Data Compression

Run Length Encoding

Huffman Encoding

Entropy

LZW

Reference: Chapter 22, *Algorithms in C, 2nd Edition*, Robert Sedgewick.

Reference: *Introduction to Data Compression*, Guy Blelloch.

Data Compression

Compression reduces the size of a file:

- To save TIME when transmitting it.
- To save SPACE when storing it.
- Most files have lots of redundancy.

Who needs compression?

- Moore's law: # transistors on a chip doubles every 18-24 months.
- Parkinson's law: data expands to fill space available.
- Text, images, sound, video, . . .

> All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value. -Carl Sagan

Basic concepts ancient (1950s), best technology recently developed.

Applications of Data Compression

Generic file compression.

- Files: GZIP, BZIP, BOA.
- Archivers: PKZIP.
- File systems: NTFS.

Multimedia.

- Images: GIF, JPEG, CorelDraw.
- Sound: MP3.
- Video: MPEG, DivX™, HDTV.

Communication.

- ITU-T T4 Group 3 Fax.
- V.42bis modem.

Databases.

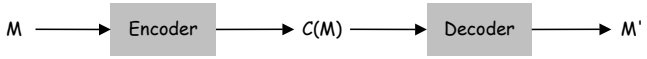
- Google.

Encoding and Decoding

Message. Binary data M we want to compress.

Encode. Generate a "compressed" representation C(M) that hopefully uses fewer bits.

Decode. Reconstruct original message or some approximation M'.



Compression ratio: bits in C(M) / bits in M.

Lossless. M = M', 50-75% or lower.
Ex: Java source code, executables.

Lossy. M ~ M', 10% or lower.
Ex: images, sound, video.

Simple Ideas

Ancient ideas.

- Human language.
- Morse code.
- Braille.

Fixed length coding.

- Use same number of bits for each symbol.
- N symbols $\Rightarrow \lceil \log N \rceil$ bits per symbol.
- 7-bit ASCII code for text.

ASCII coding

char	dec	binary
NUL	0	0000000
...	...	...
>	62	0111110
?	63	0111111
@	64	1000000
A	65	1000001
B	66	1000010
C	67	1000011
...	...	...
~	126	1111110
DEL	127	1111111

a	b	r	a	c	a	d	a	b	r	a
1100001	1100010	1110010	1100001	1100011	1100001	1110100	1100001	1100010	1110010	1100001

7 × 11 = 77 bits

Run-Length Encoding

Natural encoding: 51 × 19 + 6 = 975 bits.
Run-length encoding: 63 × 6 + 6 = 384 bits.

```
00000000000000000000000000000011111111111111100000000   28 14 9
00000000000000000000000000001111111111111111110000000   26 18 7
00000000000000000000000011111111111111111111111110000   23 24 4
00000000000000000000001111111111111111111111111111000   22 26 3
00000000000000000000011111111111111111111111111111110   20 30 1
00000000000000000001111111000000000000000000001111111   19 7 18 7
00000000000000000011111000000000000000000000000011111   19 5 22 5
00000000000000000111000000000000000000000000000000111   19 3 26 3
00000000000000000111000000000000000000000000000000111   19 3 26 3
00000000000000000111000000000000000000000000000000111   19 3 26 3
00000000000000000111000000000000000000000000000000111   19 3 26 3
00000000000000000111000000000000000000000000000001110   20 4 23 3 1
00000000000000000011100000000000000000000000000111000   22 3 20 3 3
01111111111111111111111111111111111111111111111111111   1 50
01111111111111111111111111111111111111111111111111111   1 50
01111111111111111111111111111111111111111111111111111   1 50
01111111111111111111111111111111111111111111111111111   1 50
01111111111111111111111111111111111111111111111111111   1 50
01100000000000000000000000000000000000000000000000011   1 2 46 2
```

raster of letter 'q' lying on its side — RLE

Run-Length Encoding

Run-length encoding (RLE).

- Exploit long runs of repeated characters.
- Replace run by count followed by repeated character, but don't bother if run is less than 3.
  - AAAABBBAABBBBBCCCCCCCCDABCBAAABBBBCCCD
  - 4A3BAA5B8CDABCB3A4B3CD
- Annoyance: how to represent counts.
- Runs in binary file alternate between 0 and 1, so output count only.
- "File inflation" if runs are short.

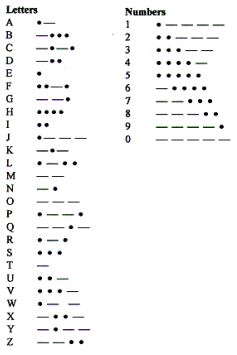
Applications.

- Black and white graphics.
  - compression ratio improves with resolution!
- JPEG (Joint Photographic Experts Group).

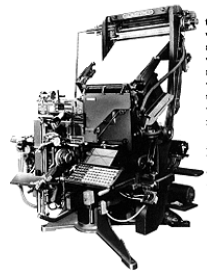
Variable Length Encoding

Variable-length encoding.

- Use DIFFERENT number of bits to encode different characters.



Char	Freq	Char	Freq
E	12.51	M	2.53
T	9.25	F	2.30
A	8.04	P	2.00
O	7.60	G	1.96
I	7.26	W	1.92
N	7.09	Y	1.73
S	6.54	B	1.54
R	6.12	V	0.99
H	5.49	K	0.67
L	4.14	X	0.19
D	3.99	J	0.16
C	3.06	Q	0.11
U	2.71	Z	0.09



Linotype machine, 1886

Variable Length Encoding

Variable-length encoding.

- Use DIFFERENT number of bits to encode different characters.

a	b	r	a	c	a	d	a	b	r	a
0 0 0	0 0 1	1 0 0	0 0 0	0 1 0	0 0 0	0 1 1	0 0 0	0 0 1	1 0 0	0 0 0

3-bit fixed length coding: $3 \times 11 = 33$ bits

a	b	r	a	c	a	d	a	b	r	a
1	0 0 1	0 1 1	0 0 0	0	1 0 0	0	1 1	0 0 1	0 1	1

variable length coding: 23 bits

char	encoding
a	1
b	001
c	0000
d	0001
r	01

Variable Length Decoding

But, then how do we decode? Variable length codes can be ambiguous.

a	b	r	a	c	a	d	a	b	r	a
1	0 0	0	0	1	1	0	1	0	1	1
c		r	a	a	d		b	a	c	r	a

char	encoding
a	1
b	0
c	10
d	01
r	00

How do we avoid ambiguity?

- One solution: ensure no encoding is a prefix of another.
- Ex: 00 is a prefix of 0001.

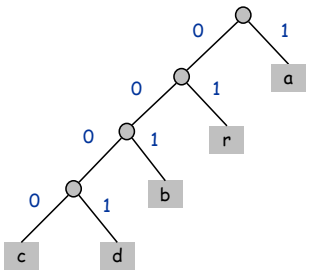
a	b	r	a	c	a	d	a	b	r	a
1	0 0 1	0 1 1	0 0 0	0	1 0 0	0	1 1	0 0 1	0 1	1

char	encoding
a	1
b	001
c	0000
d	0001
r	01

Prefix-Free Code: Implementation

How to represent? Use a binary trie.

- symbols are stored in leaves
- encoding is path to leaf



Encoding.

- Method 1: start at leaf corresponding to symbol, follow path up to the root, and print bits in reverse order.
- Method 2: create ST of symbol-encoding pairs.

Decoding.

- Start at root of tree.
- Take left branch if bit is 0; right branch if 1.
- If leaf node, print symbol and return to root.

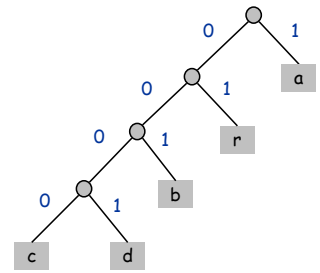
char	encoding
a	1
b	001
c	0000
d	0001
r	01

How to Transmit the Trie

How to transmit the trie?

- Send lookup table.
- Send preorder traversal of trie.
  - we use * as sentinel for internal nodes
  - what if there is no sentinel?

```
****cdbra
100101100001000110010 11
```



- If message is long, overhead of sending trie is small.

char	encoding
a	1
b	001
c	0000
d	0001
r	01

Prefix-Free Decoding Implementation: Recall COS 126

```
public class HuffmanDecoder {
   private char c;
   private HuffmanDecoder left, right;

   public HuffmanDecoder() {
      c = CharStdIn.readChar();
      if (c == '*') {
         left  = new HuffmanDecoder();
         right = new HuffmanDecoder();
      }
   }

   public void decode() {
      HuffmanDecoder x = this;
      while(!CharStdIn.isEmpty()) {
         char bit = CharStdIn.readChar();
         if      (bit == '0') x = x.left;
         else if (bit == '1') x = x.right;
         if (x.left == null && x.right == null) {
            System.out.print(x.c);
            x = this;
         }
      }
   }
```

build tree from preorder traversal

c*d*b*r*a

1001011000010001100 1011

use bits in real applications instead of chars

Huffman Coding

OK, but how do I find a good prefix-free coding scheme? Greed.

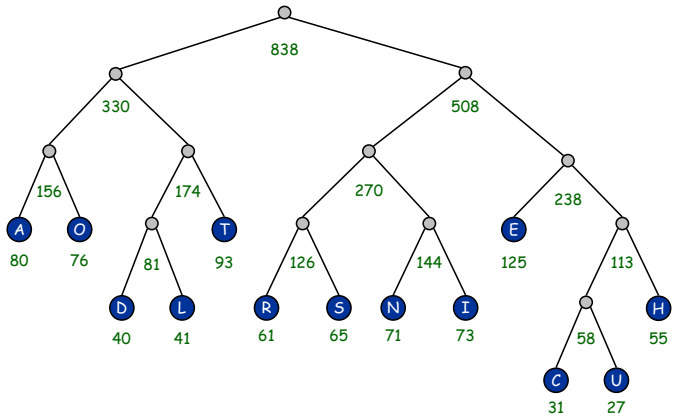
To compute Huffman prefix-free code:

- Count character frequencies p_s for each symbol s in file.
- Start with a forest of trees, each consisting of a single vertex corresponding to each symbol s with weight p_s.
- Repeat:
  - select two trees with min weight p_1 and p_2
  - merge into single tree with weight $p_1 + p_2$

Applications: JPEG, MP3, MPEG, PKZIP.

Huffman Coding Example

Char	Freq	Huff
E	125	110
T	93	011
A	80	000
O	76	001
I	73	1011
N	71	1010
S	65	1001
R	61	1000
H	55	1111
L	41	0101
D	40	0100
C	31	11100
U	27	11101
Total	838	3.62



Huffman Tree Implementation

```
private class HuffmanTree implements Comparable {
   char c;
   int freq;
   HuffmanTree left, right;

   // print preorder traversal
   void preorder() {
      System.out.print(c);
      if (left != null && right != null) {
         left.preorder();
         right.preorder();
      }
   }

   ...

}
```

Huffman Encoding Implementation

```
public HuffmanEncoder(String input) {

   // tabulate frequencies
   int[] freq = new int[SYMBOLS];
   for (int i = 0; i < input.length(); i++)
      freq[input.charAt(i)]++;

   // initialize priority queue with singleton elements
   PQ pq = new PQ();
   for (int i = 0; i < SYMBOLS; i++)
      if (freq[i] > 0)
         pq.insert(new HuffmanTree((char) i, freq[i], null, null));

   // repeatedly merge two smallest trees
   while (pq.size() > 1) {
      HuffmanTree x = (HuffmanTree) pq.delMin();
      HuffmanTree y = (HuffmanTree) pq.delMin();
      HuffmanTree parent = new HuffmanTree('*', x.freq + y.freq, x, y);
      pq.insert(parent);
   }
   tree = (HuffmanTree) pq.delMin();
}
```

internal node

root of Huffman tree

Huffman Encoding

Theorem (Huffman, 1952). Huffman coding is optimal prefix-free code.

- No variable length code uses fewer bits.

Corollary. Greed is good.

Implementation.

- Two passes.
  - tabulate symbol frequencies and build trie
  - encode file by traversing trie or lookup table
- Use priority queue for delete min and insert.
- $O(M + N \log N)$.

PQ implementation important if each symbol represents one English word

M = file size
N = # distinct symbols

Difficulties.

- Have to transmit encoding (trie).
- Not optimal (unless block size grows to infinity)!

What Data Can be Compressed?

US Patent 5,533,051 on "Methods for Data Compression."

- Capable of compressing all files.

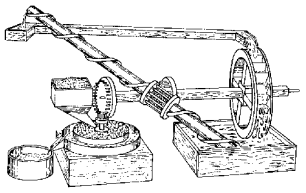
Slashdot reports of the Zero Space Tuner™ and BinaryAccelerator™.

> "ZeoSync has announced a breakthrough in data compression that allows for 100:1 lossless compression of random data. If this is true, our bandwidth problems just got a lot smaller. . . ."

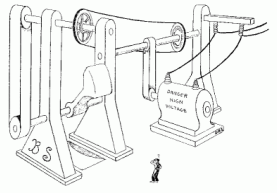
What Data Can be Compressed?

Impossible to losslessly compress all files.

- Consider all 1000 bit messages.
- 2^{1000} possible messages.
- Only $2^{999} + 2^{998} + \ldots + 1$ can be encoded with ≤ 999 bits.
- Only 1 in 2^{499} can even be encoded with ≤ 500 bits!



Closed-cycle mill by Robert Fludd, 1618



Gravity engine by Bob Schadewald

Reference: Museum of Unworkable Devices by Donald E. Simanek
http://www.lhup.edu/~dsimanek/museum/unwork.htm

A Difficult File To Compress

One million pseudo-random characters (a - p)

```
fclkkacifobjofmkgdcoiicnfmcpcjfccabckjamolnihkbgobcjbngjiceeelpfgcjiihppenefllhglfemdemgahlbpi
ggmllmnefnhjelmgjncjcidlhkglhceninidmmgnobkeglpnadanfbecoonbiehglmpnhkkamdffpacjmgojmcaabpcjce
cplfbgamlidcek1hfkkmioljdnoaagiheiapaimlcnlljniggpeanbmojgkccogpmkmoifioeikefjidbadgdcepnhdpfj
aeeapdjeofklpdeghidbgcaiemajllhnndigeihbebifemacfadnknhlbgincpmimdogimgeeomgeljfjgklkdgnhafoho
npjbmlkapddhmepdnckeajebmeknmeejnmenbmnnfefdbhpmigbbjknjmobimamjjaaaffhlhiggaljbaijnebidpaeigd
goghcihodnlhahllhhoojdfacnhadhgkfahmeaebccacgeojgikcoapknlomfignanedmajinlompjoaifiaejbcjcdibp
kofcbmjiobbpdhfilfajkhfmppcngdneeinpnfafaeladbhhifechinknpdnplamackphekokigpddmmjnbngklhibohdf
eaggmclllmdhafkldmimdbplggbbejkcmhlkjocjjlcngckfpfakmnpiaanffdjdlleiniilaenbnikgfnjfcophbgkhdg
mfpoehfmkbpiaignphogbkelphobonmfghpdgmkfedkfkchceeldkcofaldinljjcgafimaanelmfkokcjekefkbmegcgj
ifjcpjppnabldjoaafpbdafifgcoibbcmoffbbgigmngefpkmbhbghlbdjngenldhgnfbdlcmjdmoflhcogfjoldfjpaok
epndejmnbiealkaofifekdjkgedgdlgbioacflfjlafbcaemgpjlagbdgilhcfdcamhfmppfgohjphlmhegjechgdpkklj
pndphfcnnganmbmnggppnckbieknjhilafkegboilajdppcodpeoddldjfcpialoalfeomjbphkmhnpdmcpgkgeaohfdm
cnegmibjkajcdcpjcpgjminhhakihfgiiachfepffnilcooiciepoapmdjniimfbolchkibkbmhbkgconimkdchahcnhap
fdkiapikencegcjapkjkfljgdlmgncpbakhjidapbldcgeekkjaoihbnbigmhboengpmedliofgioofdcphelapijcegej
gcldcfodikalehbccpbbcfakkblmoobdmdgdkafbbkjnidoikfakjclbchambcpaepfeinmenmpoodadoecbgbmfkkeabi
laoeoggghoekamaibhjibefmoppbhfbhffapjnodlofeihmjahmeipejlfhloefgmjhjnlomapjakhhjpncomippeanbik
khekpcfgbgkmklipfbiikdkdcbolofhelipbkbjmjfoempccneaebklibmcaddlmjdcajpmhhaeedbbfpjafcndianlfcj
mmbfncpdcccodeldhmnbdjmeajmboclkggojghlohlbhgjkhkmclohkgjamfmcchkchmiadjgjhjehflcbklfifackbecg
joggpbkhlcmfhipflhmnmifpjmcoldbeghpcekhgmnahijpabnomnokldjcpppbcpgcjofngmbdcpeeeiiiclmbbmfjkhl
anckidhmbeanmlabncnccpbhoafajjicnfeenppoekmlddholnbdjapbfcajblbooiaepfmmeoafedflmdcbaodgeahimc
gpcammjljoebpfmghogfckgmomecdipmodbcempidfnlcggpgbffoncajpncomalgoiikeolmigliikjkolgolfkdgiijj
iooiokdihjbbofiooibakadjnedlodeeiijkliicnioimablfdpjiafcfineecbafaamheiipegegibioocmlmhjekfikf
effmddhoakllnifdhckmbonbchfhhclecjamjildonjjdpifngbojianpljahpkindkdoanlldcbmlmhjfomifhmncikol
jjhebidjdphpdepibfgdonjljfgifimniipogockpidamnkcpipglafmlmoacjibognbplejnikdoefccdpfkomkimffgj
gielocdemnblimfmbkfbhkelkpfoheokfofochbmifleecbglmnfbnfncjmefnihdcoeiefllemnohlfdcmbdfebdmbeeb
balggfbajdamplphdgiimehglpikbipnkkecekhilchhhfaeafbbfdmcjojfhpponglkfdmhjpcieofcnjgkpibcbiblfp
njlejkcppbhopohdghljlcokhdoahfmlglbdkliajbmnkkfcoklhlelhjhoiginaimgcabcfebmjdnbfhohkjphnklcbhc
jpgbadakoecbkjcaebbanhnfhpnfkfbfpohmnkligpgfkjadomdjjnhlnfailfpcmnololdjekeolhdkebiffebajjpclg
hllmemegncknmkkeoogilijmmkomllbkkabelmodcohdhppdakbelmlejdnmbfmcjdebefnjihnejmnogeeafldabjcgfo
aehldcmkbnbafpciefhlopicifadbppgmfngecjhefnkbjmliodhelhicnfoongngemddepchkokdjafegnpgledakmbcp
cmkckhbffeihpkajginfhdolfnlgnadefamlfocdibhfkiaofeegppcjilndepleihkpkkgkphbnkggjiaolnolbjpobjd
cehglelckbhjilafccfipgebpc....
```


A Difficult File To Compress

```
public class Rand {
   public static void main(String[] args) {
      for (int i = 0; i < 1000000; i++) {
         char c = 'a';
         c += (char) (Math.random() * 26);
         System.out.print(c);
      }
   }
}
```

231 bytes, but its output is hard to compress
(assume random seed is fixed)

```
% javac Rand.java
% java Rand   > temp.txt
% compress -c   temp.txt > temp.Z
% gzip      -c   temp.txt > temp.gz
% bzip2     -c   temp.txt > temp.bz2
```

```
% ls -l
    231 Rand.java
1000000 temp.txt
 576861 temp.Z
 570872 temp.gz
 499329 temp.bz2
```

resulting file sizes (bytes)

Information Theory

Intrinsic difficulty of compression.

- Short program generates large data file.
- Optimal compression algorithm has to discover program!
- Undecidable problem.

So how do we know if our algorithm is doing well?

- Want lower bound on # bits required by ANY compression scheme.

Language Model

How compression algorithms work?

- Exploit biases of input messages.
- Word `Princeton` occurs more frequently than `Yale`.
- White patches occur in typical images.

Compression is all about probability.

- Formulate probabilistic model to predict symbols.
  - simple: character counts, repeated strings
  - complex: models of a human face
- Use model to encode message.
- Use same model to decode message.

Example. Order 0 Markov model.

- Each symbol s generated independently at random, with fixed probability p(s).

Entropy

Entropy. (Shannon 1948) $H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$

- Information content of symbol s is proportional to $\log_2 \frac{1}{p(s)}$
- Weighted average of information content over all symbols.
- Interface between coding and model.

	p(a)	p(b)	H(S)
Model 1	1/2	1/2	1
Model 2	0.900	0.100	0.469
Model 3	0.990	0.010	0.0808
Model 4	1	0	0

	p(a)	p(b)	p(c)	p(d)	p(e)	H(S)
Model 5	1/5	1/5	1/5	1/5	1/5	2.322

Entropy and Compression

Shannon's theorem (1948). Avg # bits per symbol ≥ entropy.

- If data source S is an order 0 Markov model, then ANY compression scheme must use ≥ H(S) bits per symbol on average.
- Cornerstone result of information theory.

Huffman's coding theorem (1952). Huffman code is optimal.

- If data source S is an order 0 Markov mode, then H(S) ≤ avg # bits per symbol ≤ H(S) + 1.

Is there any hope of doing better?

- Yes. Huffman wastes up to 1 bit per symbol.
  - if H(S) is close to 0, this matters
  - can do better with "arithmetic coding"
- Yes. Source may not be order 0 Markov model.

Entropy of the English Language

Q. How much redundancy is in the English language?

> "... randomising letters in the middle of words [has] little or no effect on the ability of skilled readers to understand the text. This is easy to denmtrasote. In a pubiltacion of New Scnieitst you could ramdinose all the letetrs, keipeng the first two and last two the same, and reibadailty would hadrly be aftcfeed. My ansaylis did not come to much beucase the thoery at the time was for shape and senqeuce retigcionon. Saberi's work sugsegts we may have some pofrweul palrlael prsooscers at work. The resaon for this is suerly that idnetiyfing coentnt by paarllel prseocsing speeds up regnicoiton. We only need the first and last two letetrs to spot chganes in meniang."

A. Quite a bit.

Entropy of the English Language

How much information is in each character of English language?

- Model = English text.

How can we measure it? Shannon's experiment (1951).

- Asked humans to predict next character given previous text.
- The number of guesses required for right answer:

# of guesses	1	2	3	4	5	≥ 6
Probability	0.79	0.08	0.03	0.02	0.02	0.05

- Shannon's estimate = 0.6 - 1.3.

Lossless Compression Ratio for Calgary Corpus

Year	Scheme	Bits / char
----	ASCII	7.00
1950	Huffman	4.70
1977	LZ77	3.94
1984	LZMW	3.32
1987	LZH	3.30
1987	Move-to-front	3.24
1987	LZB	3.18
1987	Gzip	2.71
1988	PPMC	2.48
1988	SAKDC	2.47
1994	PPM	2.34
1995	Burrows-Wheeler	2.29
1997	BOA	1.99
1999	RK	1.89

Entropy	Bits/char
Char by char	4.5
8 chars at a time	2.4
Asymptotic	1.3

Statistical Methods

Estimate symbol frequencies, and assign codewords based on this.

Static model. Same distribution for all texts.

- Fast.
- Not optimal since different texts exhibit different distributions.
- Ex: ASCII, Morse code, Linotype.

Dynamic model. Generate model based on text.

- Preliminary pass needed to generate model.
- Must transmit the model.
- Ex: Huffman code.

Adaptive models. Progressively learn and update model as you read text.

- More accurate models produce better compression.
- Decoding must start from beginning.
- Ex: LZW.

LZW Algorithm

Lempel-Ziv-Welch (variant of LZ78).

- Adaptively maintain symbol table of useful strings.
- If input matches word in ST, output index instead of string.

Algorithm.

- Find longest word W in ST that is a prefix of string starting at current index.
- Output index for W followed by x = next symbol.
- Add Wx to dictionary.

Example.

- Dictionary: `a`, `aa`, `ab`, `aba`, `abb`, `abaa`, `abaab`, `abaaa`,
- String starting at current index: . . . `abaab`abbb . . .
- W = `abaab`, x = `a`.
- Output index for W, insert `abaaba` into ST.

LZW Example

Input	Send
SEND	256
i	105
t	116
t	116
y	121
_	32
b	98
i	
t	258
t	
y	260
_	
b	262
i	
t	258
_	
b	
i	266
n	110
STOP	257

Dictionary			
Index	Word	Index	Word
0		258	it
…		259	tt
32	_	260	ty
…		261	y_
97	a	262	_b
98	b	263	bi
…		264	itt
122	z	265	ty_
…		266	_bi
256	SEND	267	it_
257	STOP	268	_bin

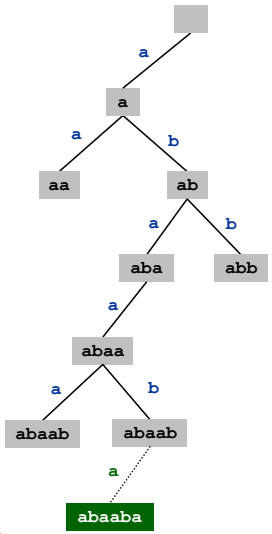
LZW Implementation

Implementation.

- Use trie to create symbol table on-the-fly.
  - prefix of every word also in ST
- Encode.
  - lookup string suffix in trie
  - output ST index at bottom
  - add new node to bottom of trie
- Decode.
  - lookup string suffix in trie
  - output ST word at bottom
  - add new node to bottom of trie

What to do when ST gets too large?

- Throw away and start over. GIF
- Throw away when not effective. Unix compress



abaaba...

LZW in the Real World

Lempel-Ziv and friends.

- LZ77. not patented ⇒ widely used in open source
- LZ78.
- LZW. patent #4,558,302 expired in US on June 20, 2003
- Deflate = LZ77 variant + Huffman.
- Some versions copyrighted.

PNG: LZ77.
Winzip, gzip, jar: deflate.
Unix compress: LZW.
Pkzip: LZW + Shannon-Fano.
GIF, TIFF, V.42bis modem: LZW.
Google: uses zlib which is based on deflate.

never expands a file

Summary

Lossless compression.

- Simple approaches. RLE
- Represent fixed length symbols with variable length codes. Huffman
- Represent variable length symbols with fixed length codes. LZW

Lossy compression.

- JPEG, MPEG, MP3.
- Signal processing, wavelets, fractals, SVD, ...
- Algorithms not covered in COS 226.

Limits on compression. Entropy.