# CS 2336
# Discrete Mathematics

## Lecture 17

Trees: Optimal Prefix Code

# Outline

- Text Encoding Problem
- Prefix Code
- Optimal Prefix Code

# Encoding to Reduce Storage

- In ASCII, each English character is represented in the same number of bits (8 bits)
- This is called fixed-length encoding

  ➔ If a text contains n characters, it takes 8n bits in total to store the text in ASCII

- However, if our target is to reduce the storage, some better schemes can be designed

  ➔ This is what the tools, such as zip, 7zip, gzip, are targeting

# Encoding to Reduce Storage

- The reason why better schemes exist is as follows:

> In real-life English texts, characters do not appear with the same frequency

- If we can make a trade-off, so that
  1. frequent characters are encoded in fewer bits
  2. infrequent characters are encoded in more bits

  then we can reduce the total storage!
- This is called a variable-length encoding

# Example

- Suppose we have a 100K char file, with characters `A`, `B`, `C`, `D`, `E` only
  - `A` occurs 45K times, others each 11K times
- Using fixed-length :

  Each character in 3 bits ; Total = 300K
- Using variable-length :

  `A` → 0, `B` → 100, `C` → 101, `D` → 110, `E` → 111

  Total = 45K × 1 + 55K × 3 = 210K (30% savings!)

# Example

- Thinking a step ahead, we may consider the following “better” scheme :

  A → 0, B → 1, C → 00, D → 01, E → 010

- This scheme requires less storage, because each character is encoded in fewer bits

- What’s wrong with this encoding ?

# Prefix Code

A → 0, B → 1, C → 00, D → 01, E → 010

- Suppose the encoded text is : 0101
- We cannot tell if the original is

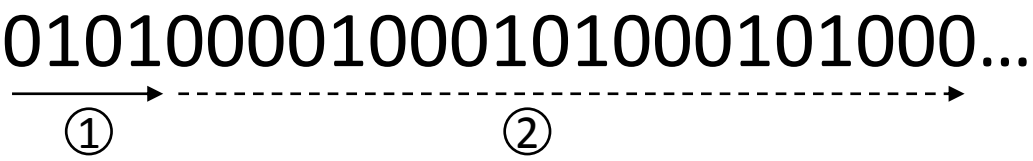
  ABAB or ABD or DAB or DD or EB

- The problem comes from

one codeword is a prefix of another

# Prefix Code

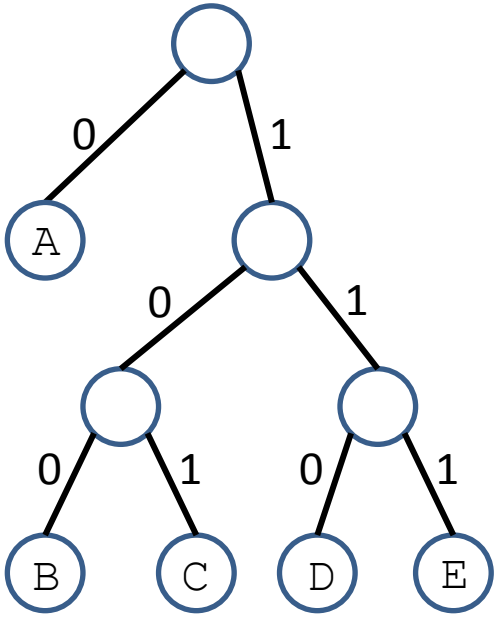
- To avoid the problem, we generally want that each codeword is NOT a prefix of another
- Such an encoding scheme is called a prefix code, or prefix-free code
- For a text encoded by a prefix code, we can easily decode it in the following way :




① Scan from left to right to extract the first code
② Recursively decode the remaining part

# Prefix Code Tree

- Naturally, a prefix code scheme corresponds to a prefix code tree
- The tree is a rooted, with
  1. each edge is labeled by a bit ;
  2. each leaf ➔ a character ;
  3. labels on root-to-leaf path ➔ codeword for the character
- E.g., A → 0, B → 100, C → 101, D → 110, E → 111



# Optimal Prefix Code

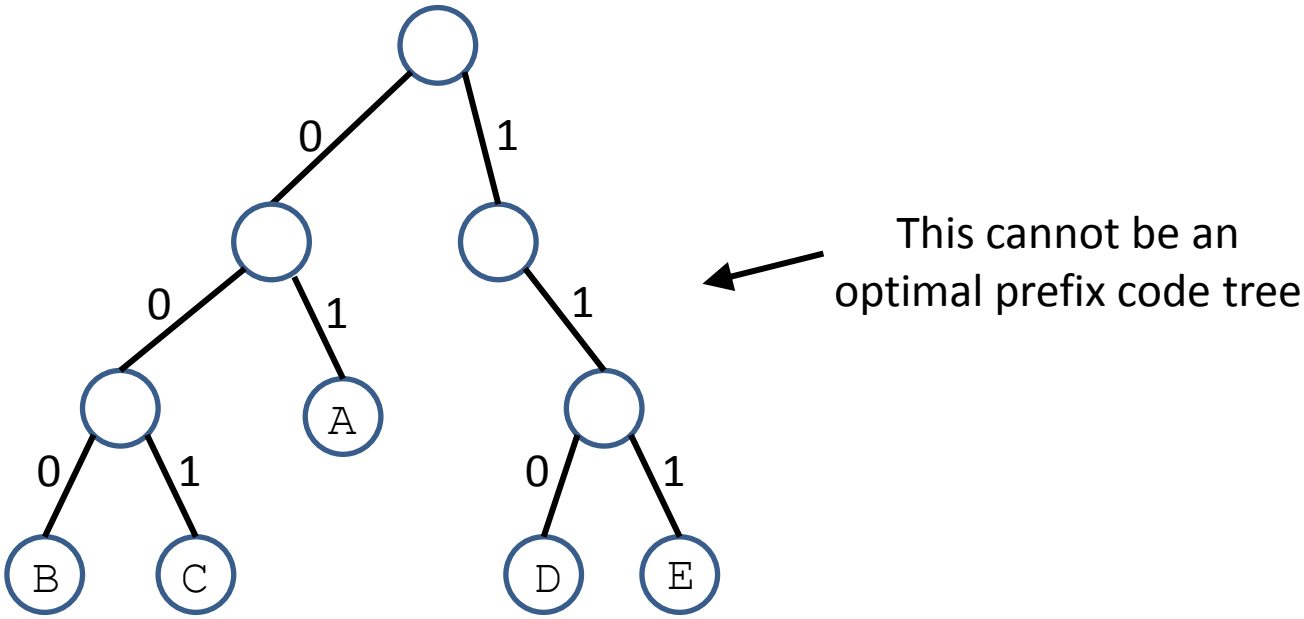
Problem : Given the frequencies of each character, design the optimal prefix code whose encoded text requires the least storage

- Equivalently, we want to find a prefix code tree that corresponds to an optimal prefix code

  What do we know about the tree ?

# Optimal Prefix Code

Observation 1 : In an optimal prefix code tree, each internal node must have two children



# Optimal Prefix Code

Observation 2 : There is an optimal prefix code tree, such that the leaves corresponding to the two least frequent characters are siblings, and the leaves are farthest from the root.

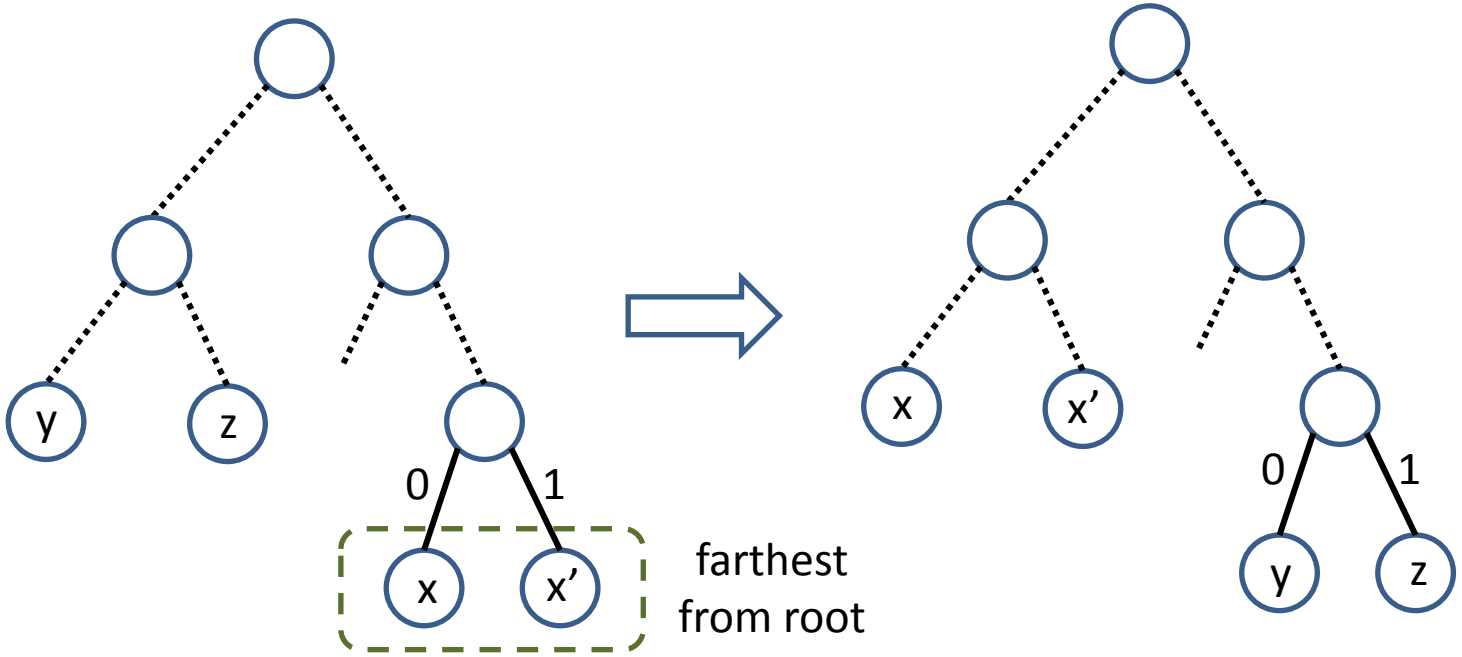
- Proof : Consider an optimal prefix code tree.

  Let y and z be the least frequent characters ;

  Let x be a character whose leaf is farthest from the root (its sibling must be a leaf for some char x')

# Optimal Prefix Code

- Then, we can obtain a desired tree, as follows :




optimal prefix code tree

as good as optimal

# Optimal Prefix Code

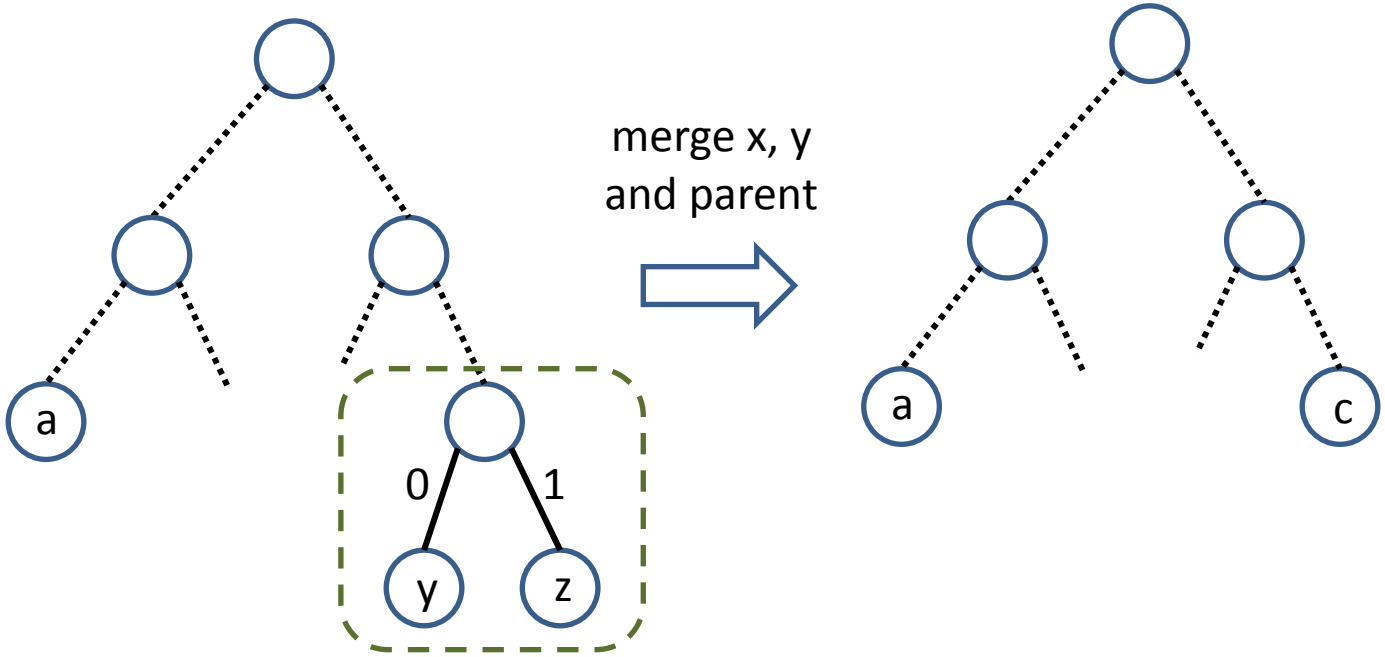
- Let y and z be the two least frequent characters

  Let T be an optimal tree such that y and z are sibling leaves and farthest from the root
- Form a new text as follows : Replace each y and z by a common character c in the original text

Observation 3: If we merge x, y, and their parent into a leaf in T, and correspond this leaf to c, we get an optimal prefix code tree for the new text

# Optimal Prefix Code

- Graphically, the observation says :




optimal prefix code tree
for original text

optimal prefix code tree
for new text

# Optimal Prefix Code

- Based on the previous observations, we get a way (discovered by David Huffman in 1952) to obtain an optimal prefix code :

1. Find the least frequent characters x and y
2. Form two leaves for x and y, and join them with a common parent p
3. Replace x and y by a common character c
4. Recursively find the optimal prefix code tree for the new text (and replace the leaf for c with p, x, y)

# Example

- Suppose the relative frequencies are as follows :

A : 40,  B : 20,  C : 15,  D : 50,  E : 25