Data Compression II

Huffman Encoding Algorithm

Mark Matthews PhD

Variable-length codes

Use different number of bits to encode different chars.

Ex. Morse code: • • • – – – • • •

Issue. Ambiguity.

S O S ?

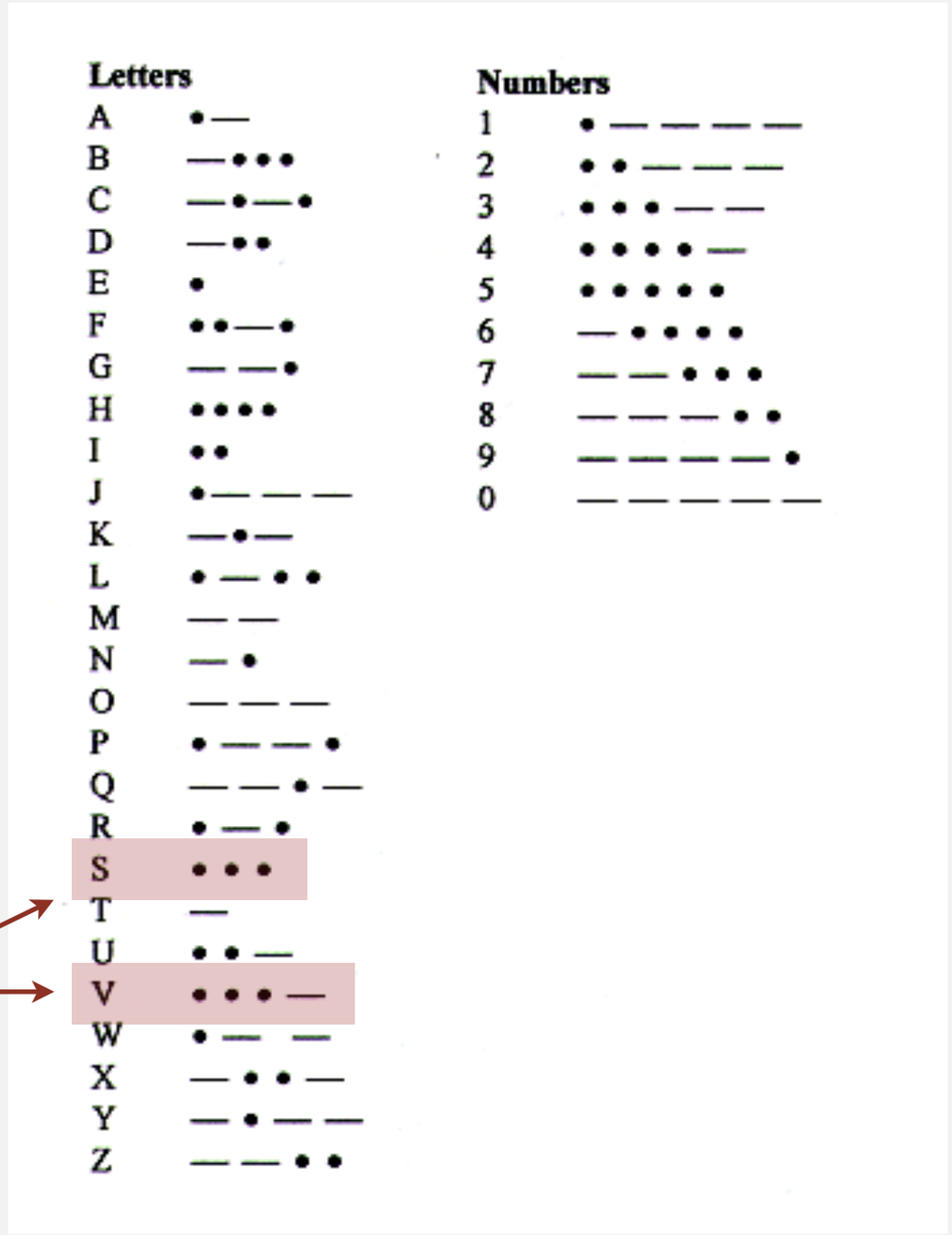
V 7 ?

I A M I E ?

E E W N I ?

In practice. Use a medium gap to separate codewords.

codeword for S is a prefix of codeword for V



Variable-length codes

Q. How do we avoid ambiguity?

A. Ensure that no codeword is a prefix of another.

Ex 1. Fixed-length code.

Ex 2. Append special stop char to each codeword.

Ex 3. General prefix-free code.

Codeword table

key	*value*
!	101
A	0
B	1111
C	110
D	100
R	1110

Compressed bitstring

01111111001100100011111100101 ←*30 bits*

A B RA CA DA B RA !

Codeword table

key	*value*
!	101
A	11
B	00
C	010
D	100
R	011

Compressed bitstring

11000111101011001100011111101 ←*29 bits*

A B R A C A D A B R A !

Prefix-free codes: trie representation

Q. How to represent the prefix-free code?

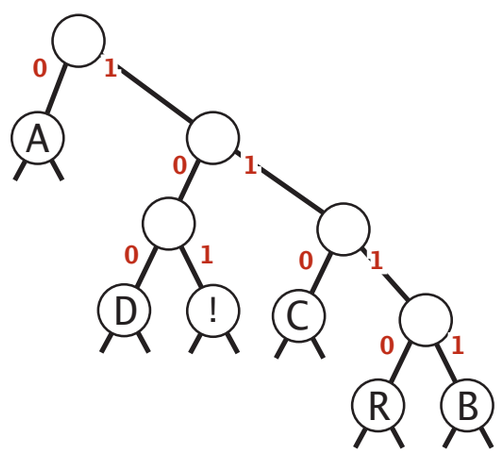
A. A binary trie!

- Chars in leaves.
- Codeword is path from root to leaf.

Codeword table

key	*value*
!	101
A	0
B	1111
C	110
D	100
R	1110

Trie representation



Compressed bitstring

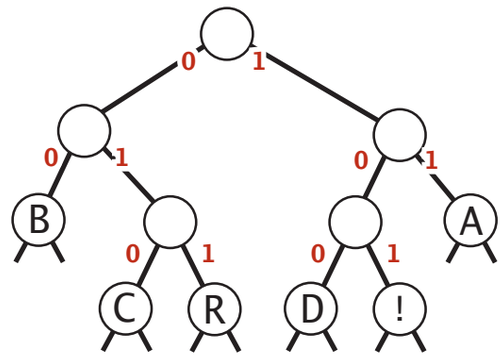
01111111001100100011111100101 ←*30 bits*

A B RA CA DA B RA !

Codeword table

key	*value*
!	101
A	11
B	00
C	010
D	100
R	011

Trie representation



Compressed bitstring

11000111101011001100011111101 ←*29 bits*

A B R A C A D A B R A !

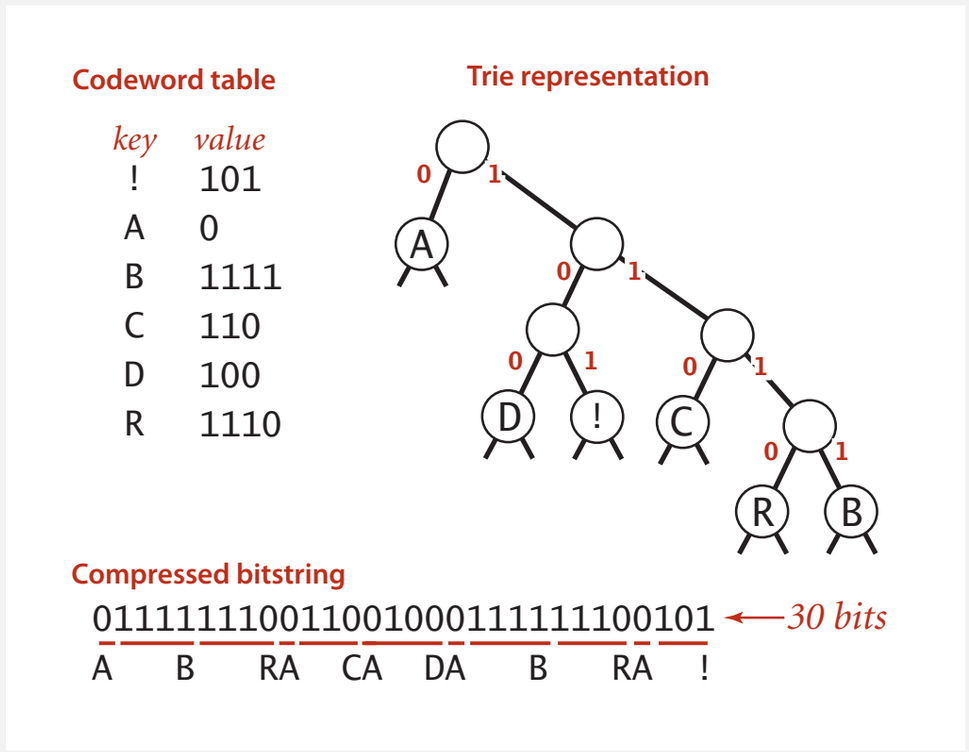
Prefix-free codes: compression and expansion

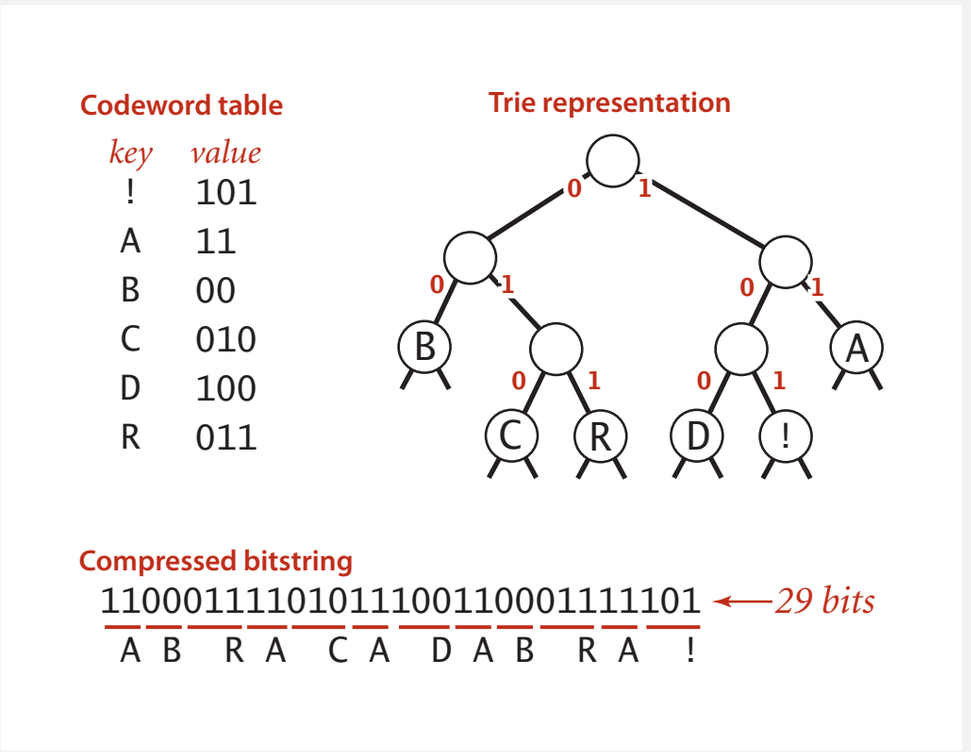
Compression.

- Method 1: start at leaf; follow path up to the root; print bits in reverse.
- Method 2: create ST of key-value pairs.

Expansion.

- Start at root.
- Go left if bit is 0; go right if 1.
- If leaf node, print char and return to root.





Huffman coding overview

Dynamic model. Use a custom prefix-free code for each message.

Compression.

- Read message.
- Built best prefix-free code for message. How?
- Write prefix-free code (as a trie) to file.
- Compress message using prefix-free code.

Expansion.

- Read prefix-free code (as a trie) from file.
- Read compressed message and expand using trie.

Huffman trie node data type

```
private static class Node implements Comparable<Node>
{
   private final char ch;   // used only for leaf nodes
   private final int freq;  // used only for compress
   private final Node left, right;

   public Node(char ch, int freq, Node left, Node right)
   {
      this.ch    = ch;
      this.freq  = freq;
      this.left  = left;
      this.right = right;
   }

   public boolean isLeaf()
   {  return left == null && right == null; }

   public int compareTo(Node that)
   {  return this.freq - that.freq;  }
}
```

initializing constructor

is Node a leaf?

compare Nodes by frequency

(stay tuned)

Prefix-free codes: expansion

```
public void expand()
{
   Node root = readTrie();
   int N = BinaryStdIn.readInt();

   for (int i = 0; i < N; i++)
   {
      Node x = root;
      while (!x.isLeaf())
      {
         if (!BinaryStdIn.readBoolean())
            x = x.left;
         else
            x = x.right;
      }
      BinaryStdOut.write(x.ch, 8);
   }
   BinaryStdOut.close();
}
```

read in encoding trie

read in number of chars

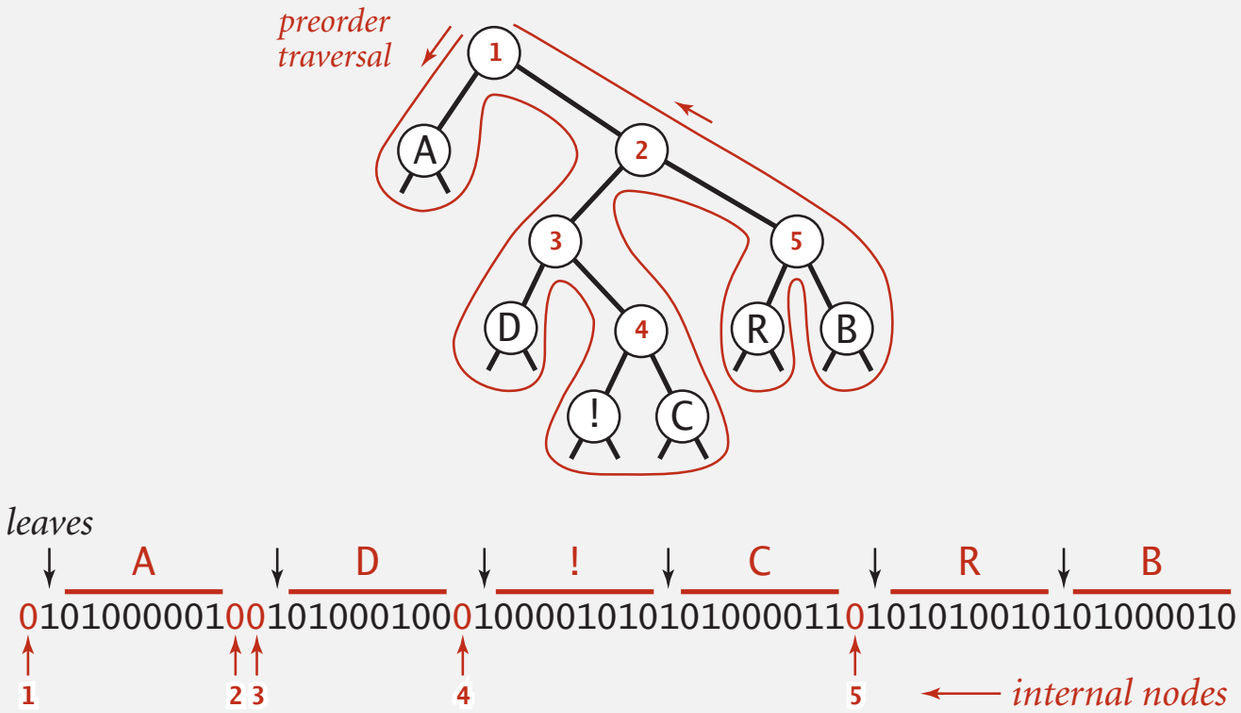
expand codeword for i^{th} char

Running tim

Prefix-free codes: how to transmit

Q. How to write the trie?

A. Write preorder traversal of trie; mark leaf and internal nodes with a bit.



Using preorder traversal to encode a trie as a bitstream

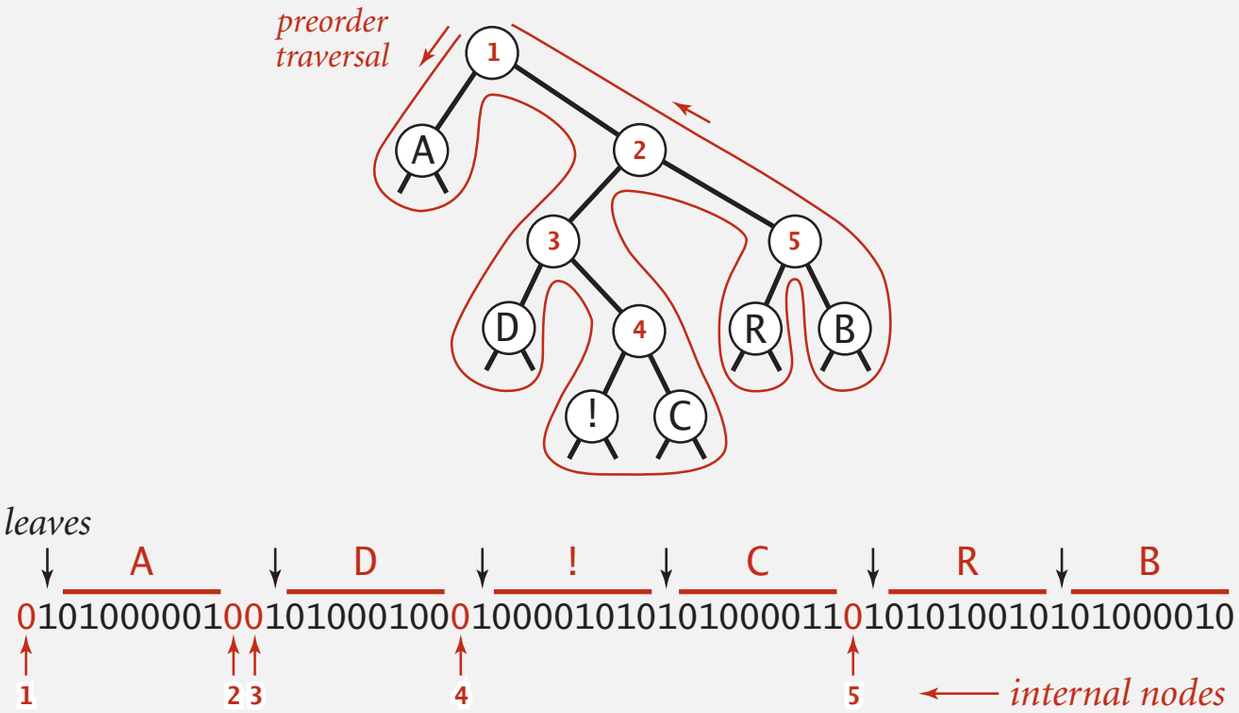
```
private static void writeTrie(Node x)
{
   if (x.isLeaf())
   {
      BinaryStdOut.write(true);
      BinaryStdOut.write(x.ch, 8);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(x.left);
   writeTrie(x.right);
}
```

Note. If message is long, overhead of transmitting trie is small.

Prefix-free codes: how to transmit

Q. How to read in the trie?

A. Reconstruct from preorder traversal of trie.



Using preorder traversal to encode a trie as a bitstream

```
private static Node readTrie()
{
   if (BinaryStdIn.readBoolean())
   {
      char c = BinaryStdIn.readChar(8);
      return new Node(c, 0, null, null);
   }
   Node x = readTrie();
   Node y = readTrie();
   return new Node('\0', 0, x, y);
}
```

arbitrary value
(value not used with internal nodes)

Shannon-Fano codes

Q. How to find best prefix-free code?

Shannon-Fano algorithm:

- Partition symbols S into two subsets S_0 and S_1 of (roughly) equal freq.
- Codewords for symbols in S_0 start with 0; for symbols in S_1 start with 1.
- Recur in S_0 and S_1.

char	freq	encoding
A	5	0...
C	1	0...

S_0 = codewords starting with 0

char	freq	encoding
B	2	1...
D	1	1...
R	2	1...
!	1	1...

S_1 = codewords starting with 1

Problem 1. How to divide up symbols?

Problem 2. Not optimal!

Huffman algorithm demo

- Count frequency for each character in input.

char	freq	encoding
A		
B		
C		
D		
R		
!		

input

A B R A C A D A B R A !

Huffman coding demo

- Count frequency for each character in input.

char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	

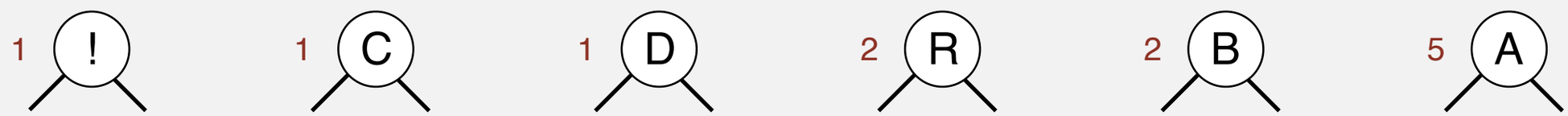
input

A B R A C A D A B R A !

Huffman coding demo

- Start with one node corresponding to each character with weight equal to frequency.

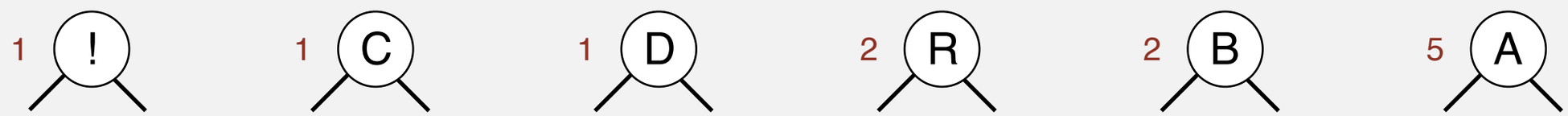
char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

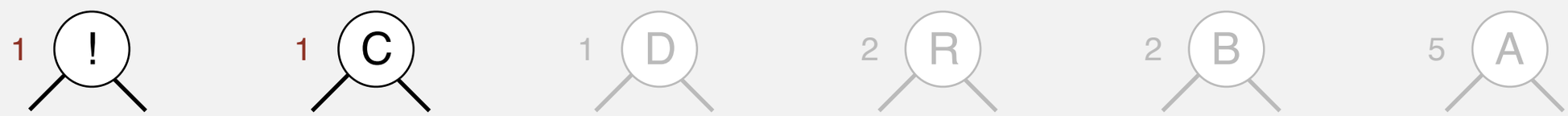
char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

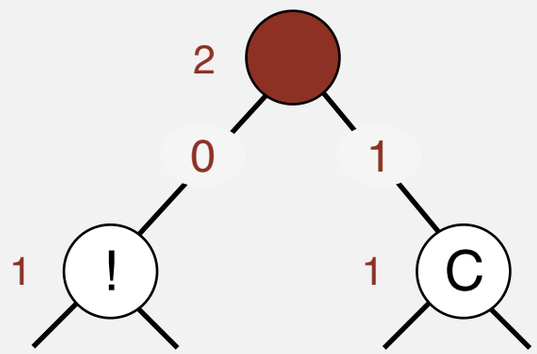
char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	

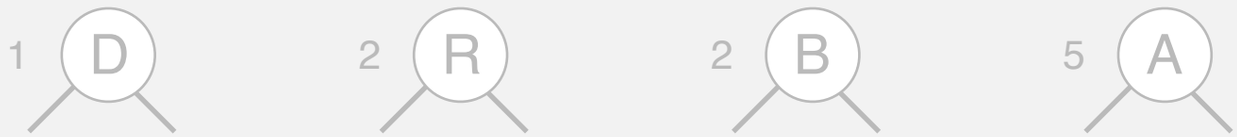


Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0

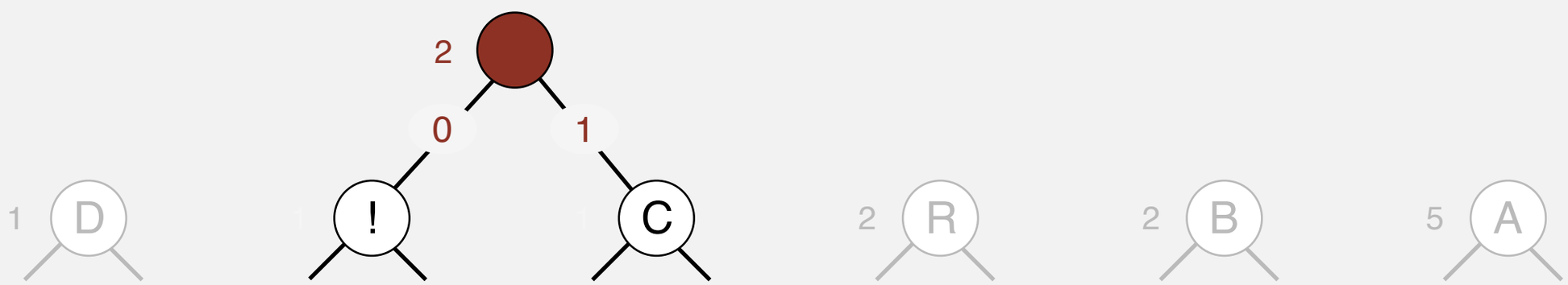




Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

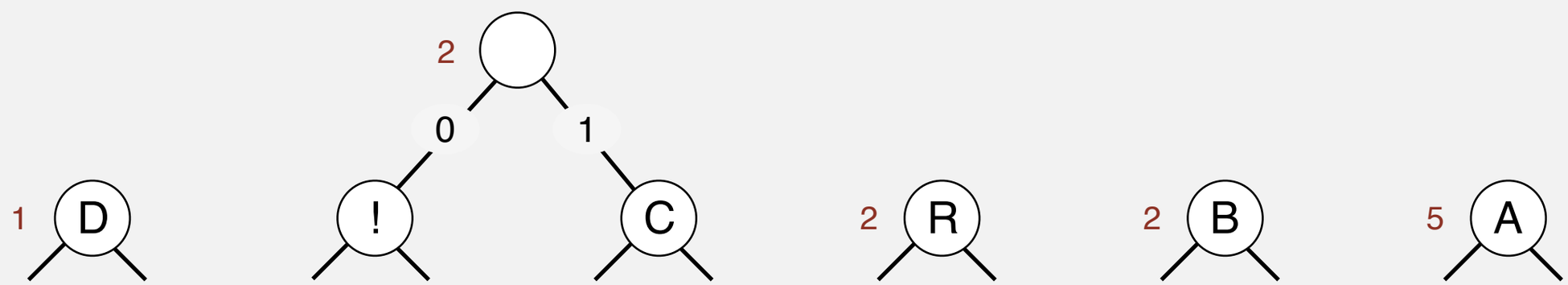
char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

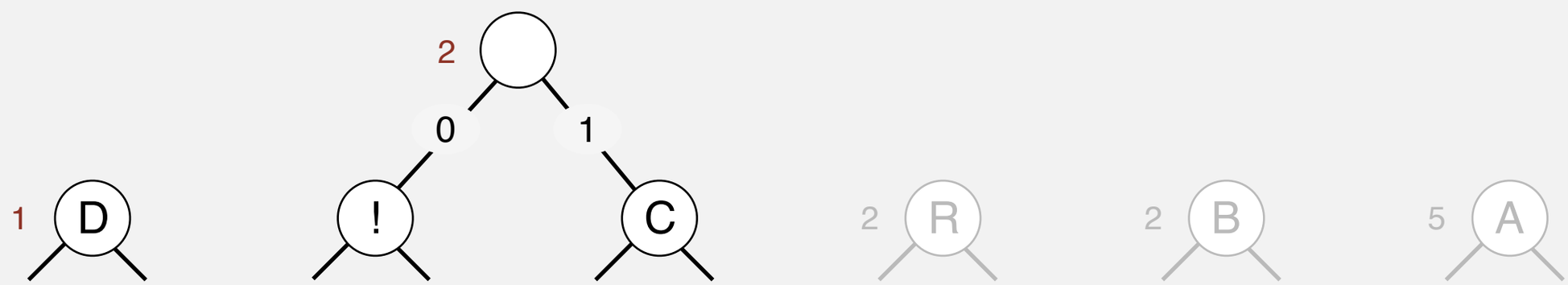
char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

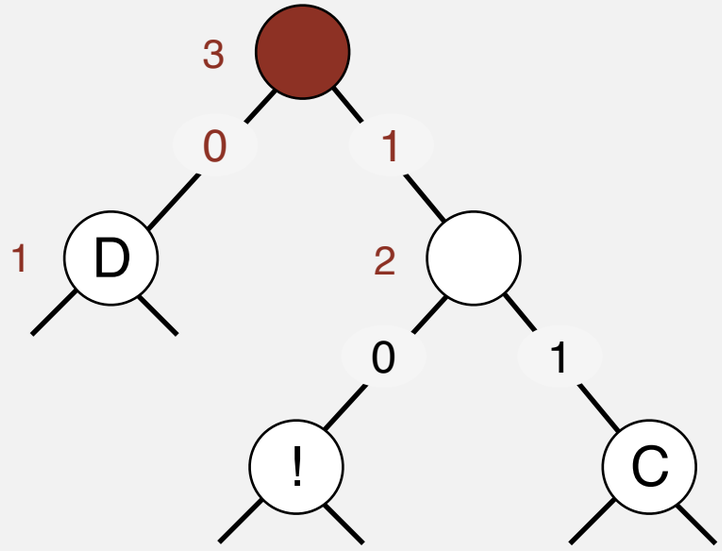
char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

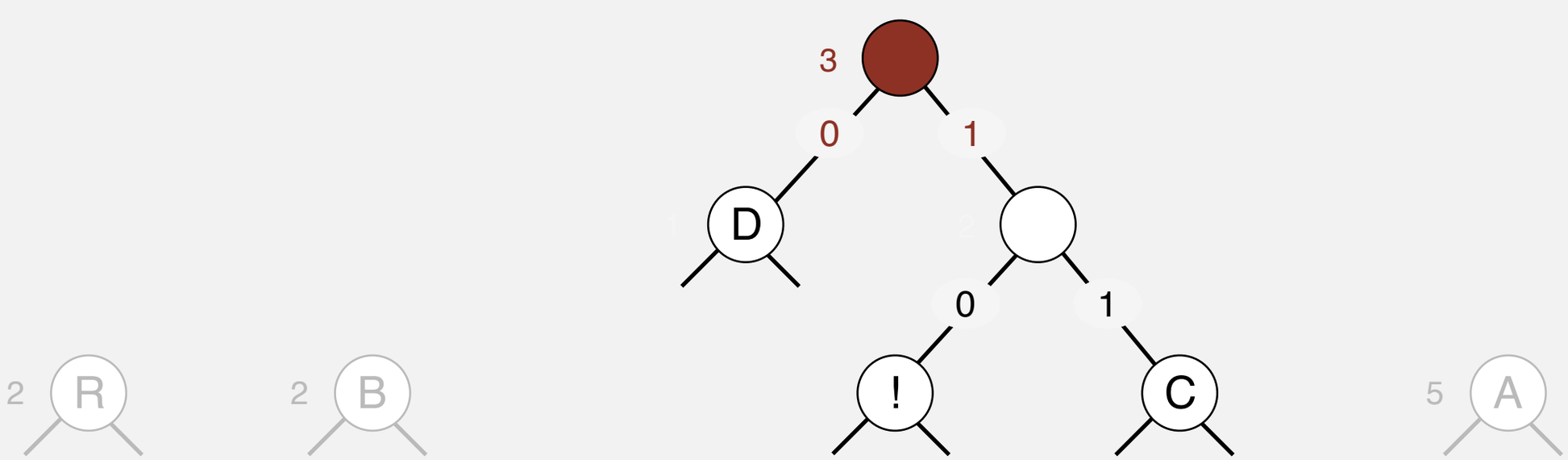
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

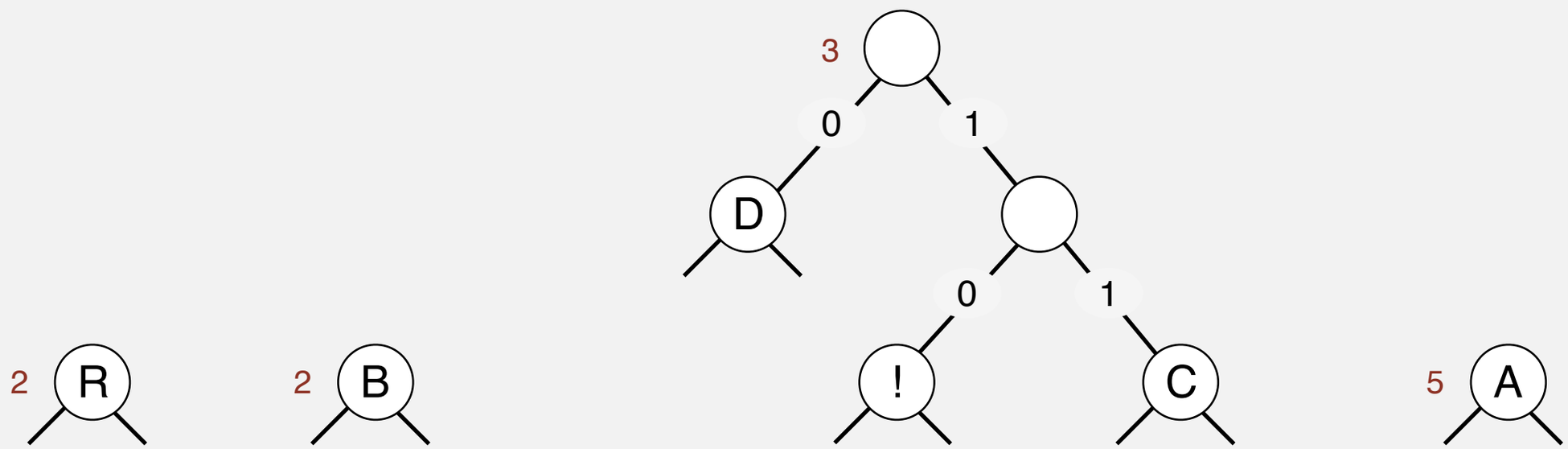
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

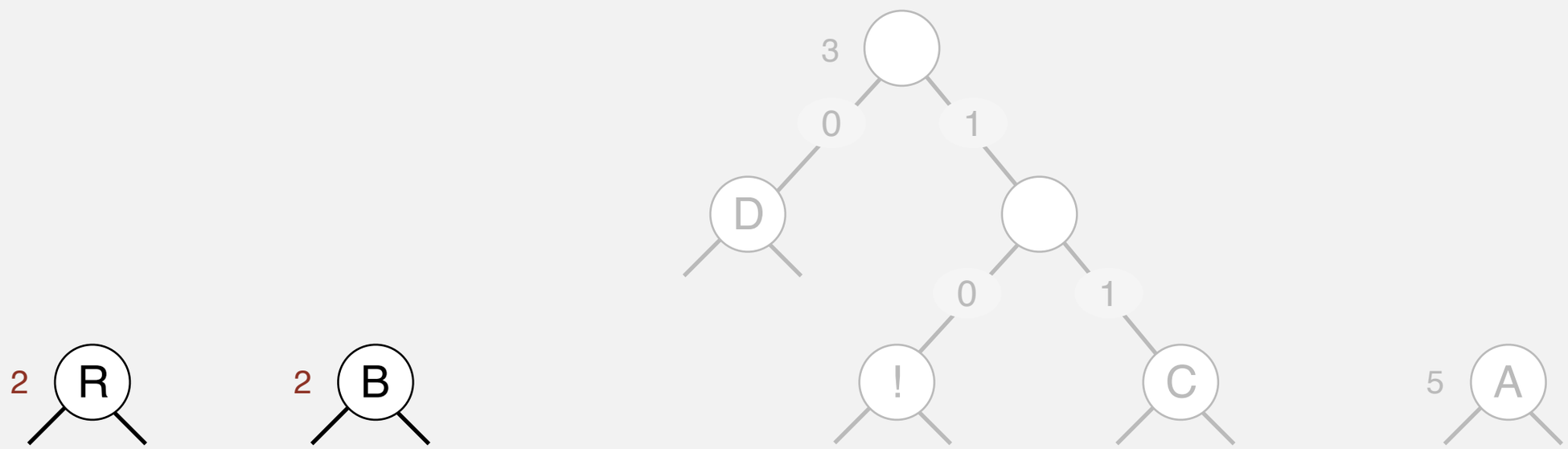
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

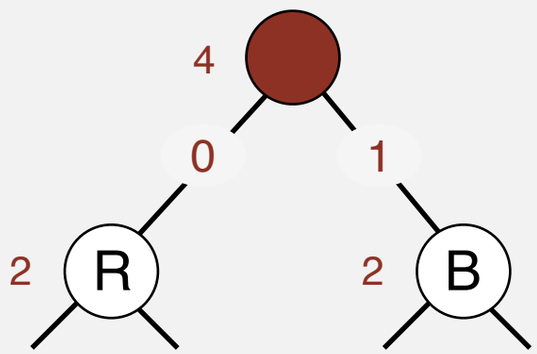
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0

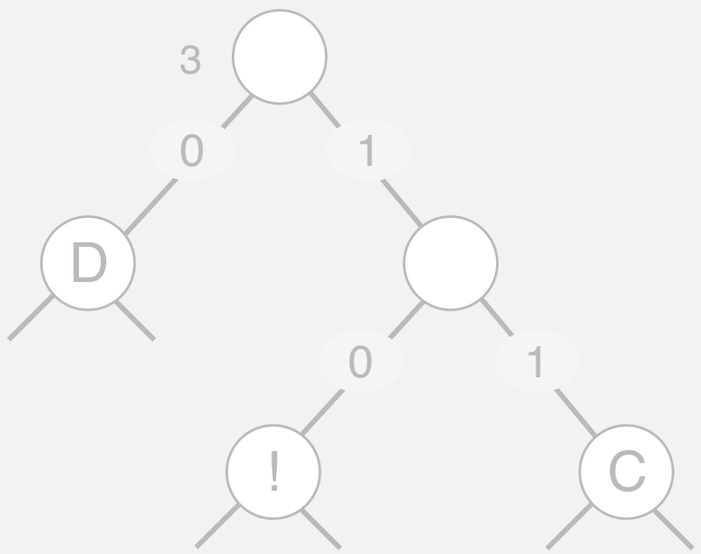


Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

char	freq	encoding
A	5	
B	2	1
C	1	1 1
D	1	0
R	2	0
!	1	1 0

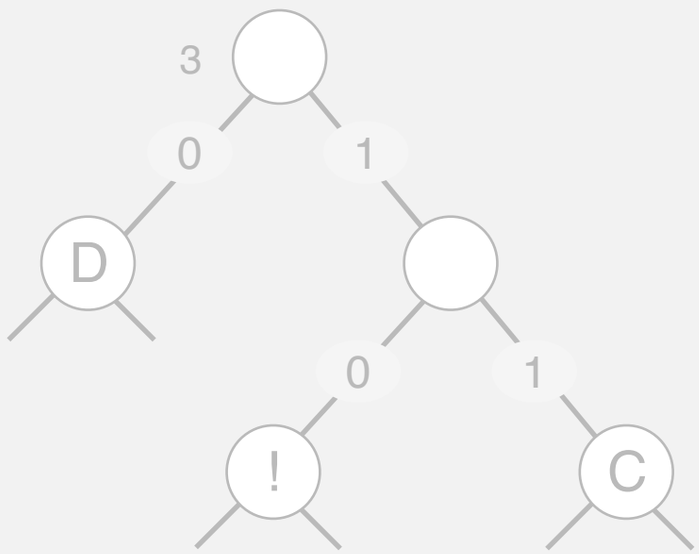


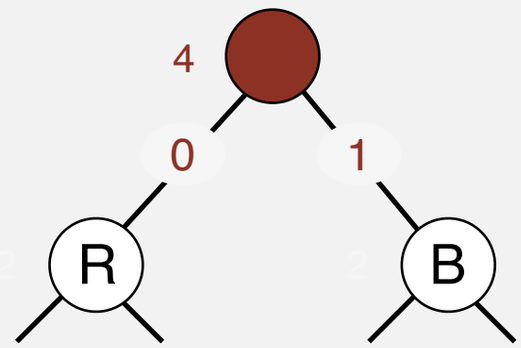


Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

char	freq	encoding
A	5	
B	2	1
C	1	1 1
D	1	0
R	2	0
!	1	1 0

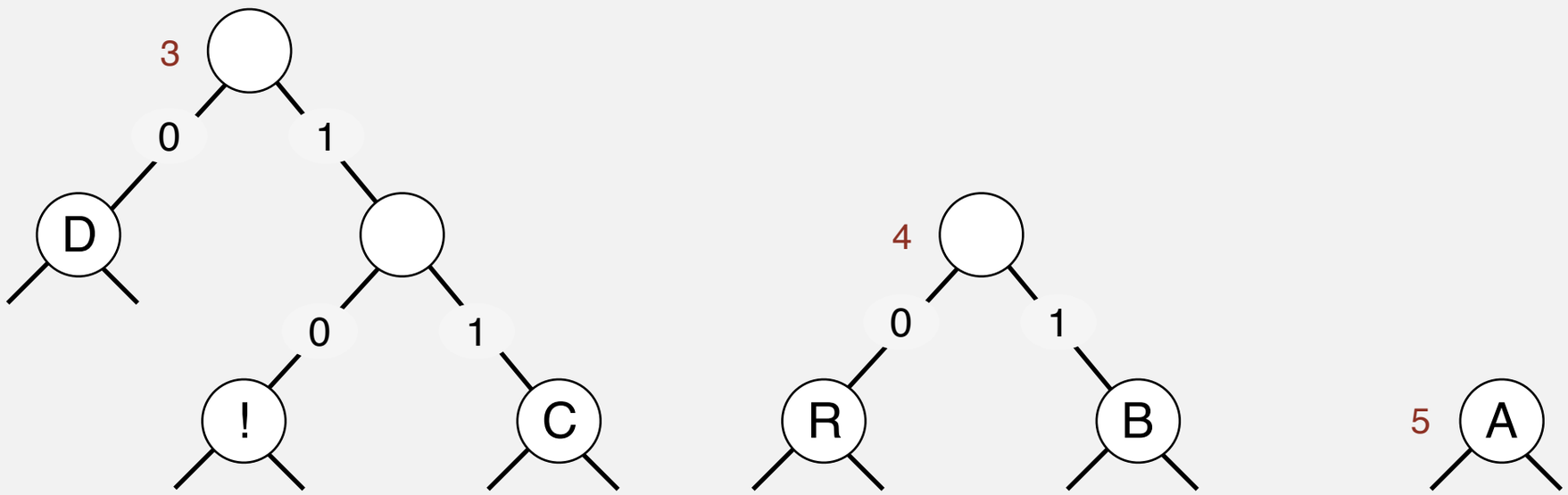




Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

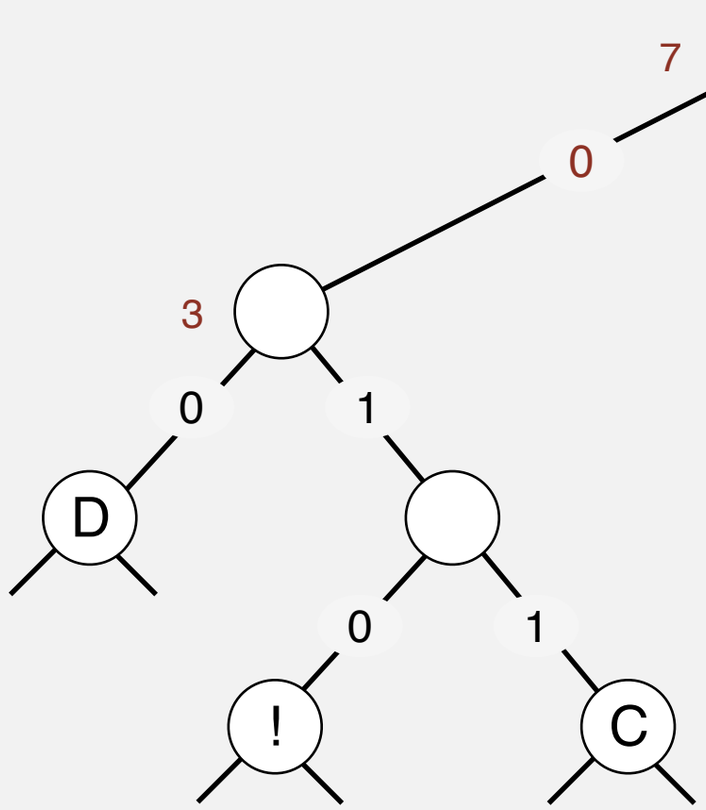
char	freq	encoding
A	5	
B	2	1
C	1	1 1
D	1	0
R	2	0
!	1	1 0

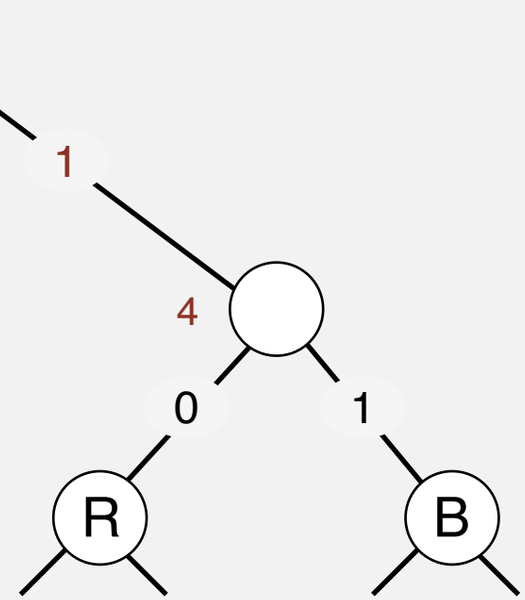


Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

char	freq	encoding
A	5	
B	2	1 1
C	1	0 1 1
D	1	0 0
R	2	1 0
!	1	0 1 0

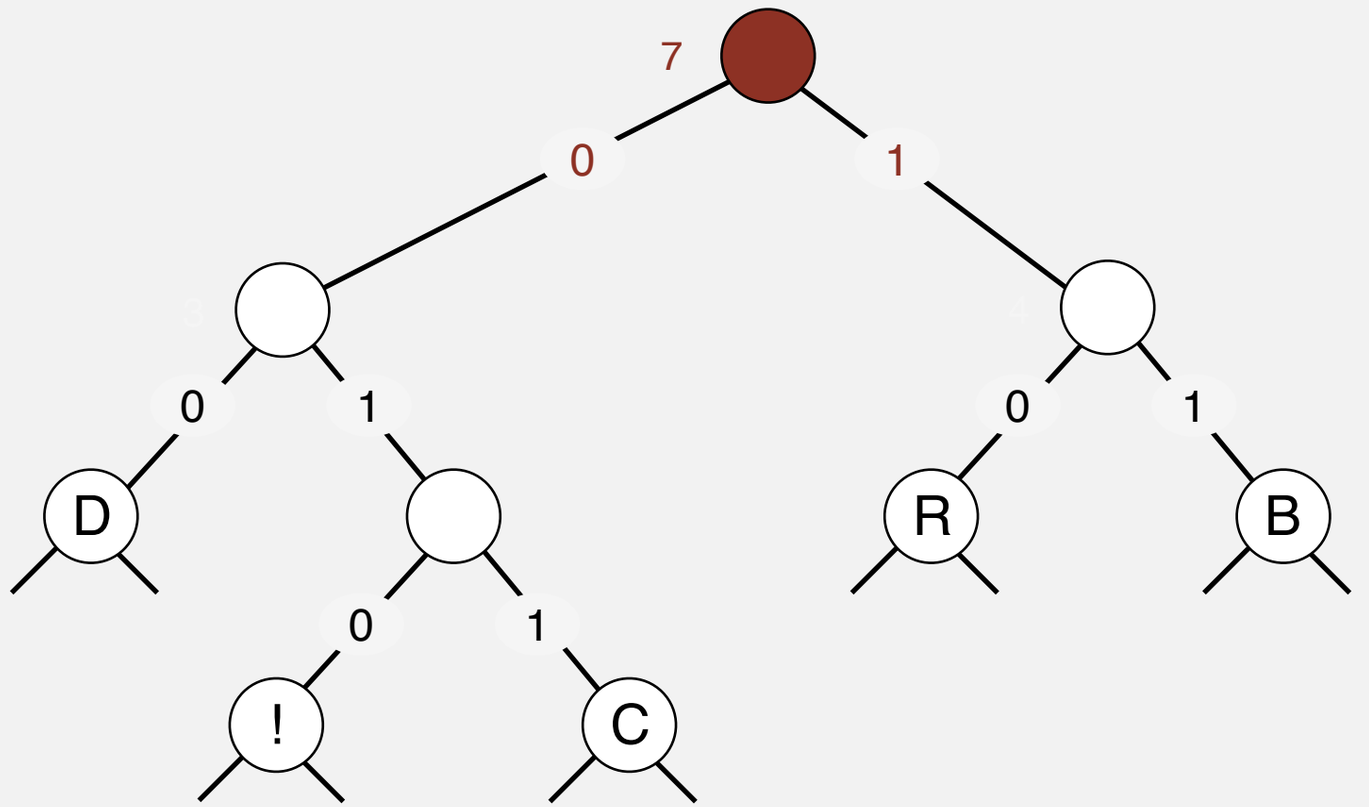




Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

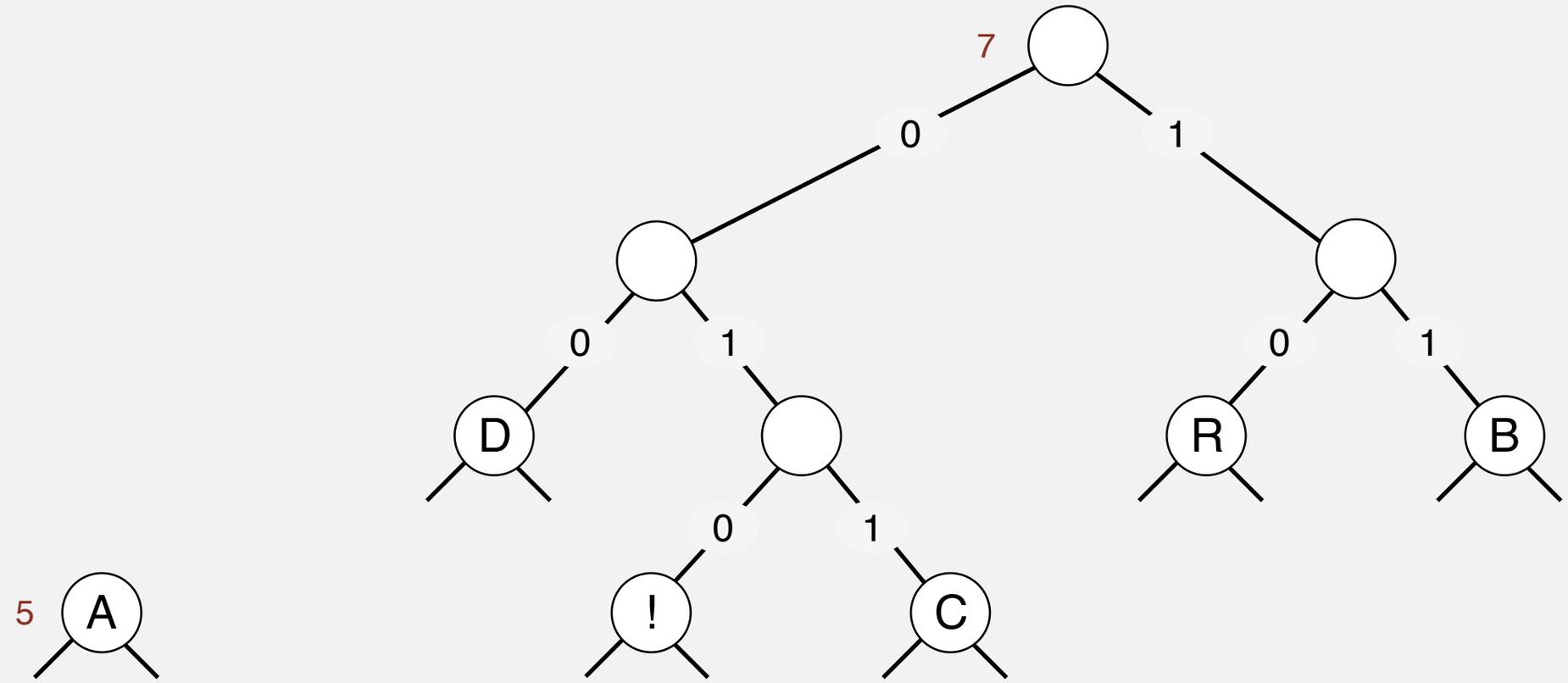
char	freq	encoding
A	5	
B	2	1 1
C	1	0 1 1
D	1	0 0
R	2	1 0
!	1	0 1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

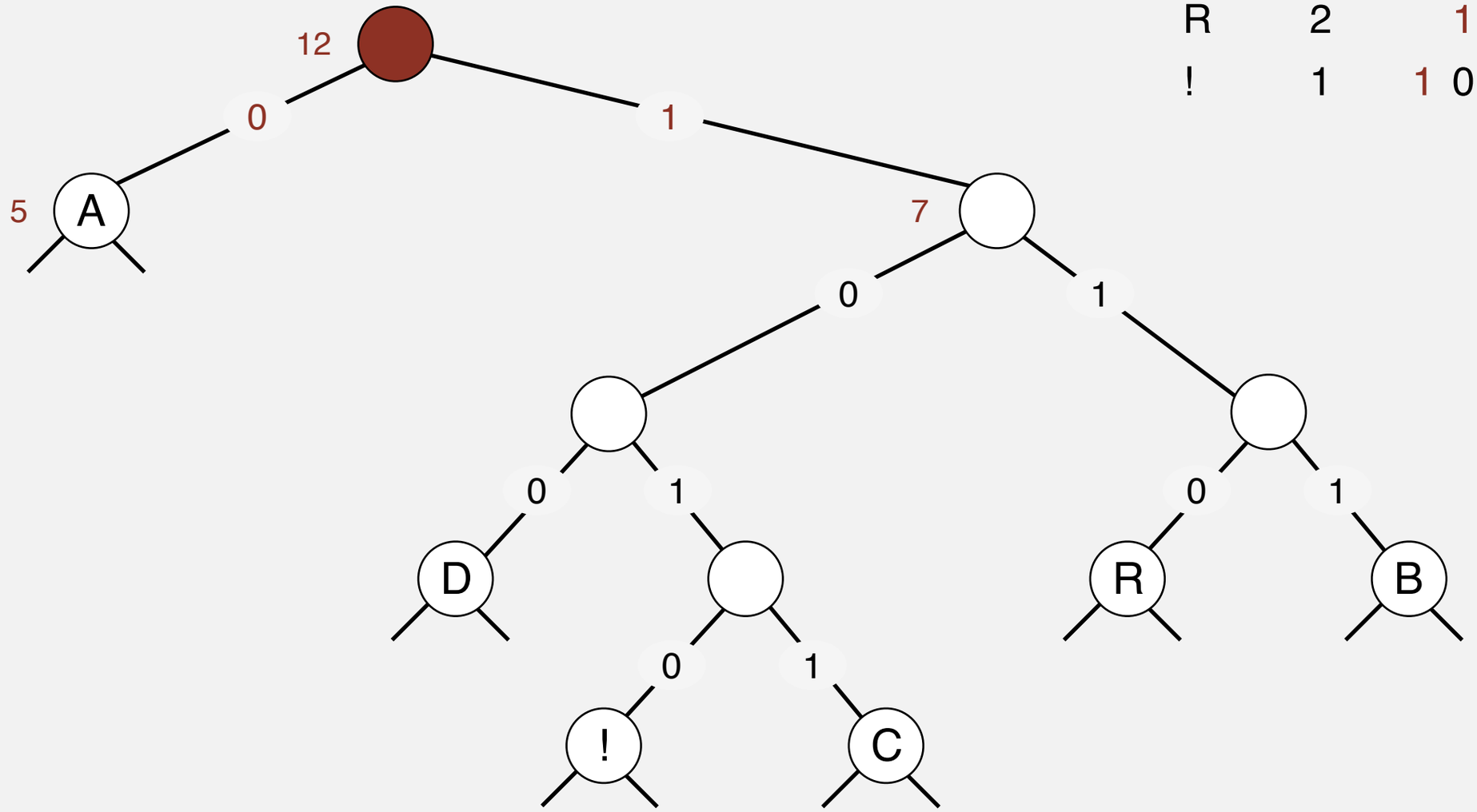
char	freq	encoding
A	5	
B	2	1 1
C	1	0 1 1
D	1	0 0
R	2	1 0
!	1	0 1 0



Huffman coding demo

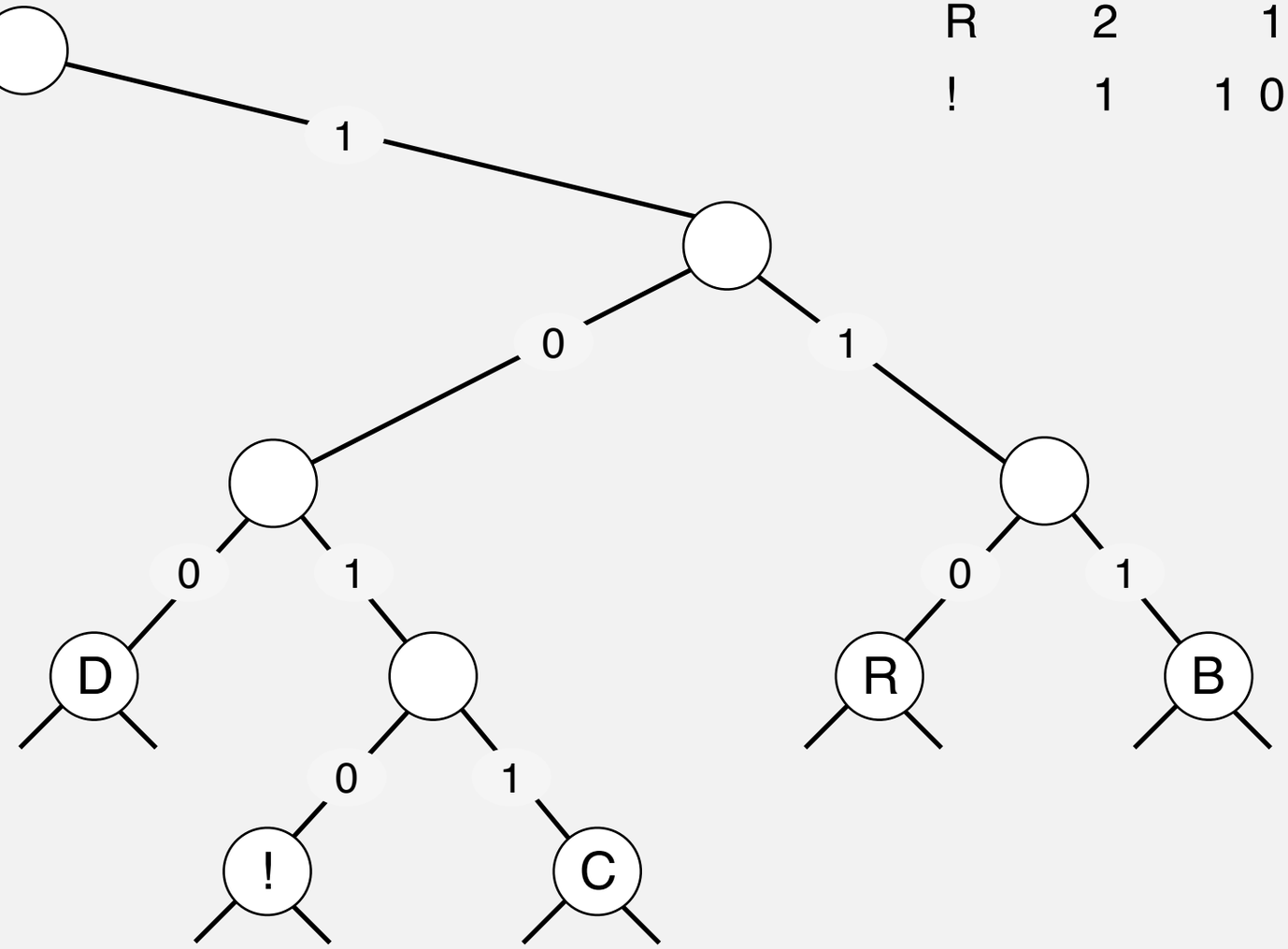
- Select two tries with min weight.
- Merge into single trie with cumulative weight.

char	freq	encoding
A	5	0
B	2	1 1 1
C	1	1 0 1 1
D	1	1 0 0
R	2	1 1 0
!	1	1 0 1 0



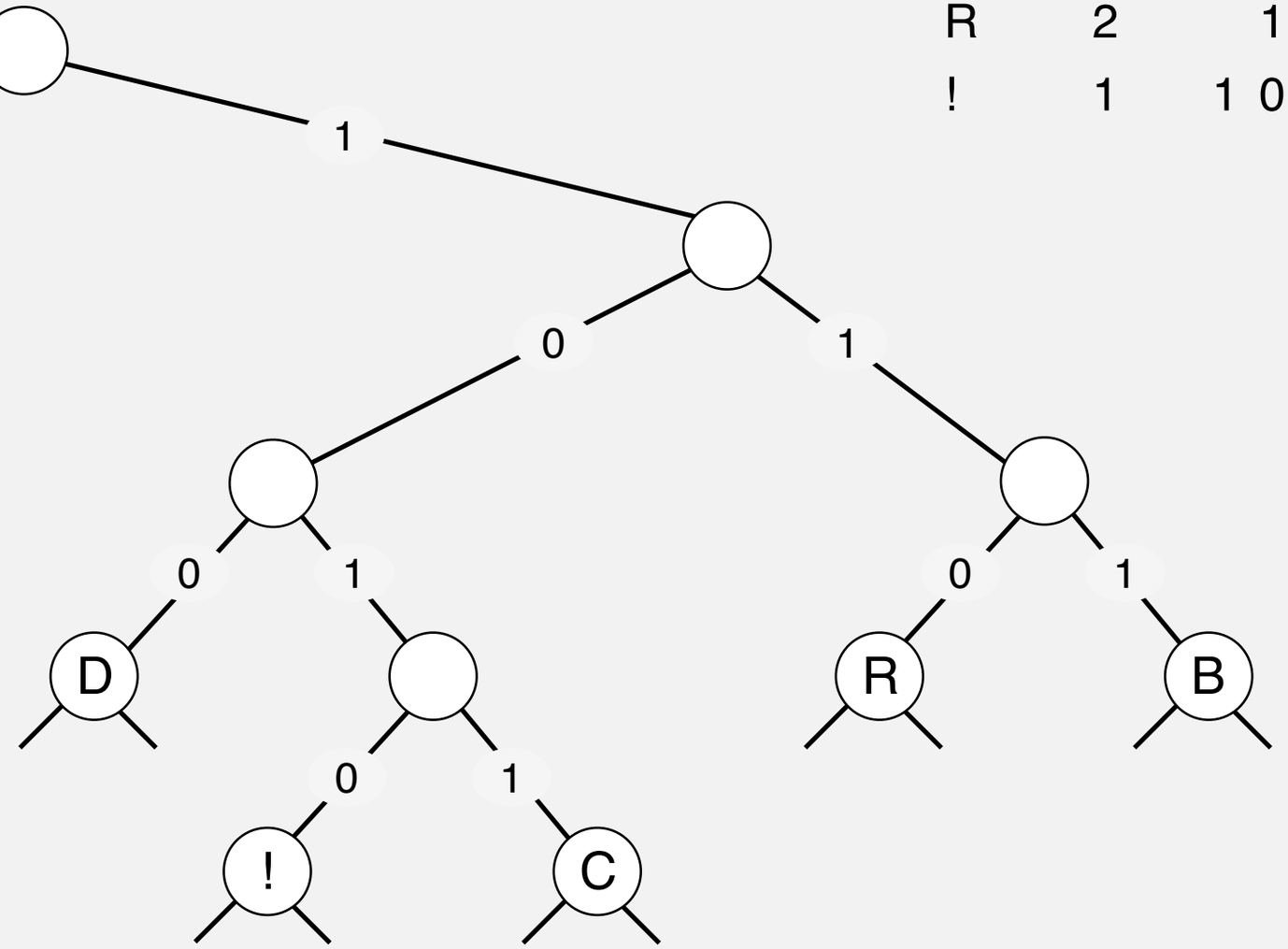
Huffman coding demo

char	freq	encoding
A	5	0
B	2	1 1 1
C	1	1 0 1 1
D	1	1 0 0
R	2	1 1 0
!	1	1 0 1 0



Huffman algorithm demo

char	freq	encoding
A	5	0
B	2	1 1 1
C	1	1 0 1 1
D	1	1 0 0
R	2	1 1 0
!	1	1 0 1 0



Huffman codes

Q. How to find best prefix-free code?

Huffman algorithm:

- Count frequency freq[i] for each char i in input.
- Start with one node corresponding to each char i (with weight freq[i]).
- Repeat until single trie formed:
  - select two tries with min weight freq[i] and freq[j]
  - merge into single trie with weight freq[i] + freq[j]

Applications:

Constructing a Huffman encoding trie: Java implementation

```
private static Node buildTrie(int[] freq)
{
   MinPQ<Node> pq = new MinPQ<Node>();
   for (char i = 0; i < R; i++)
      if (freq[i] > 0)
         pq.insert(new Node(i, freq[i], null, null));

   while (pq.size() > 1)
   {
      Node x = pq.delMin();
      Node y = pq.delMin();
      Node parent = new Node('\0', x.freq + y.freq, x, y);
      pq.insert(parent);
   }

   return pq.delMin();
}
```

initialize PQ with singleton tries

merge two smallest tries

not used for internal nodes

total frequency

two subtries

Huffman encoding summary

Proposition. [Huffman 1950s] Huffman algorithm produces an optimal prefix-free code.

no prefix-free code uses fewer bits

Implementation.

- Pass 1: tabulate char frequencies and build trie.
- Pass 2: encode file by traversing trie or lookup table.

Running time. Using a binary heap $\Rightarrow$ $N + R \log R$.

input size

alphabet size