



Análise e Projeto de Algoritmos

Algoritmos Gulosos



Algoritmo de Huffman

Gilberto Farias de Sousa Filho



História



- ❖ Criado em 1951 por David Albert Huffman.
- ❖ Foi dada a opção de estudar para a prova final ou elaborar um trabalho acadêmico.
- ▶ Qual a forma mais eficiente de representar símbolos através de códigos binários ?





Motivações

- ❖ Compressão de Dados
- ▶ Salvar Espaço de Armazenamento
- ▶ Agilizar comunicação de informação
- ❖ Machine Learning (Clusterização de Textos)

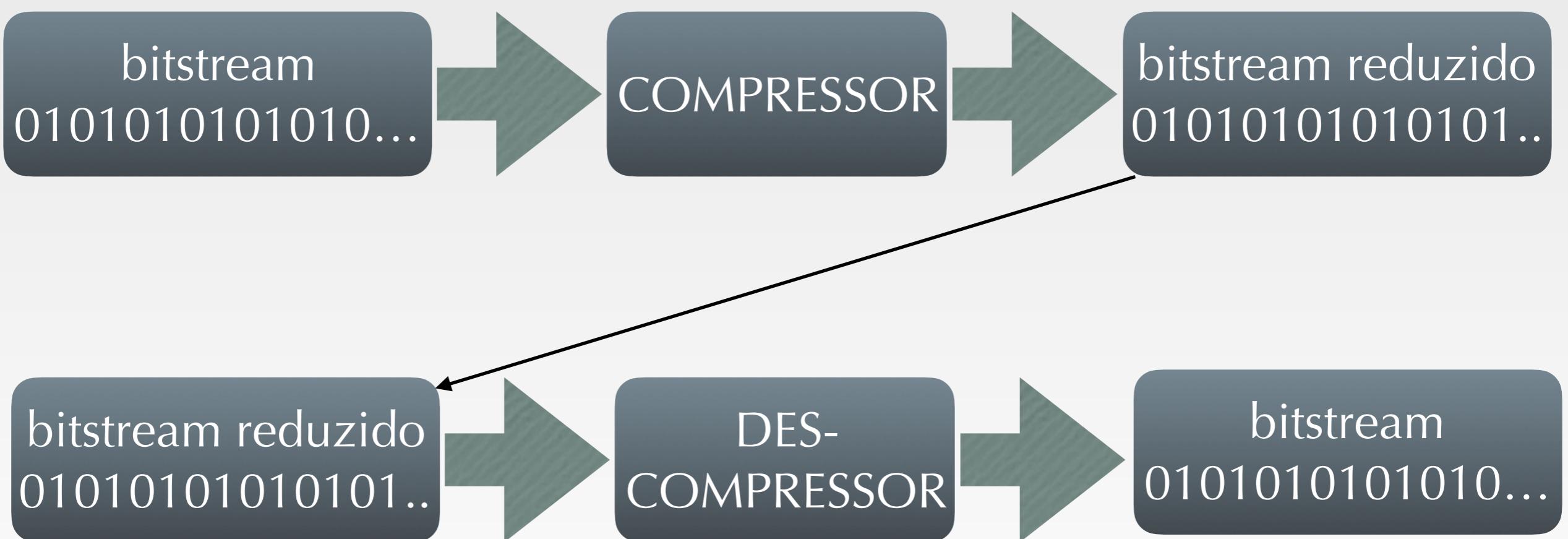


Regras do Jogo

- ❖ Dados são representados através de sequências de bits (bitstreams).
- ❖ Sequências de bits podem ser visualizadas através de sequências de bytes (bytestreams).
- ❖ O objetivo é comprimir bytestreams **sem ter perda de informação** na hora da decompressão.



Modelo Básico





Compressão Estatística



- ❖ Bytes não possuem mesma frequência de ocorrência em certos tipos de arquivos (Texto, imagens ...). Mesmo assim, eles possuem mesma quantidade de bits.
 - ▷ Ex: chars ASCII têm 1 byte. Seja esse char <e> ou <~>
- ❖ A ideia da compressão estatística é atribuir codificações binárias de tamanho variável para representar bytes.



Codificação Binária



- ❖ Considere tabela de codificação binária para chars ASCII ao lado.
- ❖ Através dela o texto <iaaoa> pode ser codificado como <100101101110101>.
- ❖ Entretanto, existem duas decodificações diferentes. <iaaoa> ou <iaaeu>.
- ❖ Por que isso acontece ?

Byte	Código
a	101
e	11
i	100
o	110
u	0101



Codificação Binária

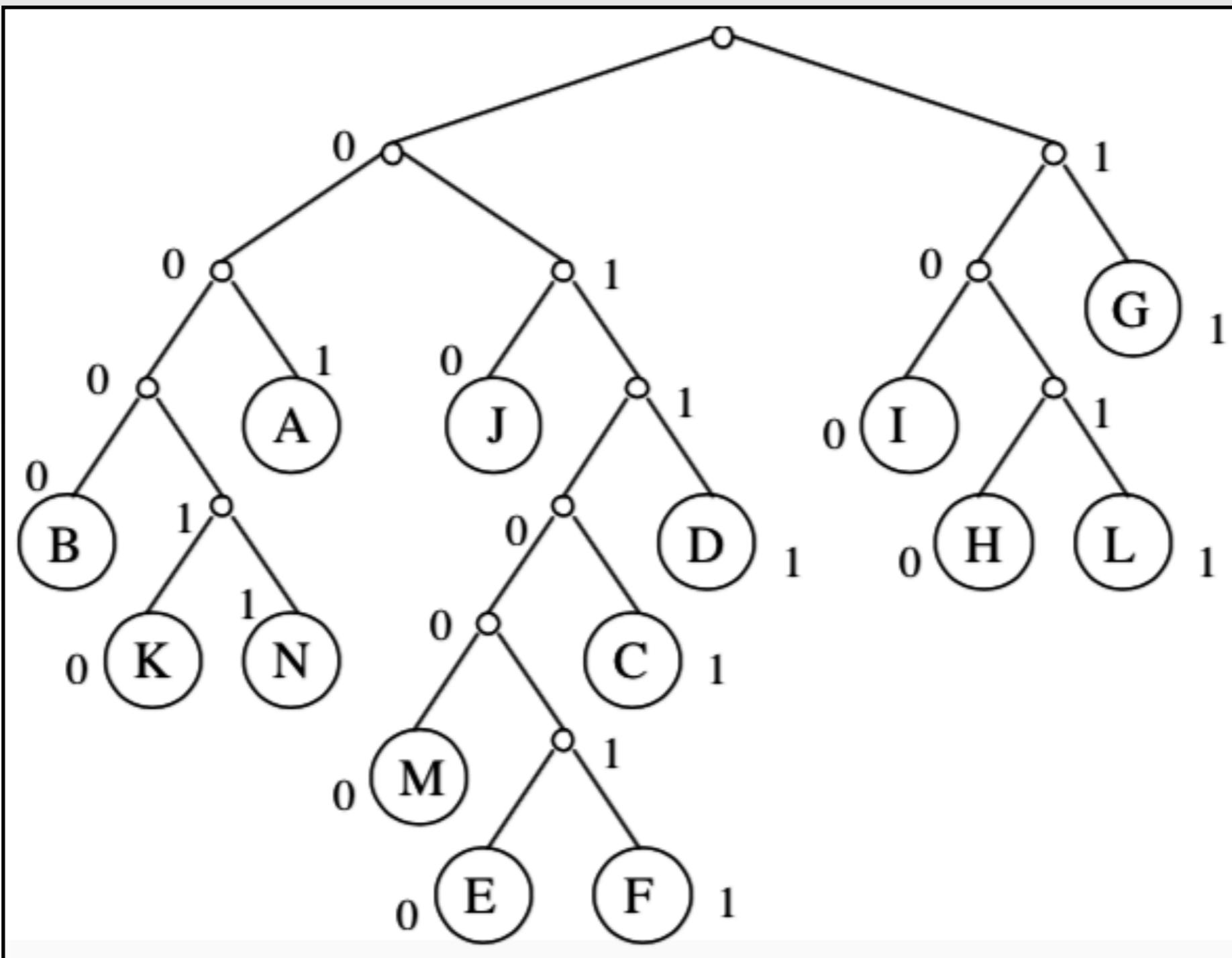


- ❖ O problema de ambiguidade na decodificação ocorre quando existe um código que é prefixo de outro código.

Byte	Código
a	101
e	11
i	100
o	110
u	0101



Codificação de Prefixo



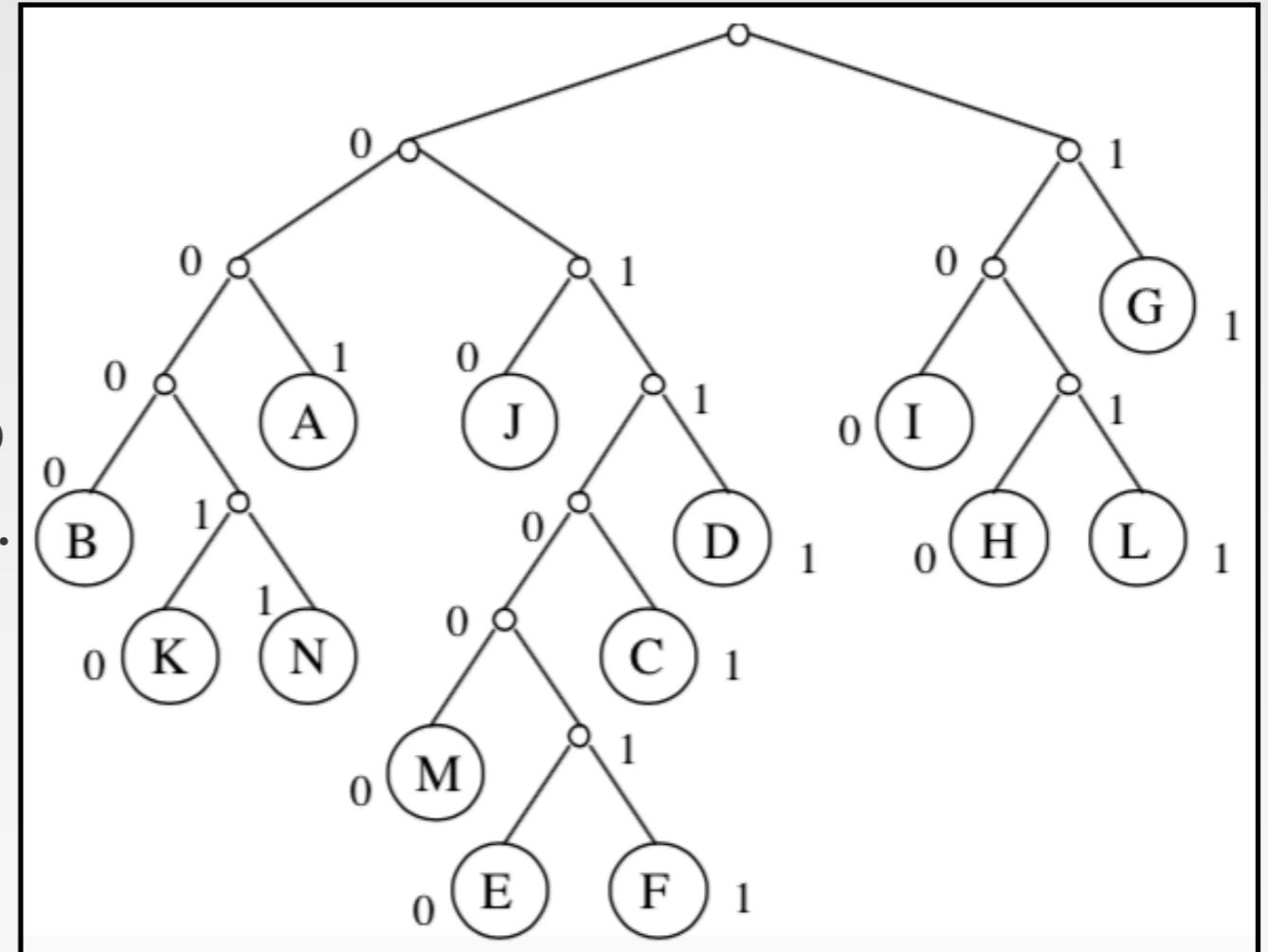


Codificação de Prefixo

- ❖ Como fazer para gerar uma codificação que seja a menor possível ?
- ❖ Formalmente, seja A o alfabeto do bytestream a ser codificado. O problema consiste em:

$$\min \sum n_i c_i$$

- ❖ Onde n_i e c_i representam, respectivamente, a frequência e codificação de um byte a_i de A .





Codificação Binária de Huffman



- ❖ O algoritmo de Huffman realiza compressão estatística.
- ❖ O algoritmo de Huffman é o melhor gerador de codificações inteiras possível.
- ❖ O algoritmo de codificação de Huffman associa uma árvore a cada byte. Inicialmente, cada árvore possui um único nó, com peso igual à probabilidade de ocorrência do byte a ela associado. A cada iteração do algoritmo, as duas árvores de menor peso são substituídas por uma nova árvore cujo peso é a soma dos seus pesos. O procedimento termina quando resta apenas uma única árvore.



Árvore de Huffman



File :

b	p	'	m	j	o	d	a	i	r	u	l	s	e	
1	1	2	2	3	3	3	4	4	5	5	6	6	8	12



Árvore de Huffman



File :

b	p	'	m	j	o	d	a	i	r	u	l	s	e	
1	1	2	2	3	3	3	4	4	5	5	6	6	8	12

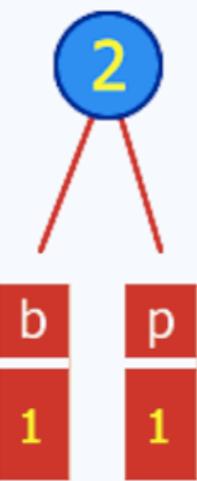


Árvore de Huffman



File :

'	m	j	o	d	a	i	r	u	l	s	e
2	2	3	3	3	4	4	5	5	6	6	8 12



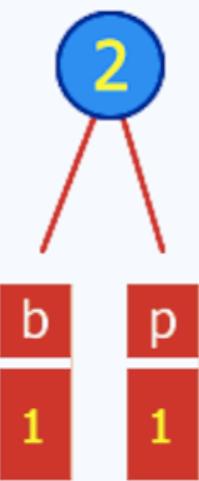


Árvore de Huffman



File :

'	m	j	o	d	a	i	r	u	l	s	e
2	2	3	3	3	4	4	5	5	6	6	8 12



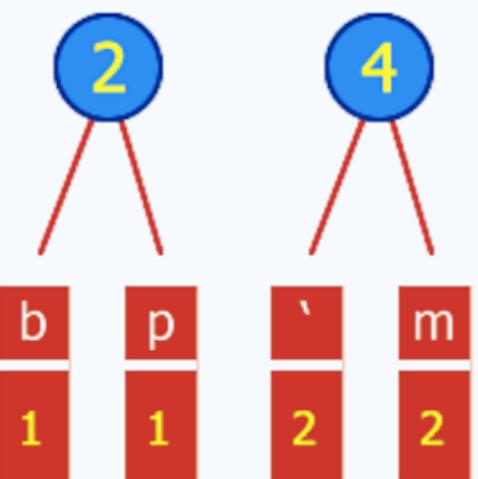


Árvore de Huffman



File :

j	o	d	a	i	r	u	l	s	e	
3	3	3	4	4	5	5	6	6	8	12



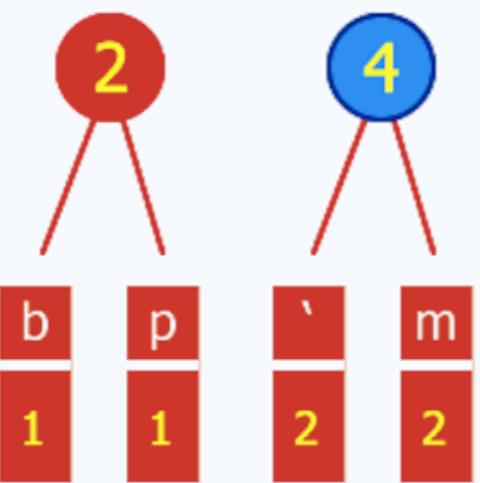


Árvore de Huffman



File :

j	o	d	a	i	r	u	l	s	e
3	3	3	4	4	5	5	6	6	8 12



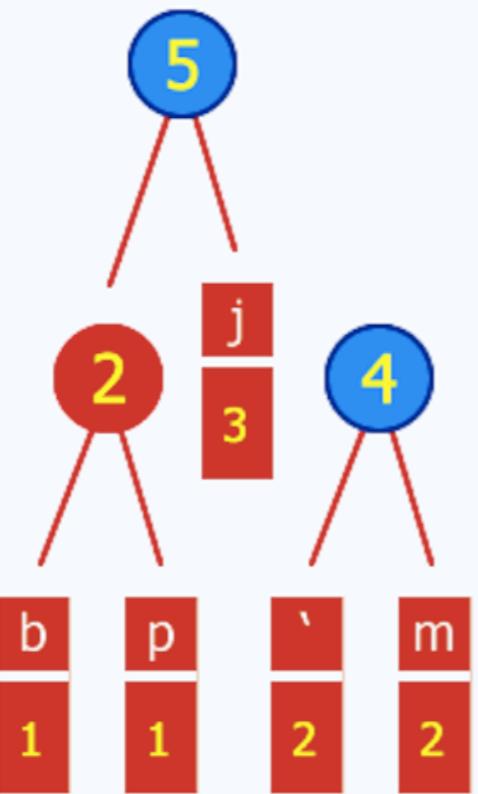


Árvore de Huffman



File :

o	d	a	i	r	u	l	s	e
3	3	4	4	5	5	6	6	8 12



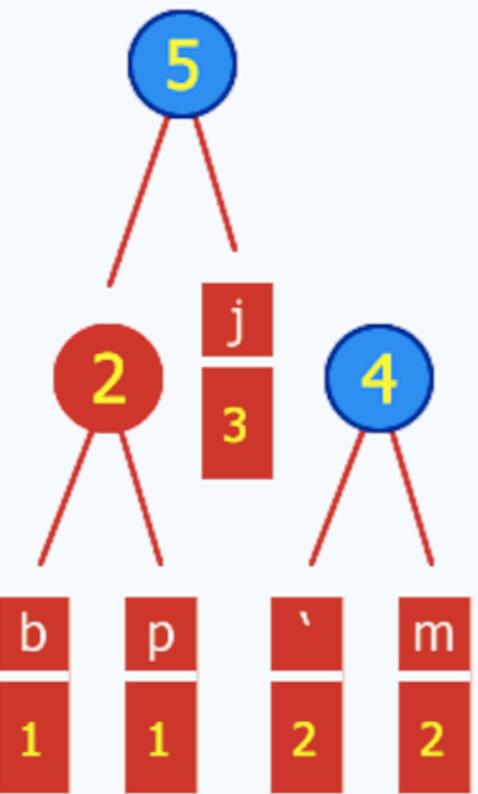


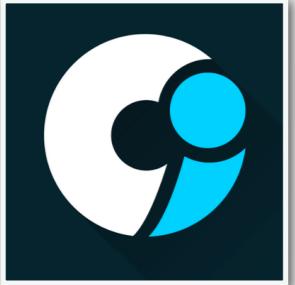
Árvore de Huffman



File :

o	d	a	i	r	u	l	s	e
3	3	4	4	5	5	6	6	8 12



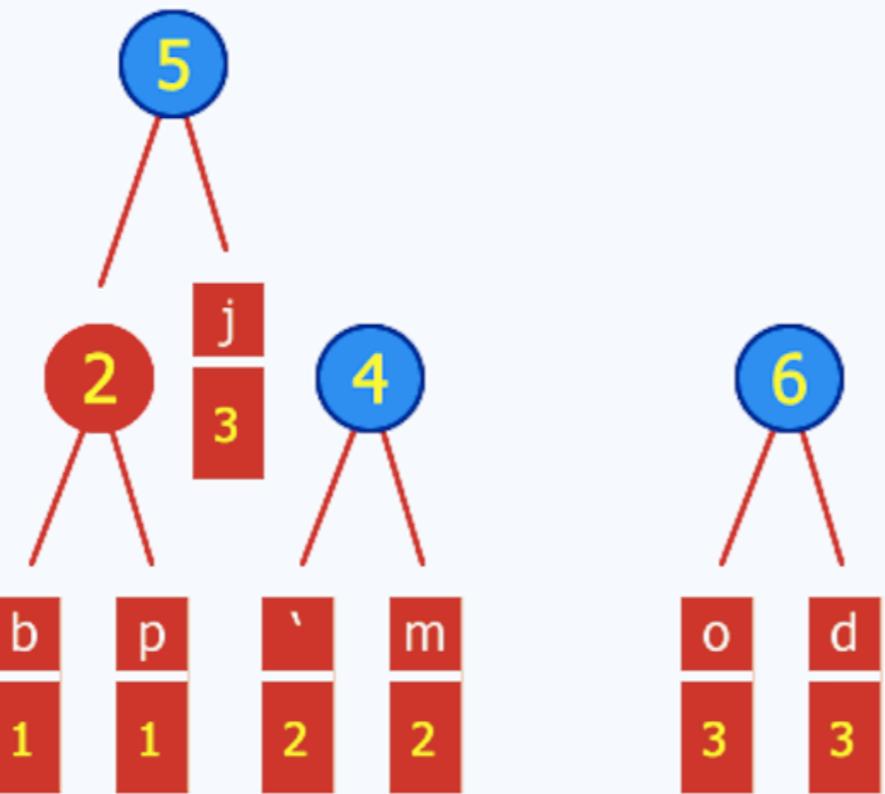


Árvore de Huffman



File :

a	i	r	u	l	s	e	
4	4	5	5	6	6	8	12



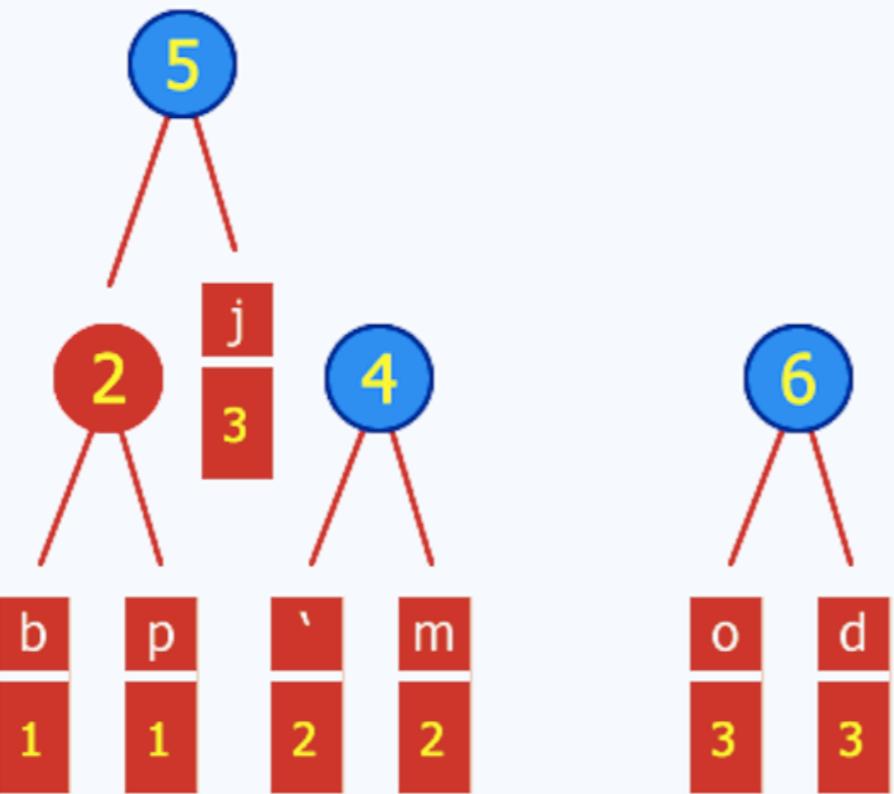


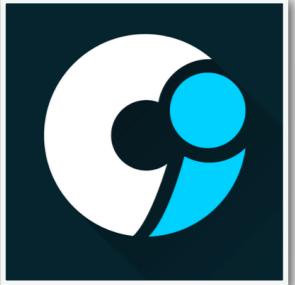
Árvore de Huffman



File :

a	i	r	u	l	s	e	
4	4	5	5	6	6	8	12



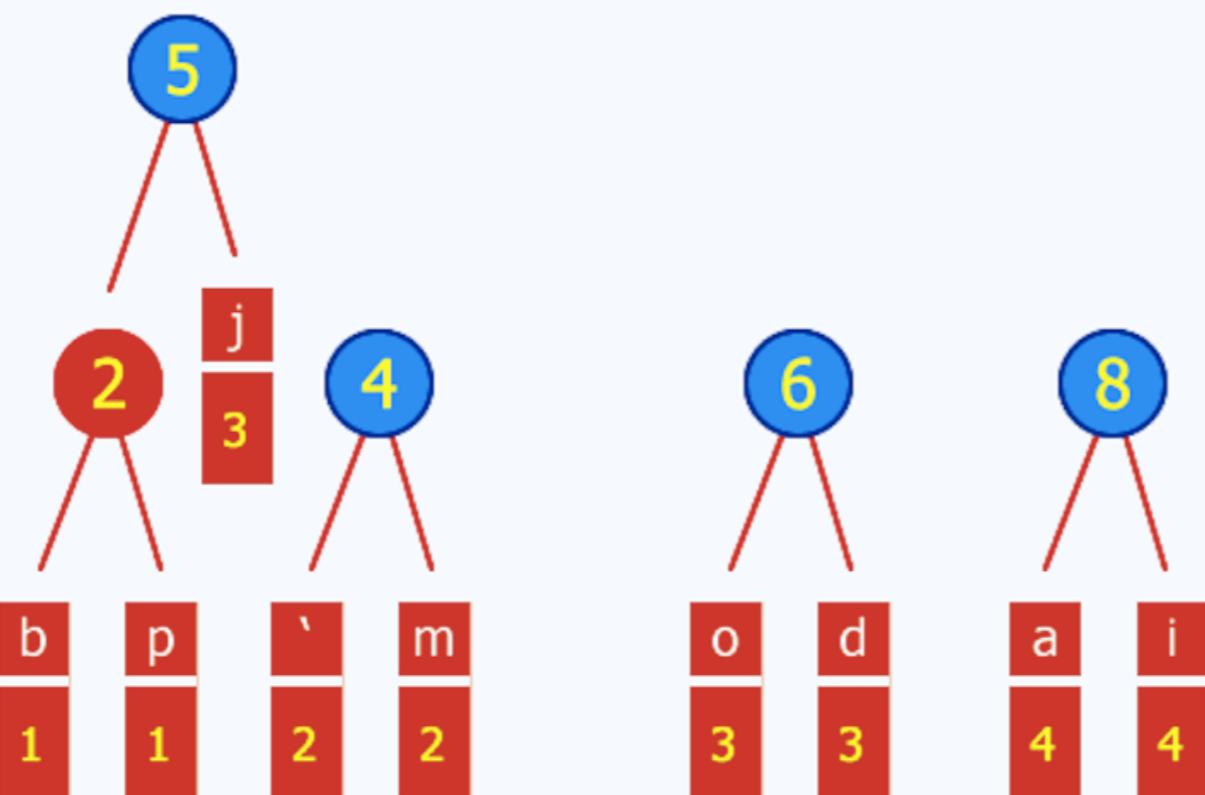


Árvore de Huffman



File :

r	u	l	s	e	
5	5	6	6	8	12



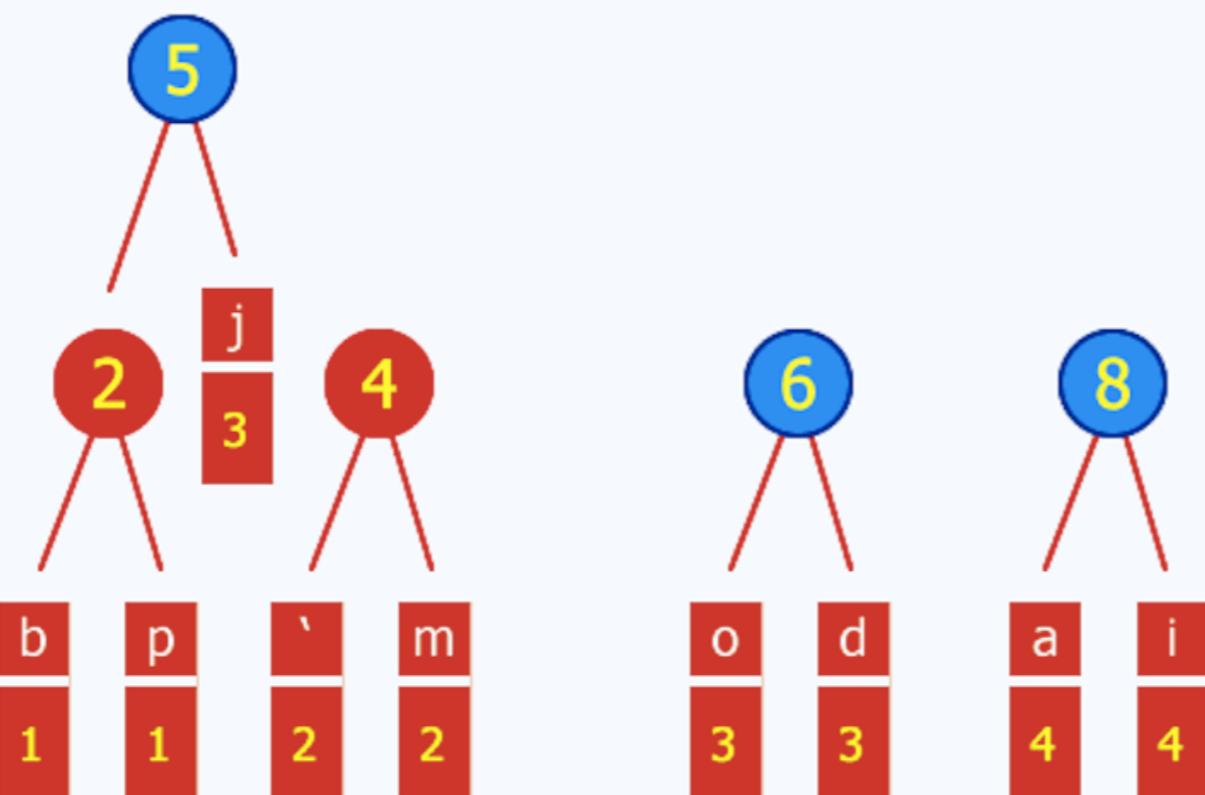


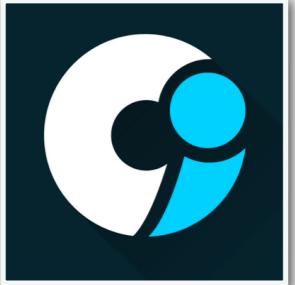
Árvore de Huffman



File :

r	u	l	s	e	
5	5	6	6	8	12



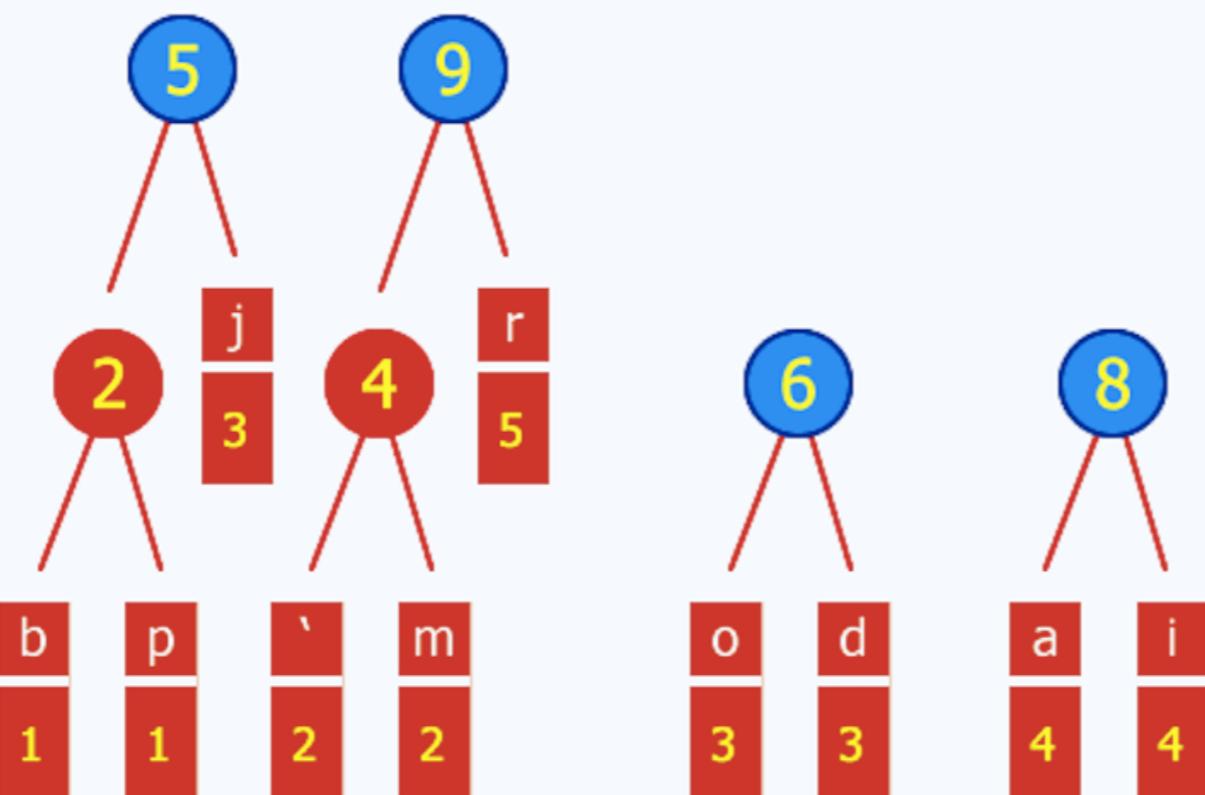


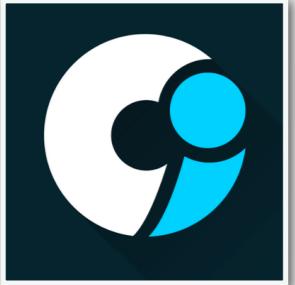
Árvore de Huffman



File :

u	l	s	e	
5	6	6	8	12



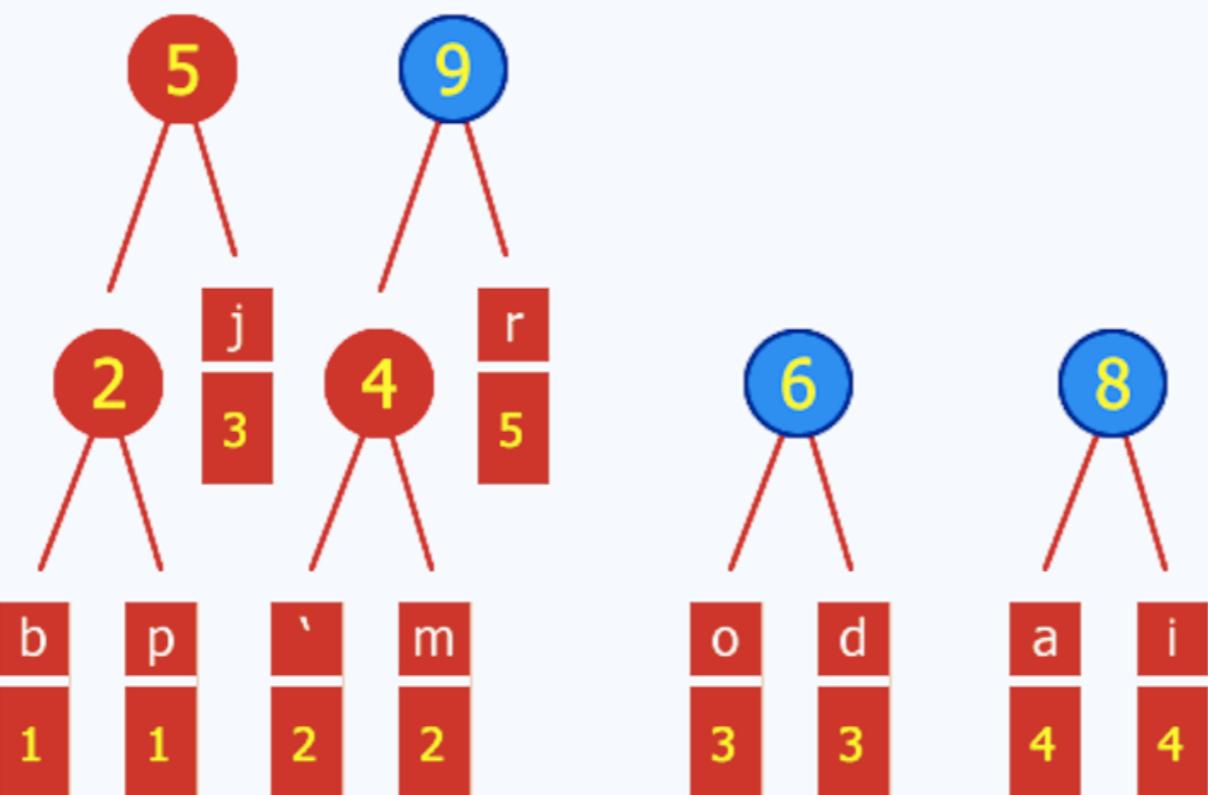


Árvore de Huffman



File :

u	l	s	e	
5	6	6	8	12



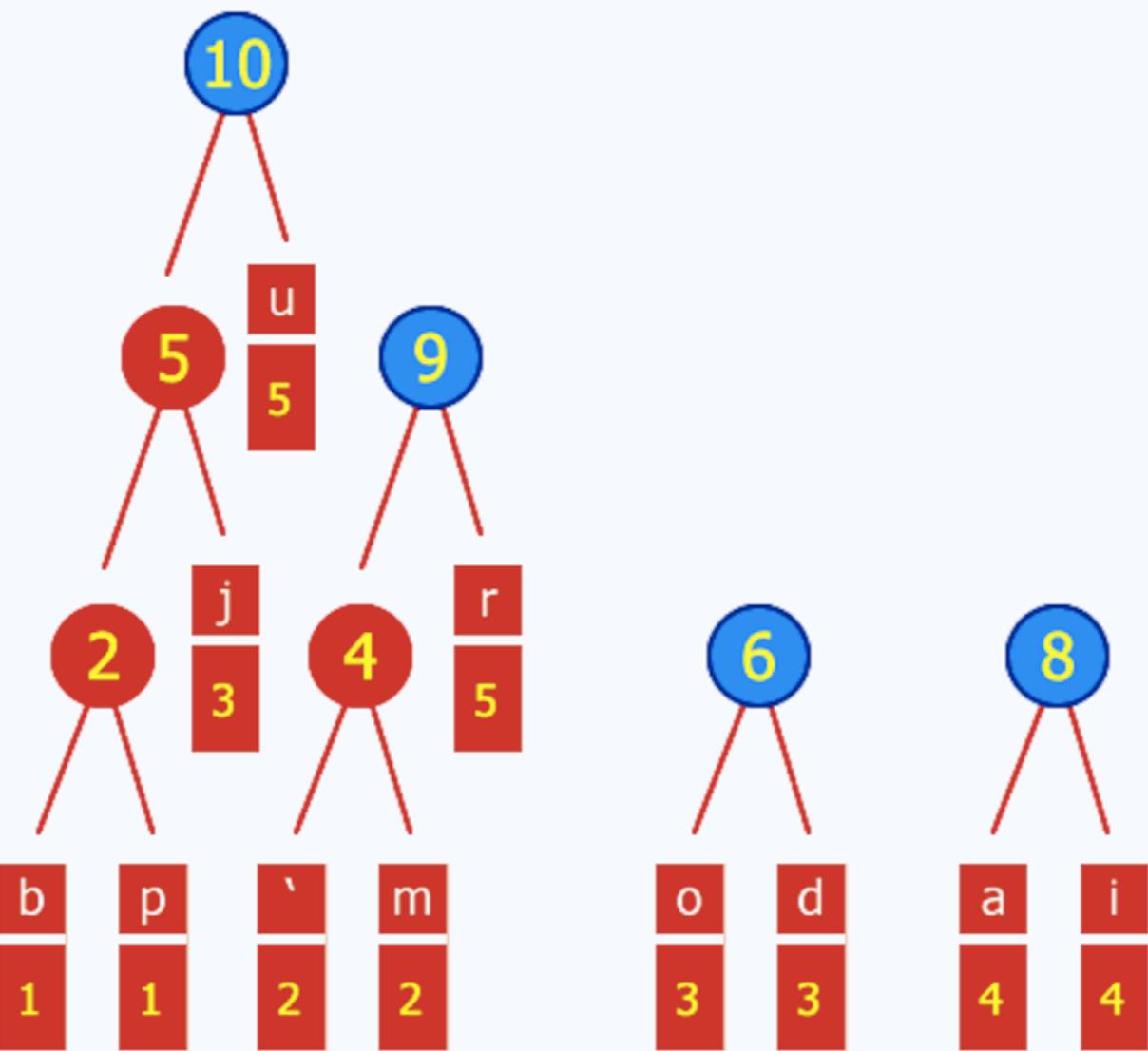


Árvore de Huffman



File :

I	s	e	
6	6	8	12



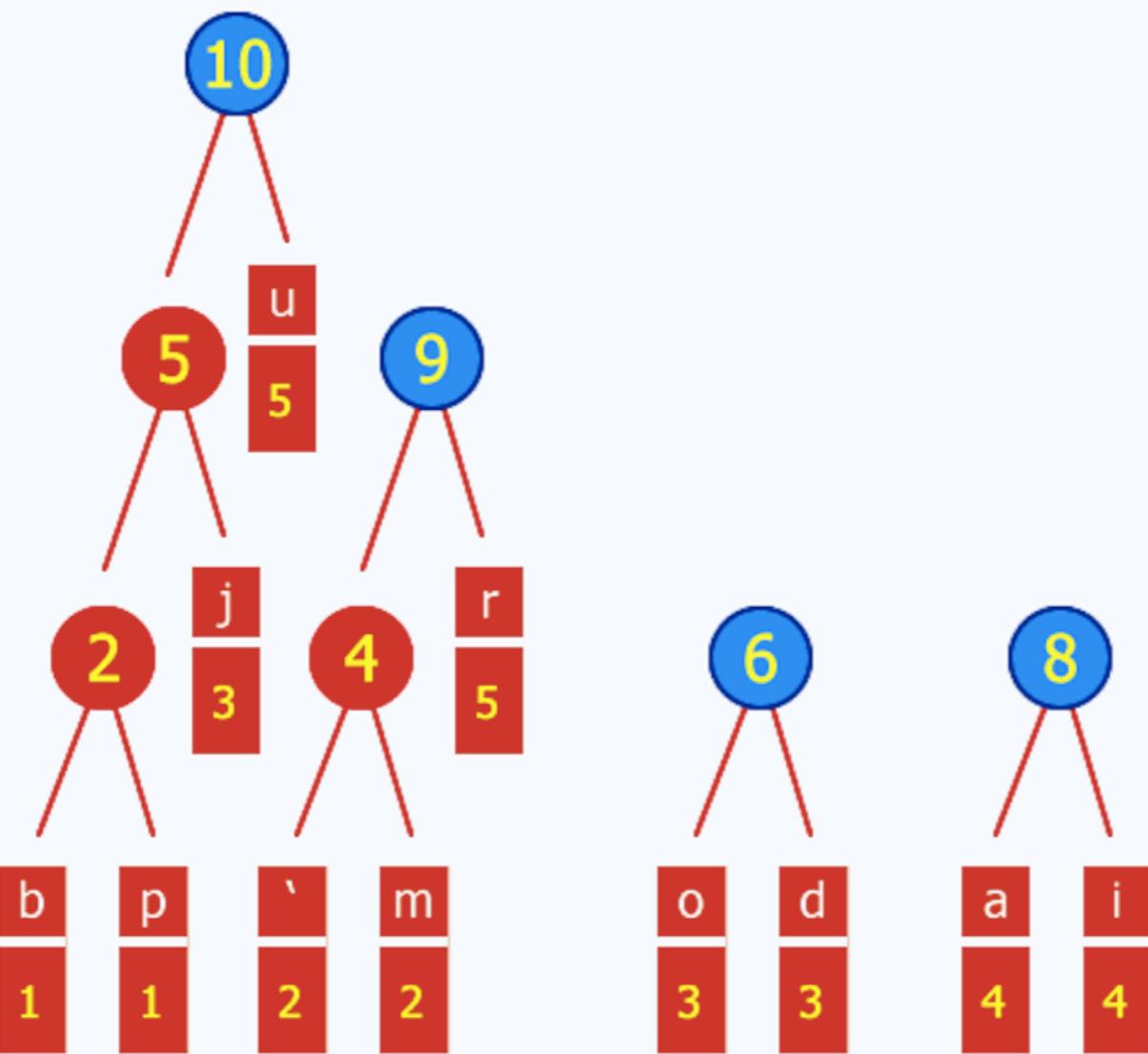


Árvore de Huffman



File :

I	s	e	
6	6	8	12



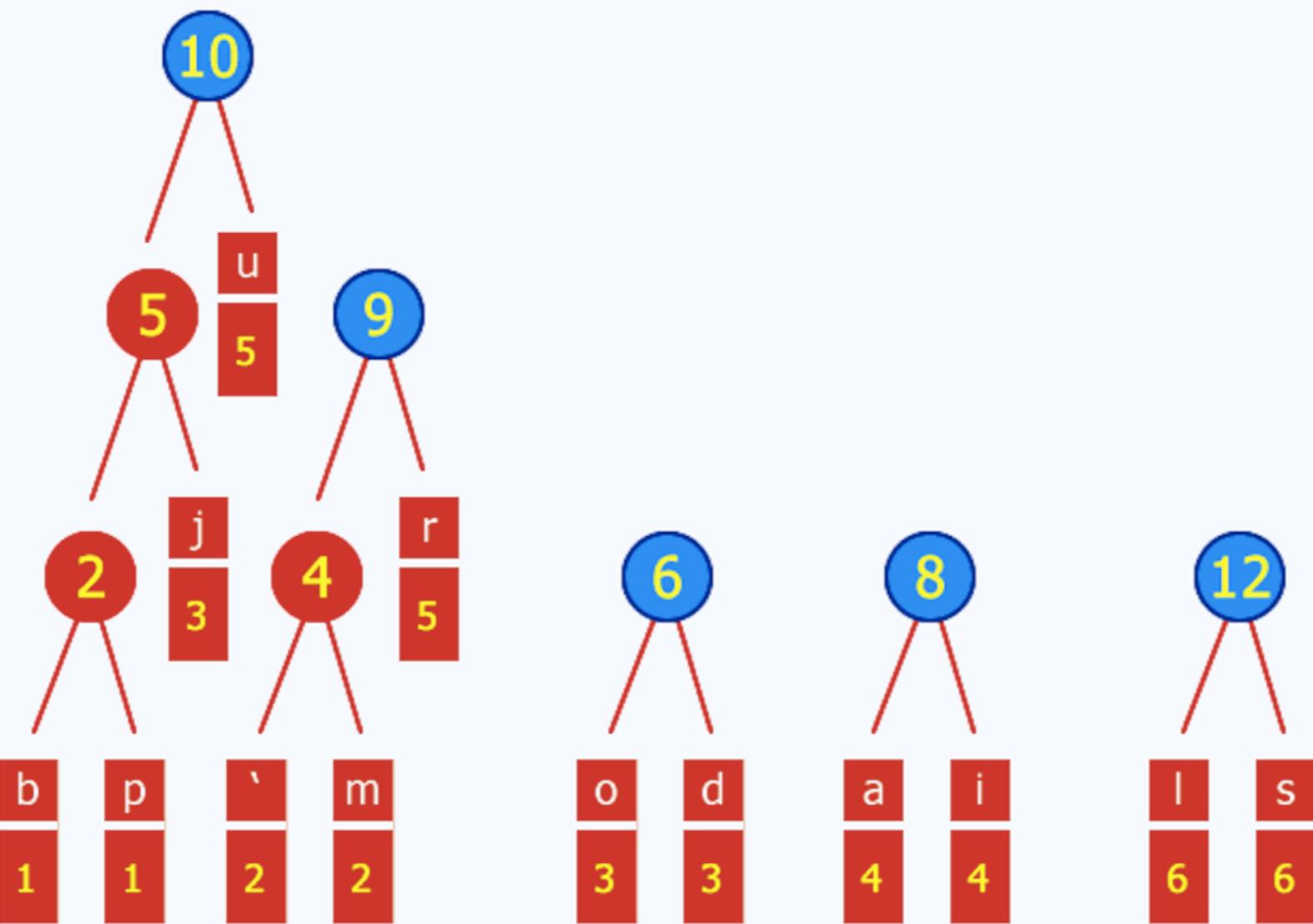


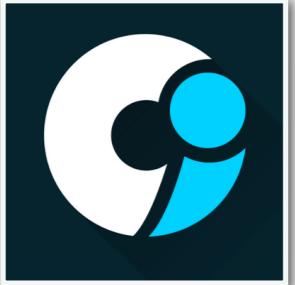
Árvore de Huffman



File :

e	
8	
	12



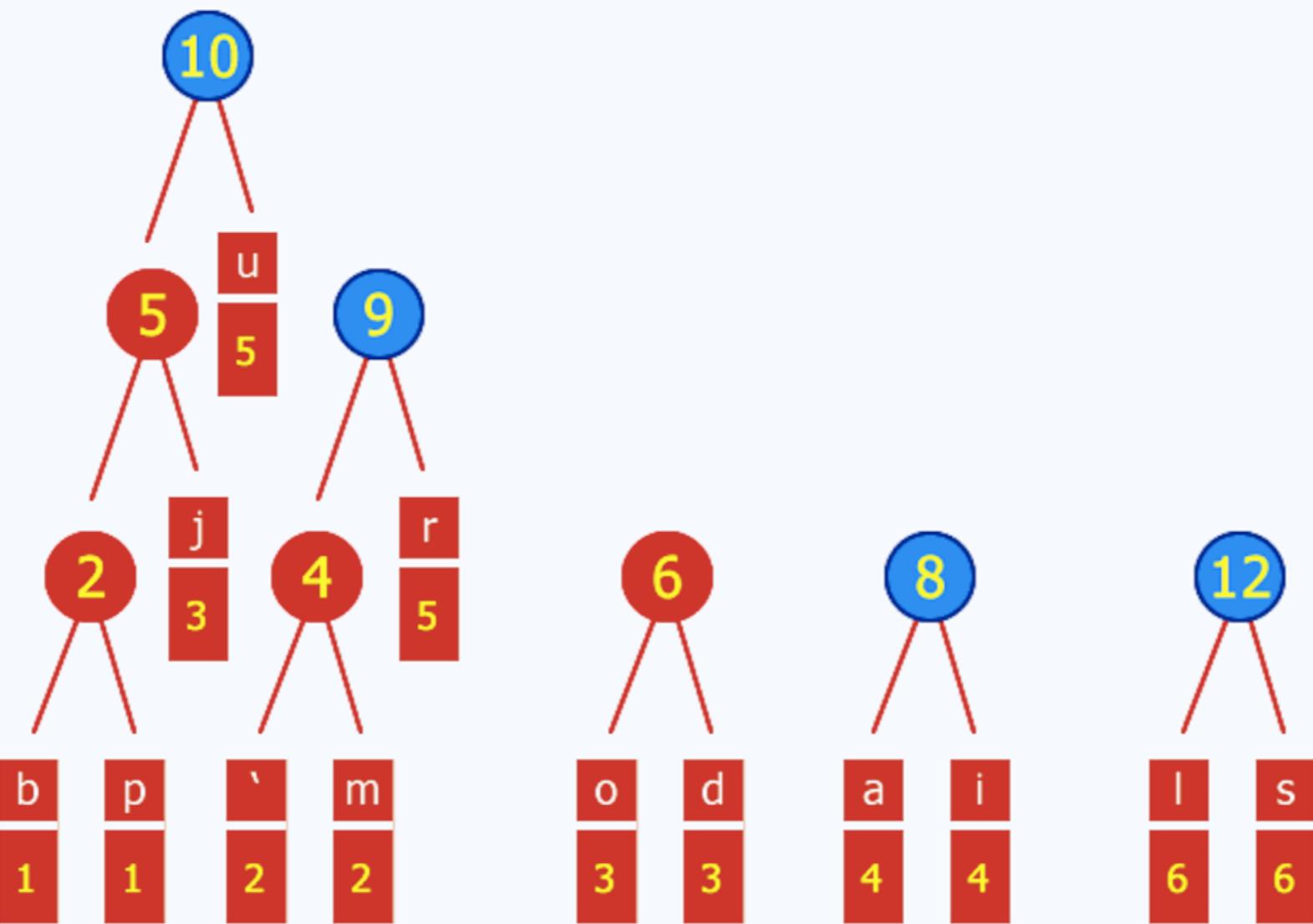


Árvore de Huffman



File :

e	
8	
	12



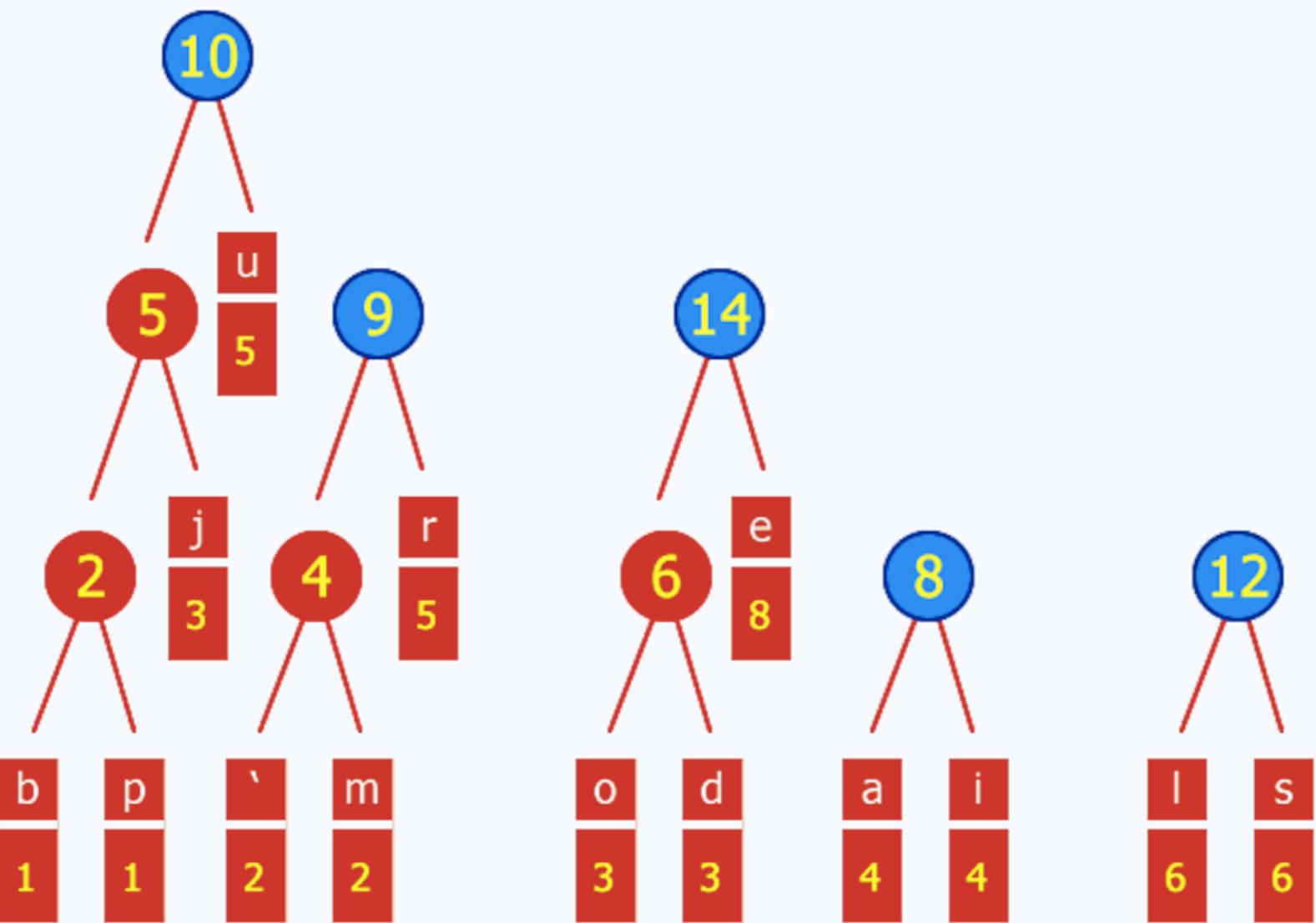


Árvore de Huffman



File :

12



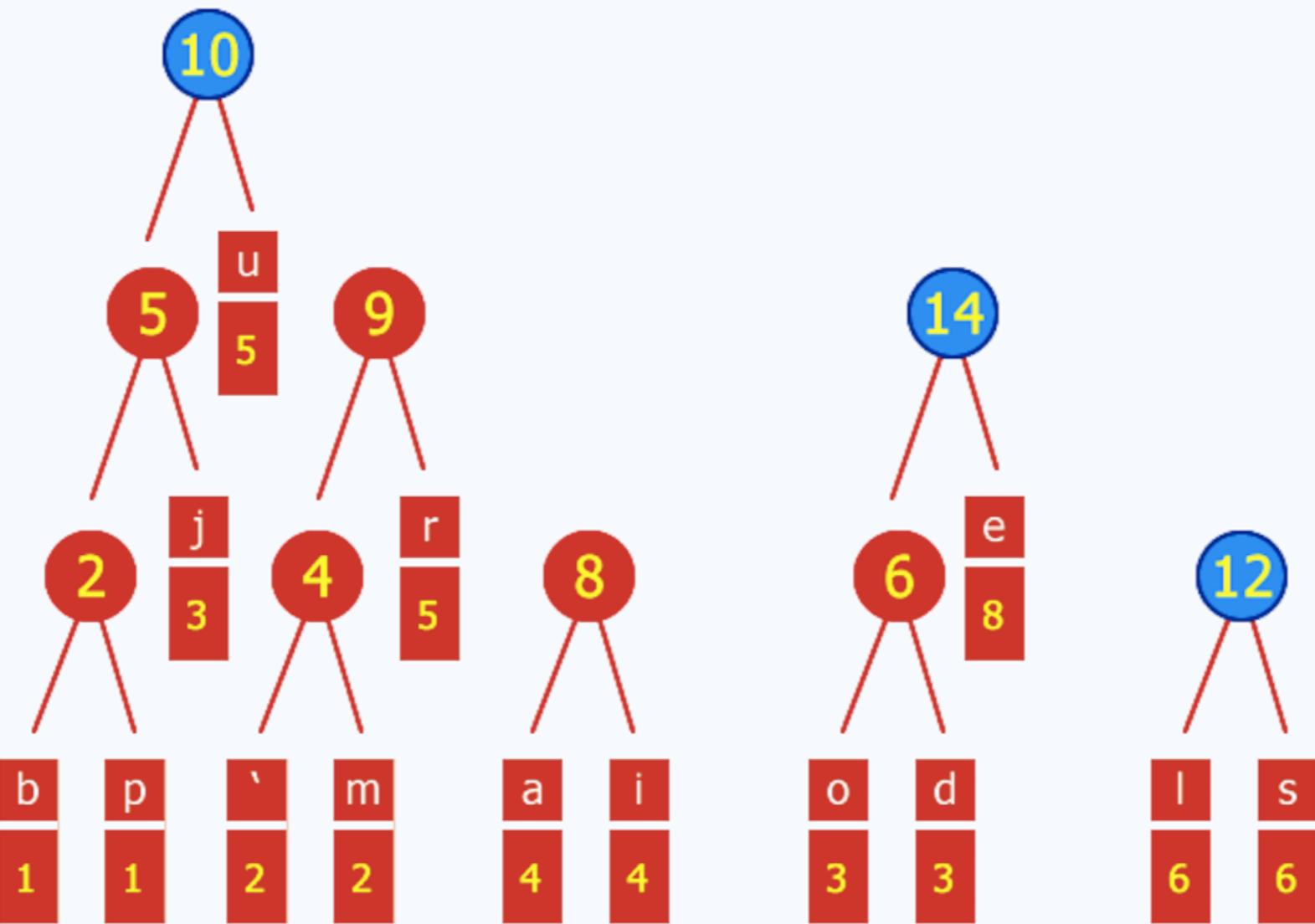


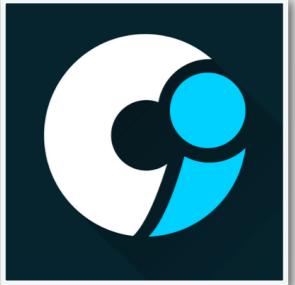
Árvore de Huffman



File :

12



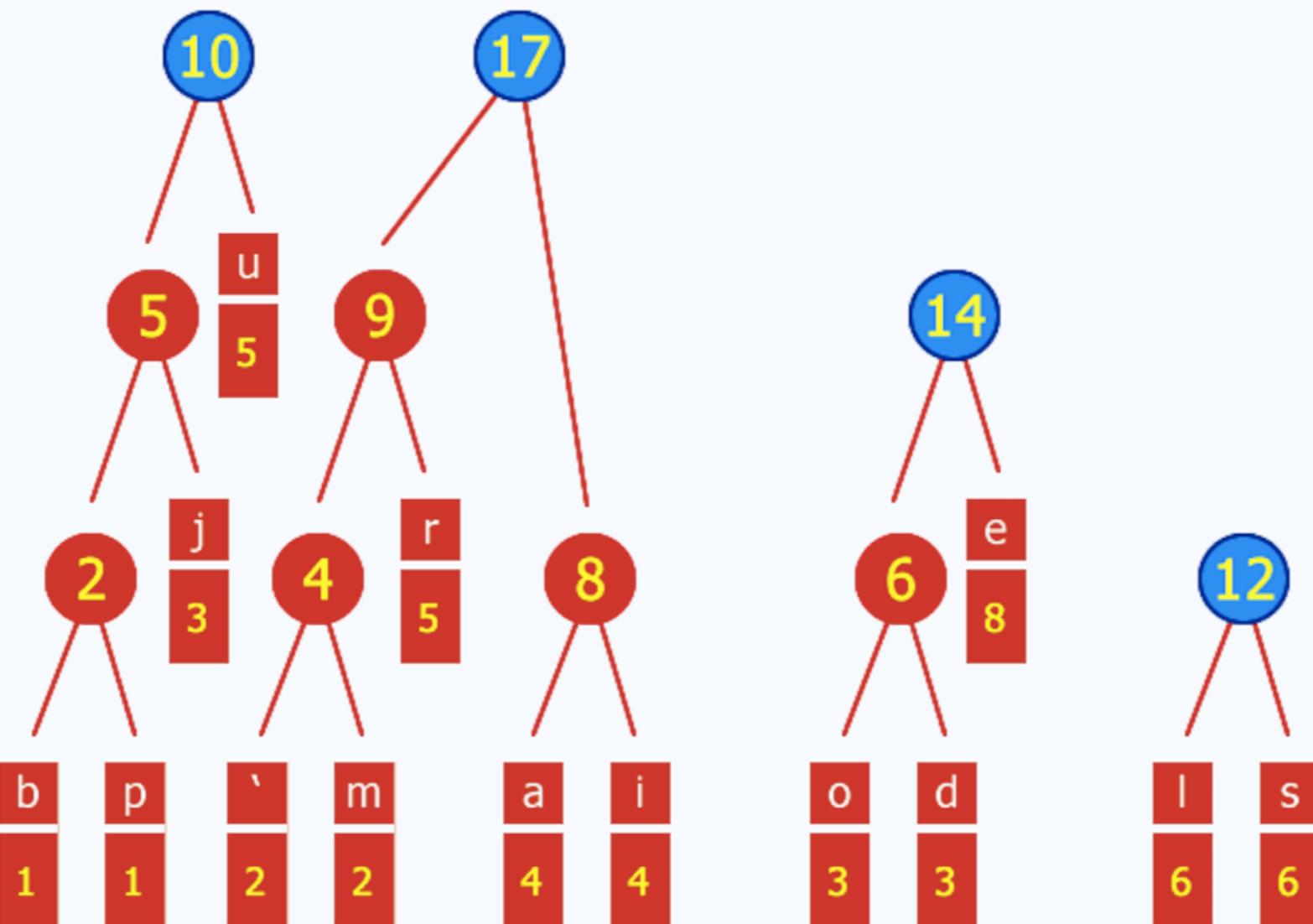


Árvore de Huffman



File :

12



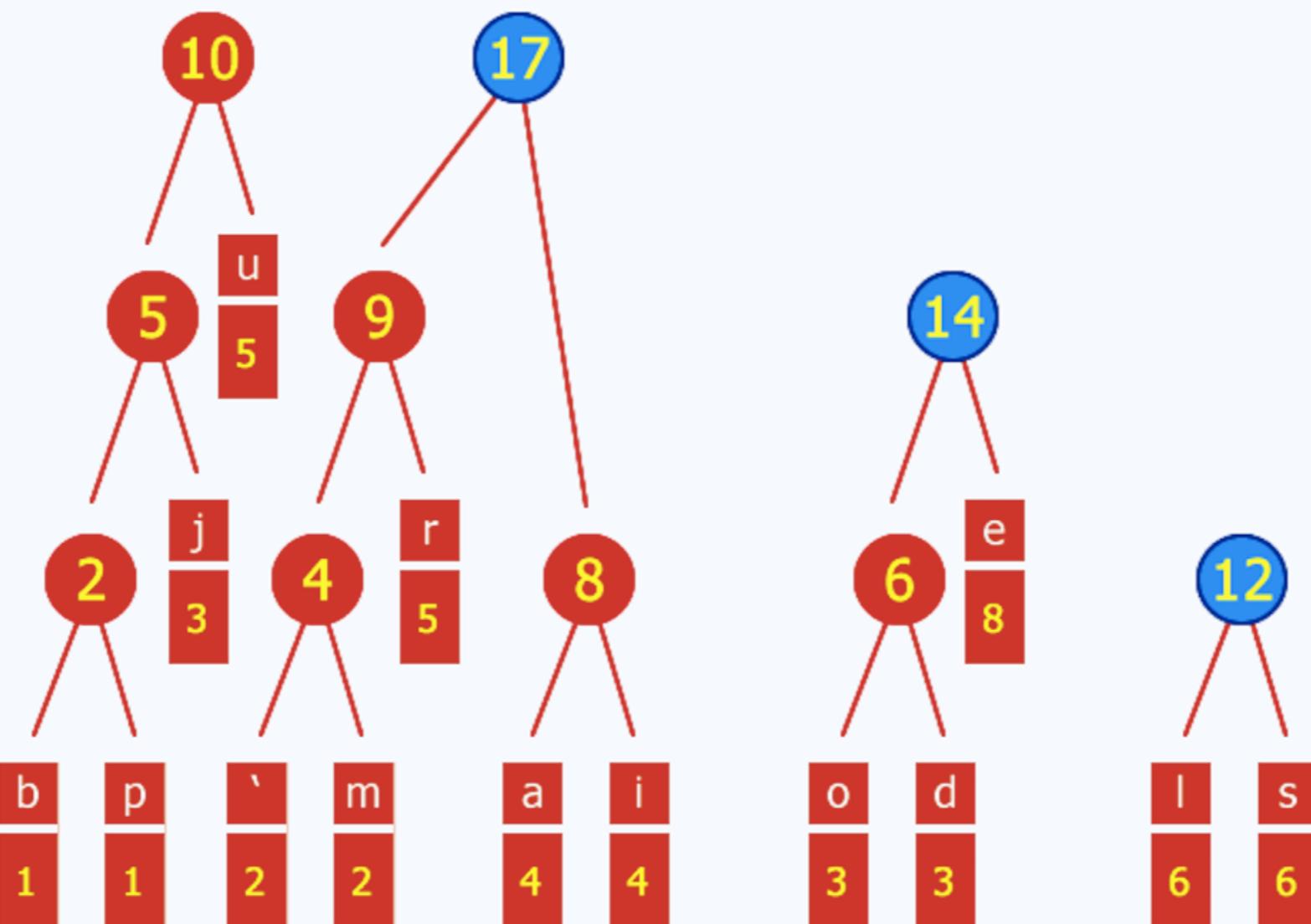


Árvore de Huffman



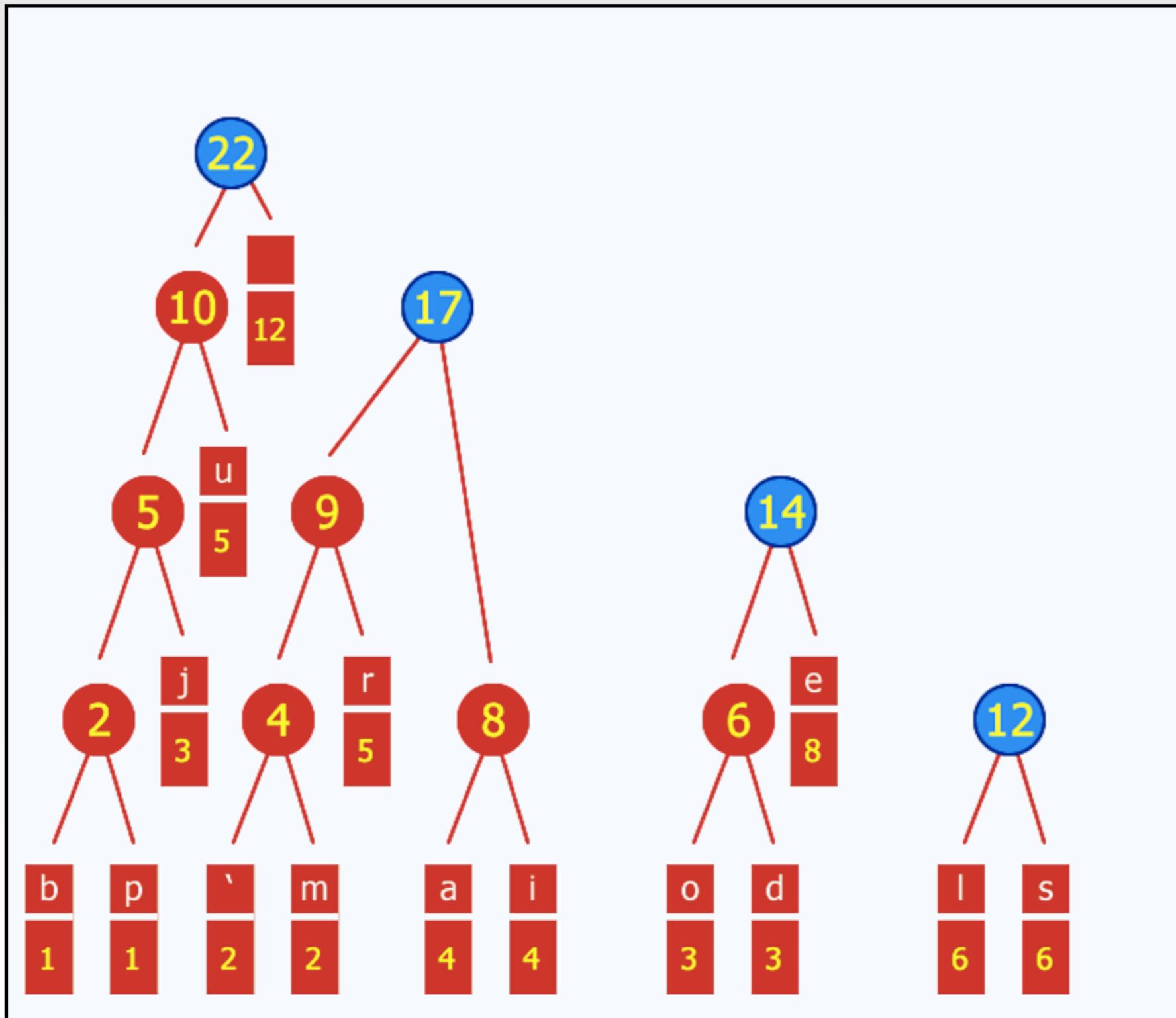
File :

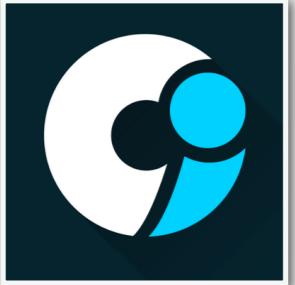
12



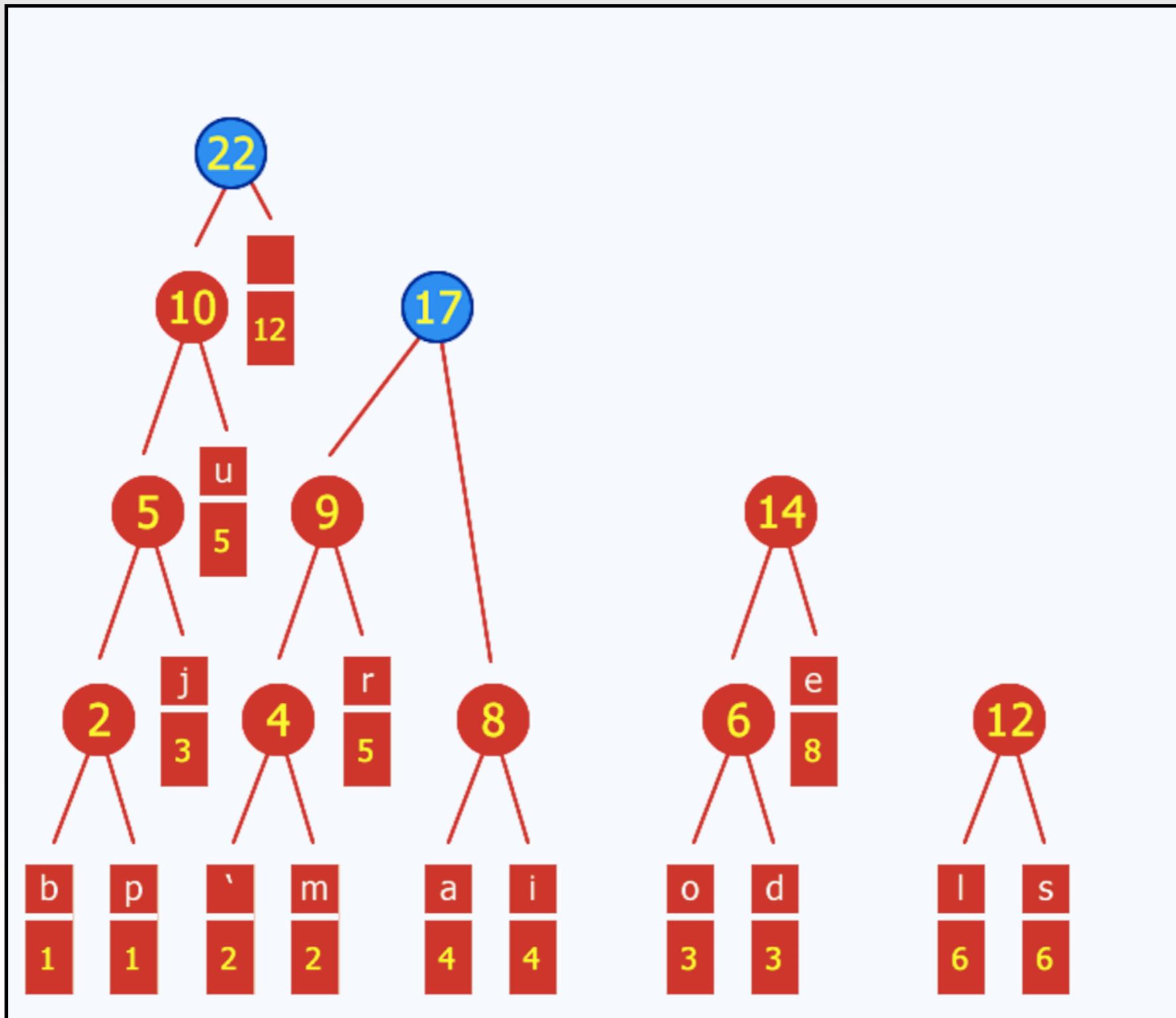


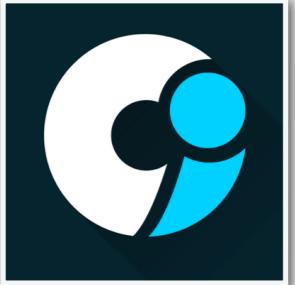
Árvore de Huffman



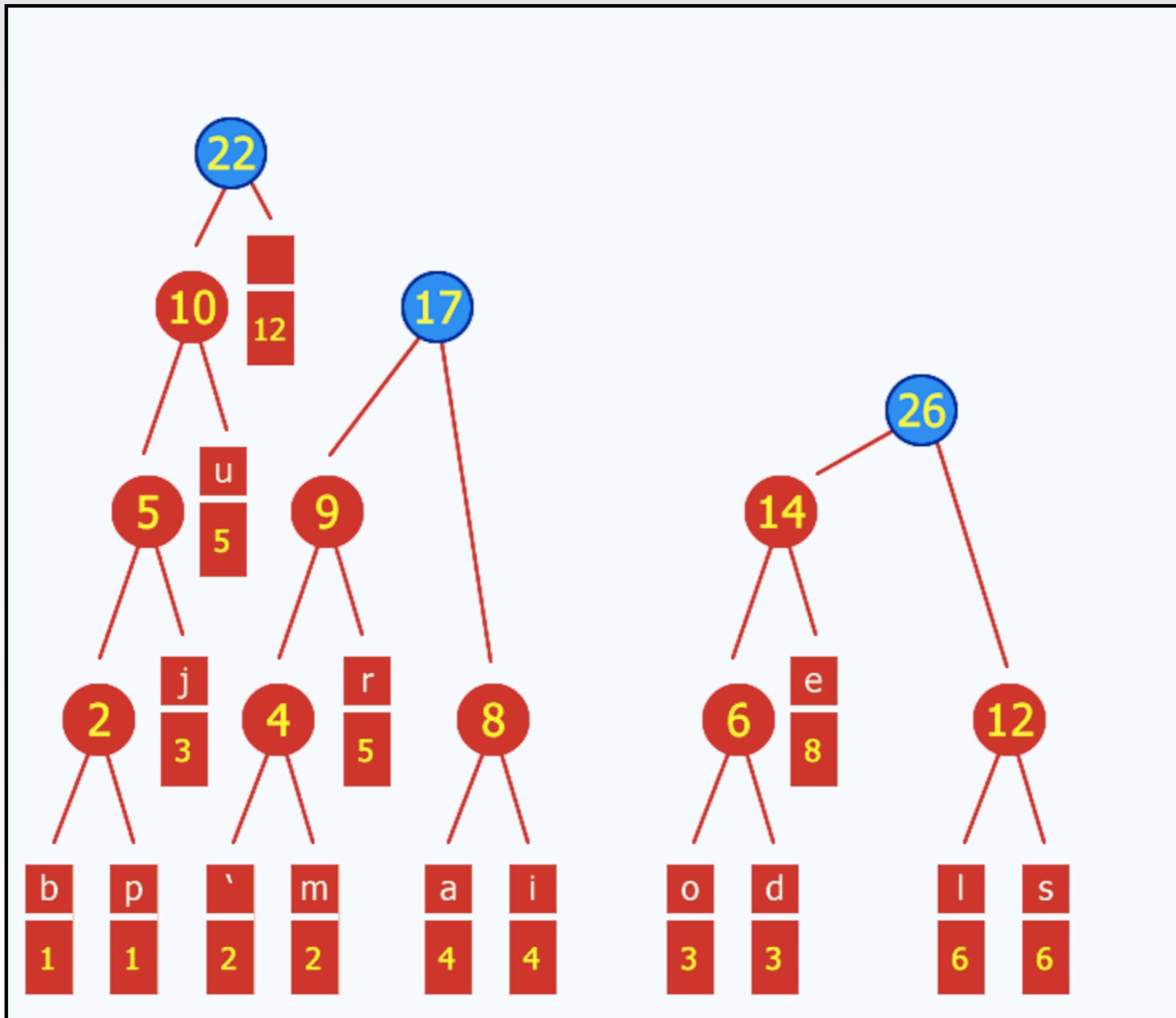


Árvore de Huffman



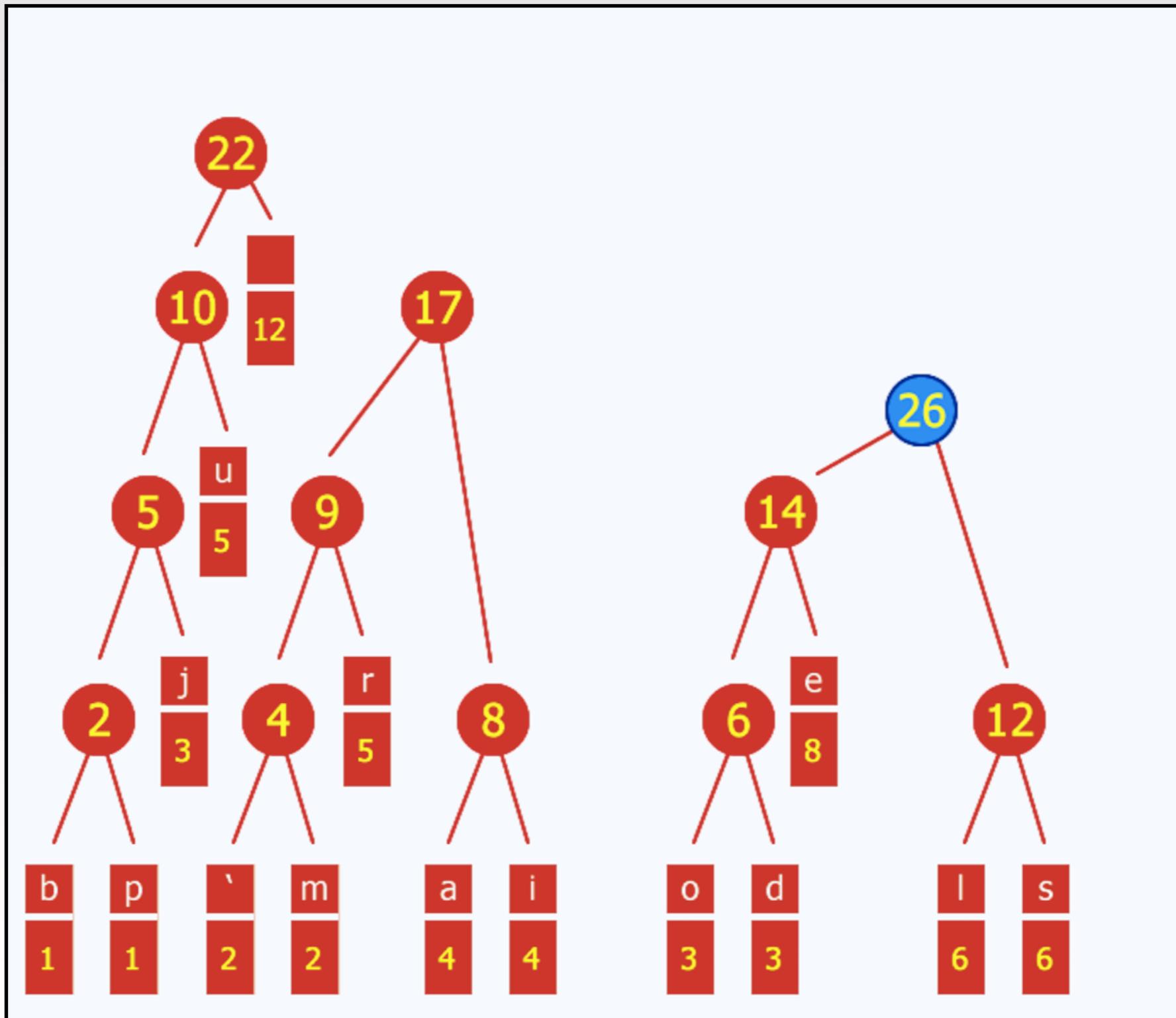


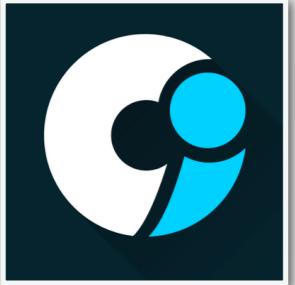
Árvore de Huffman



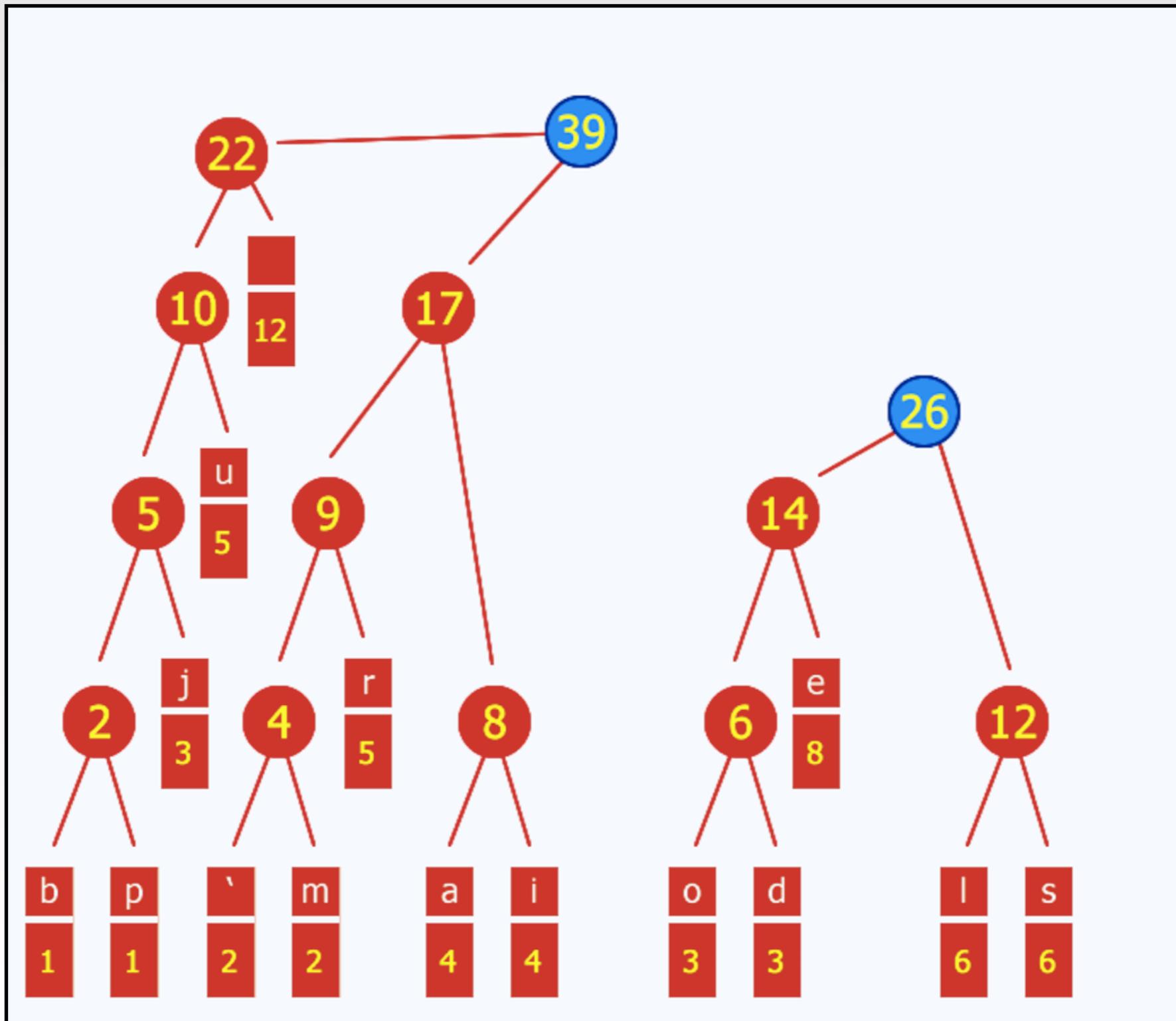


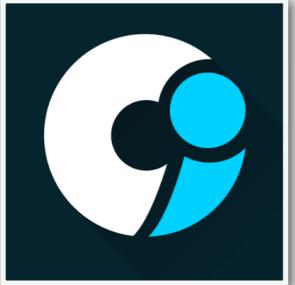
Árvore de Huffman



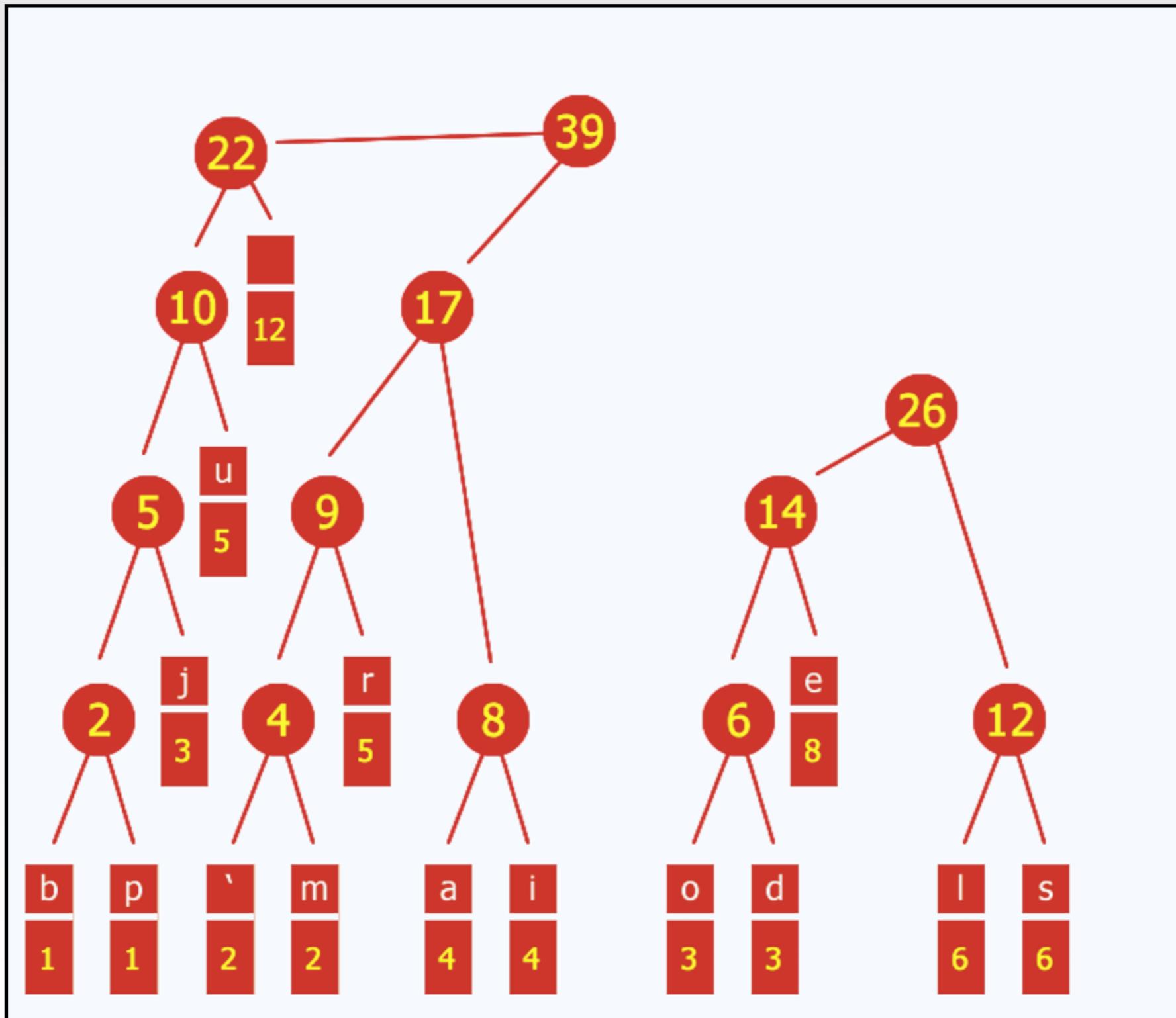


Árvore de Huffman



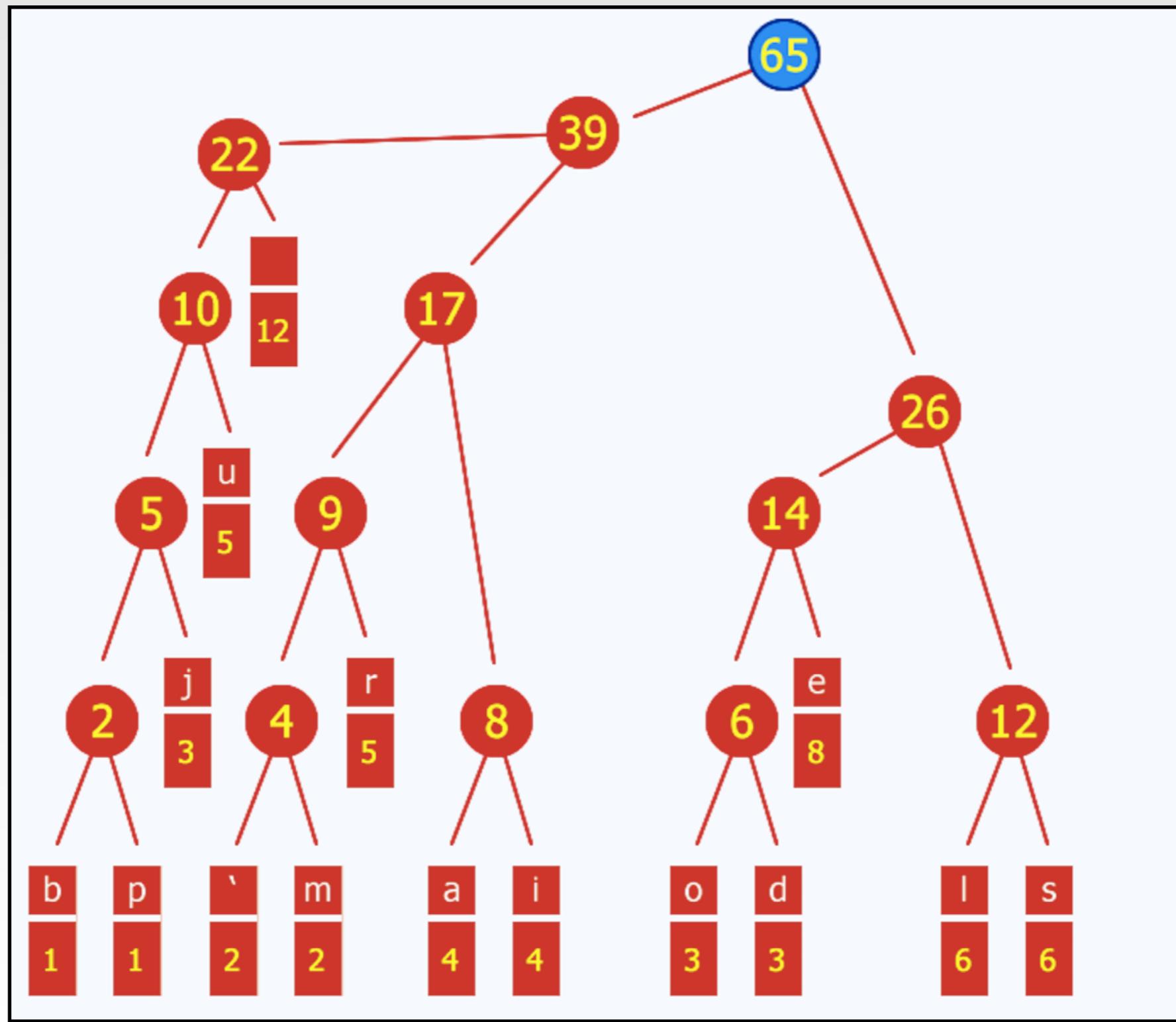


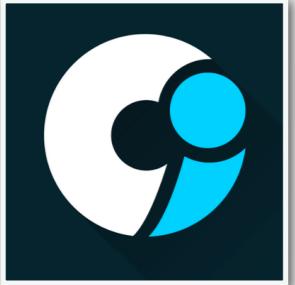
Árvore de Huffman



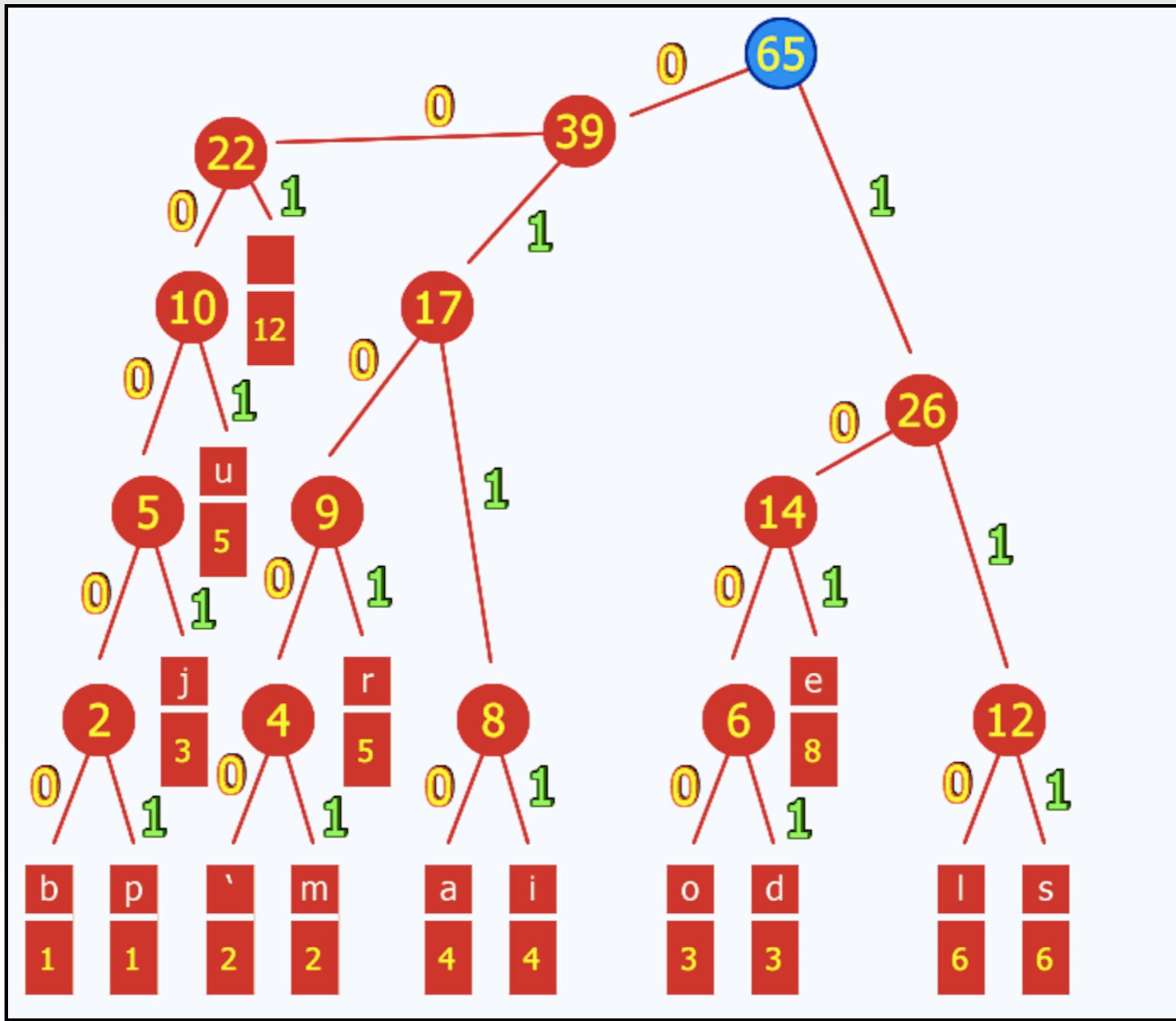


Árvore de Huffman





Árvore de Huffman





Algoritmo de Huffman



Input: a collection C of objects containing a character and its frequency.

Output: the root of a Huffman tree

Uses: priority queue Q

$n = |C|$

$Q = C$

for $i=1$ to $n-1$

$z = \text{new treenode}$

$\text{left}(z) = \text{ExtractMin}(Q)$

$\text{right}(z) = \text{ExtractMin}(Q)$

$\text{frequency}(z) = \text{frequency}(\text{left}(z)) + \text{frequency}(\text{right}(z))$

$\text{Insert}(Q, z)$

return $\text{ExtractMin}(Q)$

Running time: $(O(n \lg n))$



Referências

- ❖ Histórico
- ❖ Notas de Aula (Prof. Leonardo Vidal)