Greedy Algorithms:
Text Compression

Brute-force Algorithms
Def'n: Solves a problem in the most simple, direct, or obvious way
- Not distinguished by structure or form
- Pros
  - Often simple to implement
- Cons
  - May do more work than necessary
  - May be efficient (but typically is not)

Greedy Algorithms
Def'n: Algorithm that makes sequence of decisions, and never reconsiders decisions that have been made
- Pros
  - May run significantly faster than brute-force
- Cons
  - May not lead to correct/optimal solution
Text Compression

- **X**: text string
  - of length \( n \)
  - with \( m \) unique characters
  - defined over some alphabet \( \Sigma \)
    - may be ASCII (7 bits) or Unicode (16 bits)
- **D**: set of unique characters in \( X \)
  - \( \{d_1, d_2, ..., d_m\} \)
  - \( |D| = m \)
  - note difference in notation with book

Question:
- Can \( X \) be encoded into a binary string \( Y \), such that \( Y \) has the minimum number of bits?

Answer:
- need mapping: \( F(X) \Rightarrow Y \)
- and decoding: \( F^{-1}(Y) \Rightarrow X \)

Text Compression:

- Huffman Encoding

  Encode text string \( Y \) based upon frequency of occurrence of each character in the text string \( X \)
  - short encoding for high-frequency char
  - long encoding for low-frequency char
  - no char code has prefix of another char code
    - need to know where one char ends and next begins
Huffman Encoding Intuition

- Determine frequency \( f(d_i) \) of each unique character \( d_i \) in the text string \( X \).
- Create external nodes for each \( d_i \) that contain \( d_i \) and \( f(d_i) \).
- Iteratively, until only one tree remaining
  - combine two trees with smallest frequencies
  - set \( f(\text{newroot}) = f(\text{leftchild}) + f(\text{rightchild}) \)
- Determine encoding by following path
  - \( \text{leftchild} \Rightarrow 0 \)
  - \( \text{rightchild} \Rightarrow 1 \)

Pop Quiz

- You are given a list of items with varying frequencies
  - need to repeatedly choose two that currently have lowest frequency
  - need to repeatedly place with sum of above back into list
- How to implement?

Huffman: Pseudocode

Algorithm Huffman \((X)\)

Input: character string \( X \) of length \( n \) with \( m \) distinct characters
Output: Huffman encoding tree for \( X \)
Compute \( f(d_i) \) for each \( d_i \in D \)
Initialize priority queue \( Q \)
for each \( d_i \in D \) do
  Create single node binary tree \( T \)
  Insert \( T \) into \( Q \) with key \( f(d_i) \)
Huffman: Pseudocode (con’t)

while Q.size() > 1 do
    f1 <- Q.minKey()
    T1 <- Q.removeMin()
    f2 <- Q.minKey()
    T2 <- Q.removeMin()
    Create new binary tree T from T1&T2
    Insert T into Q with key f1 + f2
return tree Q.removeMin()

Summary: Brute & Greedy

- **Brute-force:**
  - solve problem in simplest way
  - generate entire solution set, pick best
  - will give optimal solution with (typically) poor efficiency

- **Greedy:**
  - make local, best decision, and don’t look back
  - may give optimal solution with (typically) ‘better’ efficiency
  - depends upon ‘greedy-choice property’
    - global optimum found by series of local optimum choices