

Data Structures and Algorithms

Lecture 34: Huffman algorithm

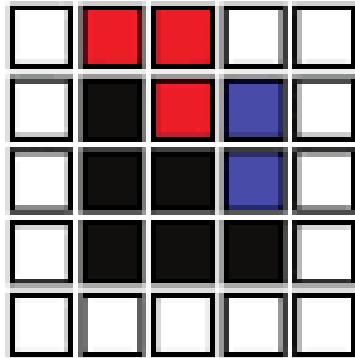
Huffman Coding

Huffman Coding is a technique of compressing data to reduce its size without losing any of the details. It was first developed by David Huffman.

Huffman Coding is generally useful to compress the data in which there are frequently occurring characters.

Fixed-Length encoding - Every character is assigned a binary code using same number of bits. Thus, a string like “aabacdad” can require 64 bits (8 bytes) for storage or transmission, assuming that each character uses 8 bits.

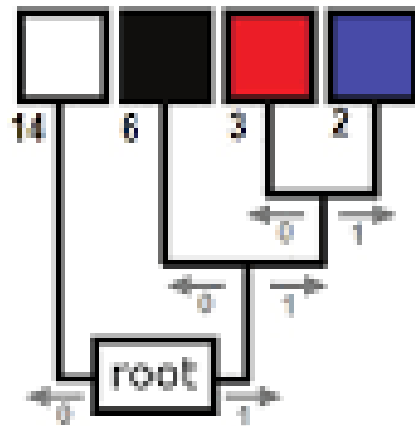
Huffman Coding





a. Image pixel grid



b. Frequency count of colors



c. Huffman Tree of color

color	freq.	bit code
	14	0
	6	10
	3	110
	2	111

d. Color code table (Mapping Table)

Huffman Coding

Variable- Length encoding - As opposed to Fixed-length encoding, this scheme uses variable number of bits for encoding the characters depending on their frequency in the given text. Thus, for a given string like “aabacdad”, frequency of characters ‘a’, ‘b’, ‘c’ and ‘d’ is 4,1,1 and 2 respectively. Since ‘a’ occurs more frequently than ‘b’, ‘c’ and ‘d’, it uses least number of bits, followed by ‘d’, ‘b’ and ‘c’.

Suppose we randomly assign binary codes to each character as follows-

a 0

b 011

c 111

d 11

Thus, the string “aabacdad” gets encoded to 00011011111011 (0 | 0 | 011 | 0 | 111 | 11 | 0 | 11), using fewer number of bits compared to fixed-length encoding scheme.

Huffman Coding

But the real problem lies with the decoding phase. If we try and decode the string 00011011111011, it will be quite ambiguous since, it can be decoded to the multiple strings, few of which are-

aaadacdad (0 | 0 | 0 | 11 | 0 | 111 | 11 | 0 | 11)

aaadbcbad (0 | 0 | 0 | 11 | 011 | 111 | 0 | 11)

aabbcb (0 | 0 | 011 | 011 | 111 | 011)

... and so on

To prevent such ambiguities during decoding, the encoding phase should satisfy the “prefix rule” which states that no binary code should be a prefix of another code. This will produce uniquely decodable codes. The above codes for ‘a’, ‘b’, ‘c’ and ‘d’ do not follow prefix rule since the binary code for a, i.e. 0, is a prefix of binary code for b i.e 011, resulting in ambiguous decodable codes.

Huffman Coding

Lets reconsider assigning the binary codes to characters 'a', 'b', 'c' and 'd'.

a 0

b 11

c 101

d 100

Using the above codes, string “aabacdad” gets encoded to 001101011000100 (0 | 0 | 11 | 0 | 101 | 100 | 0 | 100). Now, we can decode it back to string “aabacdad”.

Huffman Coding - Example

lets reconsider assigning the binary codes to characters 'a', 'b', 'c' and 'd'.

a 0

b 11

c 101

d 100

Using the above codes, string “aabacdad” gets encoded to 001101011000100 (0 | 0 | 11 | 0 | 101 | 100 | 0 | 100). Now, we can decode it back to string “aabacdad”.

Example

Suppose the string below is to be sent over a network.

B	C	A	A	D	D	D	C	C	A	C	A	C	A	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of $8 * 15 = 120$ bits are required to send this string.

Using the Huffman Coding technique, we can compress the string to a smaller size.

Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

Example

Once the data is encoded, it has to be decoded. Decoding is done using the same tree. Huffman Coding prevents any ambiguity in the decoding process using the concept of prefix code ie. a code associated with a character should not be present in the prefix of any other code. The tree created above helps in maintaining the property.

Huffman coding is done with the help of the following steps.

1. Calculate the frequency of each character in the string

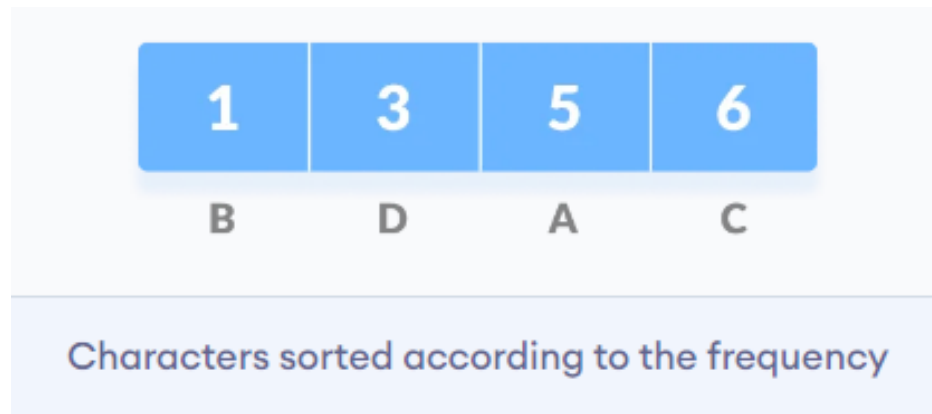
B	C	A	A	D	D	D	C	C	A	C	A	C	A	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	6	5	3
B	C	A	D

Frequency of string

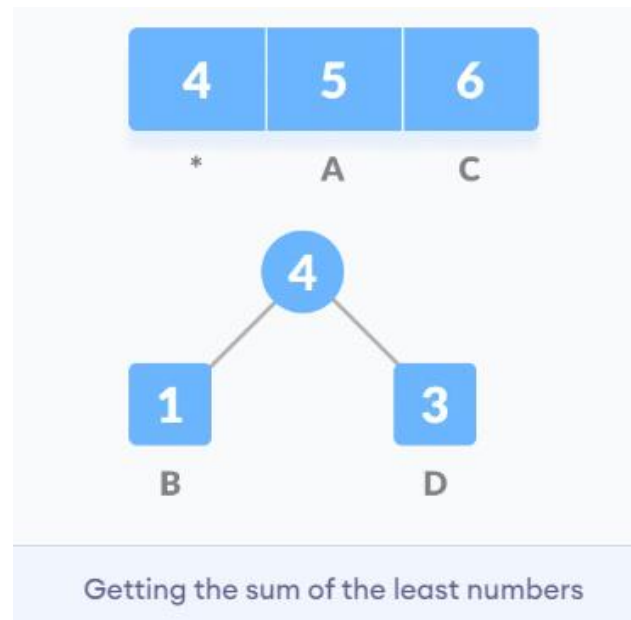
Example

2. Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.



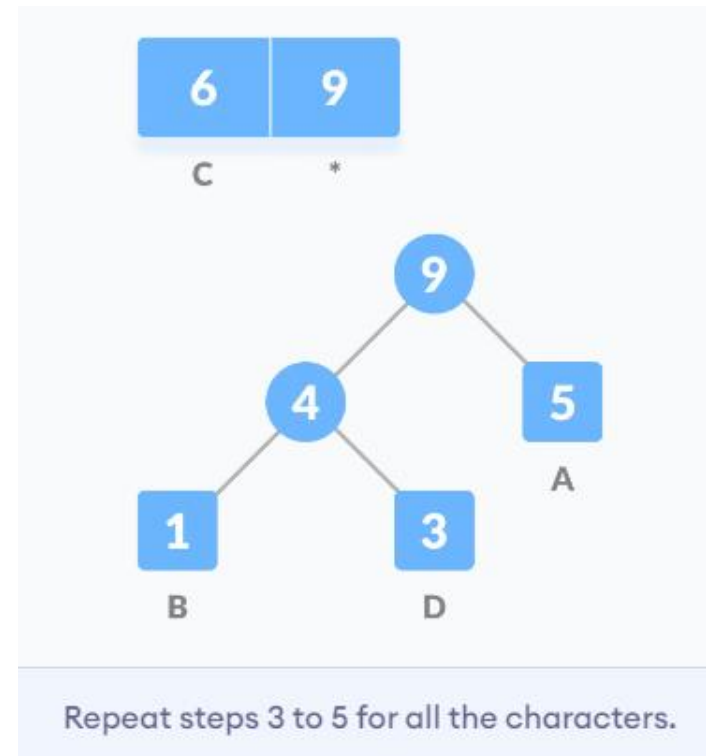
Example

3. Make each unique character as a leaf node.
4. Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.



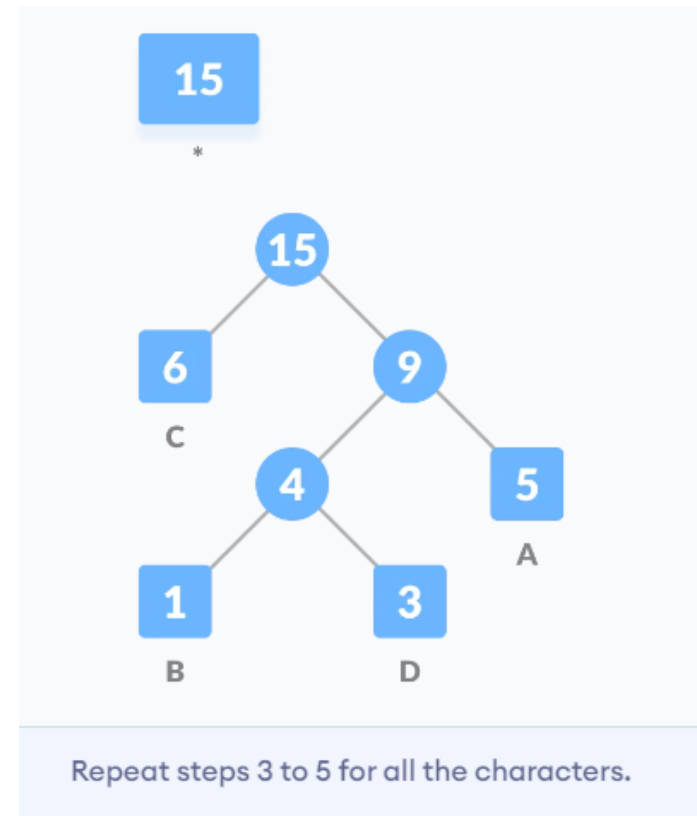
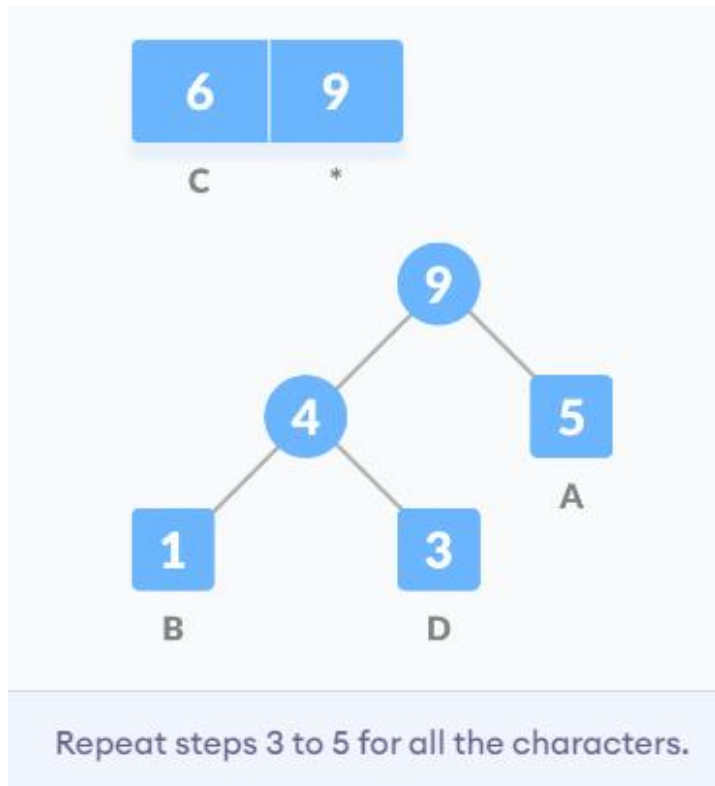
Example

5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes in the figure above).
6. Insert node z into the tree.
7. Repeat steps 3 to 5 for all the characters.



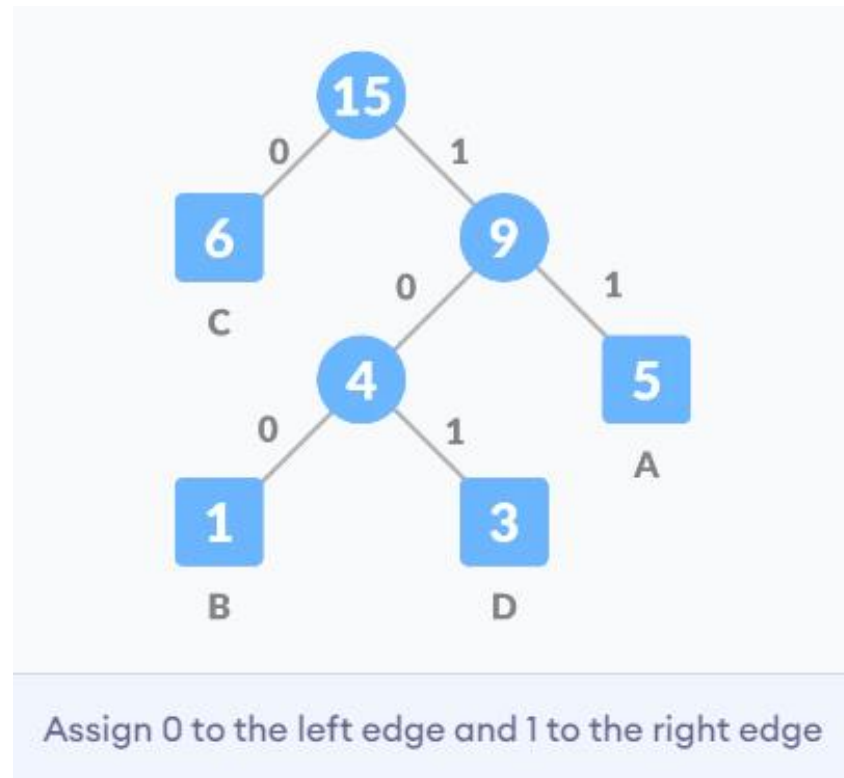
Example

5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes in the figure above).
6. Insert node z into the tree.
7. Repeat steps 3 to 5 for all the characters.



Example

8. For each non-leaf node, assign 0 to the left edge and 1 to the right edge.



For sending the above string over a network, we have to send the tree as well as the above compressed-code.

Example

The total size is given by the table below.

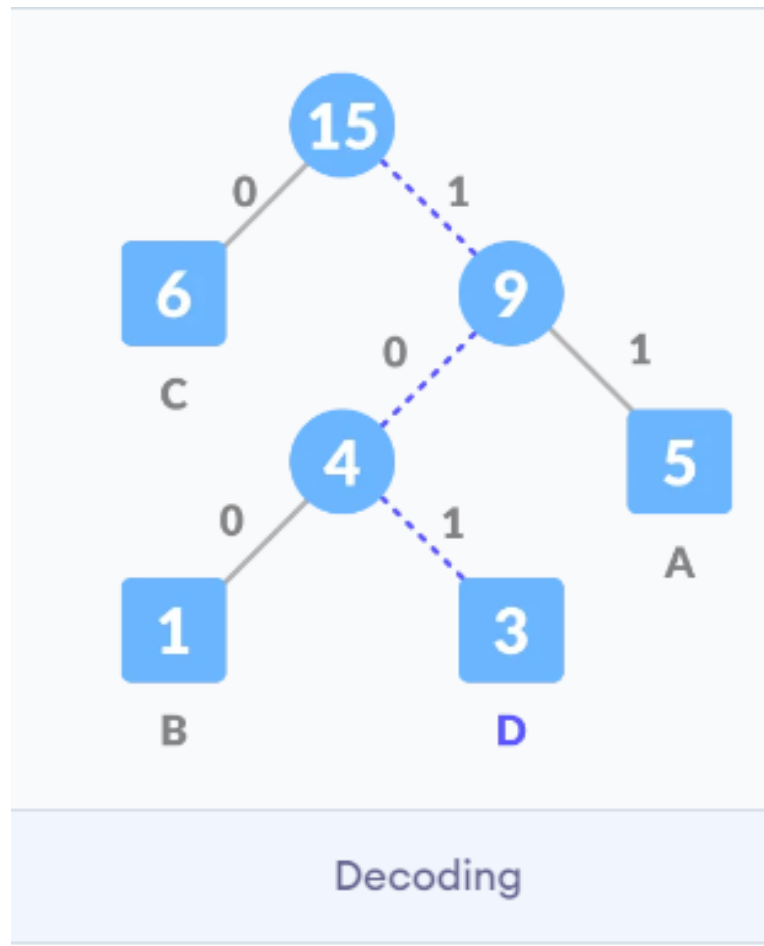
Character	Frequency	Code	Size
A	5	11	$5 \times 2 = 10$
B	1	100	$1 \times 3 = 3$
C	6	0	$6 \times 1 = 6$
D	3	101	$3 \times 3 = 9$
$4 \times 8 = 32$ bits	15 bits		28 bits

Without encoding, the total size of the string was 120 bits.

After encoding the size is reduced to $32 + 15 + 28 = 75$.

Decoding the code

For decoding the code, we can take the code and traverse through the tree to find the character. Let 101 is to be decoded, we can traverse from the root as in the figure below.



Huffman Coding Algorithm

```
create a priority queue Q consisting of each unique character.  
sort then in ascending order of their frequencies.  
for all the unique characters:  
    create a newNode  
    extract minimum value from Q and assign it to leftChild of newNode  
    extract minimum value from Q and assign it to rightChild of newNode  
    calculate the sum of these two minimum values and assign it to the value of newNode  
    insert this newNode into the tree  
return rootNode
```

Huffman Coding Applications

- Huffman coding is used in conventional compression formats like GZIP, BZIP2, PKZIP, etc.
- For text and fax transmissions.