

# Python 3 Bits & Bytes, File Compression

Eddie Guo

September 2019

## 1 Introduction to File Compression

### 1.1 Topics Covered

- (i) Data as bits
- (ii) Compression
- (iii) Huffman coding

### 1.2 Bit

- Bit is basic unit of mem in computer
- 8 bits = 1 byte
- 4 bits = 1 nibble
- 2 bits = 1 crumb

### 1.3 Characters in Text Files

- ASCII table defs binary rep of 256 ( $2^8$ ) diff chars
- Each char has 8 bit representation: 1 byte
- ASCII table is w/in Unicode table

### 1.4 Always Need All Those Bits?

- If 1 char = 1 byte, text file w/ 1000 chars contains ~1kB of data
- What if text file only contains chars 'a' & 'b'?
  - Woudn't need full byte → just use single bit: 'a' = 0, 'b' = 1
  - Using this approach, 1000 a's & b's would take up 1/8 space of normal ASCII rep

## 2 Data Compression

### 2.1 Compression Intro

- Basic goal of compression: rep file using fewer bits, even if we have to store contents in unconventional format
- Pros: use less mem to store file, transmit files faster

### 2.2 Another Example

#### 2.2.1 Method 1

- What if file has chars 'a', 'b', 'n' (ex: banana)?
- Can use 'a' = 00, 'b' = 01, 'c' = 10 → banana = 010010001000
- This gives compression rate of 1/4 that of plain-text file

#### 2.2.2 Better Compression Choice?

- Is there better way to compress text file w/ only 3 chars? YES!
- Assoc most freq character w/ 0, and remaining w/ 10 & 11
- Compression rate:
  - at least 1/3 of chars go from 8 bits to 1 bit
  - Remaining chars go from 8 bits to 2 bits

### 2.3 Calculate Compression Rate

- $n_i$  = # times 'i' occurs (where  $i = a, b, c$ )
- $n = n_a + n_b + n_n$
- Suppose 'a' is most freq. Then  $n_a \geq n/3$
- Amount of bits used in compressed string is:
$$\begin{aligned} \text{bits} &= 1 * n_a + 2 * n_b + 3 * n_n \\ &= 2 * n - n_a \\ &\leq 2 * n - n/3 \\ &\leq 5/3 n \end{aligned}$$
- # bytes is  $(5/3n)/8 = 0.2083$  or less
- Better than 0.25n we got b4 when all 3 chars were 2 bit seq

## 2.4 Decoding

- Can we decode compressed file when some chars encoded w/ 1 bit, others w/ 2 bits?
- Decode = 10010011

- 'a' = 0, 'b' = 10, 'n' = 11



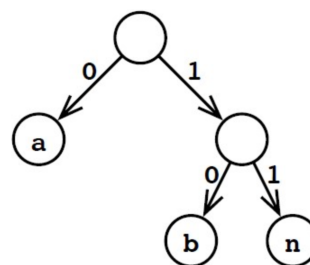
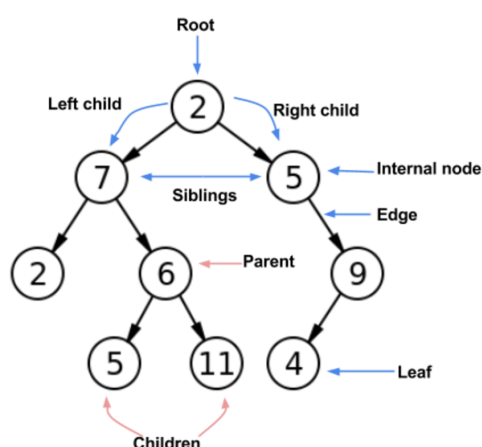
- b a b a n

## 3 Decoding Tree

### 3.1 Intro to Decoding Tree

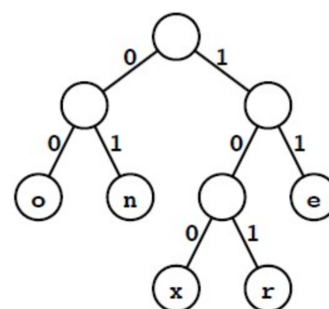
- If no bit seq is beginning of another in encoding, we can build decoding tree
- 0/1 labels on edges of root-to-leaf path = encoding of char in given leaf

- All chars are leaf nodes, all edges are 0/1
- If node not leaf, then it's an internal node
- Binary trees have at MOST 2 children



### 3.2 How to Use Decoding Tree

- Decode bit seq using bits to traverse given tree
  1. Start at root, follow 0/1 edge according to next bit in seq
  2. Output char at leaf when you reach one
  3. Return to root & repeat for next branch
- 001001101 → 00 (o) + 100 (x) + 11 (e) + 01 (n) = oxen



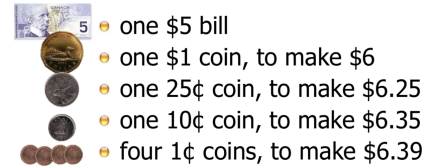
### 3.3 Build a Decoding Tree

- The key is picking encoding for each char
- **Req:** no bit seq for any char is the beginning (prefix) of another bit seq → this type of encoding scheme called **prefix code**
- **Desire:** chars that occur more freq should have shorter bit seqs

- **Optimization problem:** construct decoding tree to minimize total # of bits used to compress file
- This can be achieved using **Huffman Trees** ~ trees constructed according to simple **greedy procedure**

### 3.4 Greedy Algorithms

- At each step:
  - Take best step we can get right now, w/o regard to eventual optimization
  - Hope that by choosing local optimum at each step, you'll end up at global optimum
- Ex: count \$6.39, using fewest bills & coins
- Greedy algorithm: at each step, take largest bill/coin that does not overshoot



## 4 Huffman Coding

### 4.1 Building a Huffman Tree

1. Do freq count of all chars in file (include count of 1 for EOF sentinel)
  - (a) Ex:  $\text{freq}['a'] = 10$ ,  $\text{freq}['w'] = 2$ ,  $\text{freq}[\text{EOF}] = 1$
  - (b) Ultimately, keys will be bytes, not chars
  - (c) Total freq count of tree is sum of freqs of its leaves
2. Initially, each char is single node in trivial Huffman tree
3. While there is more than 1 tree, merge the 2 w/ the smallest freq counts
  - (a) Make each tree a child of a new root node
  - (b) Doesn't matter which tree is left/right child
  - (c) # on this new tree is total freq count of all leaves
  - (d) Repeat: pick 2 trees w/ lowest total freq count, & merge (in case of tie, doesn't matter which one you pick)
4. Merge trees  $T_1$  &  $T_2$  means creating a new root node & setting  $T_1$  &  $T_2$  as its children

### 4.2 Summary: Compress the File

- For each char, det its 0/1 compression encoding by looking at the root-to-leaf path
- Output the seq of 0/1 bits obtained by replacing the char w/ its compressed bits
- Don't forget the final seq for EOF sentinel

### 4.3 Summary: Decompress the file

- Starting from root, traverse Huffman tree. Each bit from input seq dictates when to go L/R
- When you reach a leaf, output the char, return to root, continue traversing tree according to next bit(s) in seq
- Quit when you reach the EOF leaf

### 4.4 Why Include an EOF?

- B/c last byte of compressed file might not be "complete" (ex: 35 bits in compression seq: 4 bytes & 3 bits, so EOF pads w/ 0); need multiple of 8 bits to transmit file
- $\therefore$ , decoding EOF tells us when to stop

### 4.5 Considerations

- To decompress a file using this approach, need to know struc of Huffman tree used to compress file in 1st place
- When we send compressed file, also need to send rep of Huffman tree used

### 4.6 Final Notes

- Huffman compression exploits freqs of chars  $\rightarrow$  fairly simple compression scheme, but results in reduced file sizes if file contains **subset** of the  $2^8$  diff bytes (chars); **no point in using Huffman tree if there are 256 diff chars in file**
- Huffman compression tends to work best on plain text files & bitmap images w/ small range of colours
- Other compression schemes exploit other patterns, & often target spec file types (pics, text, etc.). Some compressions allow for some data loss (ex: w/ .jpeg files)
- **No compression can make every file smaller**