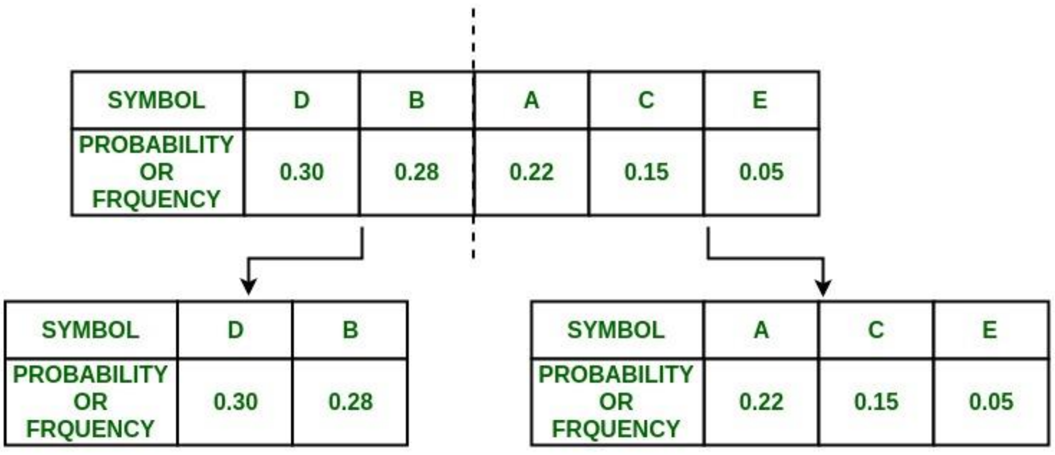
TREE AFTER STEP 2

2. Now, in {D, B} group,

P(D) = 0.30** and **P(B) = 0.28

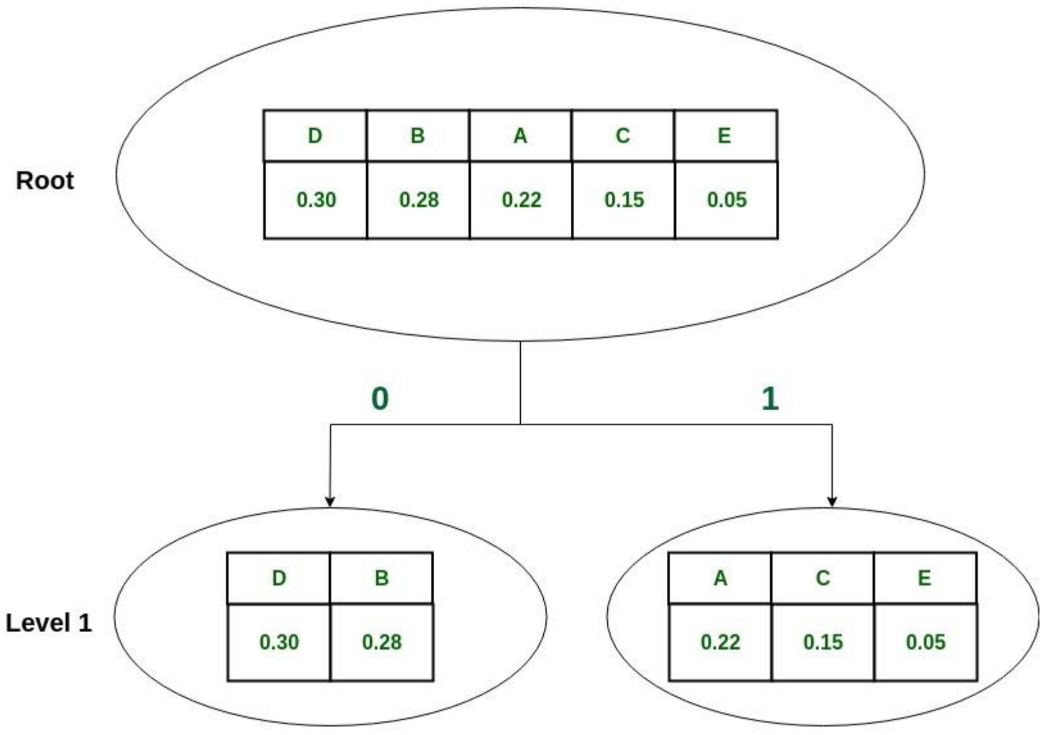
which means that **P(D)~P(B)**, so divide {D, B} into {D} and {B} and assign 0 to D and 1 to B.

Step:



THE SYMBOLS ARE DIVIDED INTO TWO SUCH THAT THE TOTAL PROBABILITY / FREQUENCY OF LEFT SIDE ALMOST SAME AS THAT OF RIGHT SIDE

Tree:



TREE AFTER STEP 3

3. In {A, C, E} group,

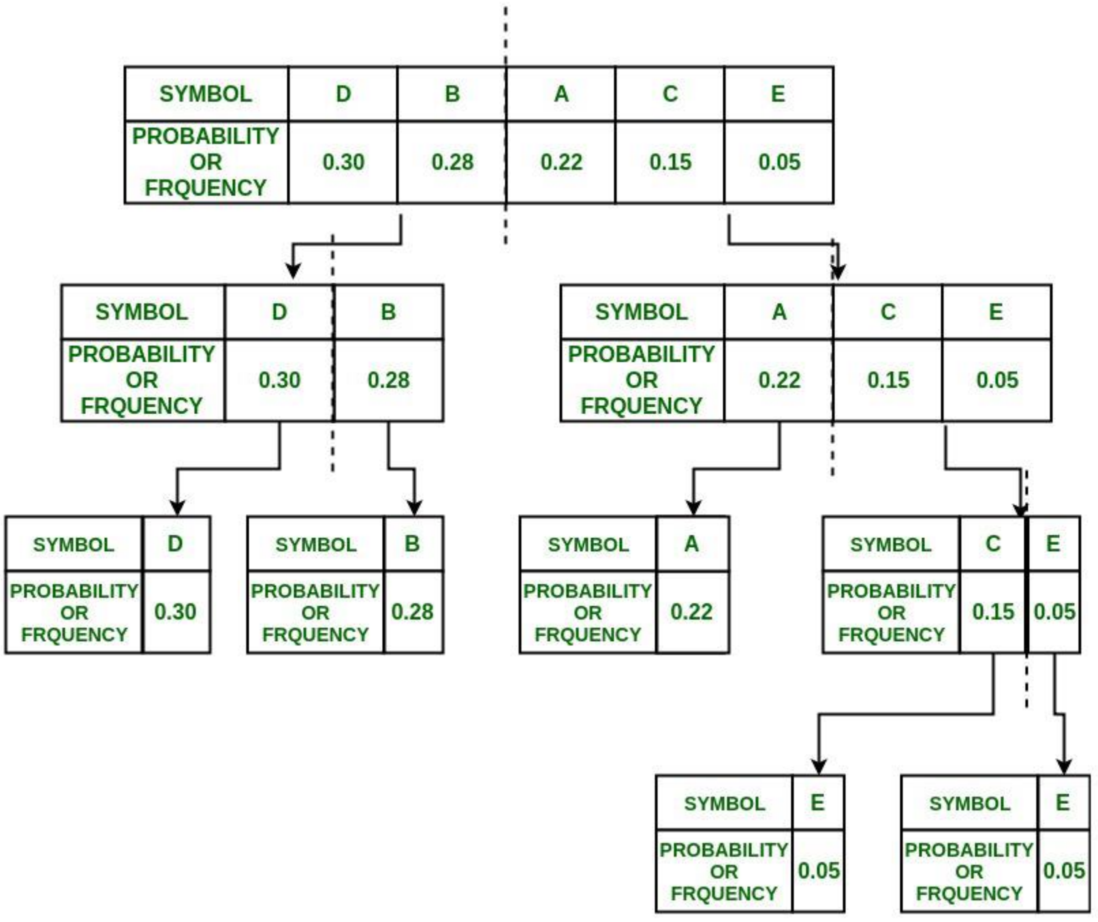
P(A) = 0.22 *and* ***P(C) + P(E) = 0.20***
So the group is divided into

{A} *and* ***{C, E}***
and they are assigned values 0 and 1 respectively.

4. In {C, E} group,

P(C) = 0.15 *and* ***P(E) = 0.05***
So divide them into {C} and {E} and assign 0 to {C} and 1 to {E}

Step:


SYMBOL D B A C E
PROBABILITY OR FRQUENCY 0.30 0.28 0.22 0.15 0.05
SYMBOL D B
PROBABILITY OR FRQUENCY 0.30 0.28
SYMBOL A C E
PROBABILITY OR FRQUENCY 0.22 0.15 0.05
SYMBOL D
PROBABILITY OR FRQUENCY 0.30
SYMBOL B
PROBABILITY OR FRQUENCY 0.28
SYMBOL A
PROBABILITY OR FRQUENCY 0.22
SYMBOL C E
PROBABILITY OR FRQUENCY 0.15 0.05
SYMBOL E
PROBABILITY OR FRQUENCY 0.05
SYMBOL E
PROBABILITY OR FRQUENCY 0.05

THE SYMBOLS ARE CONTINUED TO BE DIVIDED INTO TWO
TILL EACH SYMBOL BECOME SEPARATED

Tree:

Note: The splitting is now stopped as each symbol is separated now.
The Shannon codes for the set of symbols are:

SYMBOL	A	B	C	D	E
PROBABILITY OR FRQUENCY	0.22	0.28	0.15	0.30	0.05
SHANNON-FANO CODE	00	01	10	110	111

THE SHANNON CODES OF THE SYMBOLS
(based on their traversal from the root node)