19—String Compression

May 29, 2002

Why Compress

File	Size (bytes)
maze.ps	1746364
maze.ps.gz	37143
maze.ps.bz2	14039

Two Kinds of Compression

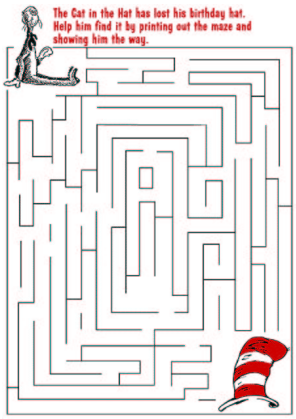
Lossy Compression

- Good for
  - Images
  - Sound
- jpeg, mpeg, mp3

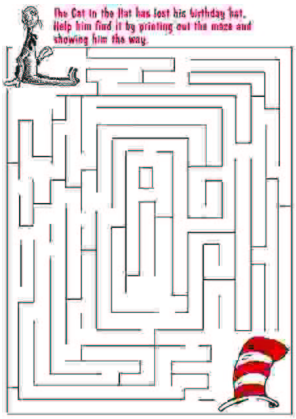
Lossless Compression

- Good for
  - Documents
  - Binaries
- zip, gz, Z, bz2

Lossy Compression



Original Image (20763 bytes)



Highly Compressed (14763 bytes)

Lossless String Compression

7108324569

- How many bits does this string take up in C++?
- Can we do better?

A Generalization

7108324569

- *Alphabet* Σ of string has 10 characters
- Need $\lceil \log_2 |\Sigma| \rceil = 4$ bits per character

Lossless String Compression

1111111897

- How many bits does this string take up in C++?
- How many bits if we encode in 4 bits?
- Can we do better?

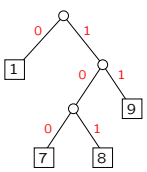
Compressing Text

See the pictures in the sorting slides. The last few iterations of Radix sort can probe memory in a very random way, as it depends on the upper digits of the sorted items, which are completely unsorted.

Letter	Freq
a	9
b	1
c	4
d	8
e	23
f	3
g	2
h	7
i	13
l	4
m	5
n	9
o	12
p	6
r	13
s	14
t	17
u	3
v	1
w	3
x	1
y	4
j,k,q,z	0

Why encode *x* the same size as *e*?

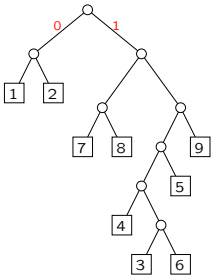
Encoding Trees



1111111897 = 0000000 101 11 100

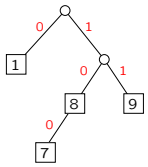
- All characters are *leaves*
- Encoding of c found from path to c

Example



234965

What's Wrong with this Tree?

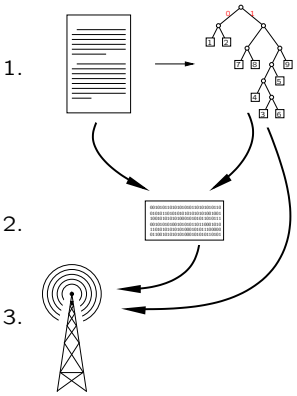


1111111897 = 0000000 10 11 100

How to Encode

Letter	Freq
1	7
7	1
8	1
9	1

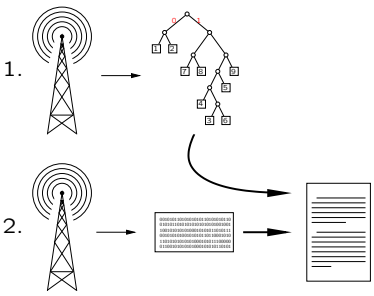
Two-Pass Encoding



1. Read input and build tree
2. Reread input and encode
3. Send tree and encoding

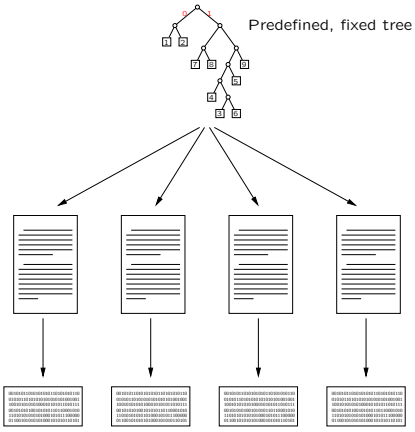
Disadvantages?

Two-Pass Decoding



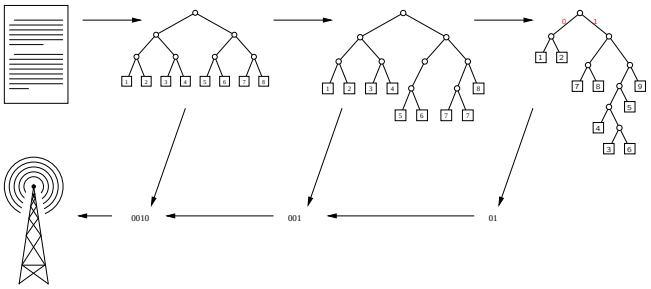
1. Read tree from input
2. Decode using tree

Static Encoding/Decoding



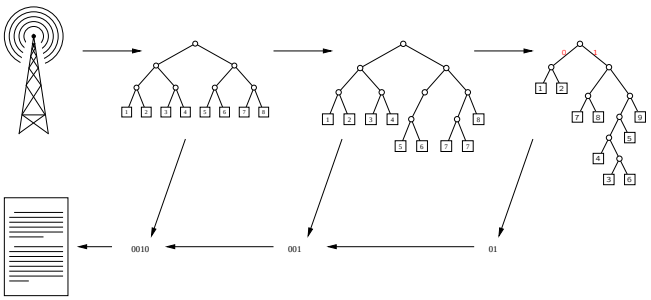
- Advantages?
- Disadvantages?

Adaptive Encoding



Build tree *while* reading & encoding input

Adaptive Decoding



Decoder builds the same tree as the encoder!

Adaptive Coding Pseudo-code

```
Encode(Text t, Output os)
{
  EncodingTree T.InitDefault();
  while (t) {
    ch = t.getch();
    os << T.Encode(ch);
    T.Update(ch);
  }
}
```

```
Decode(Input is)
{
  Text t;
  EncodingTree T.InitDefault();
  while (is) {
    ch = is.getch();
    t.Append(T.Decode(ch));
    T.Update(ch);
  }
  return t;
}
```

Limitations of Huffman Encoding

- Only considers *frequency* of letters, not *context*
- Certain *groups* may appear frequently

  ing, the, ed, . . .

- Could Huffman Code *pairs* or *triplets* of letters instead

  Disadvantages?

Lempel-Ziv Encoding

Dictionary of Sequences

0	a
1	b
2	c
⋮	
24	y
25	z
26	ed
27	ing
28	the
29	in
⋮	

1. Find longest prefix of text that's in the dictionary
2. Output *code number* of prefix
3. Update dictionary
4. Advance Text

Lempel-Ziv Pseudo-code

```
LZEncode(Text t, Ouput os)
{
  Dictionary D;
  while (!t.done()) {
    match = Longest match of t in D;
    D.Add(last_match + match[0]);
    last_match = match;
    os << D.Index(match);
  }
}
```

How to decode?

Example

COCOA AND BANNANAS

Important Details

What happens when the dictionary fills up?

- Stop inserting new entries
- Clear the dictionary and start over
- Overwrite an infrequently used sequence
- Increase size of dictionary

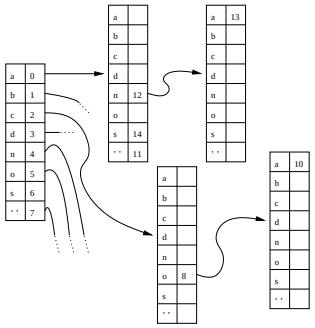
Important Details

How to implement the dictionary?

- List?
- Array?
- Balanced tree?
- Trie?

Why Try Tries?

Want longest-match lookups to be easy



Why must any prefix of a sequence be in the dictionary?

Comparing Huffman and Lempel-Ziv

our textbook	10576224 bits
two-pass Huffman	6469752 bits (61%)
adaptive Huffman	6470800 bits
Lempel-Ziv	4493168 bits (43%)