

# Design and Analysis of Algorithms

## L27: Huffman Codes

### Optimal Tree Subproblem

Dr. Ram P Rustagi  
Sem IV (2019-H1)  
Dept of CSE, KSIT/KSSEM  
[rprustagi@ksit.edu.in](mailto:rprustagi@ksit.edu.in)

# Resources

- Text book 1: Sec 9.1-5.4 - Levitin
- RI: Introduction to Algorithms
  - Cormen et al.

# Data Compression

- Consider saving a text file consisting of alphabets
  - It uses ASCII encoding,
  - each character uses 7 bits.
  - Thus, if file has 1000 characters,
    - file size is 7000 bits
- We know in english text certain characters appear more often than others e.g. a, e, i, t, h, s etc.
  - Other characters appear less often e.g.. z, x, q etc.
- Can we use a different representation than ASCII
  - Assign shorter codes to chars occurring frequently
  - Assign longer codes to chars occurring less times.
  - Will we save disk space?

# Data Compression: Communication

- Consider choosing electives with percentage of students
  - 17CS561:Java Programming (25%)
  - 17CS562:Artificial Intelligence (12.5%)
  - 17CS563:Embedded Systems (6.25%)
  - 17CS564:Dot Net framework (6.25%)
  - 17CS565:Cloud computing (50%)
- A general coding would require 3 bits, e.g.
  - 000 - Java Programming
  - 001 - AI
  - 010 - Embedded Systems
  - 011 - DotNet framework
  - 100 - Cloud Computing
- Can we employ better encoding so that average bits becomes less than 3.

# Data Compression: Communication

- Consider choosing electives with percentage of students
  - 17CS561:Java Programming (25%)
  - 17CS562:Artificial Intelligence (12.5%)
  - 17CS563:Embedded Systems (6.25%)
  - 17CS564:Dot Net framework (6.25%)
  - 17CS565:Cloud computing (50%)

- Consider following encoding

0 - Cloud Computing

10 - Java Programming

110 - AI

1110 - Embedded Systems

1110 - DotNet framework

- What is the average number of bits for encoding these?

$$1 * 1/2 + 2 * 1/4 + 3 * 1/8 + 4 * 1/16 + 4 * 1/16 = 15/8 = 1.875$$

# What is a Coding Problem...

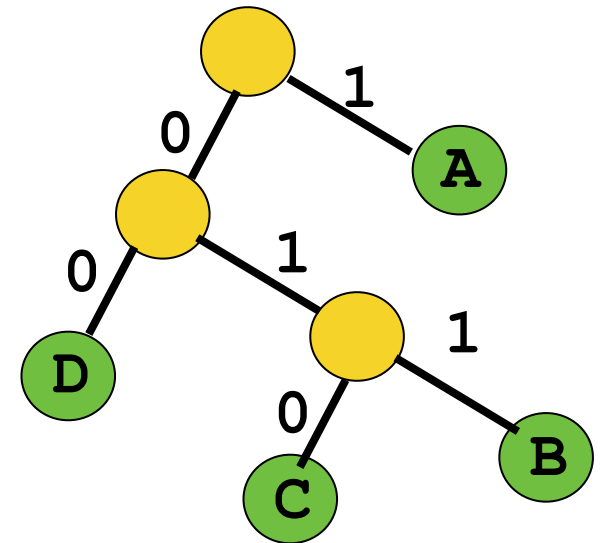
- Coding: assignment of bit strings to alphabets
- Codewords: bit strings assigned for characters of alphabet. Example:
  - if  $P(a) = 0.4$ ,  $P(b) = 0.3$ ,  $P(c) = 0.2$ ,  $P(d) = 0.1$
  - ASCII codes can be assigned as  
a:00, b:01, c:10, d:11  
Number of bits for each code is 2 (avg is 2 bits too)
  - Codes can be assigned as  
a:0, b:10, c:110, d:111
  - then the average length of this coding scheme is  
 $= 1*0.4 + 2*0.3 + 3*0.2 + 3*0.1 = 1.9$  bits

# What is a Coding Problem...

- Two kind of encodings:
  - Fixed encoding e.g. ASCII
  - Variable length encoding: Morse encoding (dots, dashes)
- Prefix free codes
  - No codeword is prefix of another code
  - Allows for efficient decoding.
- Problem: If the frequency of character occurrences are known, what is the best binary prefix code?
  - Best: Shortest average code length
  - Average code lengths represents expected number of bits required to transmit/store a character.

# Huffman Codes

- Any binary tree with edges labeled as 0, 1
  - Provides a prefix code for characters assigned to leaves
  - Just concatenate the label of edges on the path from root to a vertex
  - Example:
    - A: 1
    - B: 011
    - C: 010
    - D: 00
- Optimal binary tree can be constructed using Huffman's algorithm





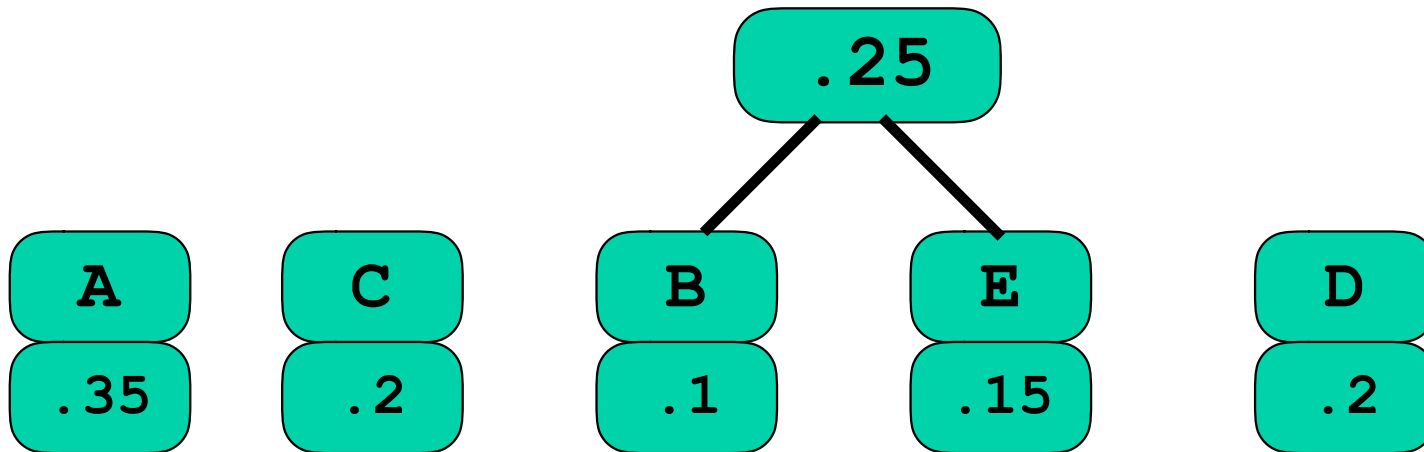
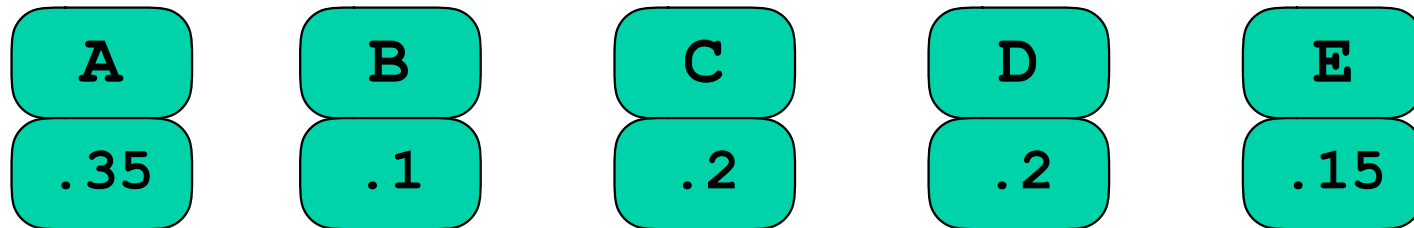
# Huffman's Algorithm

- Initialize  $n$  one node trees with alphabet characters
  - Assign tree weights as character frequencies
- Repeat the following steps  $n-1$  times
  - Join two binary trees with smallest weights into one binary tree
    - one tree would become left subtree
    - other tree would right sub-tree
  - Make the weight of new binary (after joining) as equal to sum of weights of its sub trees.
    - Mark the edge joining left subtree with label 0
    - Mark the edge joining right subtree with label 1

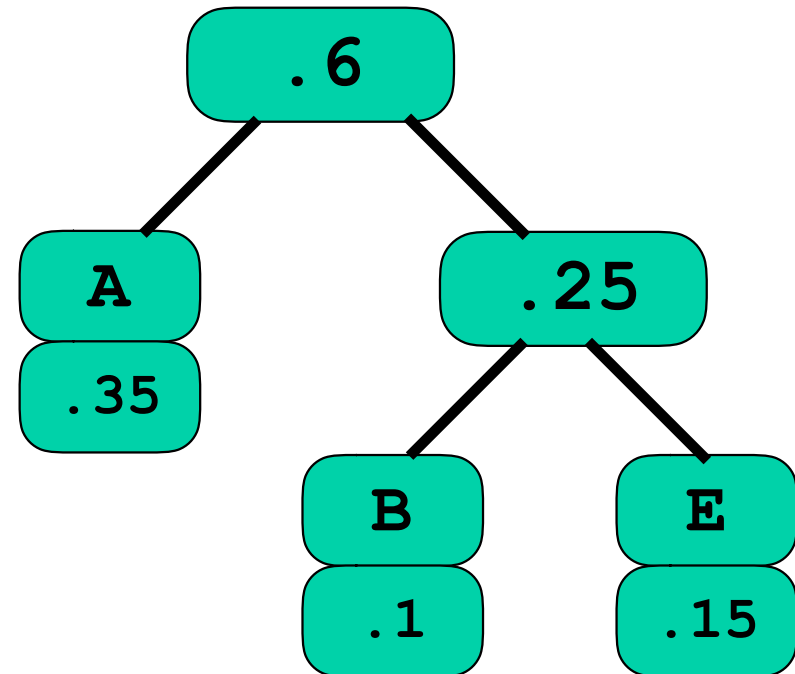
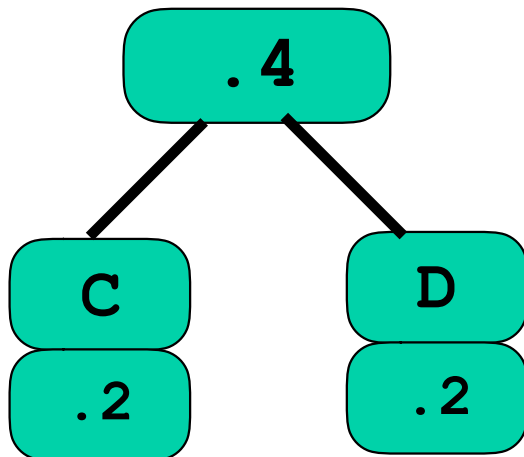
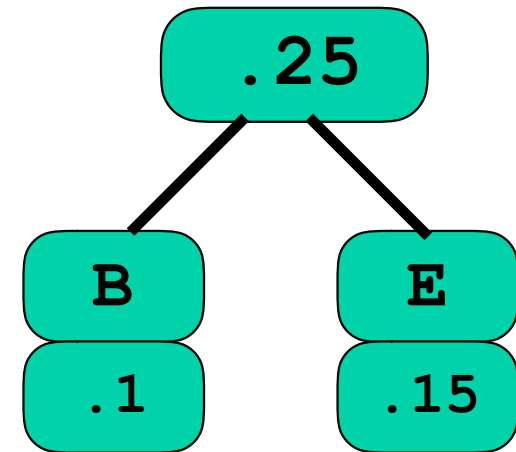
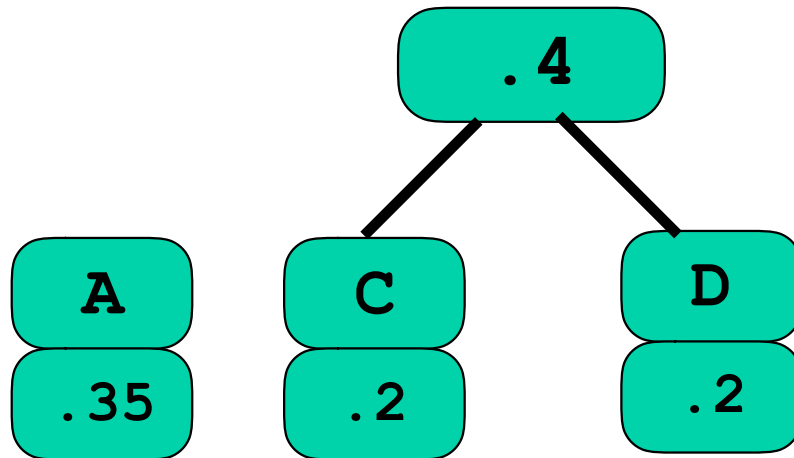
# Example: Huffman Tree

- Character frequencies (probabilities)

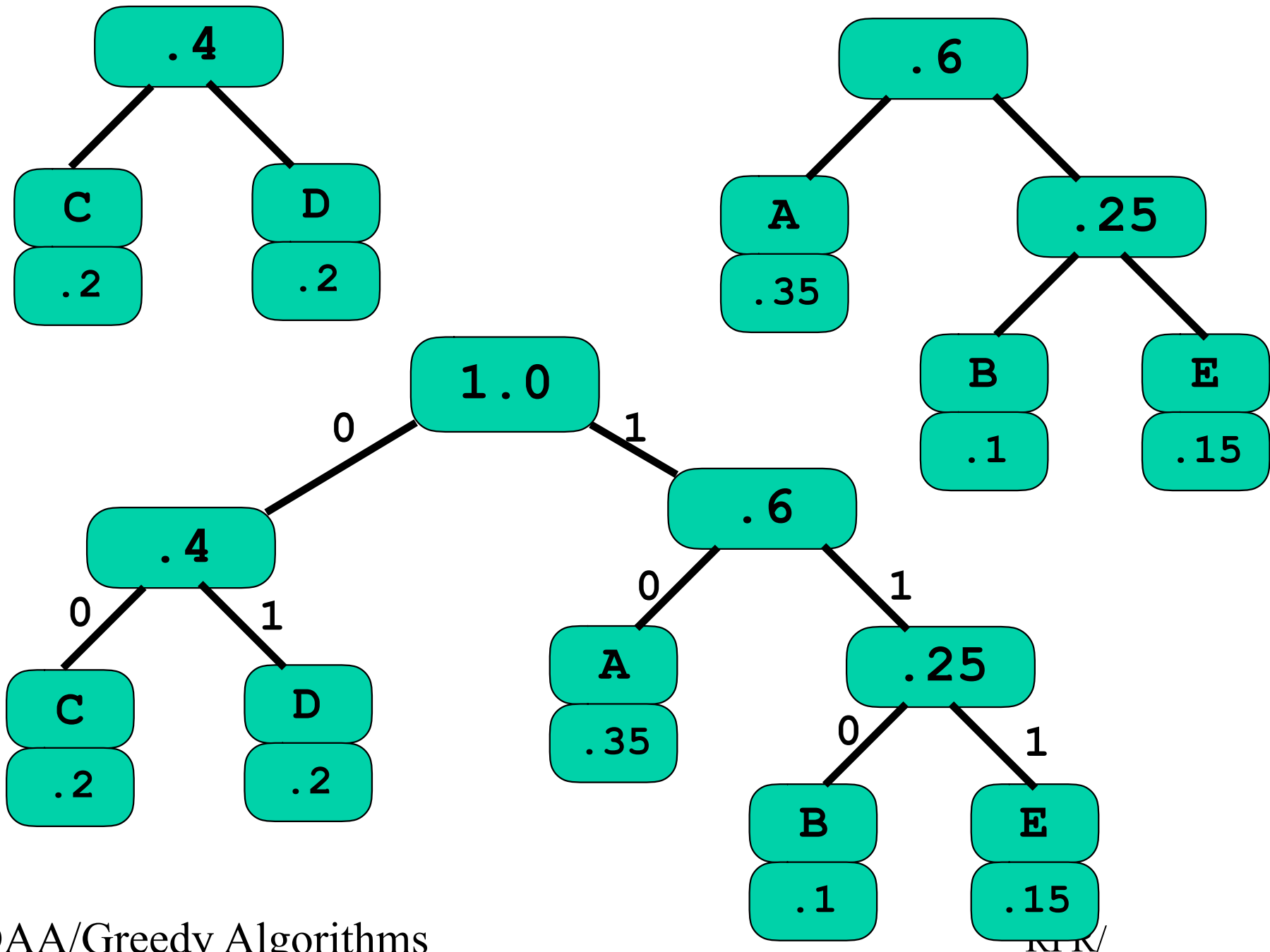
A:0.35, B:0.1, C:0.2, D:0.2, E:0.15



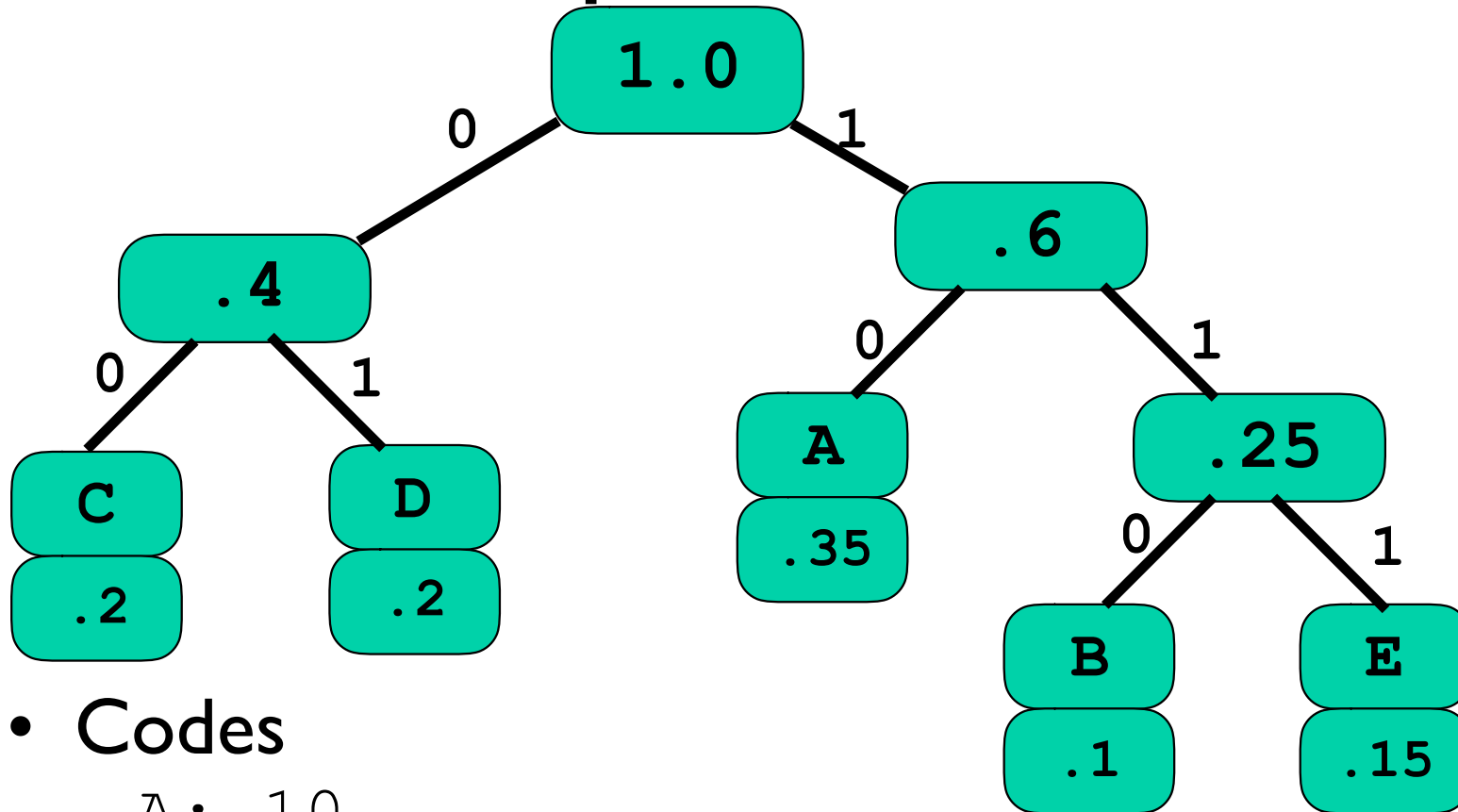
# Example: Huffman Tree



# Example: Huffman Tree



# Example: Huffman Tree



- Codes

A: 10

B: 110

C: 00

D: 01

E: 111

Note: Characters with

Higher freq. (prob.) have shorter codes

Shorter freq.(prob.) have longer codes.

# Example: Huffman Tree

- Character frequencies (probabilities)  
A:0.35, B:0.1, C:0.2, D:0.2, E:0.15
- Codes  
A: 10, B: 110, C: 00, D: 01, E: 111
- Average code length  
$$2 \cdot .35 + 3 \cdot .1 + 2 \cdot .2 + 2 \cdot .2 + 3 \cdot .15$$
$$= 0.70 + 0.3 + 0.4 + 0.4 + 0.45 = 2.25$$
- Code length for fixed length coding: 3
  - 5 characters, require 3 bits
- Compression ratio:
  - $(3.0 - 2.25) / 3.0 \cdot 100\% = 25\%$
- Q: Represent character sequence ACDBA

# Huffman Tree/Codes

- Some characteristics
  - Codewords of two least frequent characters is same
    - They are at same level
    - What happens if  $\neq$  more than 2 least frequent chars?
  - Codeword length of a more frequent character is always smaller than codeword length less frequent characters. Proof by contradiction.
  - If alphabet's frequency is sorted,
    - Huffman tree can be constructed in Linear time
  - The max length of a codeword in huffman encoding of  $n$  characters can be  $n-1$ . Consider when each frequency is different.

# Algorithm: Huffman Code

```
Algo Huffman (W [ 0 : n-1 ] )  
// i/p: an array W [ 0 : n-1 ] of weights  
// o/p: A Huffman tree with leaves having weights of W  
Initialize Priority Queue Q of size n with 1-node  
trees and weights equal to elements of W [ 0 : n-1 ]  
while Q has more than 1 element do  
    T1 ← minimum weight tree in Q  
    delete T1 from Q  
    T2 ← minimum weight tree in Q  
    delete T2 from Q  
    create a new tree T with T1 / T2 as left/right subtree of T  
        with weight(T) = weight(T1) + weight(T2)  
    Insert T into Q
```



# Complexity Analysis (General)

- In each iteration
  - Removing two tree
  - Adding one tree
  - Effectively reducing number of trees by 1.
  - Thus, total iteration =  $n-1$
  - Removing min weight node takes  $\lg n$
  - Thus, time complexity  $O(n \lg n)$

# Complexity Analysis (Improved)

1. Maintain priority  $Q$  as sorted array in ascending order of weights. (Assume it is given),
  - a. otherwise it takes  $O(n \cdot \lg n)$  time.
2. Maintain another array  $Q'$  of size 1 (initially empty)
3. Remove the first 2 elements from  $Q$ .
4. Merge these 2 into new element  $N$  and add  $N$  to  $Q'$
5. Now consider 2 elements from  $Q$  and 1 from  $Q'$
6. Remove the two with minimum weights and merge these to create new element  $N$ .
7. If one of two elements picked from  $Q'$ , add  $N$  to  $Q'$   
else add  $N$  to  $Q$ .
8. Repeat the steps 5, 6, 7 till only 1 tree is left.

Time Complexity:  $O(N)$  excluding the sorting of weights

# Summary

- Huffman codes
- Huffman tree
- Algo
- Complexity analysis