Huffman Encoding

Week 2 Min-Yen KAN

(Self-Study Module)

Fixed and variable bit widths

- To encode English text, we need 26 lower case letters, 26 upper case letters, and a handful of punctuation
- We can get by with 64 characters (6 bits) in all
- Each character is therefore 6 bits wide
- We can do better, provided:
  - Some characters are more frequent than others
  - Characters may be different bit widths, so that for example, **e** use only one or two bits, while **x** uses several
  - We have a way of decoding the bit stream
    - Must tell where each character begins and ends

Example Huffman encoding

A = 0
B = 100
C = 1010
D = 1011
R = 11

ABRACADABRA = 01001101010010110100110

- This is eleven letters in 23 bits
- A fixed-width encoding would require 3 bits for five different letters, or 33 bits for 11 letters
- Notice that the encoded bit string *can* be decoded!

Why it works

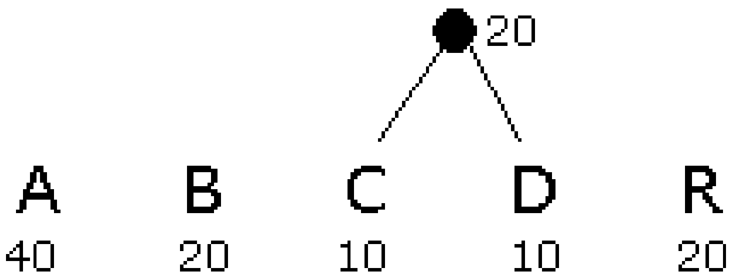
- In this example, A was the most common letter
- In ABRACADABRA:
  - 5 As code for A is 1 bit long
  - 2 Rs code for R is 2 bits long
  - 2 Bs code for B is 3 bits long
  - 1 C code for C is 4 bits long
  - 1 D code for D is 4 bits long

Creating a Huffman encoding

- For each encoding unit (letter, in this example), associate a frequency (number of times it occurs)
  - You can also use a percentage or a probability
- Create a binary tree whose children are the encoding units with the smallest frequencies
  - The frequency of the root is the sum of the frequencies of the leaves
- Repeat this procedure until all the encoding units are in the binary tree

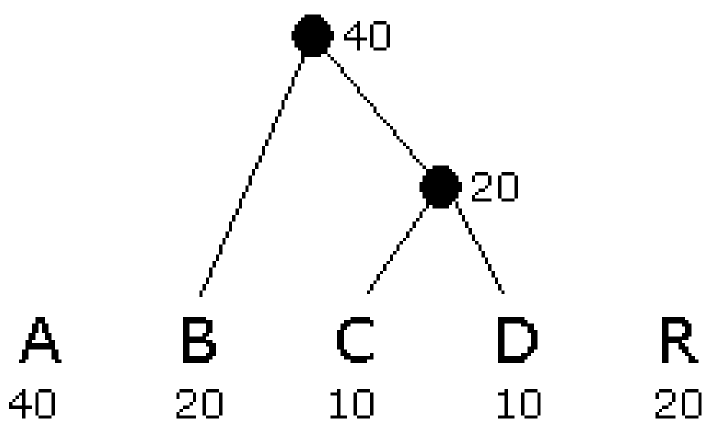
Example, step I

- Assume that relative frequencies are:
  - A: 40
  - B: 20
  - C: 10
  - D: 10
  - R: 20
- Smallest number are 10 and 10 (**C** and **D**),
  - connect those



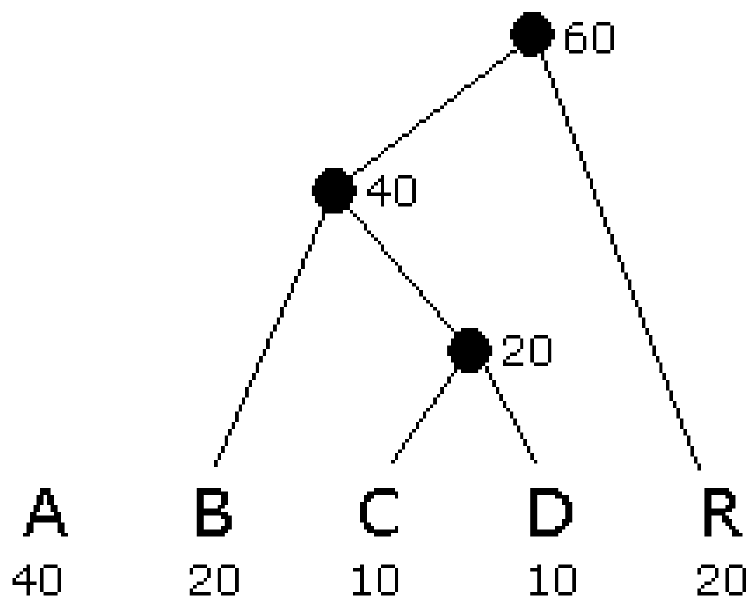
Example, step II

- C and D have already been used, and the new node above them (call it **C+D**) has value 20
- The smallest values are **B**, **C+D**, and **R**, all of which have value 20
  - Connect any two of these



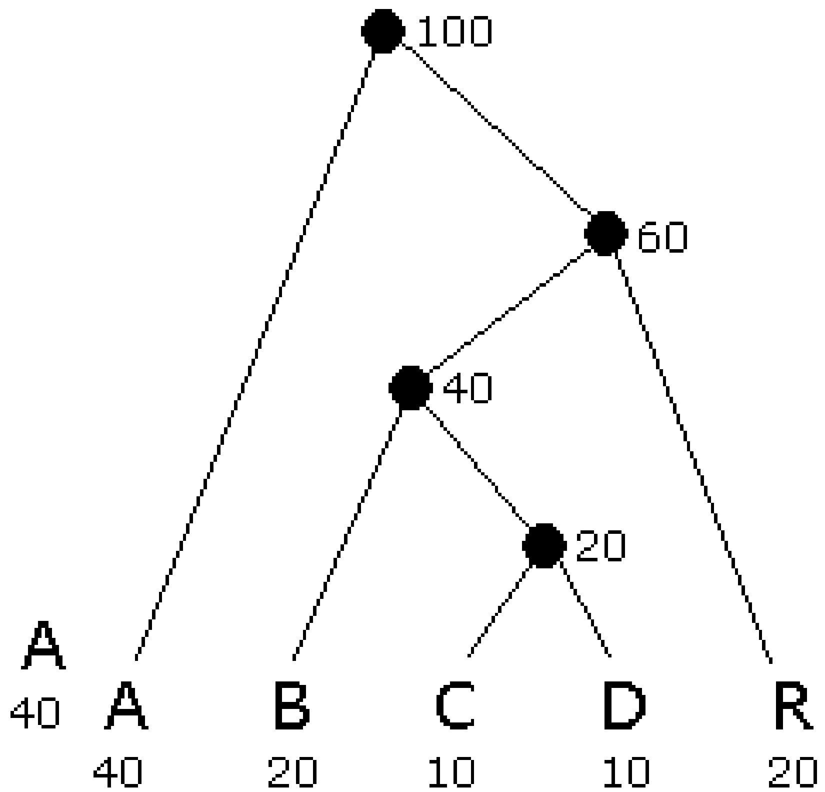
Example, step III

- The smallest values is **R**, while **A** and **B+C+D** all have value 40
- Connect **R** to either of the others



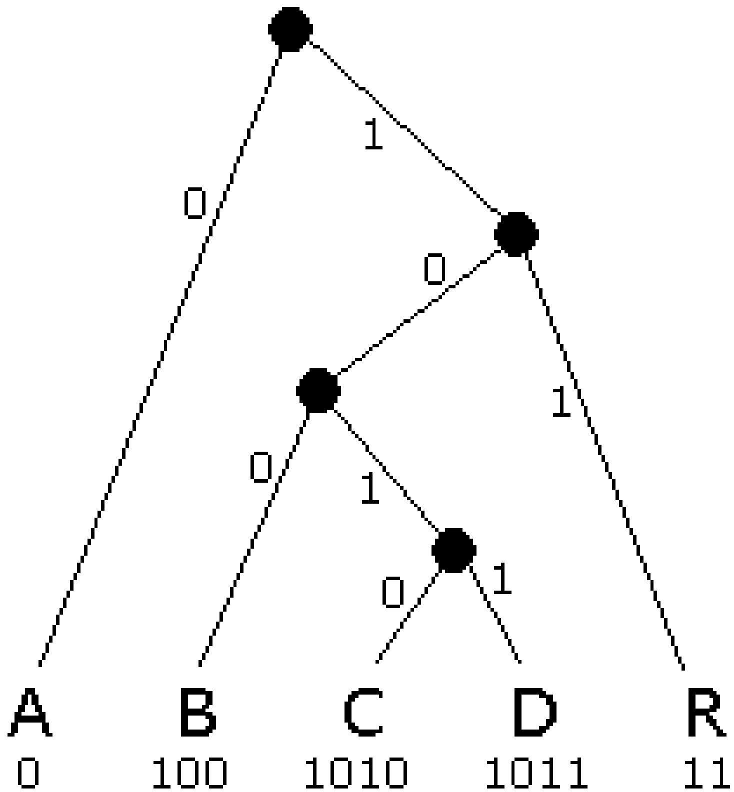
Example, step IV

- Connect the final two nodes



Example, step V

- Assign 0 to left branches, 1 to right branches
- Each encoding is a path from the root



- **A = 0**
  B = 100
  C = 1010
  D = 1011
  R = 11
- Each path terminates at a leaf
- Do you see why encoded strings are decodable?

Unique prefix property

- **A = 0** **B = 100**
  C = 1010 **D = 1011**
  R = 11
- No bit string is a prefix of any other bit string
- For example, if we added E=01, then A (0) would be a prefix of E
- Similarly, if we added F=10, then it would be a prefix of three other encodings (B=100, C=1010, and D=1011)
- The unique prefix property holds because, in a binary tree, a leaf is not on a path to any other node

Practical considerations

- Is encoding practical for long texts or short ones?
  - Short: impractical
    - To decode it, you would need the code table
    - The code table is bigger than the message
  - Long: practical
    - The encoded string is large relative to the code table

Question: What about if we agree on code table beforehand?