**CS430 Lecture 17 Activities**

Fractional Knapsack Problem

n items wi weight vi values Total weight limit = W

Maximize value Total weight ≤ W Fractional amounts of items are allowed

1. Prove that the Fractional Knapsack Problem has optimal substructure.

2. Try various “common sense” greedy approaches that divide the problem into a sub-problem(s) and try to come up with counter-examples or prove the greedy choice is correct using the “cut and paste” proof.

Huffman Codes Problem

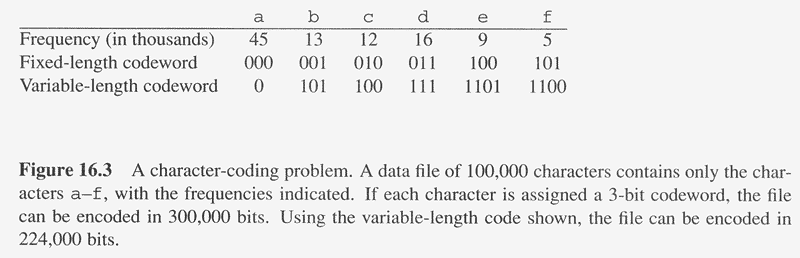
Data Encoding Background

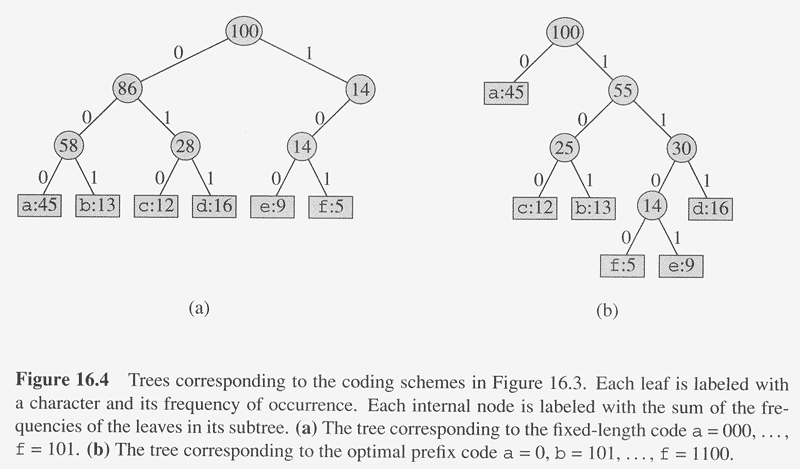
* Data is a sequence of characters
* Fixed Length – Each character is represented by a unique binary string. Easy to encode, concatenate the codes together. Easy to decode, break off 3-bit codewords and decode each one.
* Variable Length – Give frequent characters shorter codewords, infrequent characters get long codewords. However, how do we decode if the length of the codewords are variable?
* Prefix Codes - No codeword is a prefix of another codeword. Easy to encode, concatenate the codes together. Easy to decode, since no codeword is a prefix of another, strip off one bit a time and match to unique prefix code

Huffman Codes

* Are a Data Compression technique using a greedy algorithm to construct an optimal variable length prefix code
* Use frequency of occurrence of characters to build an optimal way to represent each character as a binary string
* Use a Binary tree method – 0 means go to left child, 1 means go to right child (not a binary search tree).
* Cost of Tree in bits B(T) = ∑ freq(c) \* depth(c)  
    for all c∈C

Example





3. Prove that the Huffman Codes Problem has optimal substructure.

The greedy approach that works is: Build the tree bottom up by using a minimum priority queue to merge 2 least frequent objects (objects are leaf nodes or other subtrees) together into new subtree.

|  |  |
| --- | --- |
|  |  |

Huffman <http://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html>

4. Proof this greedy approach leads to an optimal Huffman Code tree: Build the tree bottom up by using a minimum priority queue to merge 2 least frequent objects (objects are leaf nodes or other subtrees) together into new subtree.