Huffman Encoding Example + Algorithm

Suppose, we have to encode string **abracadabra.** Determine the following:

1. Huffman code for All the characters
2. Average code length for the given String
3. Length of the encoded string

(i) Huffman Code for All the Characters

In order to determine the code for each character, first, we construct a **Huffman tree.**

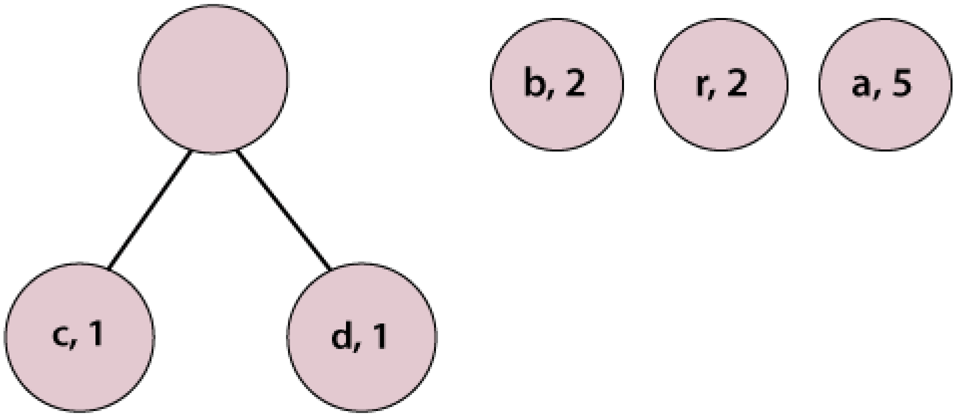
Step 1: Make pairs of characters and their frequencies.

(a, 5), (b, 2), (c, 1), (d, 1), (r, 2)

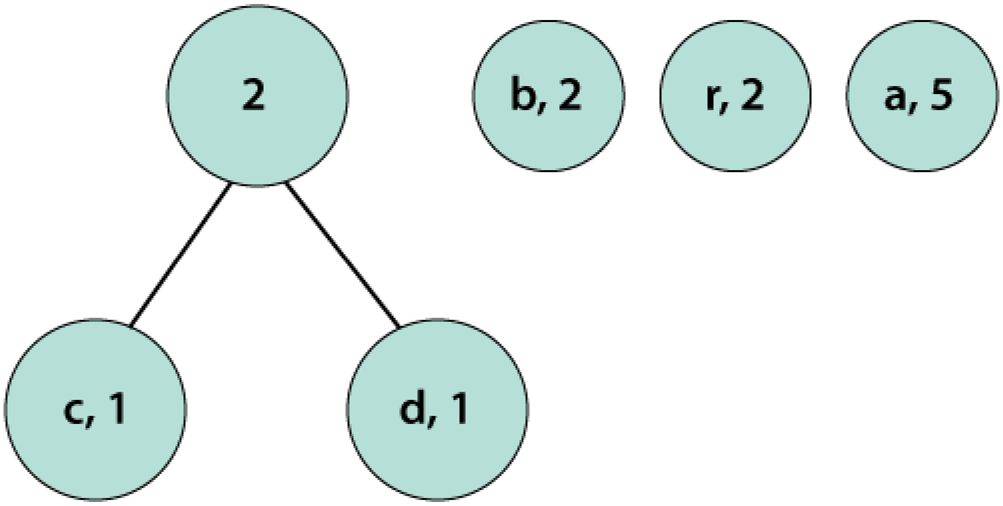
Step 2: Sort pairs with respect to frequency, we get:

(c, 1), (d, 1), (b, 2) (r, 2), (a, 5)

Step 3: Pick the first two characters and join them under a parent node.

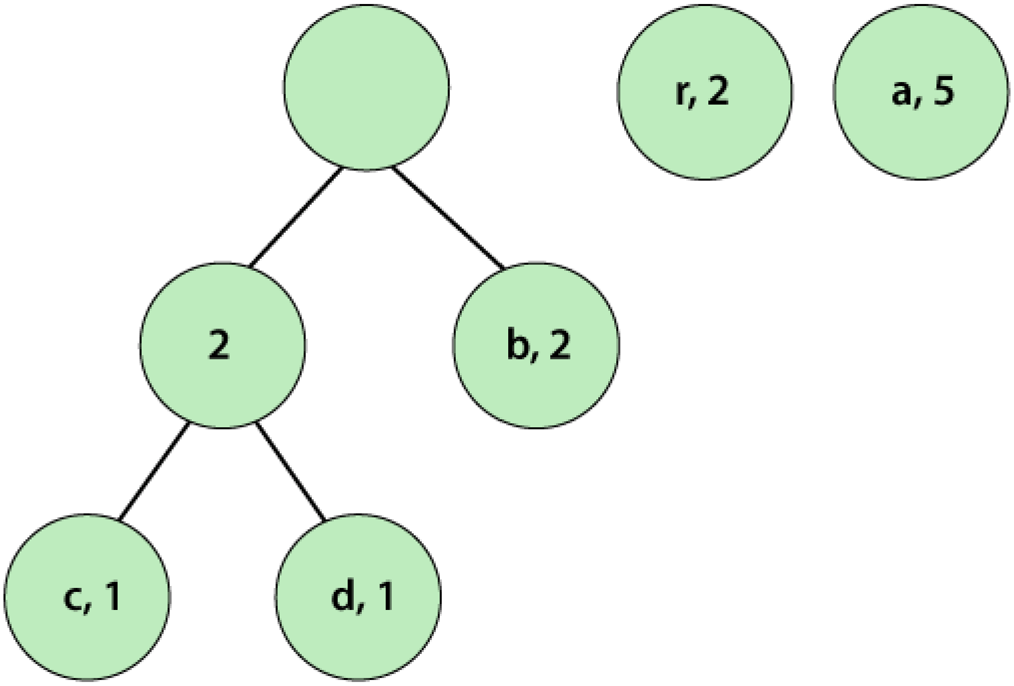


We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 1+1=**2.**

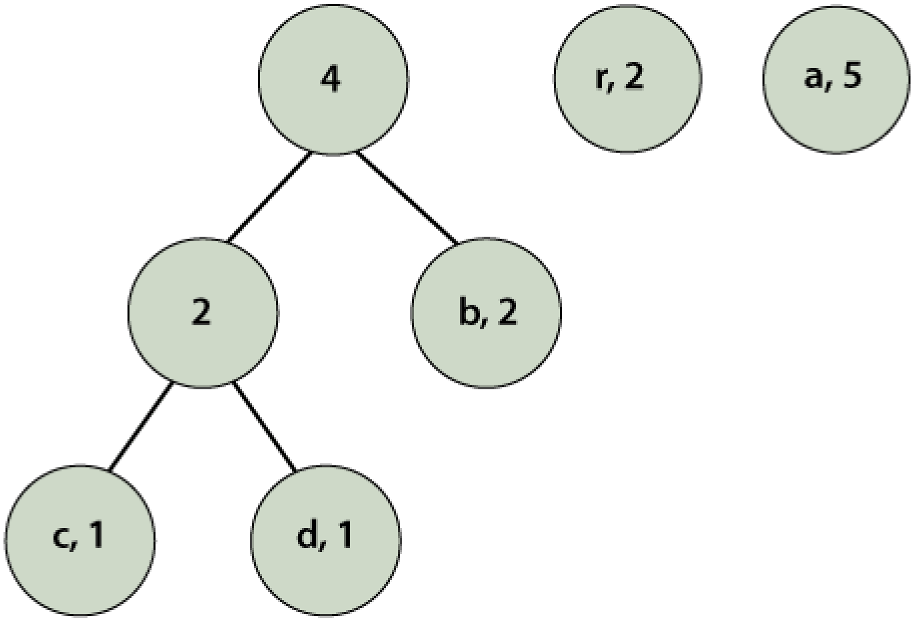


Step 4: Repeat Steps 2 and 3 until, we get a single tree.

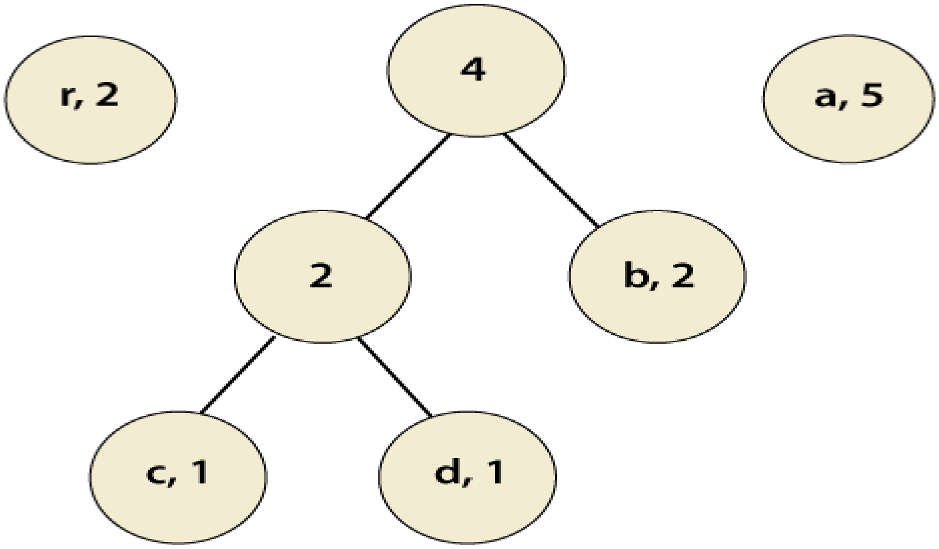
We observe that the pairs are already in a sorted (by step 2) manner. Again, pick the first two pairs and join them.



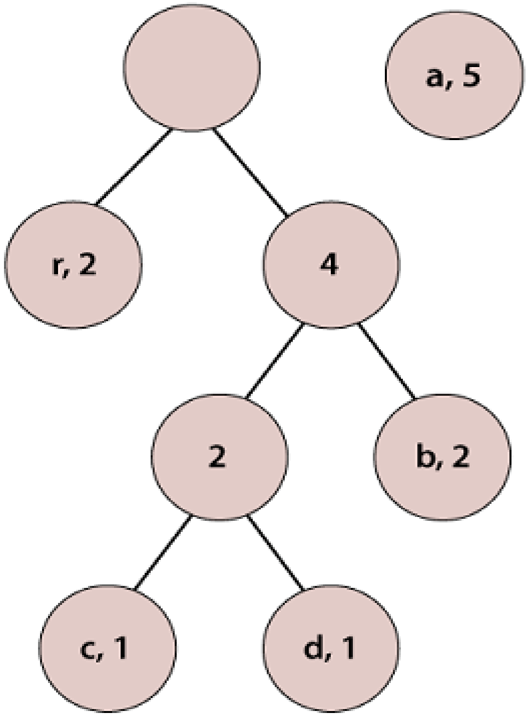
We observe that a parent node does not has a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 2+2=**4.**



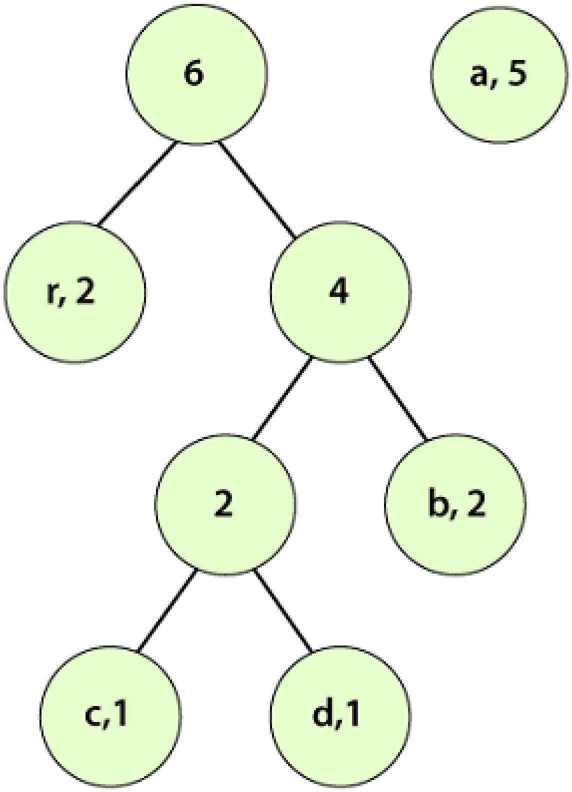
Again, we check if the pairs are in a sorted manner or not. At this step, we need to sort the pairs.



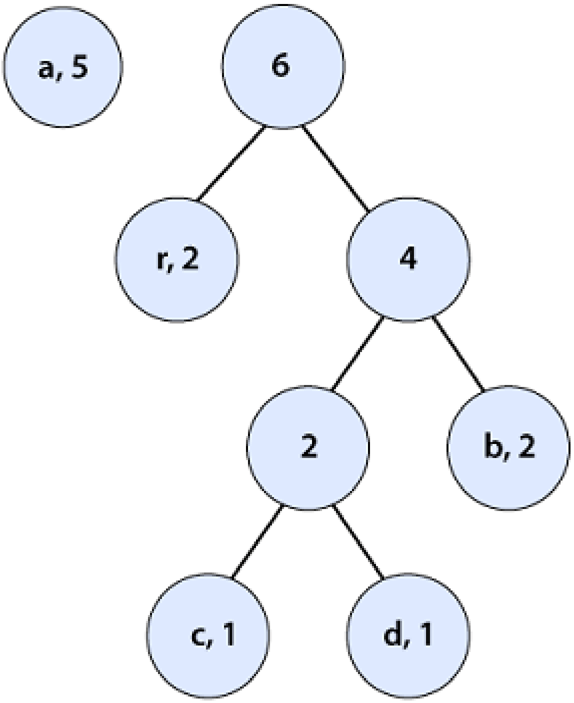
According to step 3, pick the first two pairs and join them, we get:



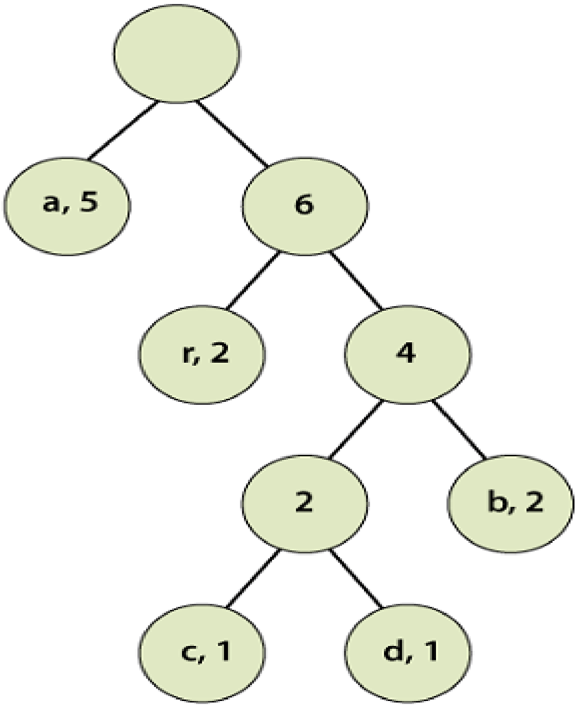
We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 2+4=**6.**



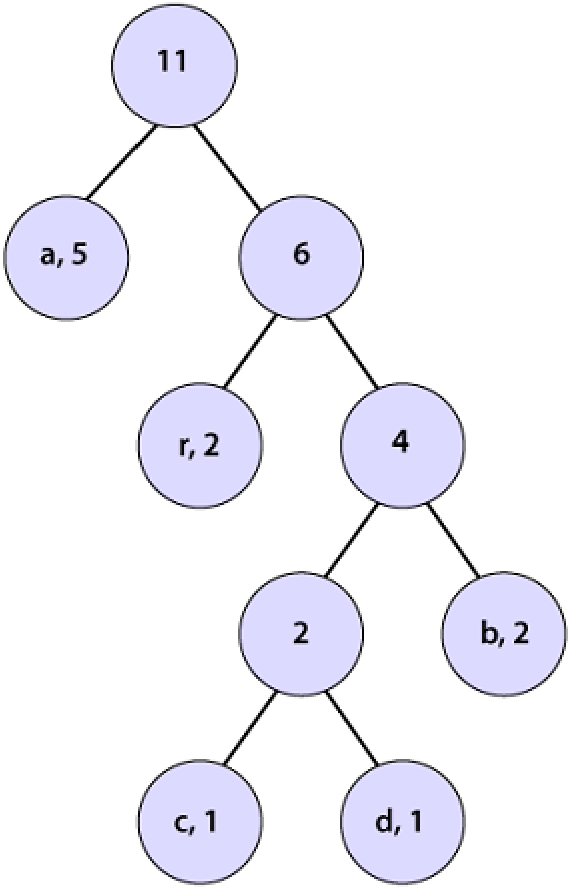
Again, we check if the pairs are in a sorted manner or not. At this step, we need to sort the pairs. After sorting the tree looks like the following:



According to step 3, pick the first two pairs and join them, we get:

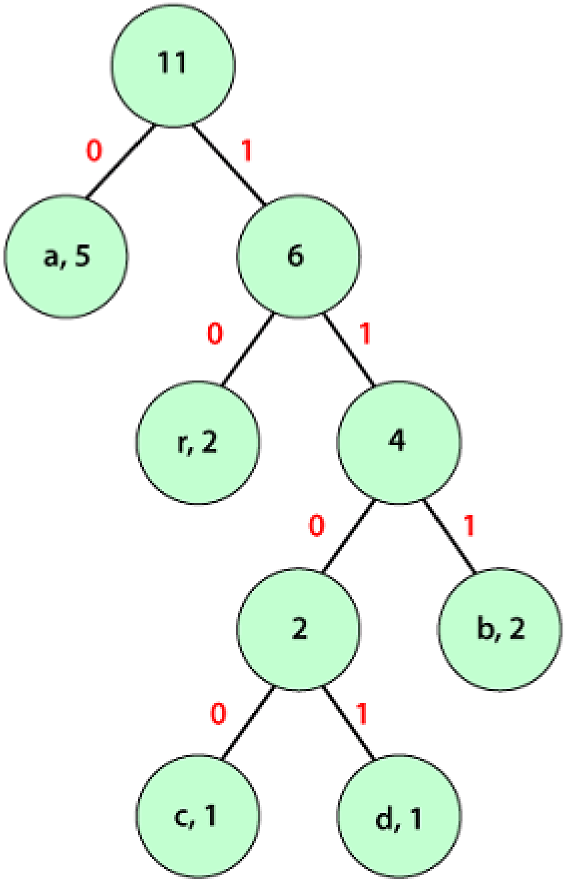


We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 5+6=**11.**



Therefore, we get a single tree.

At last, we will find the code for each character with the help of the above tree. Assign a weight to each edge. Note that each **left edge-weighted is 0** and the **right edge-weighted is 1.**



We observe that input characters are only presented in the leave nodes and the internal nodes have null values. In order to find the Huffman code for each character, traverse over the Huffman tree from the root node to the leaf node of that particular character for which we want to find code. The table describes the code and code length for each character.

Character	Frequency	Code	Code Length
A	5	0	1
B	2	111	3

C	1	1100	4
D	1	1101	4
R	2	10	2

We observe that the most frequent character gets the shortest code length and the less frequent character gets the largest code length.

Now we can encode the string **(abracadabra)** that we have taken above.

1. 0 111 10 0 1100 0 1101 0 111 10 0

(ii) Average Code Length for the String

The average code length of the Huffman tree can be determined by using the formula given below:

1. Average Code Length = ∑ (frequency × code length) / ∑ (frequency)

= { (5 x 1) + (2 x 3) + (1 x 4) + (1 x 4) + (2 x 2) } / (5+2+1+1+2)

= 2.09090909

(iii) Length of the Encoded String

The length of the encoded message can be determined by using the following formula:

1. length= Total number of characters in the text x Average code length per character

   = 11 x 2.09090909

   = 23 bits

Huffman Encoding Algorithm

```
Huffman (C)
n=|C|
Q=C
for i=1 to n-1
    do
        z=allocate_Node()
        x=left[z]=Extract_Min(Q)
        y=right[z]=Extract_Min(Q)
        f[z]=f[x]+f[y]
        Insert(Q,z)
        return Extract_Min(Q)
```

The Huffman algorithm is a greedy algorithm. Since at every stage the algorithm looks for the best available options.

The time complexity of the Huffman encoding is **O(nlogn).** Where n is the number of characters in the given text.

Huffman Decoding

Huffman decoding is a technique that converts the encoded data into initial data. As we have seen in encoding, the Huffman tree is made for an input string and the characters are decoded based on their position in the tree. The decoding process is as follows:

- Start traversing over the tree from the **root** node and search for the character.
- If we move left in the binary tree, add **0** to the code.
- If we move right in the binary tree, add **1** to the code.

The child node holds the input character. It is assigned the code formed by subsequent 0s and 1s. The time complexity of decoding a string is **O(n),** where n is the length of the string.