Compression & Huffman Codes

Data compression

- **Huffman encoding is a simple example of data compression: Represents data in fewer bits.**

- **Huffman codes can be used to compress information**
  - **JPEGs do use Huffman as part of their compression process**

- **The basic idea is that instead of storing each character in a file as an 8-bit ASCII value, we will instead store the more frequently occurring characters using fewer bits and less frequently occurring characters using more bits**

Compression

- **Definition**
  - **Reduce size of data**
    (number of bits needed to represent data)
- **Benefits**
  - **Reduce storage needed**
  - **Reduce transmission cost / latency / bandwidth**

Sources of Compressibility

- **Redundancy**
  - **Recognize repeating patterns**
  - **Exploit using**
    - **Dictionary**
    - **Variable length encoding**
- **Human perception**
  - **Less sensitive to some information**
  - **Can discard less important data**

Types of Compression

- **Lossless**
  - **Preserves all information**
  - **Exploits redundancy in data**
  - **Applied to general data**
- **Lossy**
  - **May lose some information**
  - **Exploits redundancy & human perception**
  - **Applied to audio, image, video**

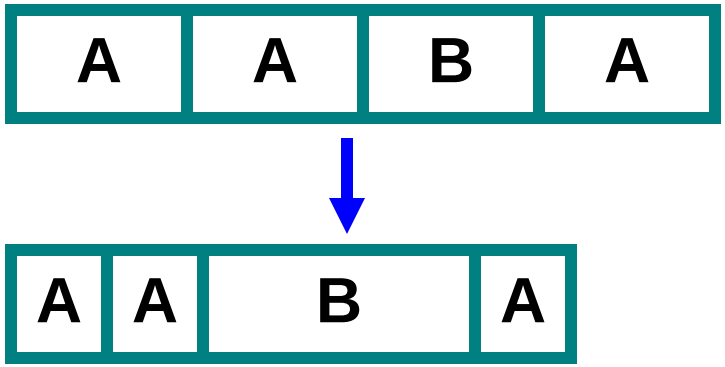
Lossless Compression Techniques

- **LZW (Lempel-Ziv-Welch) compression**
  - **Build pattern dictionary**
  - **Replace patterns with index into dictionary**
- **Run length encoding**
  - **Find & compress repetitive sequences**
- **Huffman codes**
  - **Use variable length codes based on frequency**

Huffman Code

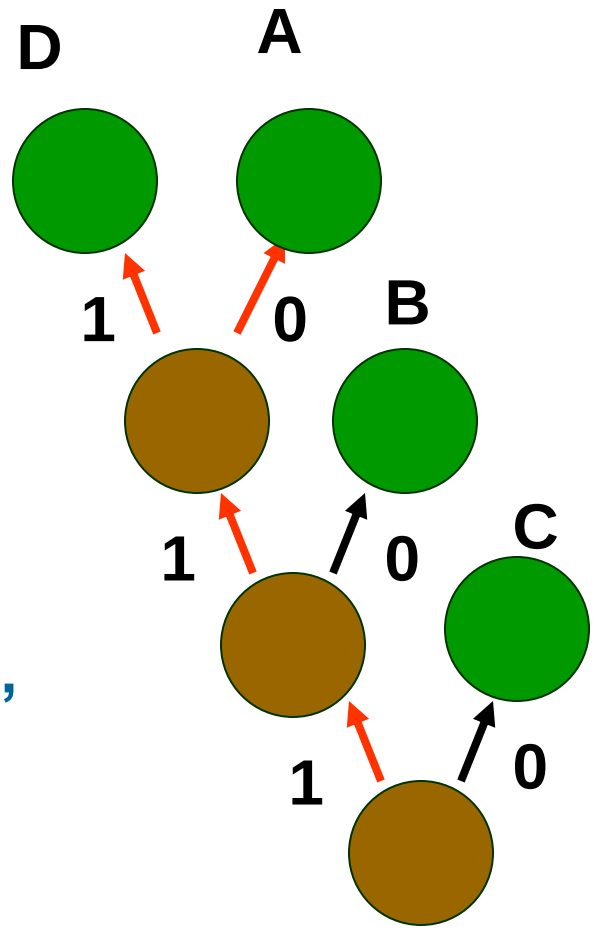
- **Approach**
  - **Variable length encoding of symbols**
  - **Exploit statistical frequency of symbols**
  - **Efficient when symbol probabilities vary widely**

- **Principle**
  - **Use fewer bits to represent frequent symbols**
  - **Use more bits to represent infrequent symbols**



Huffman Code Data Structures

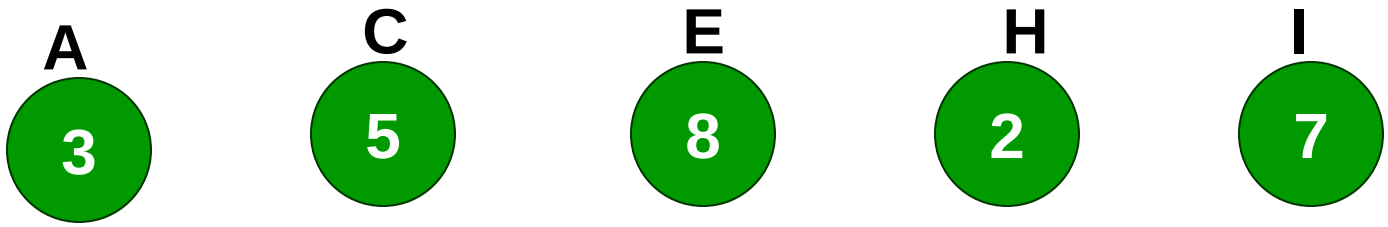
- **Binary (Huffman) tree**
  - Represents Huffman code
  - Edge ⇒ code (0 or 1)
  - Leaf ⇒ symbol
  - Path to leaf ⇒ encoding
  - Example
    - A = “110”, B = “10”, C = “0”



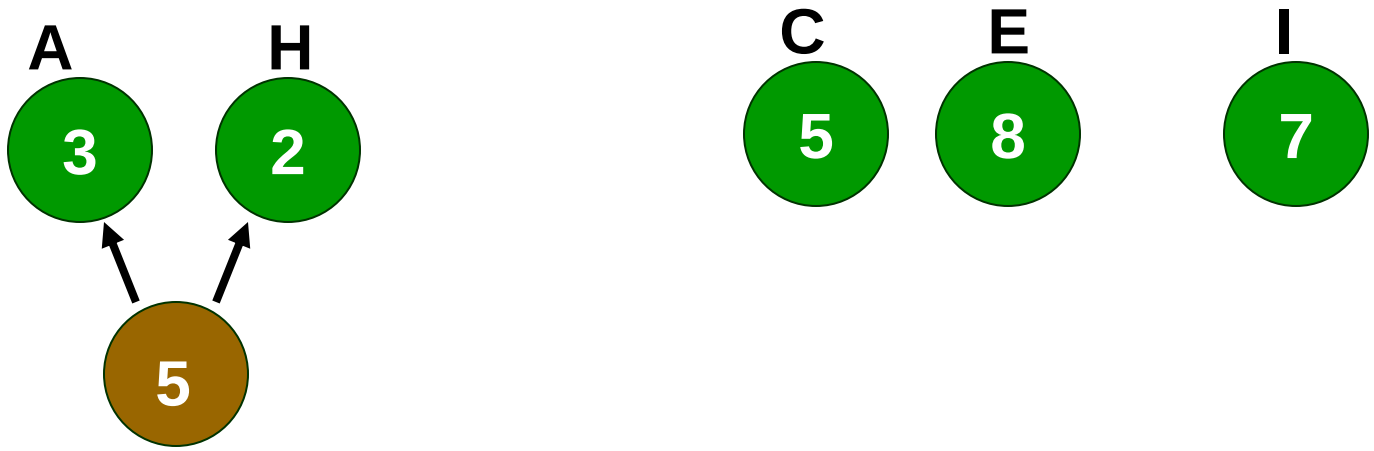
Huffman Code Algorithm Overview

- **Encoding**
  - **Calculate frequency of symbols in file**
  - **Create binary tree representing “best” encoding**
  - **Use binary tree to encode compressed file**
    - **For each symbol, output path from root to leaf**
    - **Size of encoding = length of path**
  - **Save binary tree**

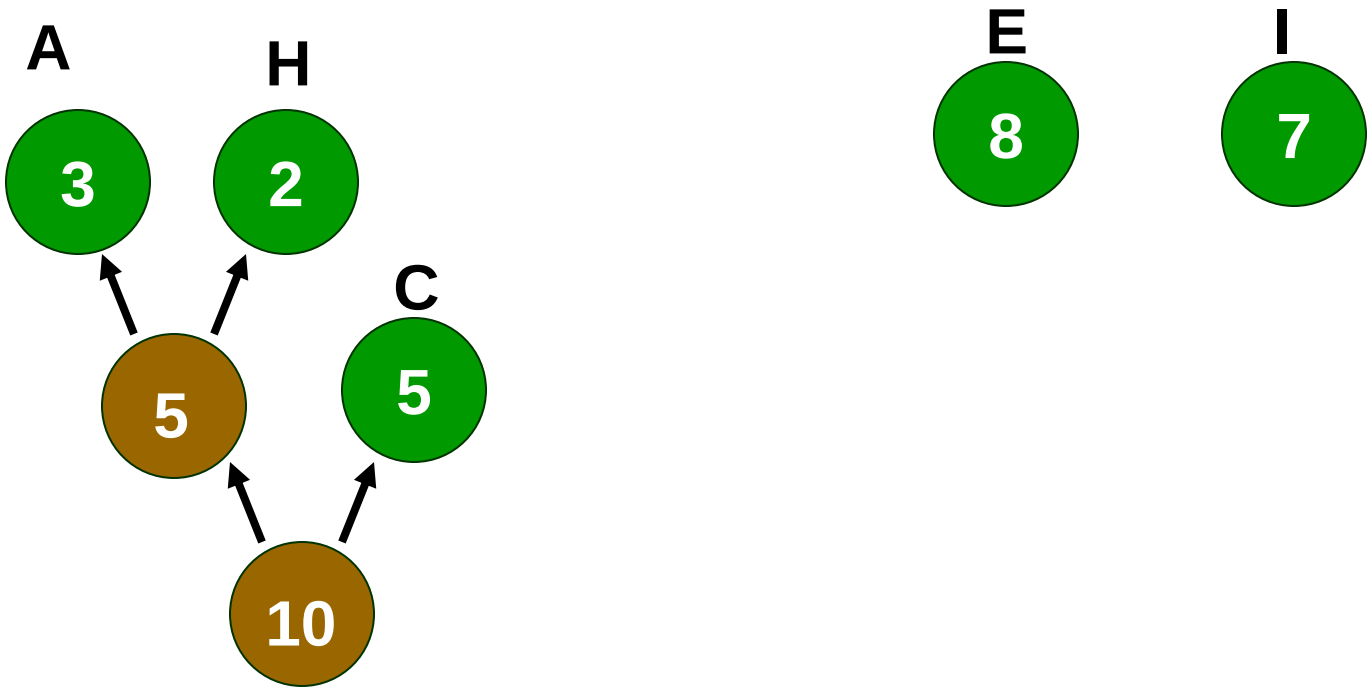
Huffman Tree Construction 1



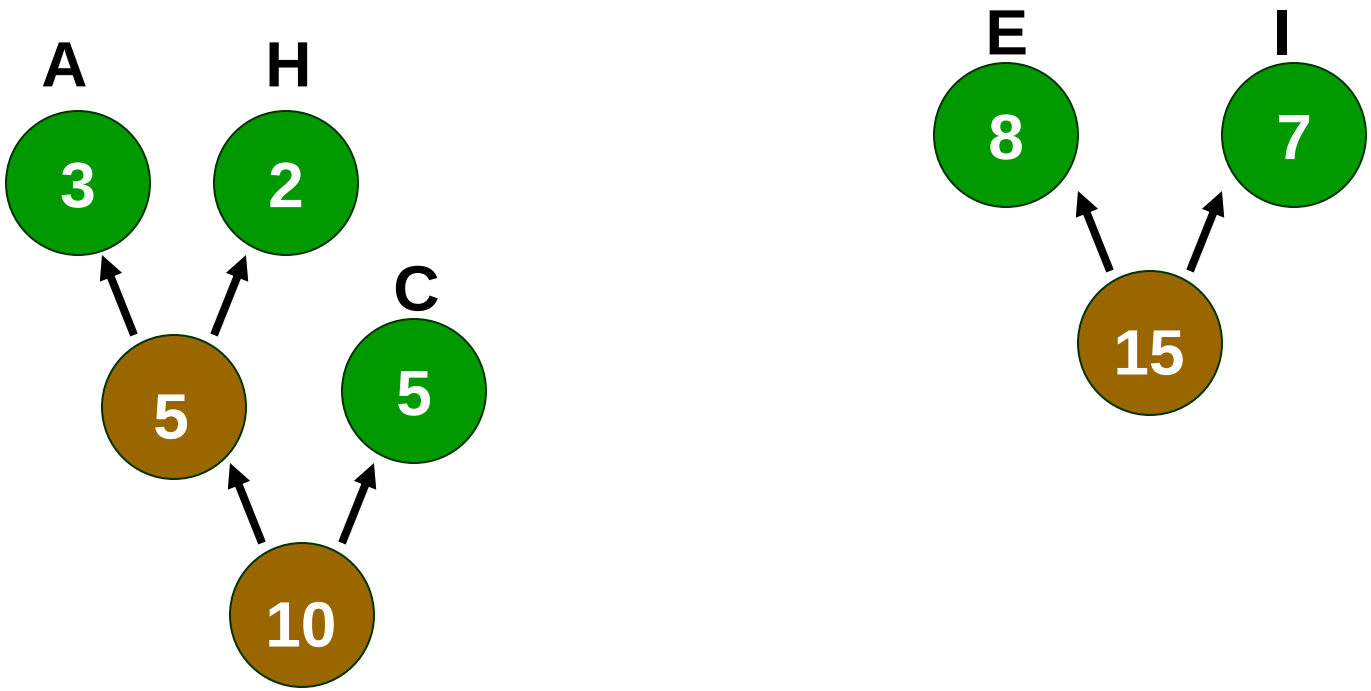
Huffman Tree Construction 2



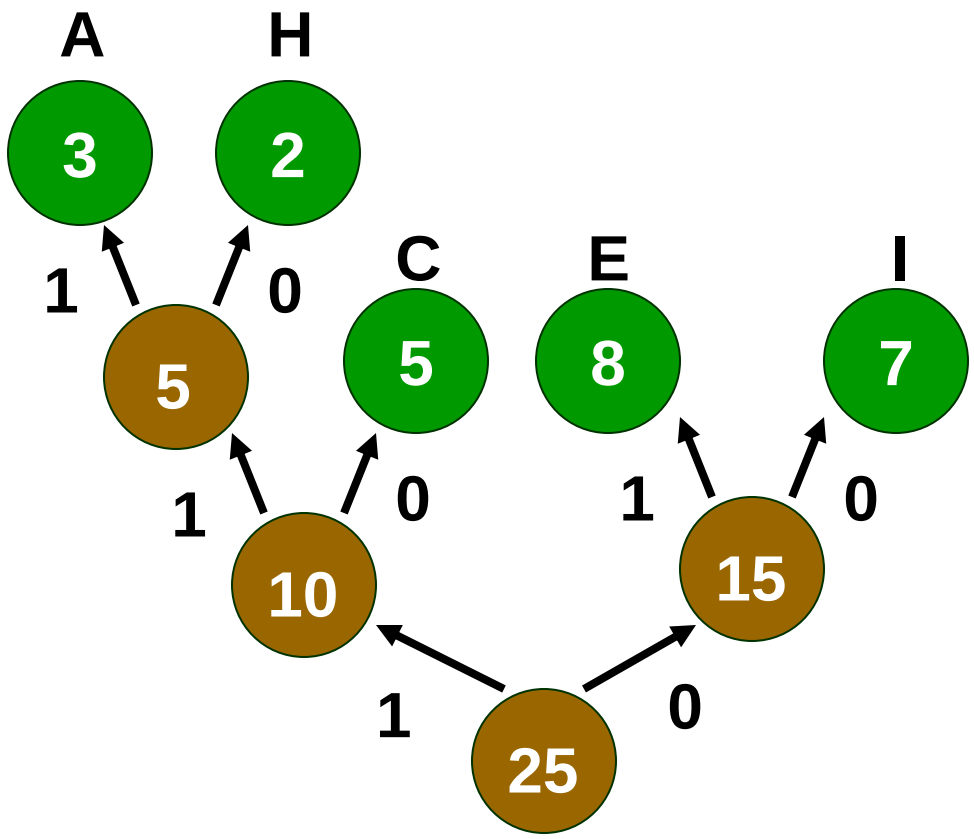
Huffman Tree Construction 3



Huffman Tree Construction 4



Huffman Tree Construction 5



E = 01

I = 00

C = 10

A = 111

H = 110

Huffman Coding Example

- **Huffman code**

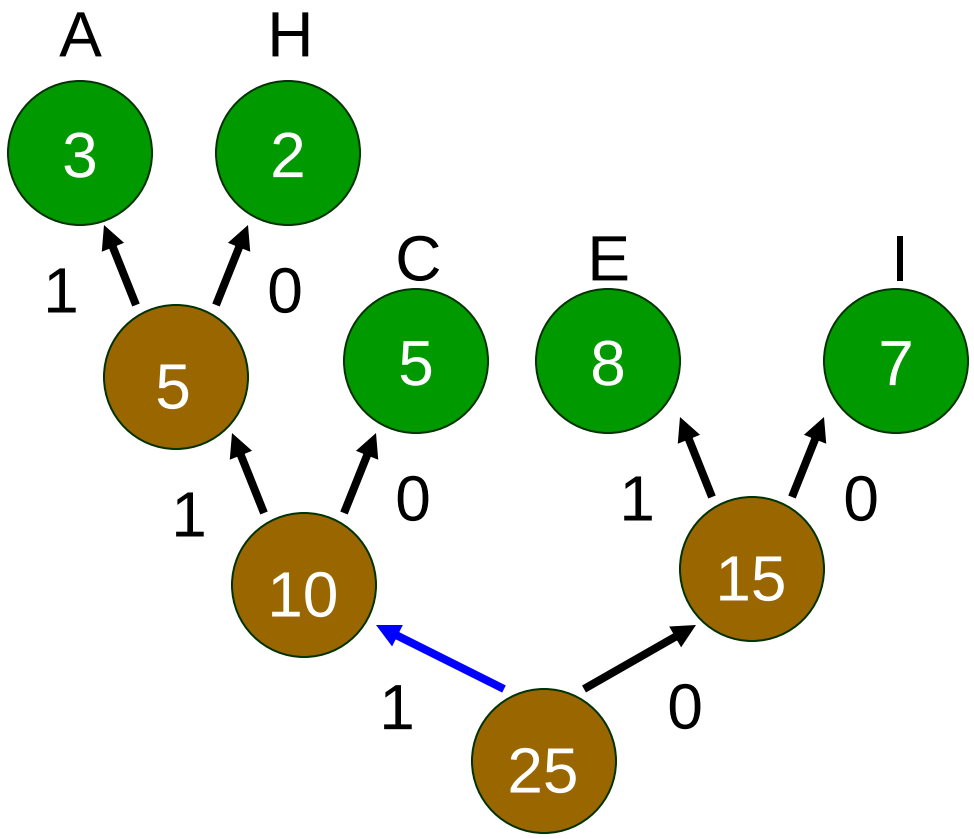
E	=	01
I	=	00
C	=	10
A	=	111
H	=	110

- **Input**
  - **ACE**
- **Output**
  - **(111)(10)(01) = 1111001**

Huffman Code Algorithm Overview

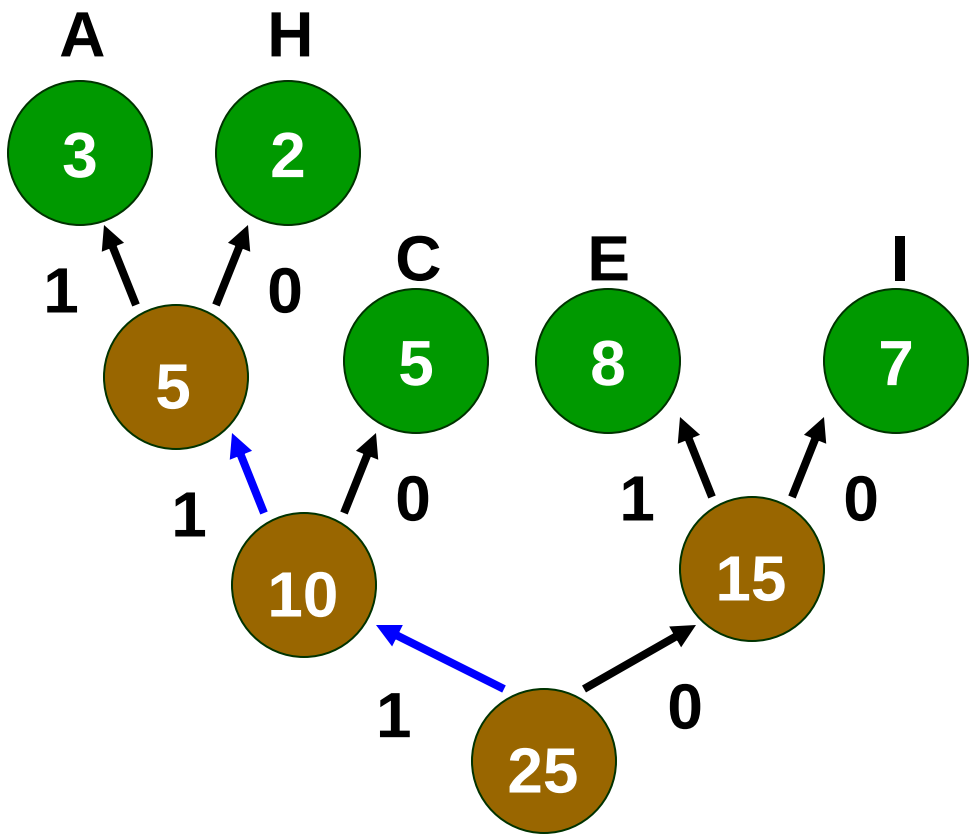
- **Decoding**
  - **Read compressed file & binary tree**
  - **Use binary tree to decode file**
    - **Follow path from root to leaf**

Huffman Decoding 1



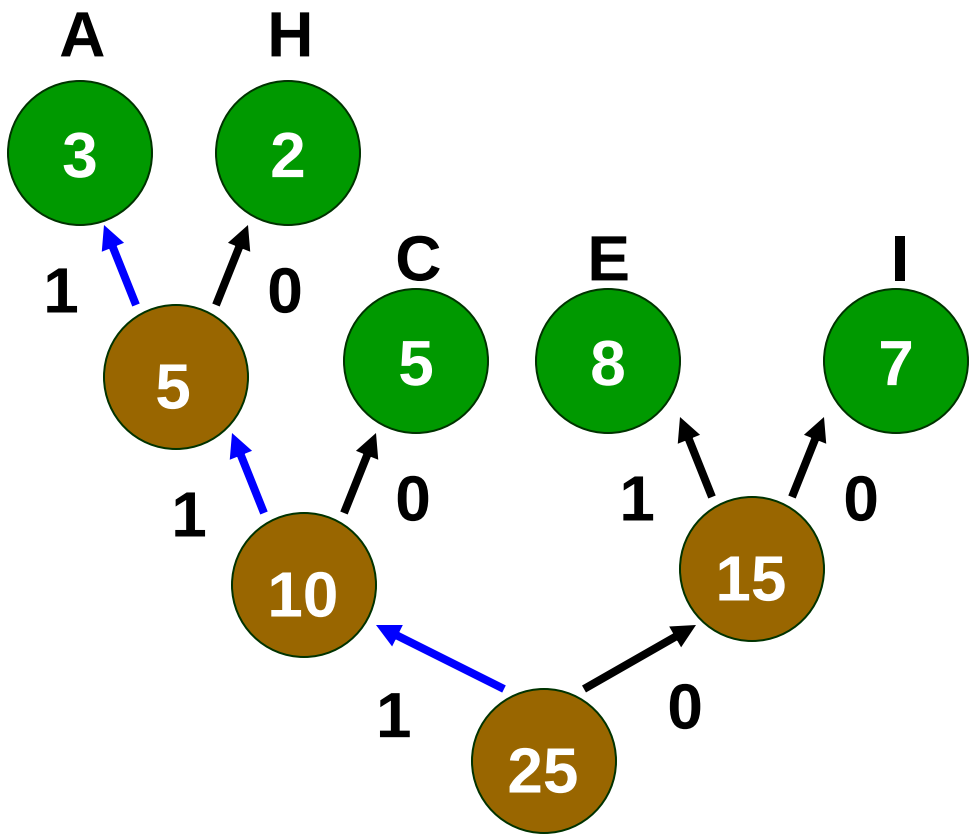
1111001

Huffman Decoding 2



1111001

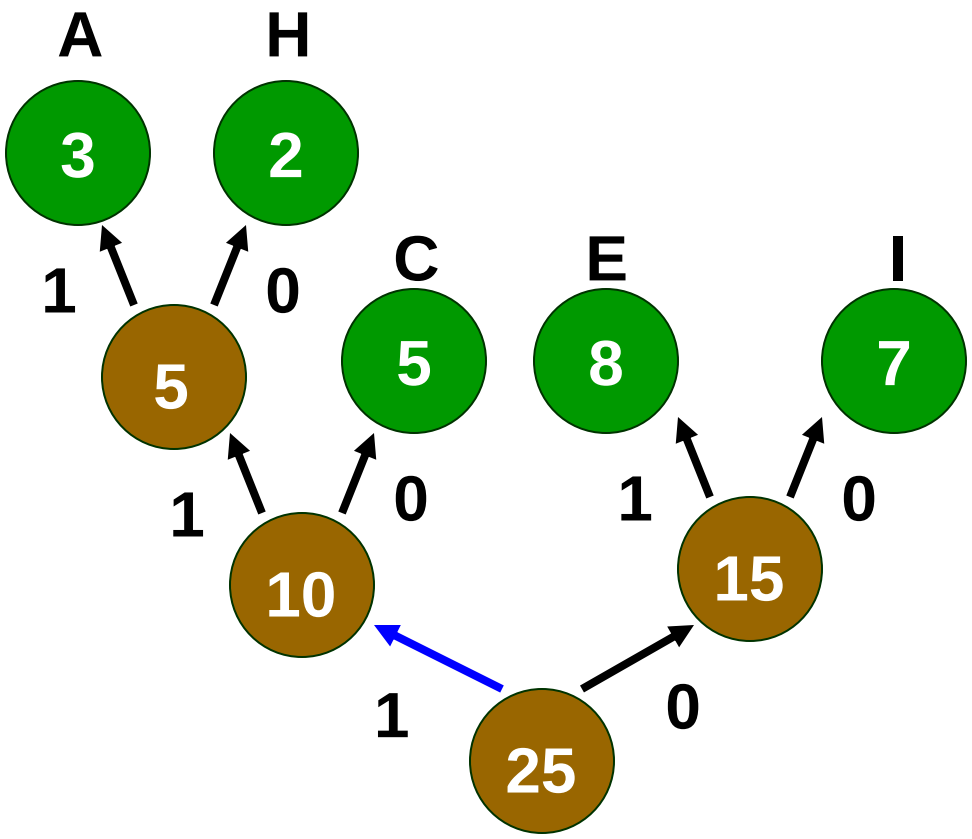
Huffman Decoding 3



1111001

A

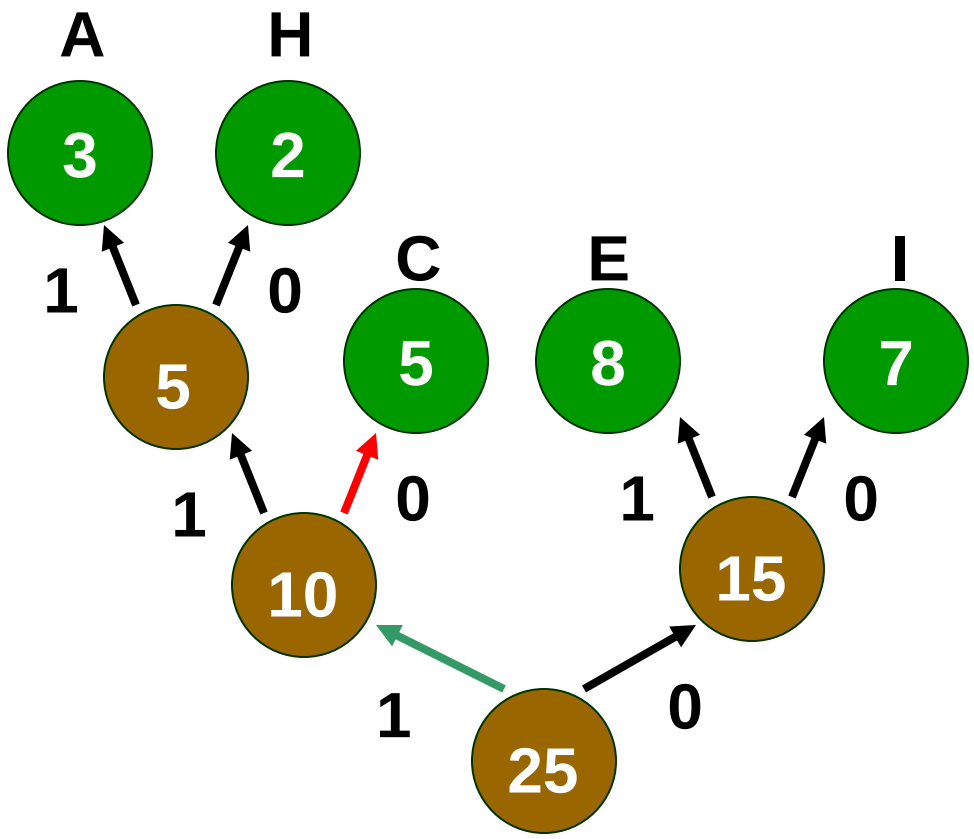
Huffman Decoding 4



1111001

A

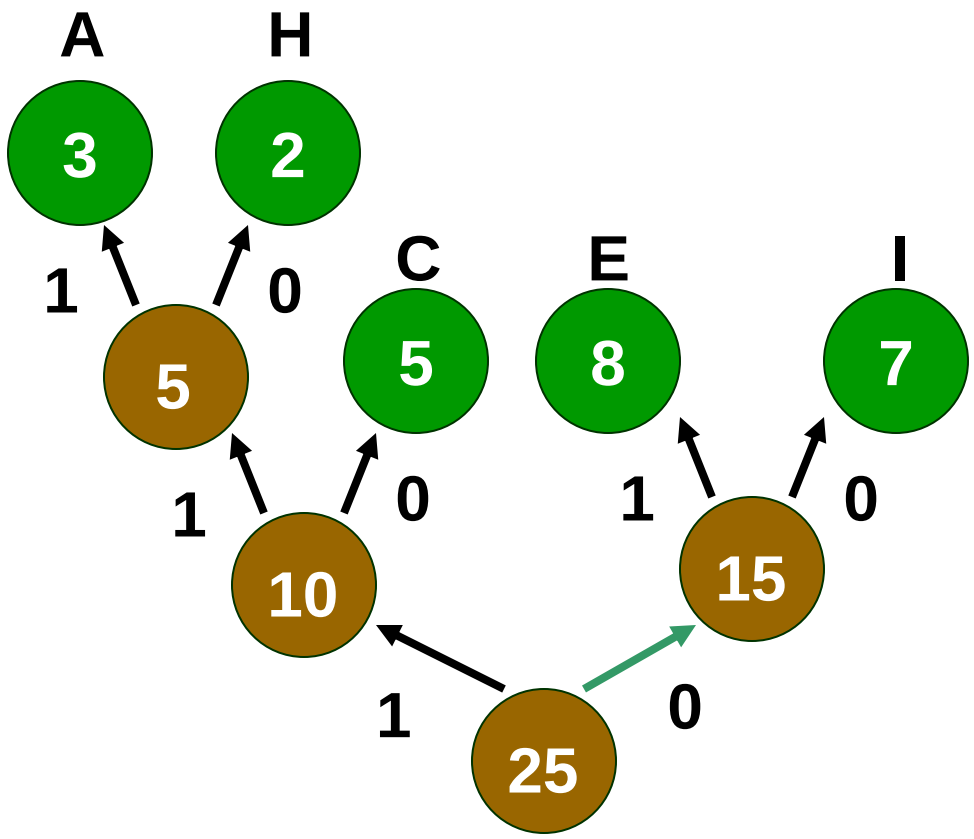
Huffman Decoding 5



1111001

AC

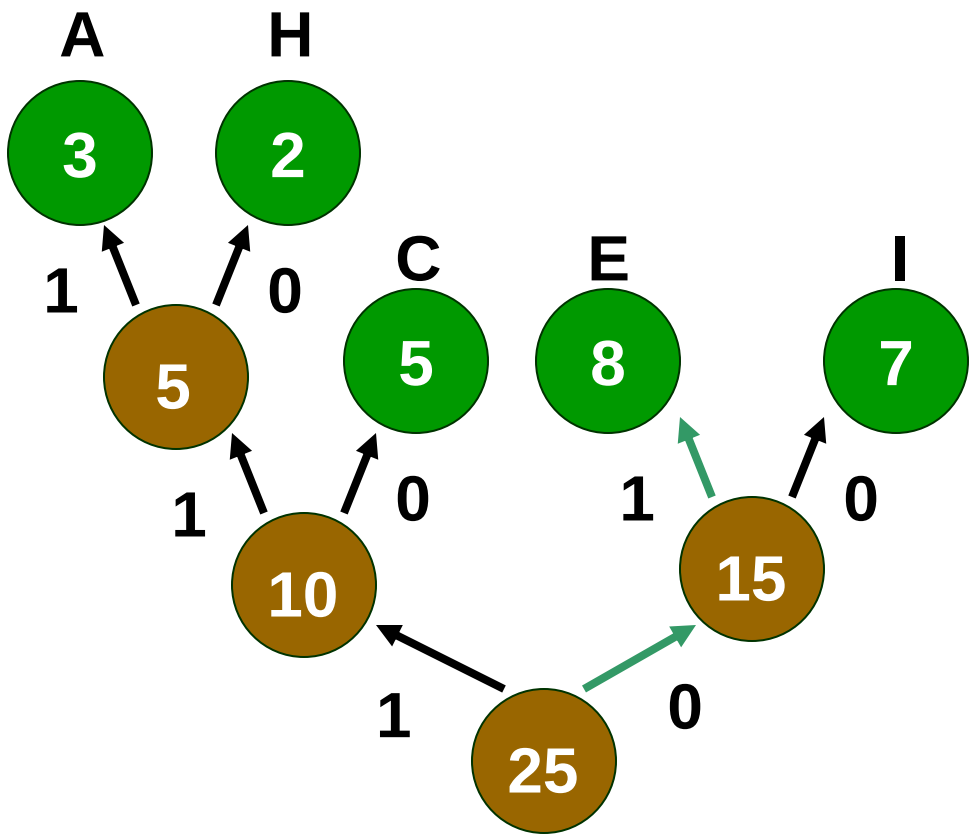
Huffman Decoding 6



1111001

AC

Huffman Decoding 7



1111001

ACE

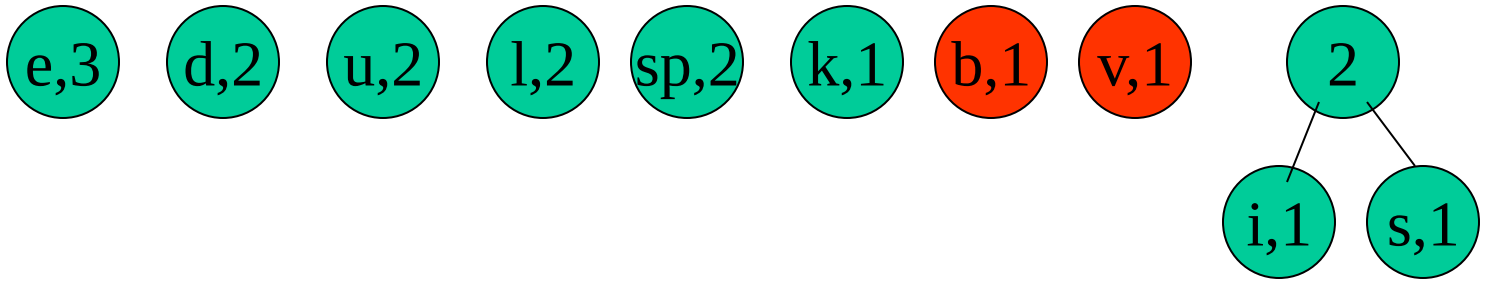
Huffman Coding

- Uncompressing works by reading in the file bit by bit
  - Start at the root of the tree
  - If a 0 is read, head left
  - If a 1 is read, head right
  - When a leaf is reached decode that character and start over again at the root of the tree
- Thus, we need to save Huffman table information as a header in the compressed file
  - Doesn't add a significant amount of size to the file for large files (which are the ones you want to compress anyway)
  - Or we could use a fixed universal set of codes/frequencies

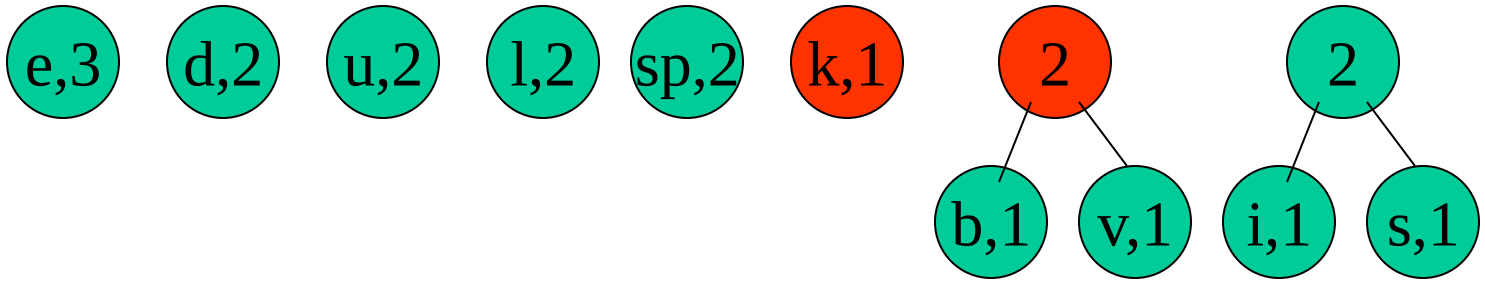
- **2nd Example**

Huffman Coding

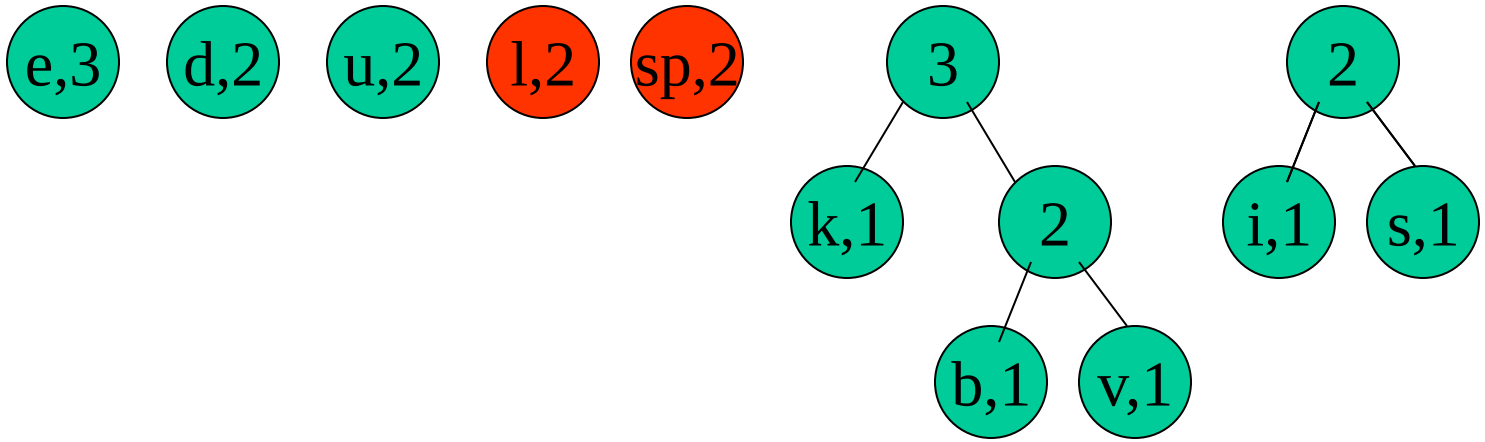
Huffman Coding



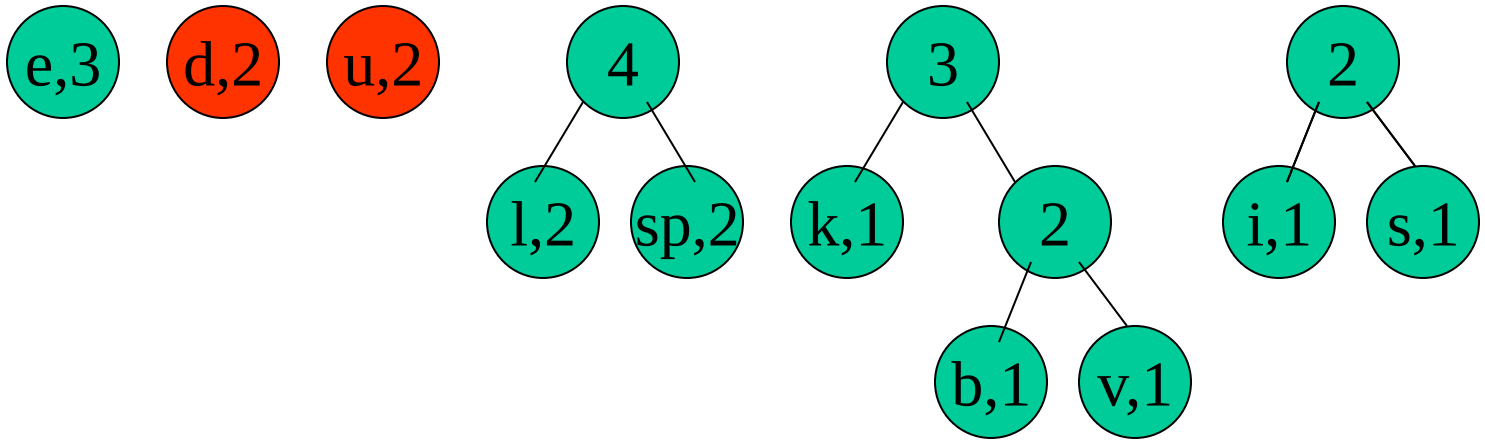
Huffman Coding



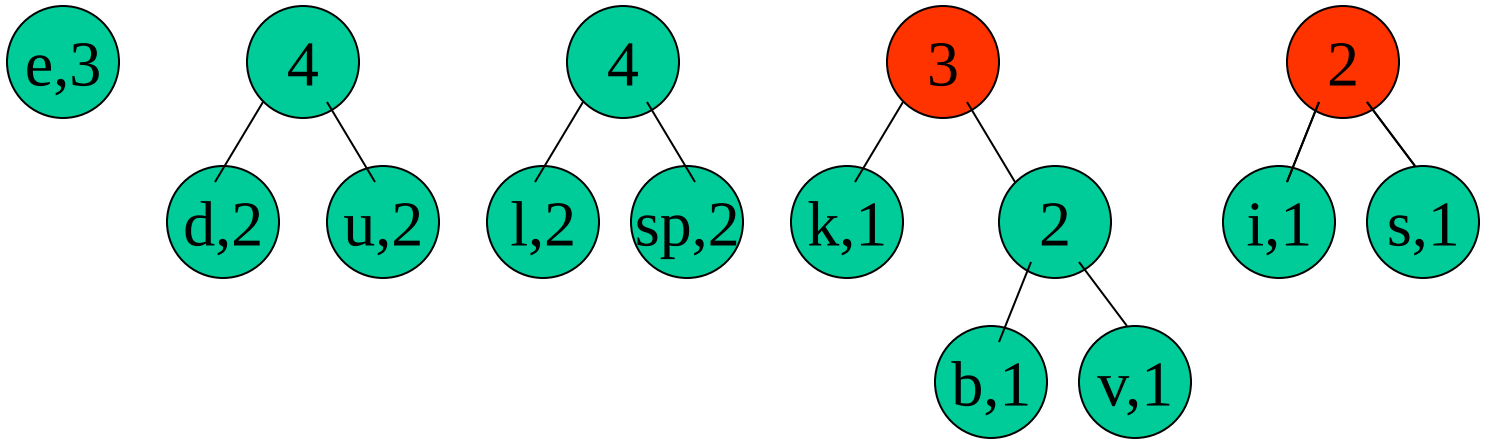
Huffman Coding



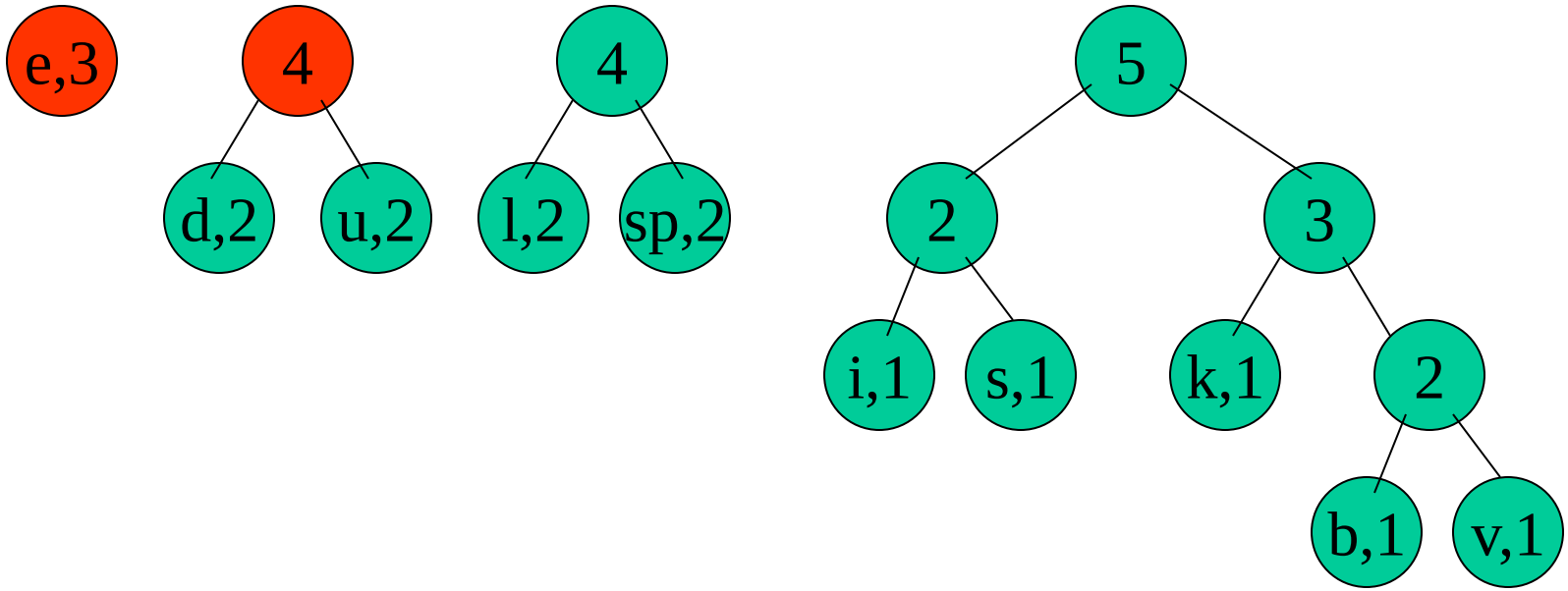
Huffman Coding



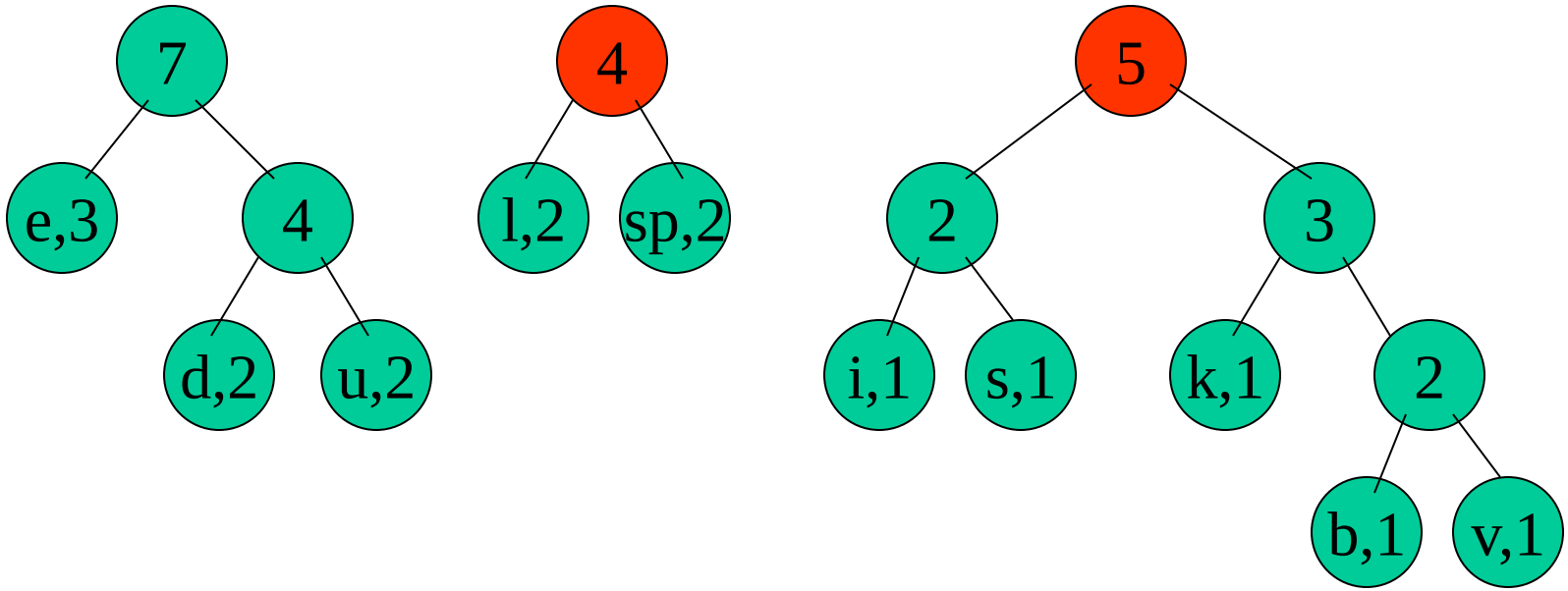
Huffman Coding



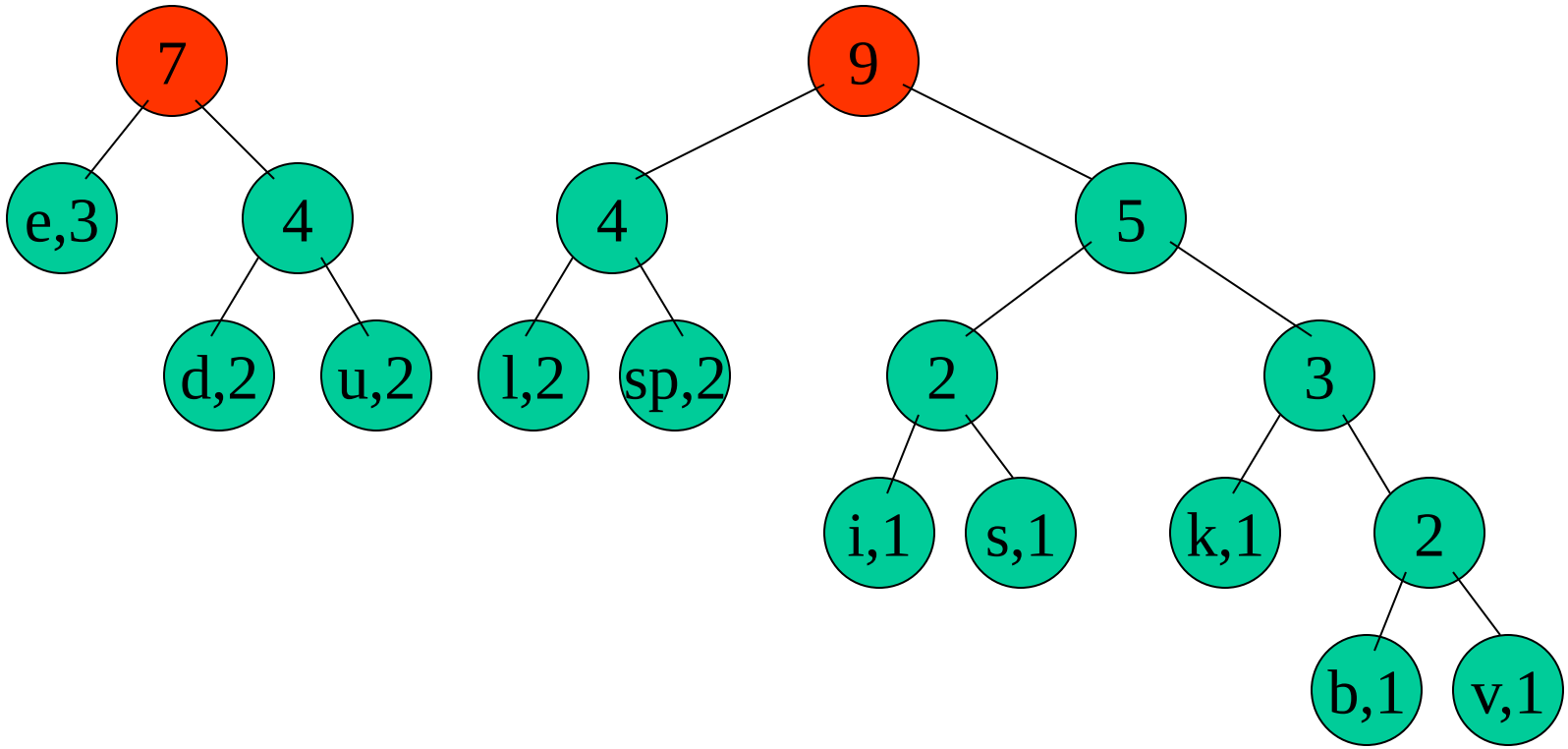
Huffman Coding



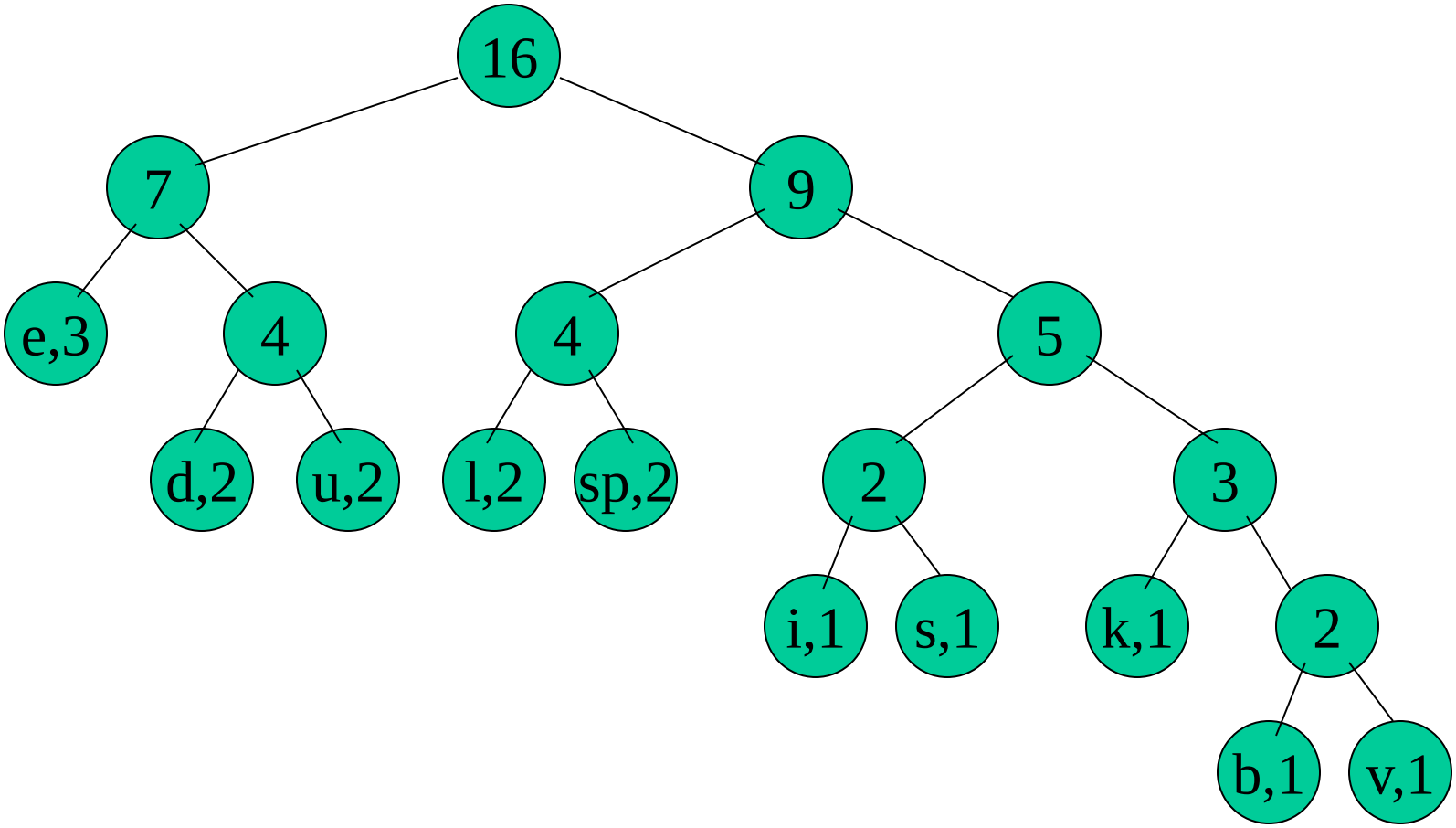
Huffman Coding



Huffman Coding



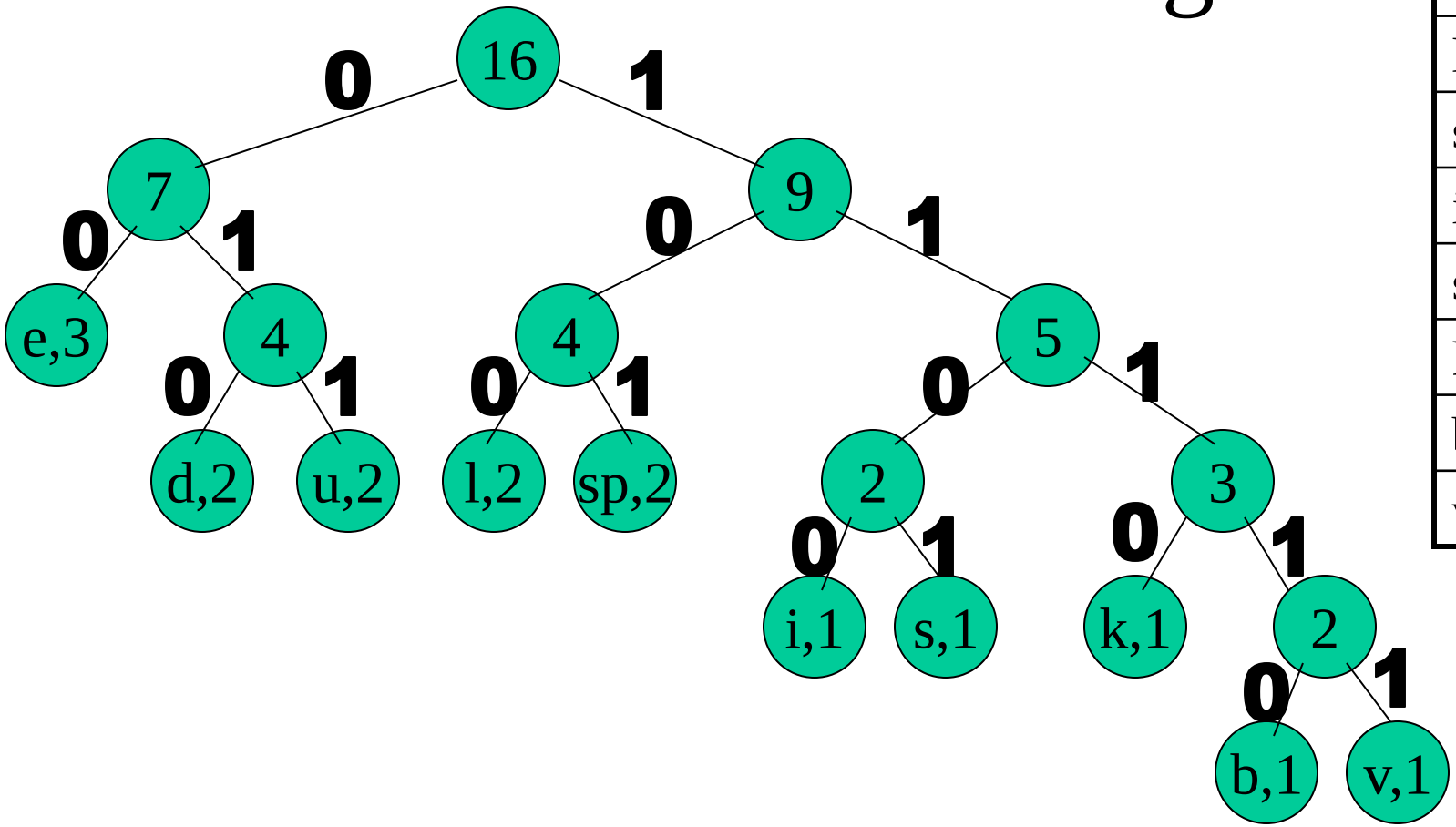
Huffman Coding



Huffman Coding

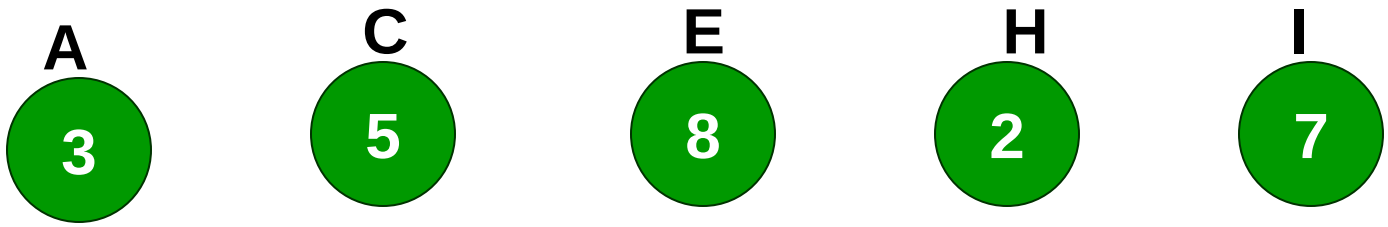
- Now we assign codes to the tree by placing a 0 on every left branch and a 1 on every right branch
- A traversal of the tree from root to leaf give the Huffman code for that particular leaf character
- Note that no code is the prefix of another code

Huffman Coding



e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

Huffman Tree Construction 1



E = 01

I = 00

C = 10

A = 111

H = 110

(3*8)+(5*8)+ (8*8)+(2*8)+ (7*8) =200

(3*3)+(5*2)+(8*2)+(2*3)+(7*2) = 55

- **Suppose A, B, C, D, E, F, G and H are 8 data items, and suppose they are assigned weights as follows:**

A	B	C	D	E	F	G	H
22	5	11	19	2	11	25	5

How to construct the tree T with minimum weighted path length using the above data and Huffman algorithm?