Huffman Coding - A Greedy Algorithm

Slides based on Kevin Wayne / Pearson-Addison Wesley

Greedy Algorithms

Greedy Algorithms

- Build up solutions in small steps
- Make local decisions
- Previous decisions are never reconsidered

- Greedy algorithms may be optimal, they may not be.

Huffman Coding

The Task at Hand

- Encoding symbols using bits

- Suppose we want to represent symbols using a computer
  - Letters of the English alphabet (more broadly Unicode characters)
  - Pixels of an image
  - Audio information from a sound clip

- Ultimately these are converted to bits

- How do we represent these symbols?
  - How do we do it efficiently – using the fewest number of bits?
  - And make sure that we can decode the bits to recover our original symbols
  - This is a fundamental aspect of data compression

Huffman Coding

Types of Codes that can be used to represent symbols:

- Fixed length codes
  - Every symbol is encoded using an equal number of bits
  - E.g. ASCII code
    - 256 symbols
    - Encoded using 8 bits
    - A = 01000001
    - ! = 00100001
    - 8 = 00111000
    - q = 01110001

- Decoding
  - Simply chop the data into blocks of 8 bits each and decode one symbol for each block
  - Ease of decoding is an advantage of fixed length codes

Huffman Coding

Types of Codes that can be used to represent symbols:

- Fixed length codes
  - Every symbol is encoded using an equal number of bits
  - E.g. color image pixels – most consumer cameras generate 24-bit color images
    - 8 bits for Red
    - 8 bits for Green
    - 8 bits for Blue
    - If an 8-Megapixel image isn't compressed, it requires 24 Megabytes for storage!

- Question – how can we encode symbols more efficiently?
  - We'll define efficiency as the average number of bits needed per symbol

Huffman Coding

Consider Encoding English Text using Fixed Length Codes

- Suppose we restrict ourselves to just the 26 uppercase letters, plus 6 extra characters (space, comma, period, !, ?, ').
- Using a fixed length code:
  - 5 bits per symbol

- Is this the most efficient way to represent these symbols?

- Why Not?
  - It assumes that their frequency of occurrence is uniform.

- What might we try instead?
  - We'd conceivably do better if we could somehow assign shorter codewords to frequently occurring symbols at the expense of longer codewords for infrequently occurring symbols

Letter Probabilities in the English Alphabet

Symbol	Probability	Symbol	Probability
Space	0.1859	N	0.0574
A	0.0642	O	0.0632
B	0.0127	P	0.0152
C	0.0218	Q	0.0008
D	0.0317	R	0.0484
E	0.1031	S	0.0514
F	0.0208	T	0.0796
G	0.0152	U	0.0228
H	0.0467	V	0.0083
I	0.0575	W	0.0175
J	0.0008	X	0.0013
K	0.0049	Y	0.0164
L	0.0321	Z	0.0005
M	0.0198		

N. Abramson, *Information Theory and Coding,* McGraw Hill, 1963.

Huffman Coding

Variable Length Codes

- Assign shorter codewords to more frequent symbols, longer codewords to less frequent symbols

Symbol	Probability	Code I	Code II	Code III
A	0.60	00	0	0
B	0.30	01	1	10
C	0.05	10	10	110
D	0.05	11	11	111

- What's the problem with Code II?
  - Suppose we receive the encoding: 11011
  - Is that "BBABB" or "BCD" or "DAD" or …
  - Code II is not ***uniquely decodable***

Huffman Coding

Variable Length Codes

- Assign shorter codewords to more frequent symbols, longer codewords to less frequent symbols

Symbol	Probability	Code I	Code II	Code III
A	0.60	00	0	0
B	0.30	01	1	10
C	0.05	10	10	110
D	0.05	11	11	111

- Code III is a ***prefix-free*** variable length code
  - (Somewhat confusingly, we will refer to this as a ***prefix code***)
  - No codeword is a prefix of any other codeword
  - Prefix codes are uniquely decodable

Huffman Coding

Variable Length Codes

- Assign shorter codewords to more frequent symbols, longer codewords to less frequent symbols

Symbol	Probability	Code I	Code II	Code III
A	0.60	00	0	0
B	0.30	01	1	10
C	0.05	10	10	110
D	0.05	11	11	111

- What is the efficiency of fixed length (Code I) versus prefix-code III
  - Fixed length requires 2 bits per symbol
  - Code III requires .6*1 + .3*2 + .1*3 = 1.5 bits per symbol

Huffman Coding

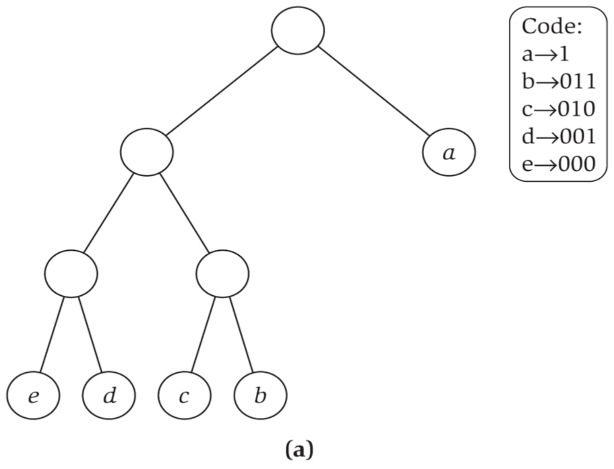
Optimal Prefix Codes

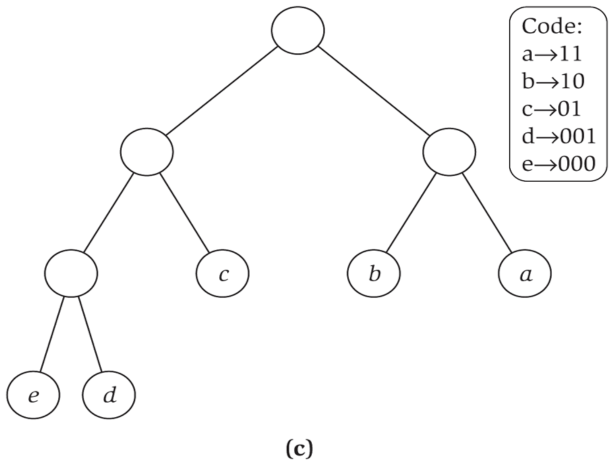
- Among all possible prefix codes, can we devise an algorithm that will give us an optimal prefix code?
  - One that most efficiently encodes the symbols with the lowest average bits per symbol

- Huffman codes are optimal prefix codes

Huffman Coding

Representing Prefix Codes as Binary Trees





- Each leaf node represents a symbol
- Each path represents an encoding for that symbol
  - Traveling to the left child is a '0'
  - Traveling to the right child is a '1'
- Why is it a prefix code?
  - For a symbol to a prefix of another, it would have to be a non-leaf

Figure 4.16 Parts (a), (b), and (c) of the figure depict three different prefix codes for the alphabet $S = \{a, b, c, d, e\}$.

Huffman Coding

The Huffman Coding Algorithm Generates a Prefix Code (a binary tree)

- Overall idea – bottom-up approach:
  - Start with all symbols as leaf nodes
  - Associate with each symbol its frequency of occurrence
  - REPEAT until only one symbol remaining
    - ✏ Select the two least frequently occurring symbols (ties can be broken arbitrarily) and link them together as children of a new common parent symbol
    - ✏ Associate with this new parent symbol the combined frequency of the two children
    - ✏ Remove the children from the collection of symbols being considered and replace with the new parent symbol

- Each step introduces a new parent symbol but removes two children
- Each step is building the tree from the leaves up to the root (when there is just one symbol – the root – the algorithm terminates)

Huffman Coding

The Huffman Coding Algorithm Generates a Prefix Code (a binary tree)

- Codewords for each symbol are generated by traversing from the root of the tree to the leaves
- Each traversal to a left child corresponds to a '0'
- Each traversal to a right child corresponds to a '1'

```
Huffman ( [$a_1$,$f_1$],[$a_2$,$f_2$],…,[$a_n$,$f_n$])
1. if n=1 then
2.     code[$a_1$] ← “”
3. else
4.     let $f_i$,$f_j$ be the 2 smallest f's
5.     Huffman ( [$a_i$,$f_i$+$f_j$],$\underbrace{[a_1,f_1],\ldots,[a_n,f_n]}_{\text{omits } a_i, a_j}$ )
6.     code[$a_j$] ← code[$a_i$] + “0”
7.     code[$a_i$] ← code[$a_i$] + “1”
```

Huffman Coding

The Huffman Coding Algorithm is a Greedy Algorithm

- At each step it makes a local decision to combine the two lowest frequency symbols

Complexity

- Assuming n symbols to start with
- Requires O(n) to identify the two smallest frequencies
- $T(n) \leq T(n-1) + dn$
  - $O(n^2)$

- Can we do better?
  - Consider storing frequencies in a heap
  - O(n log n) to build the heap
  - Each iteration requires two extract min operation (log n each) and one add new element (log n).
    - ✎ O(n log n) to generate the Huffman codes

Properties of Optimal Prefix Codes

Lemma 1: Let x, y be symbols with $f_x > f_y$. Then in an optimal prefix code, $\text{length}(C_x) \le \text{length}(C_y)$

- If x occurs more often than y, its codeword must be no longer than the codeword for y in an optimal prefix code

Lemma 2: If w is a longest codeword in an optimal prefix code, there must be another codeword with equal length

Lemma 3: Let x, y be symbols with the smallest frequencies. Then there exists an optimal prefix code in which they differ only in the final bit

Theorem: The prefix code output by the Huffman algorithm is optimal.

- Can be shown by induction. See book (page 174-175) for proof.

Extensions and Drawbacks of Huffman Codes

Each codeword is an integer number of bits

- Suppose we are encoding a sequence containing just two symbols: x and y, where x has a frequency of 99% and y has a frequency of 1%.
- The Huffman code will use one bit to represent x, and one bit to represent y
- But we know most of the time the symbol is x. Can we do better?
  - Arithmetic coding can be used to approach an average bits per symbol equal to the ***entropy*** of the data. Entropy represents the amount of information contained in the data.
  - If $f_x = .5$ and $f_y = .5$, then the encoding of one symbol carries one bit of information. The entropy for that distribution is one bit per symbol
  - If $f_x = 1$ and $f_y = 0$, then the encoding of one symbol carries zero bits of information. The entropy for that distribution is zero – since it is a certain outcome, there is no information encoded.
  - Distributions in between will have entropy between 0 and 1 bit.

Extensions and Drawbacks of Huffman Codes

As described, Huffman coding assumes each symbol is independent

- Many data sources, however, are not independent.
  - English text, for example. After a "q", we know it is highly likely the next letter is a "u".
- Suppose we had symbols w, x, y and z, and each occurred 25% of the time, but they always occurred in pairs:
  - xxyywwzzxxzzwwwwxxyywwyyzzxxyyzzxxwwyywwyy
  - We could use two bits to encode each symbol, but really we only need to have one codeword for each pair of symbols
- Huffman coding can often be improved by considering 1st order dependencies (neighboring symbols not being independent).

Extensions and Drawbacks of Huffman Codes

As described, Huffman codes do not adapt to changes in distributions

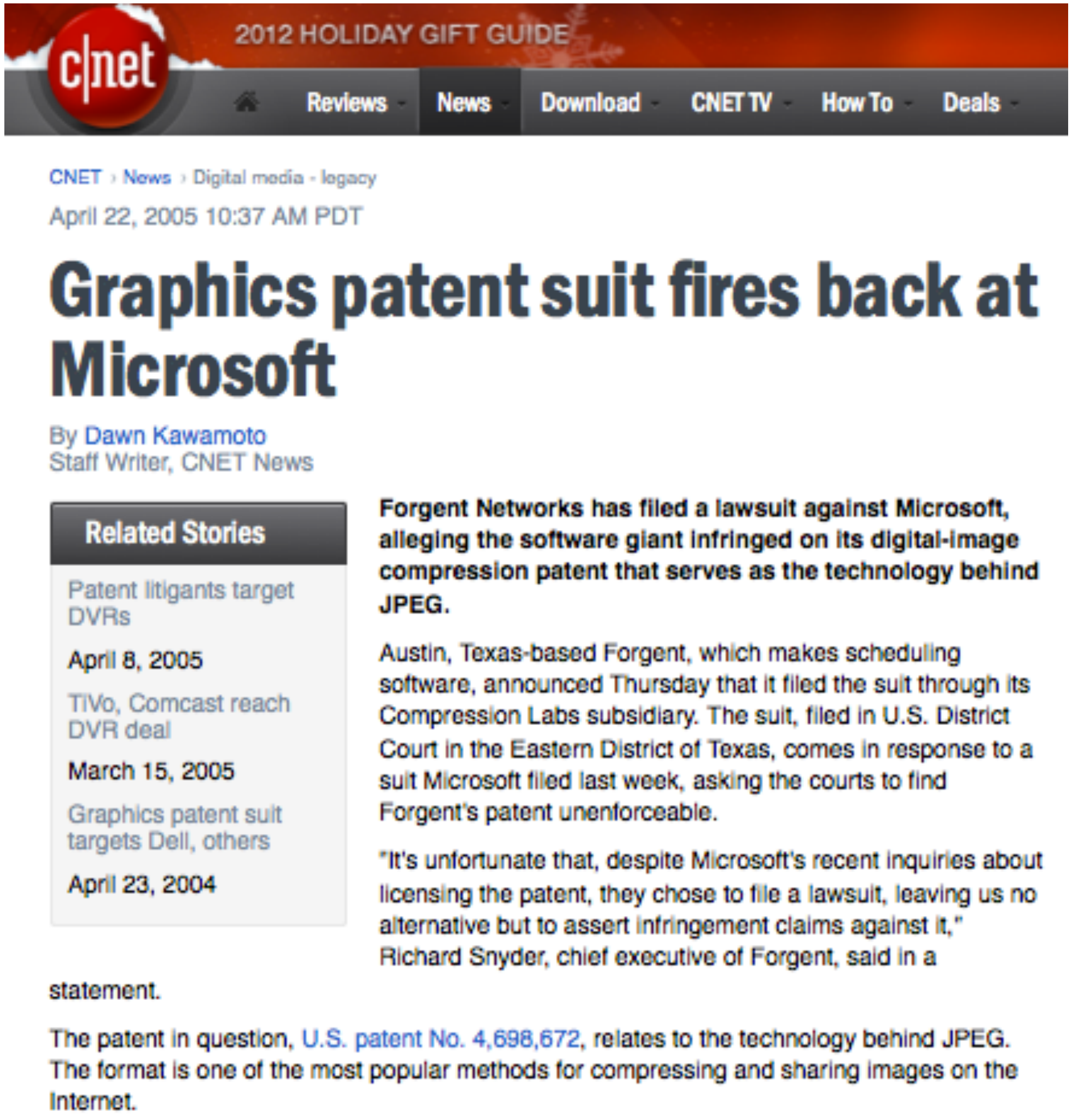
- Suppose the symbols we are encoding have one frequency distribution at the beginning of the sequence, but a different frequency distribution starting in the middle of the sequence
  - One Huffman code will be optimal for the average distribution
  - We'd ideally like to have multiple Huffman codes, or be able to have an adaptive Huffman code, that adjusts to the real frequency distribution as it changes over time

Extensions and Drawbacks of Huffman Codes

All of these concepts apply in the field of data compression and image compression

- Images have distributions that are often not 50%-50%
  - Imagine encoding and sending a black and white fax.
  - Typically most of the pixels are white, a small percentage black.
- Images have correlation.
  - The color of one pixel is highly correlated to the color of its neighbor.
  - Coding should not treat them as independent.
- Images have distributions that vary over time (space).
  - One part of an image might be sky with a certain color distribution.
  - Another part might be grass, beach, water, buildings, etc. with another distribution of colors.

Huffman Coding in the Courts



CNET › News › Digital media - legacy

April 22, 2005 10:37 AM PDT

Graphics patent suit fires back at Microsoft

By Dawn Kawamoto
Staff Writer, CNET News

Related Stories

Patent litigants target DVRs
April 8, 2005

TiVo, Comcast reach DVR deal
March 15, 2005

Graphics patent suit targets Dell, others
April 23, 2004

Forgent Networks has filed a lawsuit against Microsoft, alleging the software giant infringed on its digital-image compression patent that serves as the technology behind JPEG.

Austin, Texas-based Forgent, which makes scheduling software, announced Thursday that it filed the suit through its Compression Labs subsidiary. The suit, filed in U.S. District Court in the Eastern District of Texas, comes in response to a suit Microsoft filed last week, asking the courts to find Forgent's patent unenforceable.

"It's unfortunate that, despite Microsoft's recent inquiries about licensing the patent, they chose to file a lawsuit, leaving us no alternative but to assert infringement claims against it," Richard Snyder, chief executive of Forgent, said in a statement.

The patent in question, U.S. patent No. 4,698,672, relates to the technology behind JPEG. The format is one of the most popular methods for compressing and sharing images on the Internet.

- JPEG image coding standard uses Huffman coding
- Forgent sued companies using JPEG (anybody with digital cameras in their products) for royalties based on patent US 4698672