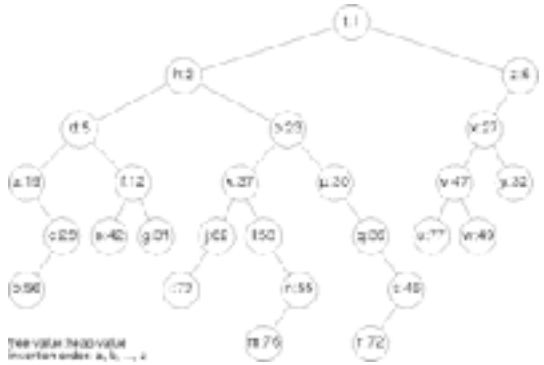
APPLICATION OF DATA STRUCTURES BINARY CODE TREES

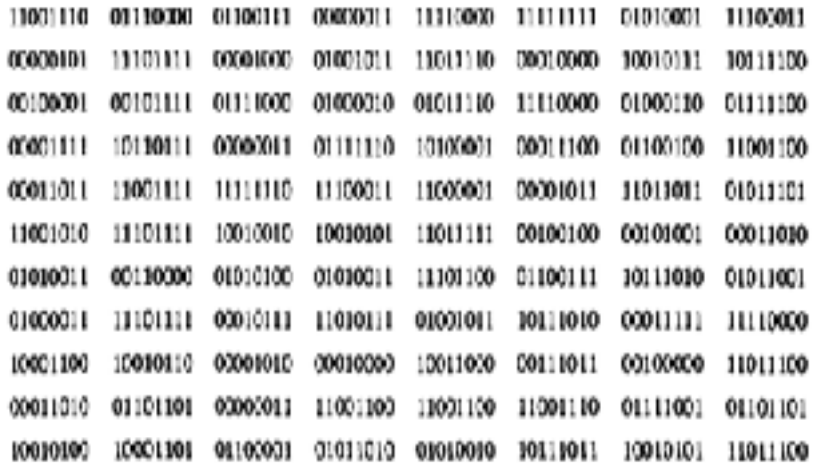
Data Compression

- Data compression: Represent digital media using the fewest number of bits!

Text, video, audio

We will do this using trees!





All data is bits!

Fixed length encoding

- Fixed length: each symbol is represented using a fixed number of bits, example ASCII encoding
- For example for the symbols ‘s’, ‘p’, ’a’, ‘m’ one possible encoding is:

spamspamspam
spamspamspam

Text file

0001101100011011000110110001101100011011
0001101100011011000110110001101100011011
0001101100011011000110110001101100011011

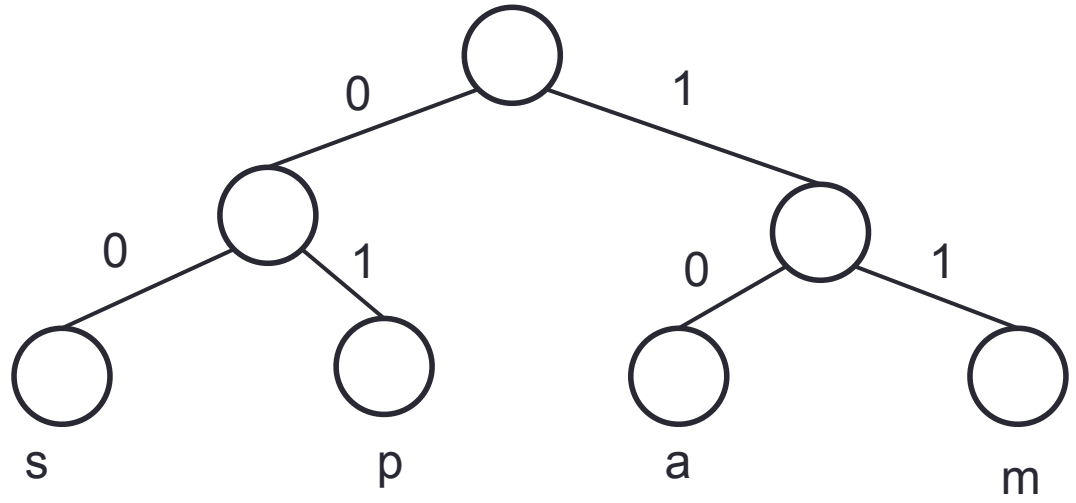
Encoded file

Symbol	Codeword
s	00
p	01
a	10
m	11

For a dictionary consisting of M symbols, what is the minimum number of bits needed to encode each symbol (assume fixed length binary codes) ?

A. 2^M B. M C. M/2 D. ceil($\log_2$ M) E. None of these

Binary codes as Binary Trees

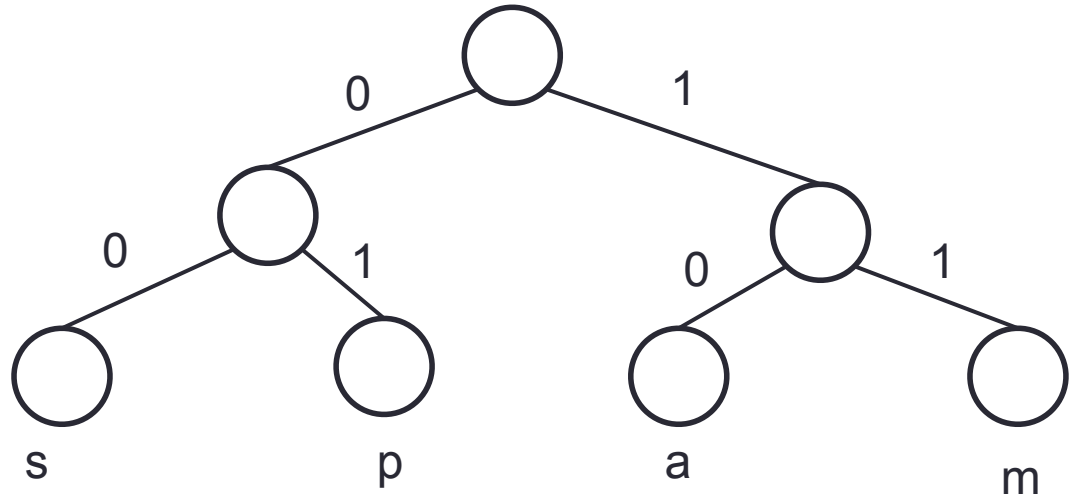


Code A

Symbol	Codeword
s	00
p	01
a	10
m	11

- Symbols are leaf nodes
- Root to leaf node gives the codeword for each symbol
- Once we have the tree we can encode and decode data
- Given the tree
  - Encode the string ‘papa’
  - Decode the binary sequence ‘01101100’

Encoding and decoding on Binary Coded Trees

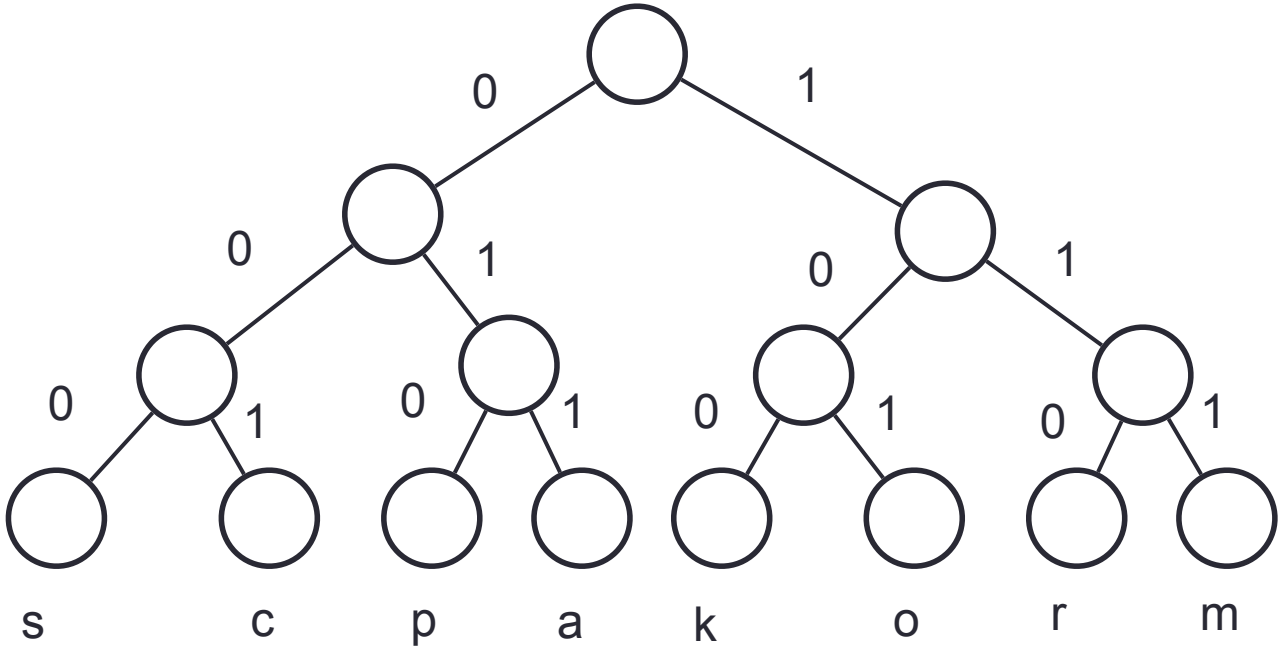


Code A

Symbol	Codeword
s	00
p	01
a	10
m	11

- Encoding a string (e.g. papa) -
  - locate the leaf node for that symbol (e.g. p)
  - Traverse up to the root, recording the bits you see along the way
  - Reverse to get the code
- Decoding a binary string: e.g. **00 00 00 10 01**
  - Traverse down the tree from the root, using each bit as a direction to go left or right
  - Record each leaf node that you encounter and repeat!

Decoding on binary trees, another example



Decode the bitstream 110101001100 using the given binary tree

A. scam
B. mork
C. rock
D. rkop

- Do we need to be constrained to fixed length encoding?
- What if certain symbols appeared more often than others?

Variable length codes

ssssssssssssssss
sspppppaampamm

Text file

Symbol	Counts
s	18
p	6
a	3
m	3

Symbol	Frequency
s	0.6
p	0.2
a	0.1
m	0.1

Code A

Symbol	Codeword
s	00
p	01
a	10
m	11

Code B

Symbol	Codeword
s	0
p	1
a	10
m	11

Average length (code A) = 2 bits/symbol

Average length (code B) = 0.6 *1 +0.2 *1 + 0.1* 2+ 0.1*2
= 1.2 bits/symbol

Comparing encoding schemes

ssssssssssssssss
ssppppaampamm

Text file

Symbol	Counts
s	18
p	6
a	3
m	3

Symbol	Frequency
s	0.6
p	0.2
a	0.1
m	0.1

Code A

Symbol	Codeword
s	00
p	01
a	10
m	11

Code B

Symbol	Codeword
s	0
p	1
a	10
m	11

Is code B better than code A?
A. Yes
B. No
C. Depends

Hint: Try decoding the binary pattern 0110 using Code B

Variable length codes

Variable length codes only work if no symbol's codeword is a prefix of another

Code A

Symbol	Codeword
s	00
p	01
a	10
m	11

Code B

Symbol	Codeword
s	0
p	1
a	10
m	11

Decoding 01:
'sp' OR 'a'?

Code B is not a valid code because we cannot uniquely decode certain bit streams

In search of a better code

Is code C better than code A and code B? (Why or why not?)

A. Yes
B. No

Code A

Symbol	Codeword
s	00
p	01
a	10
m	11

Code B

Symbol	Codeword
s	0
p	1
a	10
m	11

Code C

Symbol	Codeword
s	0
p	10
a	110
m	111

Symbol	Frequency
s	0.6
p	0.2
a	0.1
m	0.1

Variable length codes

Symbol	Frequency
s	0.6
p	0.2
a	0.1
m	0.1

Code A

Symbol	Codeword
s	00
p	01
a	10
m	11

Code B

Symbol	Codeword
s	0
p	1
a	10
m	11

Code C

Symbol	Codeword
s	0
p	10
a	110
m	111

Average length (code A) = 2 bits/symbol

Average length (code B) = 0.6 *1 +0.2 *1 + 0.1* 2+ 0.1*2
= 1.2 bits/symbol

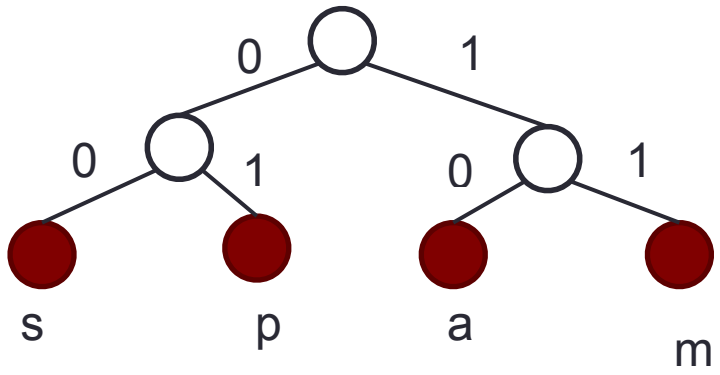
Average length (code C) = 0.6 *1 +0.2 *2 + 0.1* 3+ 0.1*3
= 1.6 bits/symbol

Lower bits per symbol is better

Advantage of thinking of codes as trees

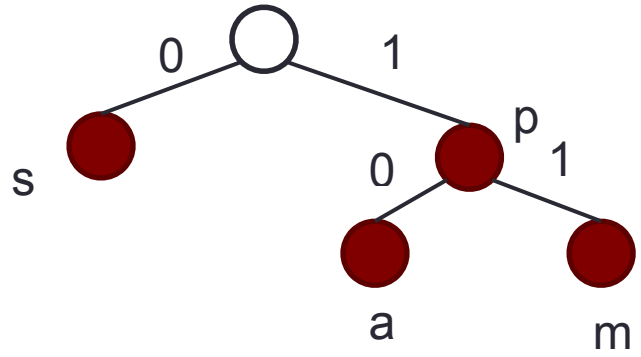
Code A

Symbol	Codeword
s	00
p	01
a	10
m	11



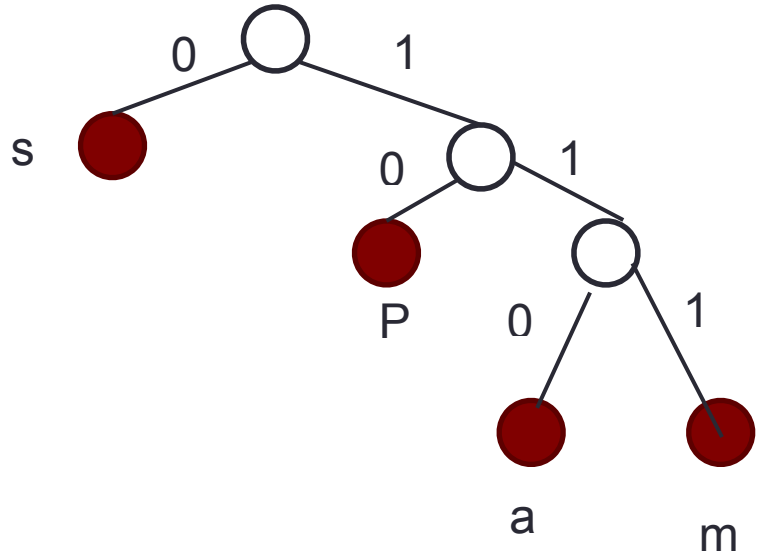
Code B

Symbol	Codeword
s	0
p	1
a	10
m	11



Code C

Symbol	Codeword
s	0
p	10
a	110
m	111



Problem Definition

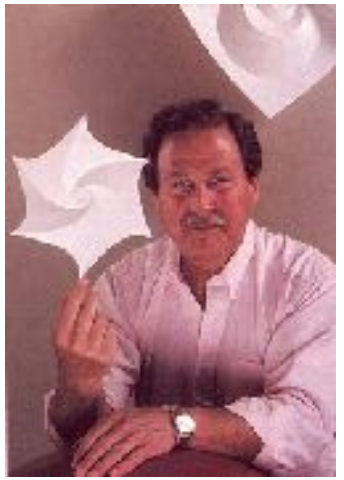
mapsmppam
ssampamsmam
…

TEXT FILE

Symbol	Frequency
s	0.6
p	0.2
a	0.1
m	0.1

Huffman coding is one of the fundamental ideas that people in computer science and data communications are using all the time - Donald Knuth

David Huffman



Given a frequency distribution over M symbols, find the optimal binary code i.e. one that minimizes the average code length and is decodable (no code is the prefix of another code)

Problem Definition (reworded for trees)

Input: The frequency (f_i) of occurrence of each symbol (S_i)

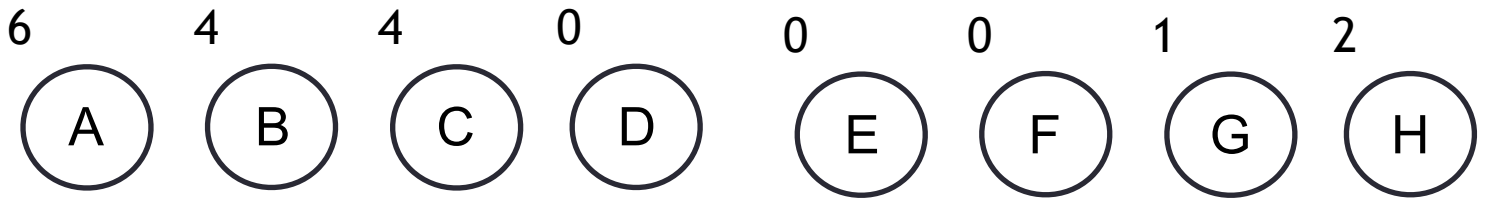
Output: Binary tree *T* that minimizes the following objective function:

$$L(T) = \sum_{i=1:M} f_i \cdot Depth(S_i \text{ in } T)$$

Solution: Huffman Codes

Huffman's algorithm: Bottom up construction

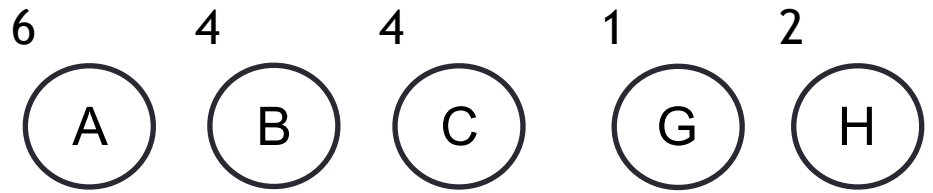
- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node



A: 6; B: 4; C: 4; D: 0; E: 0; F: 0; G: 1; H: 2

Huffman's algorithm: Bottom up construction

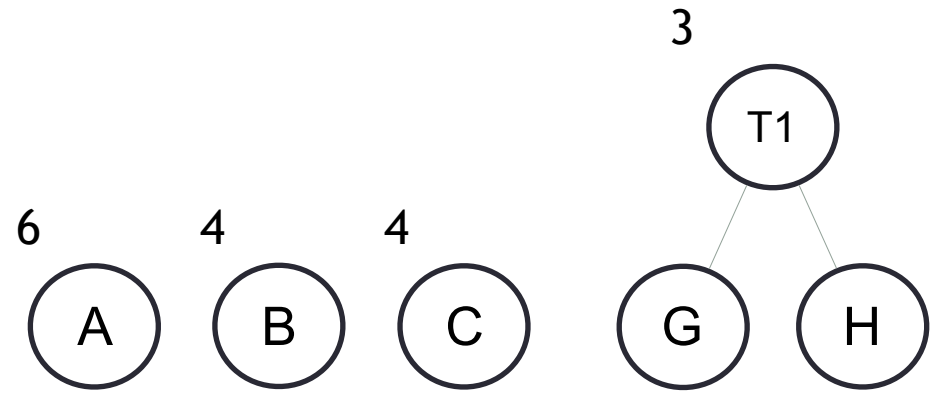
- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them



A: 6; B: 4; C: 4; D: 0; E: 0; F: 0; G: 1; H: 2

Huffman's algorithm: Bottom up construction

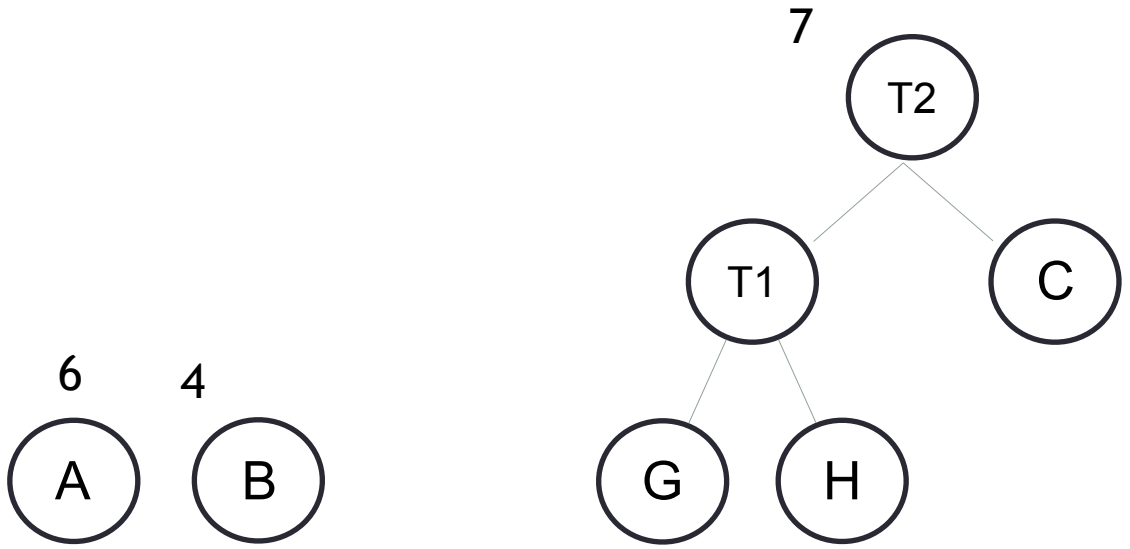
- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them



A: 6; B: 4; C: 4; D: 0; E: 0; F: 0; G: 1; H: 2

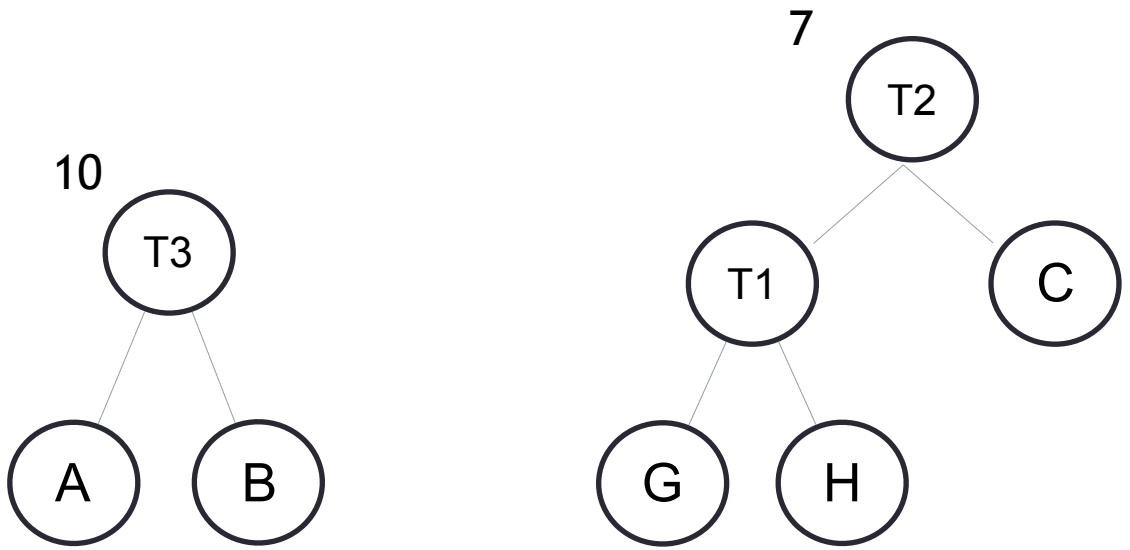
Huffman's algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree



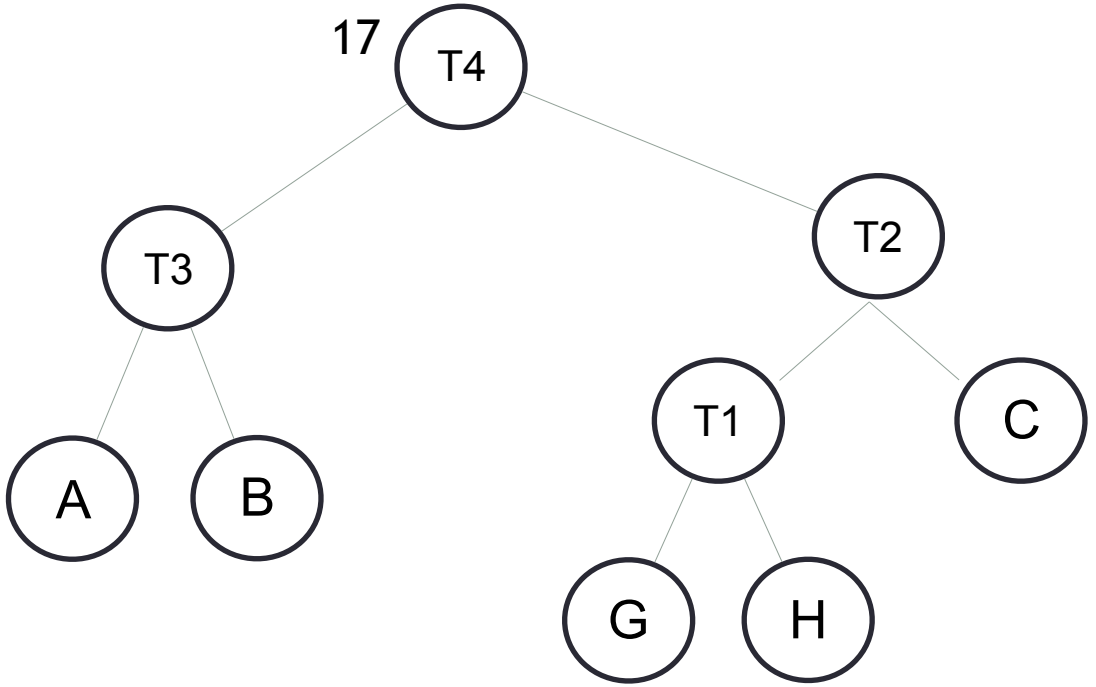
Huffman's algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree

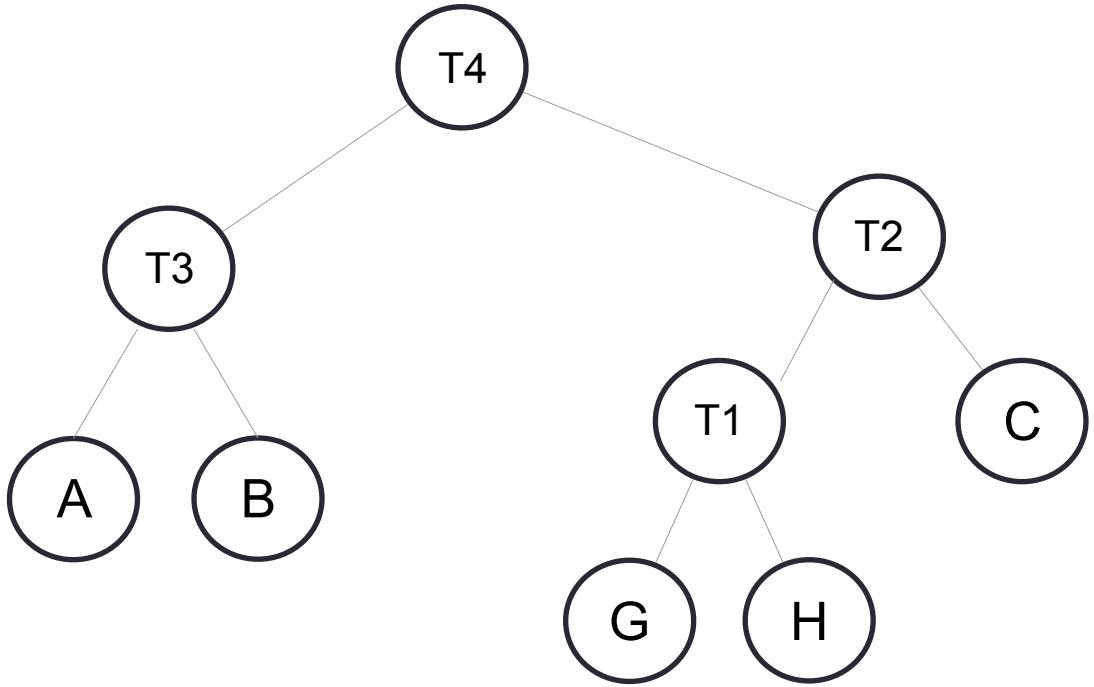


Huffman's algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree



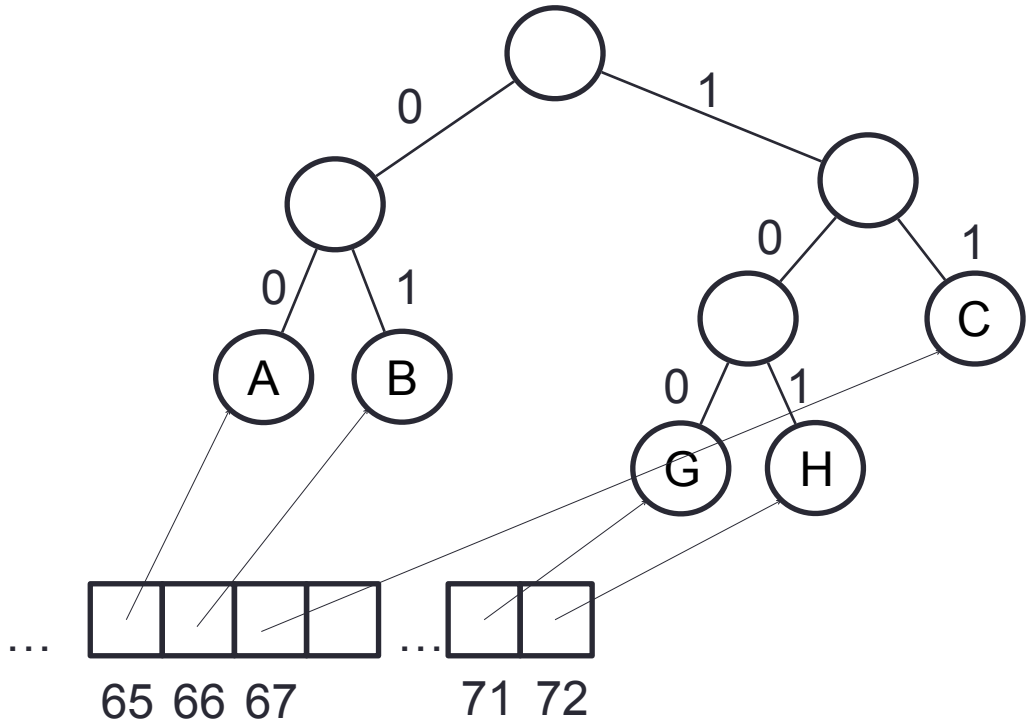
Encoding a symbol- think implementation!



- Compression using trees:
  - Devise a “good” code/tree
  - Encode symbols using this tree

A very bad way is to start at the root and search down the tree until you find the symbol you are trying to encode, why?

Encoding a symbol



A much better way is to maintain a list of leaves and then to traverse the tree to the root (and then reverse the code)

```
vector<HCNode*> leaves;
…
leaves = vector<HCNode*>(256, (HCNode*)0);
```