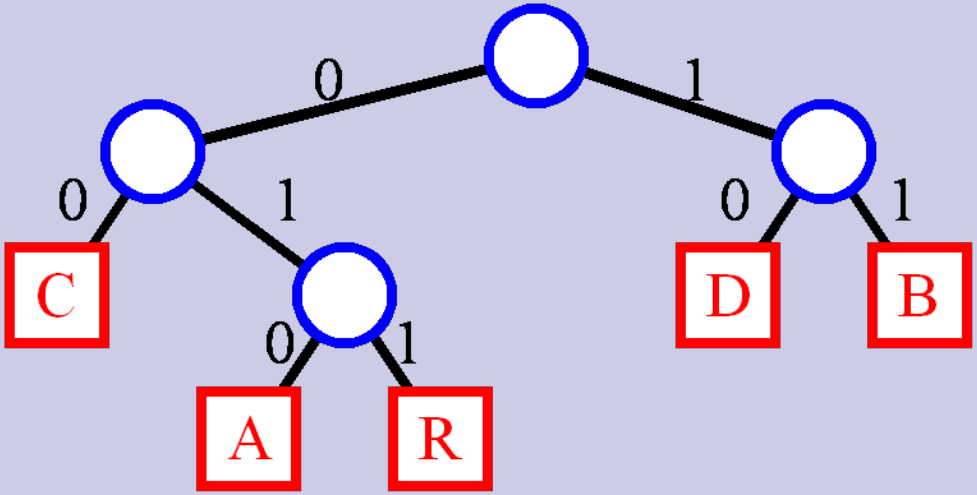
# Data Compression

- File Compression
- Huffman Tries




ABRACADABRA

01011011010000101001011011010

# File Compression

- Text files are usually stored by representing each character with an 8-bit ASCII code (type man ascii in a Unix shell to see the ASCII encoding)
- The ASCII encoding is an example of **fixed-length encoding**, where each character is represented with the same number of bits
- In order to reduce the space required to store a text file, we can exploit the fact that some characters are more likely to occur than others
- **variable-length encoding** uses binary codes of different lengths for different characters; thus, we can assign fewer bits to frequently used characters, and more bits to rarely used characters.

# File Compression: Example

- An Encoding Example

  text: java

  encoding: **a = “0”, j = “11”, v = “10”**

  encoded text: **110100** (6 bits)

- How to decode (problems in ambiguity)?

  encoding: **a = “0”, j = “01”, v = “00”**

  encoded text: **010000** (6 bits)

  could be "java", or "jvv", or "jaaaa"

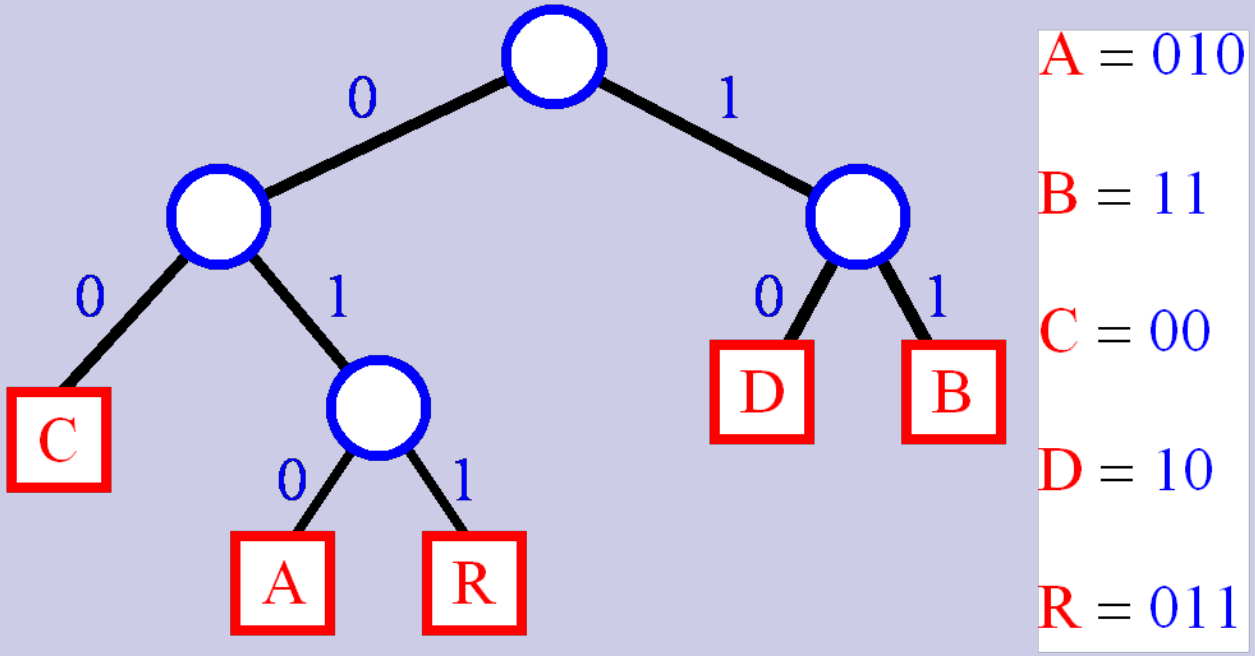
# Encoding Trie

- To prevent ambiguities in decoding, we require that the encoding satisfies the **prefix rule**: no code is a prefix of another.
  - a = “0”, j = “11”, v = “10” satisfies the prefix rule
  - a = “0”, j = “01”, v= “00” does **not** satisfy the prefix rule (the code of 'a' is a prefix of the codes of 'j' and 'v')

# Encoding Trie(2)

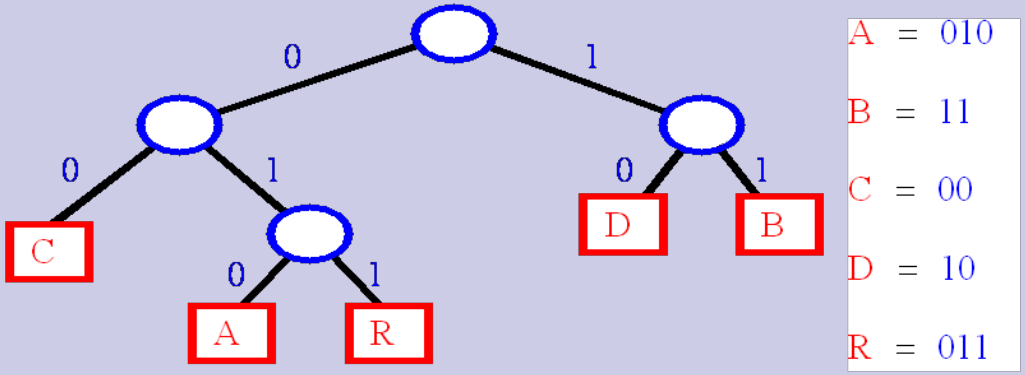
We use an **encoding trie** to satisfy this prefix rule

- the characters are stored at the external nodes.
- a left child (edge) means 0
- a right child (edge) means 1



# Example of Decoding

- trie

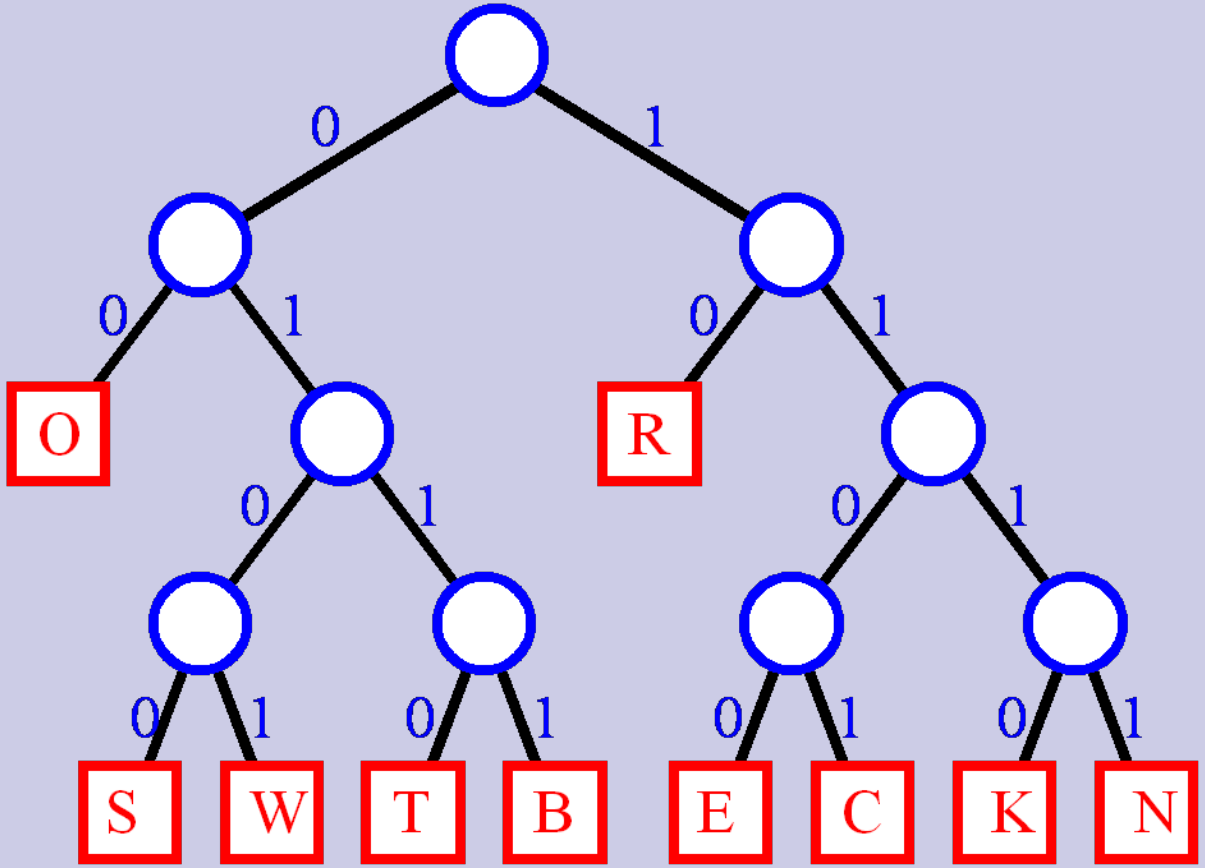



- encoded text: 010110110100001010011011010
- text:

***ABRACADABRA***

# Trie this!

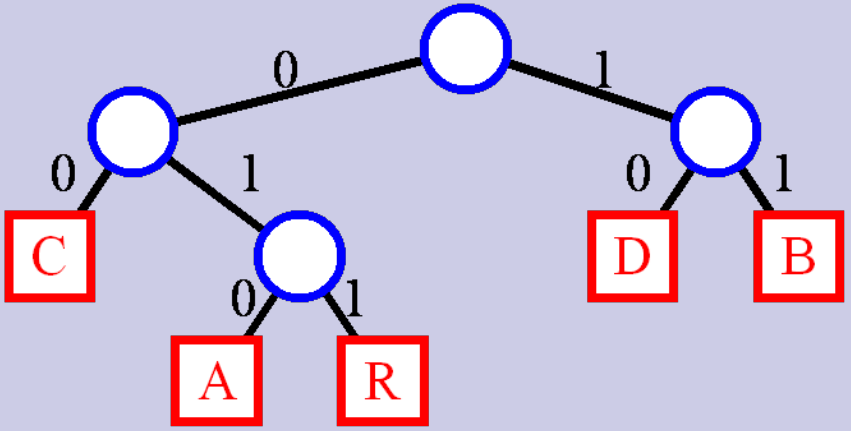
1000011111100100110001110111100010101001100



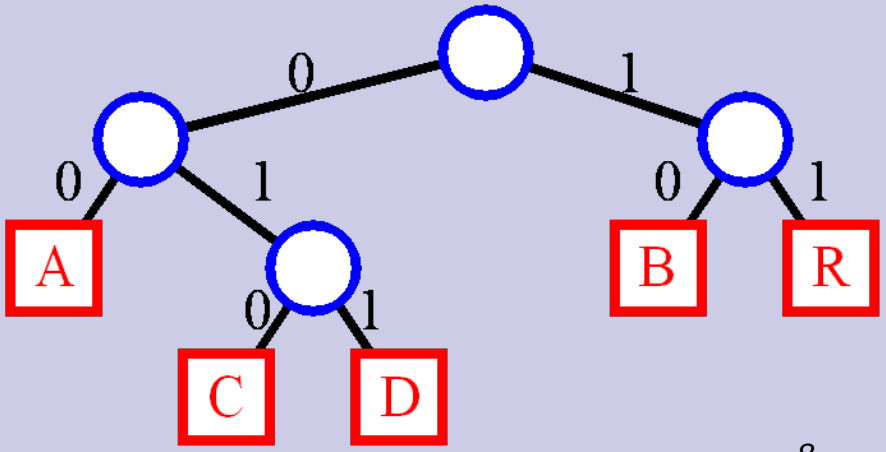
# Optimal Compression

- An issue with encoding tries is to ensure that the encoded text is as short as possible:

ABRACADABRA
01011011010000101001011010101010
**29 bits**




ABRACADABRA
00101100010000110010110000
**24 bits**



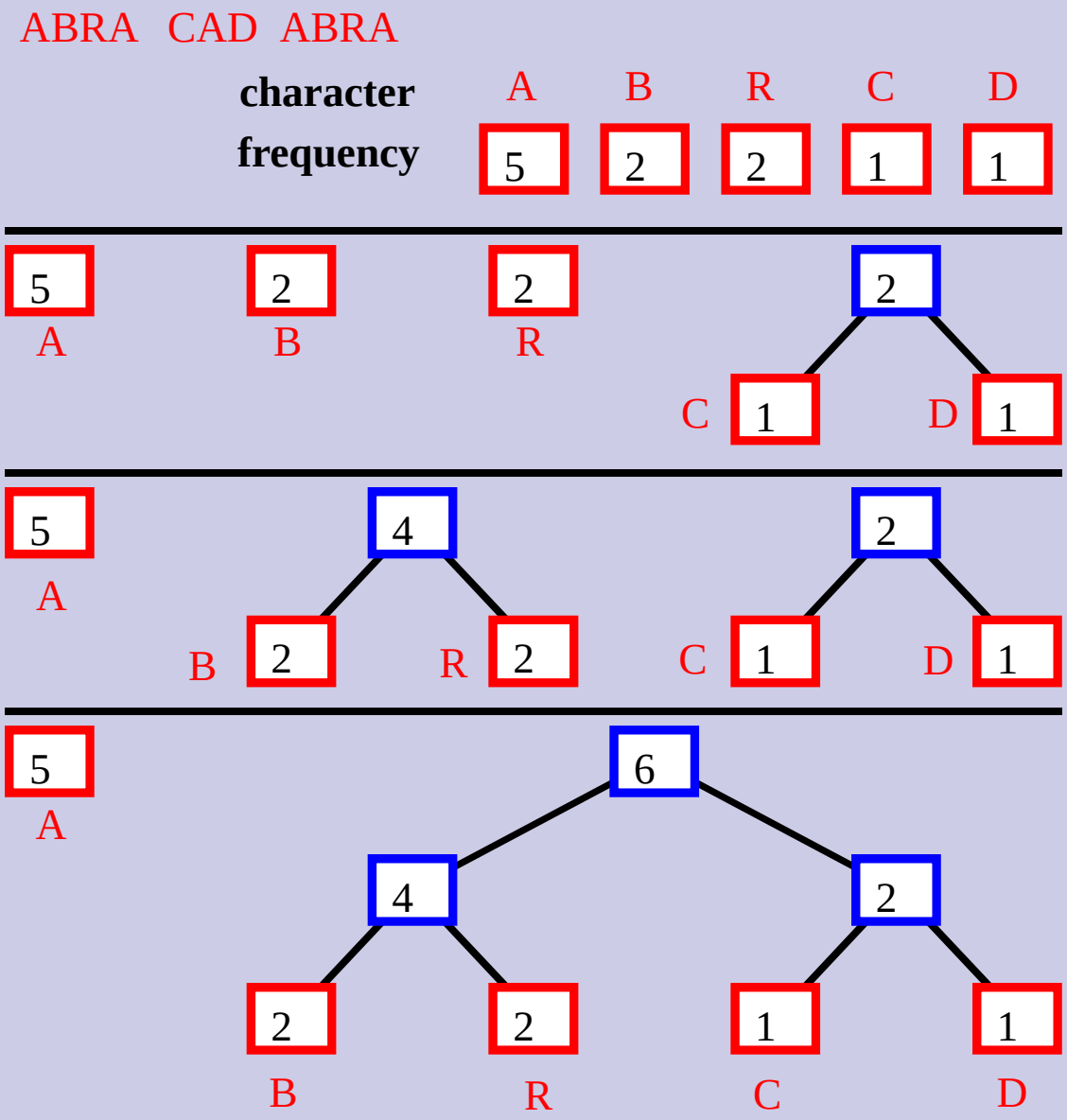
# Construction algorithm

- Given frequencies of characters we wish to compute a trie so that the length of the encoding is minimum possible.
- Each character is a leaf of the trie
- The number of bits used to encode a character is its level number (root is 0).
- Thus if $f_i$ is the frequency of the $i^{th}$ character and $l_i$ is the level of the leaf corresponding to it then we want to find a tree which minimises $\sum_i f_i l_i$

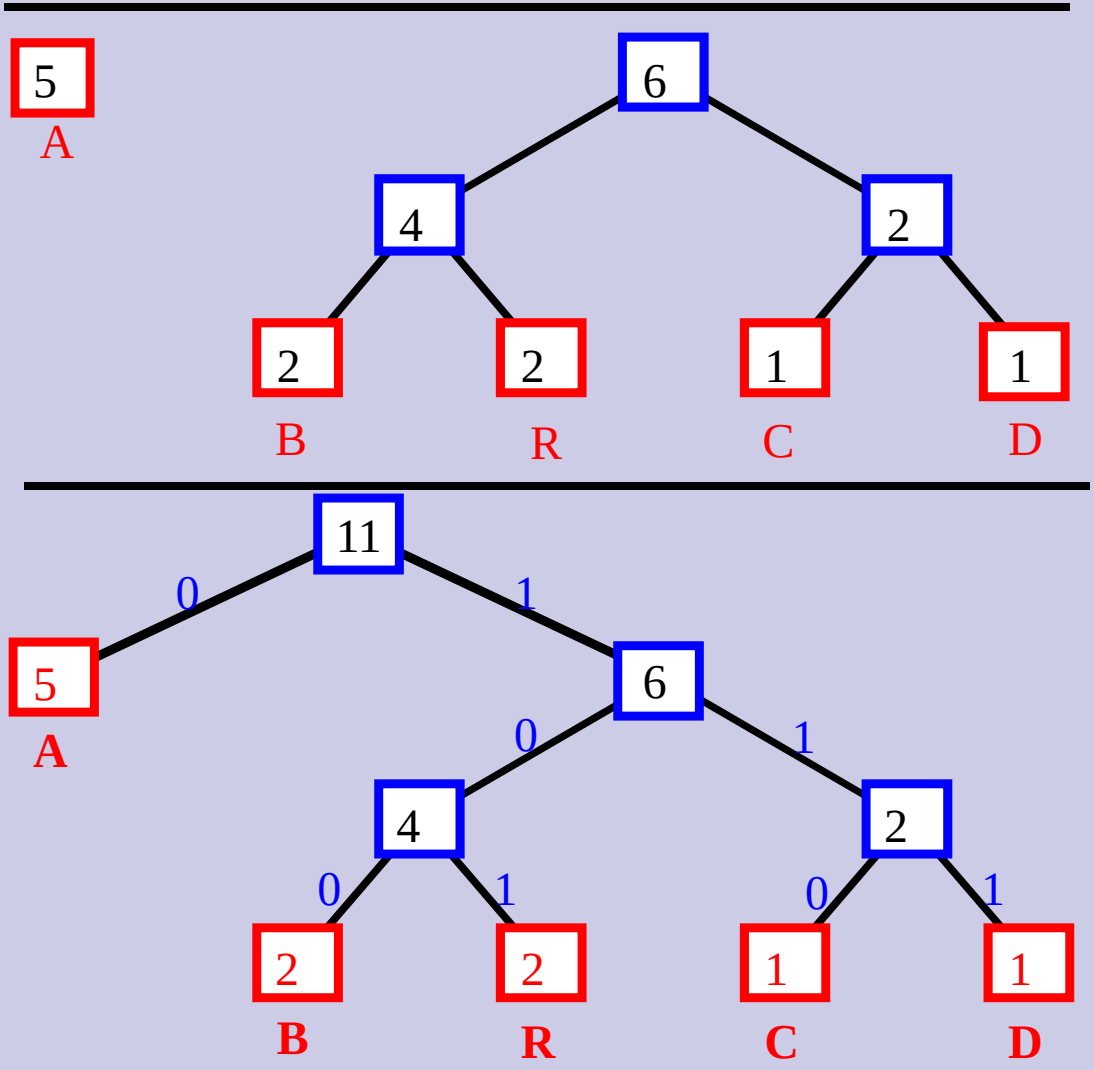
# Total weighted external path length

- The quantity $\sum_i f_i l_i$ is called the total external weighted path length of a tree.
- We view each leaf as having a weight equal to the frequency of the corresponding character.
- Given weights $f_1, f_2, \ldots, f_n$ we wish to find a tree whose weighted external path length is minimum.
- We denote this by WEPL($f_1, f_2, \ldots, f_n$)

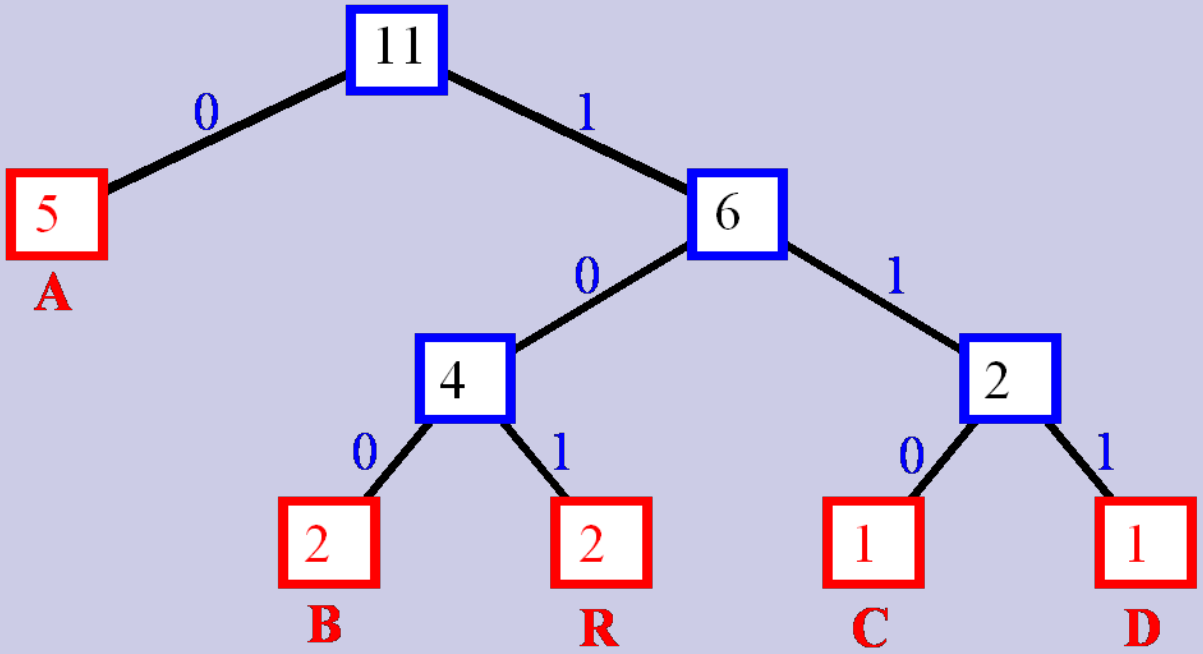
# Huffman Encoding Trie



# Huffman Encoding Trie (contd.)



# Final Huffman Encoding Trie

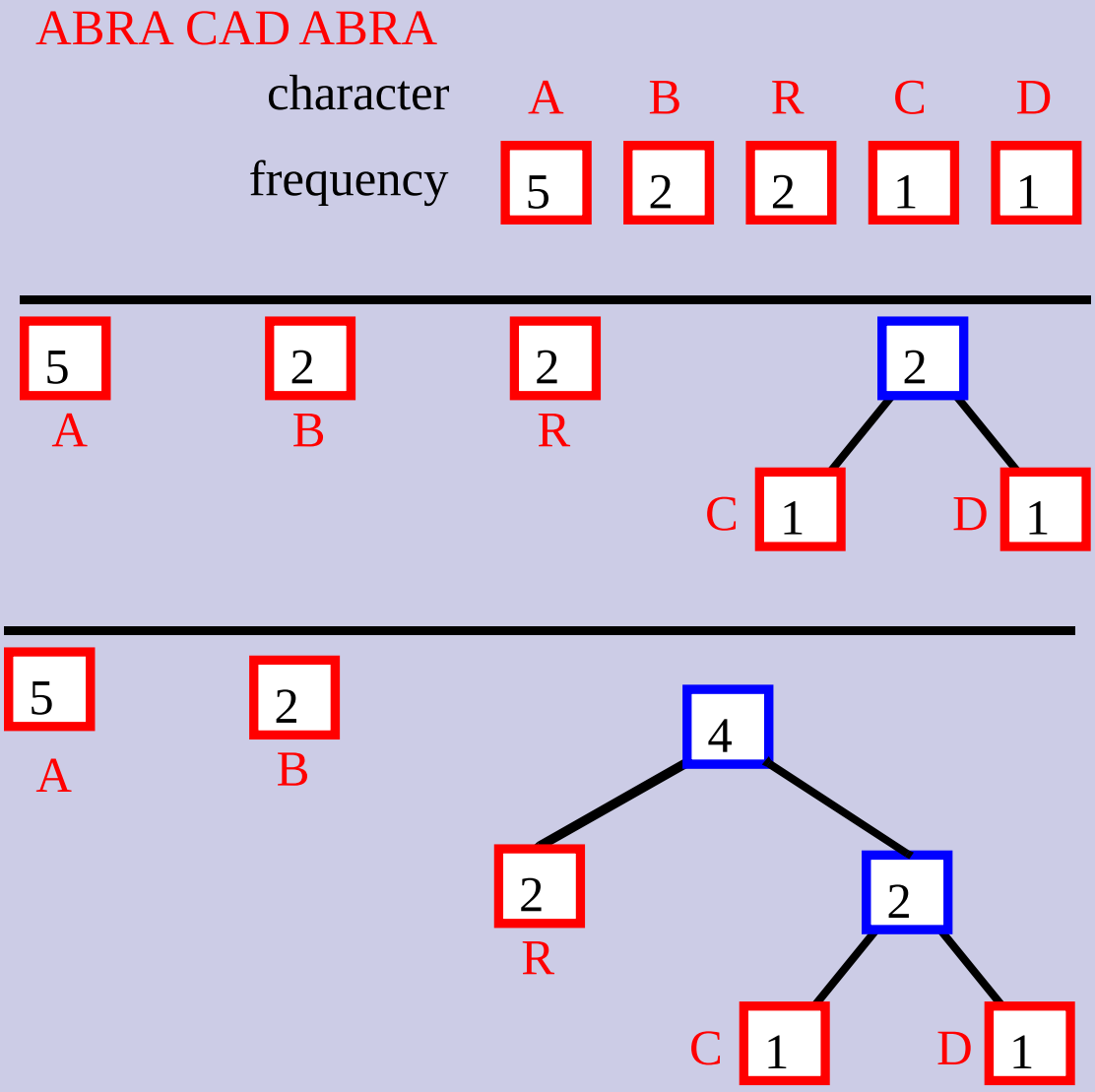



A B R A C A D A B R A

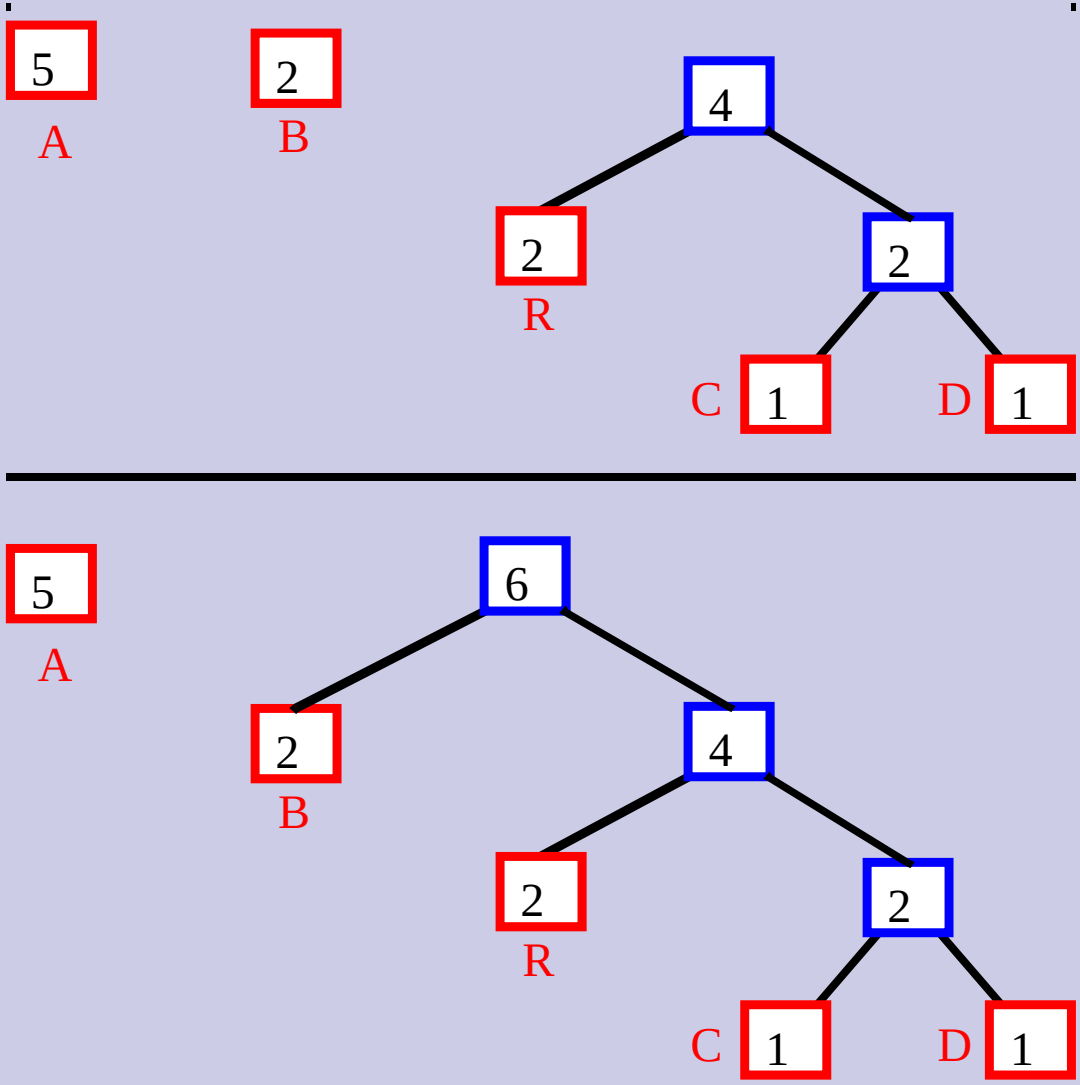
0 100 101 0 110 0 111 0 100 1010

**23 bits**

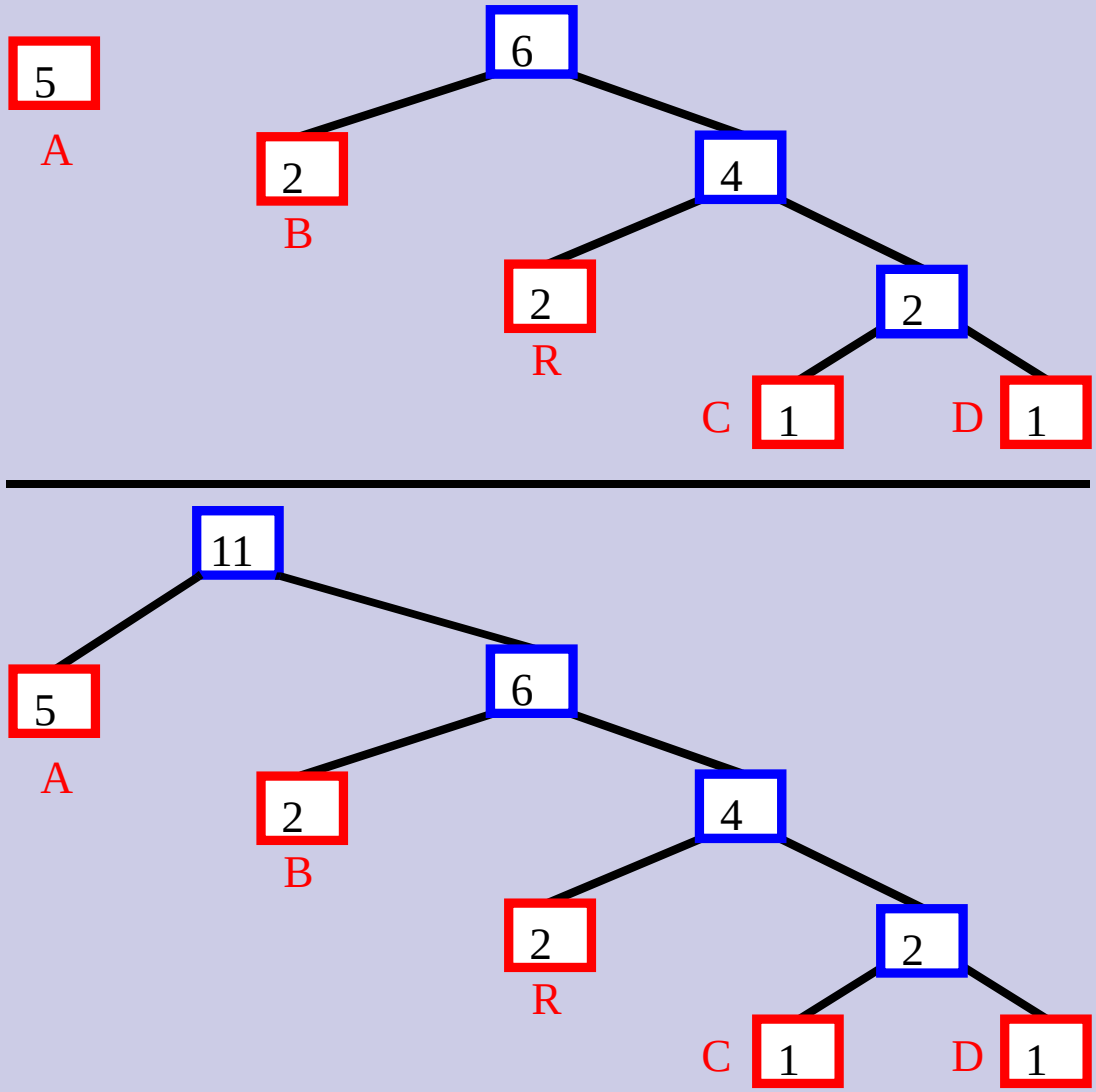
# Another Huffman Encoding Trie



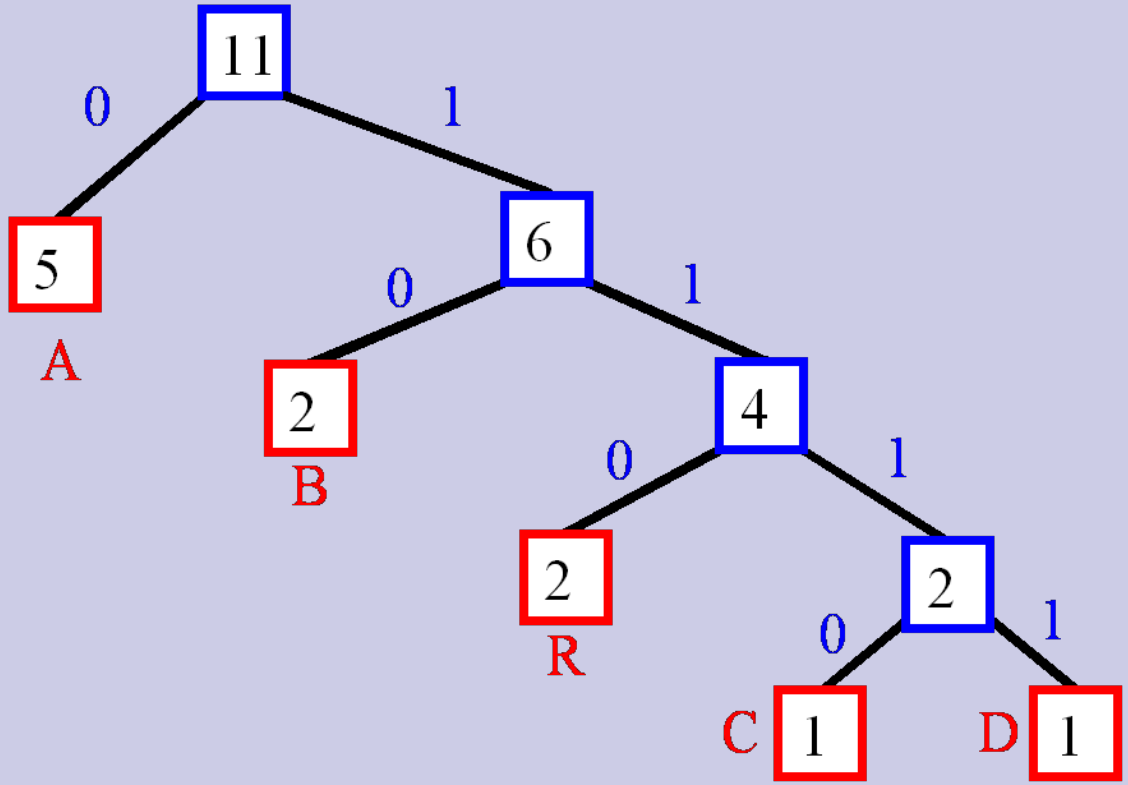
# Another Huffman Encoding Trie



# Another Huffman Encoding Trie



# Another Huffman Encoding Trie




A B R A C A D A B R A

0 10 110 0 1100 0 1111 0 10 110 0

**23 bits**

# Correctness of the algorithm

- Why does this algorithm compute a tree with the minimum weighted external path length?
- We prove this by induction on the number of leaves/characters.
- The claim is true when we have only two leaves.
- Suppose claim is true when we have n-1 characters.

# Correctness of the algorithm(2)

- When we have n characters at the first step we replace the two characters with frequencies $f_1$,$f_2$ with one character of frequency $f_1+f_2$.
- Beyond this point the algorithm behaves as if it had only n-1 characters. Hence it computes a tree with min. total weighted external path length i.e. WEPL($f_1+f_2$, $f_3$,…$f_n$)
- Hence the tree computed has weighted external path length $f_1+f_2$+ WEPL($f_1+f_2$, $f_3$,…$f_n$)

# Correctness of the algorithm(3)

- We now argue that

$WEPL(f_1,f_2, f_3,\ldots f_n) = f_1+f_2+ WEPL(f_1+f_2, f_3,\ldots f_n)$

- This follows from the fact that in the optimum tree the leaves with the two lowest weights are siblings