Huffman Encoding

- Used for compression
- Strings of fixed alphabet
- Create encoding for each letter
- Encodings are prefix-free
- Encodings are variable length
- Most frequent letters have shortest encodings

Example:
- Alphabet: \{A, B, C, D\}
- Frequencies: \{A: 7\%, B: 15\%, C: 31\%, D: 47\%\}
- String Length: 1,000,000 = 10^6

Huffman Encoding

Example: \{A: 7\%, B: 15\%, C: 31\%, D: 47\%\} 10^6 letters

ASCII (non-Huffman): 7 bits per letter
encoded size = 7 \cdot 10^6 bits

2-bit (non-Huffman): 4 letters, need 4 unique encodings, only need 2 bits
A = 00, B = 01, C = 10, D = 11
encoded size = 2 \cdot 10^6 bits
Huffman Encoding Greedy

Example: \{A: 7\%, B: 15\%, C: 31\%, D: 47\%\}

10^6 letters

Additional Possible Minimal Encodings

```
D          D
  O 1       O 1
 C   0      C   0
 A B       B
```

How many total?
Why are they all minimal?

Huffman Encoding Greedy

Encoding Tree Structure
- Letters are in leaf nodes
- Least frequent letters are deepest
- Left branch is '0', right is '1'
- Encoding for letter made from path from root to leaf
- Encoding is prefix-free because no internal node contains a letter

Huffman Encoding Greedy

Algorithm
```python
def Huffman(letters, freq):
    h = heap()  # ordered by freq
    for l in letters:
        h.add(l)
    while h.size() > 1:
        i = h.pop(); j = h.pop()
        n = (i, j); freq[n] = freq[i] + freq[j]
        h.add(n)
    tree = h.pop()
```