Data Compression: Huffman Coding

10.1 in Weiss (p.389)

Why compress files?

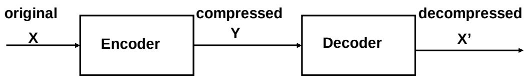
Why compress files?

- For long term storage (disc space is limited)
- For transferring files over the internet (bigger files take longer)
- A smaller file more likely to fit in memory/cache

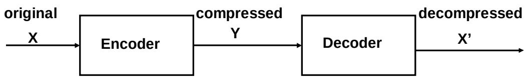
What is a file?

- C++ program code
- Executable program
- Email - text
- HTML document
- Pictures (lossy); JPEG
- Video (lossy); MPEG
- Audio (lossy); MP3

Data Compression



Data Compression



- **Lossless** compression X = X'
- **Lossy** compression X != X'
- **Compression Ratio** |X|/|Y|
  - Where |X| is the # of bits in X.

Lossy Compression

- Some data is lost, but not too much.

Standards:

- JPEG (Joint Photographic Experts Group) – stills
- MPEG (Motion Picture Experts Group)
  - Audio and video
- MP3 (MPEG-1, Layer 3)

Lossless Compression

- No data is lost.

Standards:

- Gzip, Unix compress, zip, GIF, Morse code
- Examples:
  - Run-length Encoding (RLE)
  - Huffman Coding

RLE

- Idea: Compactly represent long ‘runs’ of the same character
- “aaarrrrr!” as ‘a’x3 ‘r’x5 then ‘!’

RLE

- Idea: Compactly represent long ‘runs’ of the same character
- “aaarrrrr!” as ‘a’x3 ‘r’x5 then ‘!’
- Say…
  - Replace all ‘runs’ of the same character by 2 characters: the 1) character and 2) the length
  - ‘bee’ becomes ‘b’,1,’e’,2

RLE

- Idea: Compactly represent long ‘runs’ of the same character
- “aaarrrrr!” as ‘a’x3 ‘r’x5 then ‘!’
- Say…
  - Replace all ‘runs’ of the same character by 2 characters: the 1) character and 2) the length
  - ‘bee’ becomes ‘b’,1,’e’,2
  - When is this good?
  - When is this really bad?

Another idea: Use fewer bits per character

ASCII = fixed 8 bits per character

Example: “hello there”

- 11 characters * 8 bits = 88 bits

Can we encode this message using fewer bits?

Another idea: Use fewer bits per character

ASCII = fixed 8 bits per character

Example: “hello there”

- – 11 characters * 8 bits = 88 bits

Can we encode this message using fewer bits?

- We could look JUST at the message
- there are only 6 possible characters + one space = 7 things; only need 3 bits
- Encode: aabddcaa = could do as 16 bits (each character = 2 bits each)
- Huffman can do as 14 bits

Huffman Coding

- Uses *frequencies* of symbols in a string to build a **prefix code**.
- **Prefix Code** – no code in our encoding is a prefix of another code.

Letter	code
a	0
b	100
c	101
d	11

Huffman Coding

- Uses *frequencies* of symbols in a string to build a **prefix code**.
- **Prefix Code** – no code in our encoding is a prefix of another code.

Letter	code
a	0
b	100
c	101
d	11

Huffman Coding

- Uses *frequencies* of symbols in a string to build a **prefix code**.
- **Prefix Code** – no code in our encoding is a prefix of another code.

Letter	code
a	0
b	100
c	101
d	11

Decoding a Prefix Code

```
Loop
  start at root of tree
     loop
          if bit read = 1 then go right
          else, go left
     until node is a leaf
     Report character found!
Until end of the message
```

Decode: 11100010100110

Letter	code
a	0
b	100
c	101
d	11

Decode: 11100010100110

Letter	code
a	0
b	100
c	101
d	11

Huffman Trees

Cost of a Huffman Tree containing n symbols

$$\mathbf{C(T) = p_1 * r_1 + p_2 * r_2 + p_3 * r_3 + \ldots + p_n * r_n}$$

Where:

$\mathbf{p_i}$ = the probability that a symbol occurs

$\mathbf{r_i}$ = the length of the path from the root to the node

---

Example Cost

Letter	Frequency	code
a	.50	0
b	.125	100
c	.125	101
d	.25	11

Cost: 1.75

Constructing a tree

- Determine frequency of each letter/symbol
- Place each as an unconnected leaf node
- Repeatedly merge two nodes with lowest frequency into one node with sum of frequencies
- Huffman Coding is optimal*

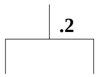
Constructing a tree example

- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total

a: .4 space: .2 j: .2 v: .1 r: .1

Constructing a tree example

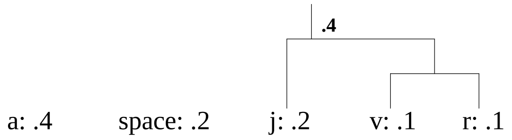
- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total



a: .4 space: .2 j: .2 v: .1 r: .1

Constructing a tree example

- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total

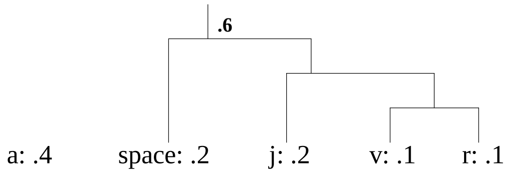


.4

a: .4 space: .2 j: .2 v: .1 r: .1

Constructing a tree example

- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total

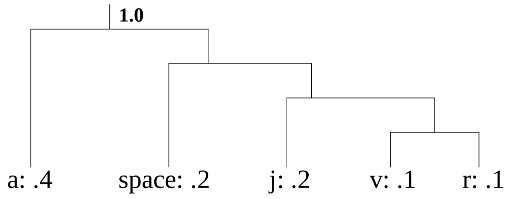


.6

a: .4 space: .2 j: .2 v: .1 r: .1

Constructing a tree example

- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total

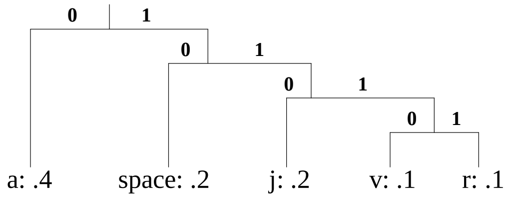


1.0

a: .4 space: .2 j: .2 v: .1 r: .1

Constructing a tree example

- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total



0 1

0 1

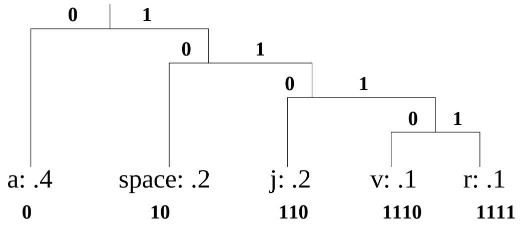
0 1

0 1

a: .4 space: .2 j: .2 v: .1 r: .1

Constructing a tree example

- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total



0 1

0 1

0 1

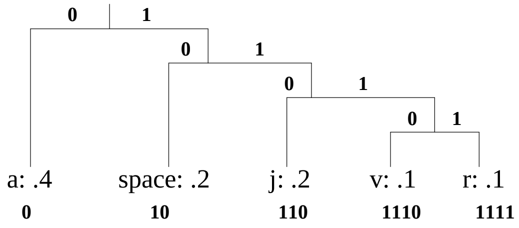
0 1

a: .4 space: .2 j: .2 v: .1 r: .1

0 10 110 1110 1111

Constructing a tree example

- Encode “a java jar”
- 4 a’s, 2 spaces, 2 j’s, 1 v, 1 r; 10 total



0 1

0 1

0 1

0 1

a: .4 space: .2 j: .2 v: .1 r: .1

0 10 110 1110 1111

Cost = .4*1 + .2*2 + .2*3 + .1*4 + .1*4 = 2.2

Run-time?

- To decode an encoded message length n:

Run-time?

- To decode an encoded message length n: O(n)
- To encode message length n, with c possible characters

Run-time?

- To decode an encoded message length n: O(n)
- To encode message length n, with c possible characters
  - Count frequencies:
  - Build tree:
  - Encode:

Run-time?

- To decode an encoded message length n: O(n)
- To encode message length n, with c possible characters
  - Count frequencies: O(n)
  - Build tree: O(clogc) (with priority queue)
  - Encode: O(n)