Data Compression

Reference: Chapter 22, *Algorithms in C, 2nd Edition*, Robert Sedgewick.
Reference: *Introduction to Data Compression*, Guy Blelloch.

Robert Sedgewick and Kevin Wayne • Copyright © 2006 • http://www.Princeton.EDU/~cos226

Compression reduces the size of a file:
- To save space when storing it.
- To save time when transmitting 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.

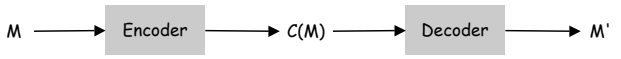
Multimedia.
- Images: GIF, JPEG.
- Sound: MP3.
- Video: MPEG, DivX™, HDTV.

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). (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. Natural language, source code, executables.

Lossy. M ≈ M', 10% or lower.
Ex. Images, sound, video.

Ancient Ideas

Ancient ideas.

- Braille.
- Morse code.
- Natural languages.
- Mathematical notation.
- Decimal number system.

"Poetry is the art of lossy data compression."

Natural Encoding

Natural encoding. $(19 \times 51) + 6 = 975$ bits.

needed to encode number of characters per line

```
00000000000000000000000000000011111111111111000000000
00000000000000000000000000001111111111111111110000000
00000000000000000000000011111111111111111111111110000
00000000000000000000001111111111111111111111111111000
00000000000000000000011111111111111111111111111111110
00000000000000000000111111100000000000000000001111111
00000000000000000001111100000000000000000000000011111
00000000000000000011100000000000000000000000000000111
00000000000000000011100000000000000000000000000000111
00000000000000000011100000000000000000000000000000111
00000000000000000011100000000000000000000000000000111
00000000000000000011110000000000000000000000000001110
00000000000000000001110000000000000000000000000111000
01111111111111111111111111111111111111111111111111111
01111111111111111111111111111111111111111111111111111
01111111111111111111111111111111111111111111111111111
01111111111111111111111111111111111111111111111111111
01111111111111111111111111111111111111111111111111111
01100000000000000000000000000000000000000000000000011
```

19-by-51 raster of letter 'q' lying on its side

Run-Length Encoding

Natural encoding. $(19 \times 51) + 6 = 975$ bits.
Run-length encoding. $(63 \times 6) + 6 = 384$ bits.

63 6-bit run lengths

00000000000000000000000000000011111111111111000000000	28 14 9
00000000000000000000000000001111111111111111110000000	26 18 7
00000000000000000000000011111111111111111111111110000	23 24 4
00000000000000000000001111111111111111111111111111000	22 26 3
00000000000000000000011111111111111111111111111111110	20 30 1
00000000000000000001111111100000000000000000001111111	19 7 18 7
00000000000000000011111000000000000000000000000011111	19 5 22 5
00000000000000000011100000000000000000000000000000111	19 3 26 3
00000000000000000011100000000000000000000000000000111	19 3 26 3
00000000000000000011100000000000000000000000000000111	19 3 26 3
00000000000000000011100000000000000000000000000000111	19 3 26 3
00000000000000000011110000000000000000000000000001110	20 4 23 3 1
00000000000000000001110000000000000000000000000111000	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

19-by-51 raster of letter 'q' lying on its side RLE

Run-Length Encoding

Run-length encoding (RLE).

- Exploit long runs of repeated characters.
- Binary alphabet: runs alternate between 0 and 1; output counts.
- "File inflation" possible if runs are short.

Applications.

- JPEG.
- ITU-T T4 fax machines. (black and white graphics)

Fixed Length Coding

Fixed length encoding.

- Use same number of bits for each symbol.
- N symbols ⇒ ⌈lg N⌉ bits per symbol.

7-bit ASCII encoding

char	dec	encoding
NUL	0	0000000
...	...	...
a	97	1100001
b	98	1100010
c	99	1100011
d	100	1100100
...	...	...
~	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

Fixed Length Coding

Fixed length encoding.

- Use same number of bits for each symbol.
- N symbols ⇒ ⌈lg N⌉ bits per symbol.

3-bit abracadabra encoding

char	encoding
a	000
b	001
c	010
d	011
r	100

a	b	r	a	c	a	d	a	b	r	a
000	001	100	000	010	000	011	000	001	100	000

3 × 11 = 33 bits

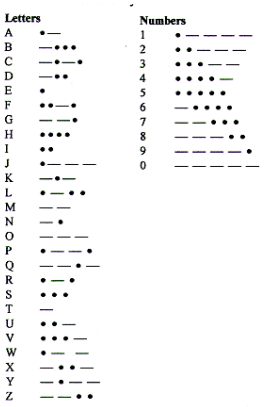
Variable Length Encoding

Variable-length encoding. Use different number of bits to encode different characters.

Ex. Morse code.

Ambiguity. • • • – – – • • •

- SOS
- IAMIE
- EEWNI
- T7O



Uniquely Decodable Codes

Variable-length encoding. Use different number of bits to encode different characters.

Q. How do we avoid ambiguity?

A1. Append special stop symbol to each codeword.

A2. Ensure no encoding is a prefix of another.

101 is a prefix of 1011

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

variable length coding: 28 bits

char	encoding
a	0
b	111
c	1011
d	100
r	110
!	1010

ITU-T T4 Group 3 Fax

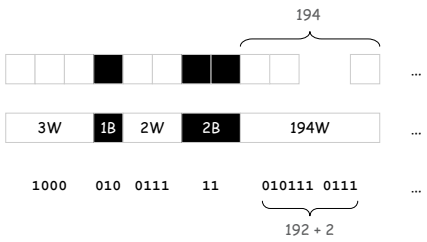
Group 3 fax. Transmit image comprised of up to 1728 pels per line, typically mostly white.

picture element = black or white

RLE. Compute run-lengths of white and black pels.

Prefix-free code. Encode run-lengths using following prefix-free code.

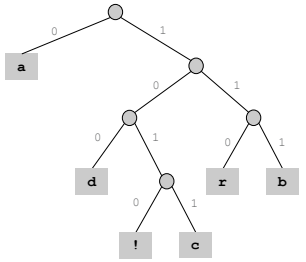
run	white	black
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
…	…	…
63	00110100	000001100111
64	11011	0000001111
128	10010	000011001000
192	010111	000011001001
…	…	…
1728	010011011	0000001100101



Prefix-Free Code: Encoding and Decoding

How to represent? Use a binary trie.

- Symbols are stored in leaves.
- Encoding is path to leaf.



Encoding.

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

Decoding.

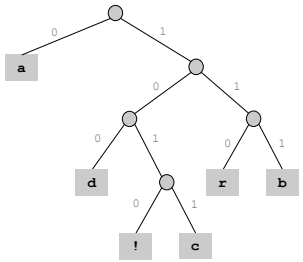
- Start at root of tree.
- Go left if bit is 0; go right if 1.
- If leaf node, print symbol and return to root.

char	encoding
a	0
b	111
c	1011
d	100
r	110
!	1010

How to Transmit the Trie

How to transmit the trie?

- Send preorder traversal of trie.
  - we use * as sentinel for internal nodes
  - what if there is no sentinel?
- Send number of characters to decode.
- Send bits (packed 8 to the byte).




```
*a**d*!c*rb
12
011111001011010001111100101O
```

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

char	encoding
a	0
b	111
c	1011
d	100
r	110
!	1010

Prefix-Free Decoding Implementation

```
public class HuffmanDecoder {
   private Node root = new Node();

   private class Node {
      char ch;
      Node left, right;

      Node() {
         ch = StdIn.readChar();
         if (ch == '*') {
            left  = new Node();
            right = new Node();
         }
      }

      boolean isInternal() { }
   }
```

build tree from preorder traversal

*a**d*!c*rb

Prefix-Free Decoding Implementation

```
public void decode() {
   int N = StdIn.readInt();
   for (int i = 0; i < N; i++) {
      Node x = root;
      while (x.isInternal()) {
         char bit = StdIn.readChar();
         if      (bit == '0') x = x.left;
         else if (bit == '1') x = x.right;
      }
      System.out.print(x.ch);
   }
}
```

use bits in real applications instead of chars

```
12
011111001011010001111001010
```


Huffman Codes

David Huffman

Huffman Coding

Q. How to create a good prefix-free code?

A. Huffman code. [David Huffman, 1950]

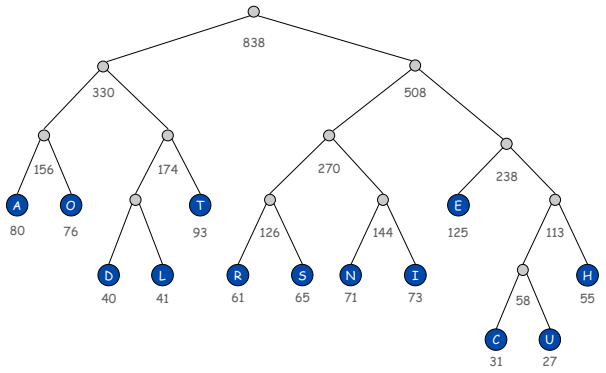
To compute Huffman code:

- Count frequencies p_s for each symbol s in message.
- Start with a node 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, GZIP, ...

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 Construction

```
// tabulate frequencies
int[] freq = new int[128];
for (int i = 0; i < input.length(); i++)
   freq[input.charAt(i)]++;

// initialize priority queue with singleton elements
MinPQ<Node> pq = new MinPQ<Node>();
for (int i = 0; i < 128; i++)
   if (freq[i] > 0)
      pq.insert(new Node((char) i, freq[i], null, null));

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

sentinel character; frequency count; two subtrees

Huffman Encoding

Theorem. [Huffman] Huffman coding is optimal prefix-free code.
Corollary. "Greed is good."

no prefix free code uses fewer bits

Implementation.

- Pass 1: tabulate symbol frequencies and build trie
- Pass 2: encode file by traversing trie or lookup table.

Running time. Use binary heap $\Rightarrow$ $O(M + N \log N)$.

input size; distinct symbols

ITU-T T4 Group 3 Fax: Revisited

Group 3 fax. Transmit image comprised of up to 1728 pels per line.

run	white	black
0	00110101	0000110111
1	000111	010
…	…	…
63	00110100	000001100111
64	11011	0000001111
128	10010	000011001000
…	…	…
1728	010011011	0000001100101

Q. Why this prefix-free code?
A. Huffman code based on frequency of each run-length over a large number of documents.

Limits on Compression

Claude Shannon

What Data Can be Compressed?

Theorem. Impossible to losslessly compress all files.

Pf.

- Consider all 1,000 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 be encoded with ≤ 500 bits!

A Difficult File To Compress

One million pseudo-random characters (a - p)

```
fclkkacifobjofmkgdcoiicnfmcpcjfccabckjamolnihkbgobcjbngjiceeelpfgcjiihppenefllhglfemdemgahlbpi
ggmllmnefnhjelmgjncjcidlhkglhceninidmmgnobkeglpnadanfbecoonbiehglmpnhkkamdffpacjmgojmcaabpcjce
cplfbgamlidceklhfkkmioljdnoaagiheiapaimlcnlljniggpeanbmojgkccogpmkmoifioeikefjidbadgdcepnhdpfj
aeeapdjeofklpdeghidbgcaiemajllhnndigeihbebifemacfadnknhlbgincpmimdogimgeeomgeljfjgklkdgnhafoho
npjbmlkapddhmepdnckeajebmeknmeejnmenbmnnfefdbhpmigbbjknjmobimamjjaaaffhlhiggaljbaijnebidpaeigd
goghcihodnlhahllhhoojdfacnhadhgkfahmeaebccacgeojgikcoapknlomfignanedmajinlompjoaifiaejbcjcdibp
kofcbmjiobbpdhfilfajkhfmppcngdneeinpnfafaeladbhhifechinknpdnplamackphekokigpddmmjnbngklhibohdf
eaggmclllmdhafkldmimdbplggbbejkcmhlkjocjjlcngckfpfakmnpiaanffdjdlleiniilaenbnikgfnjfcophbgkhdg
mfpoehfmkbpiaignphogbkelphobonmfghpdgmkfedkfkchceeldkcofaldinljjcgafimaanelmfkokcjekefkbmegcgj
ifjcpjppnabldjoaafpbdafifgcoibbcmoffbbgigmngefpkmbhbghlbdjngenldhgnfbdlcmjdmoflhcogfjoldfjpaok
epndejmnbiealkaofifekdjkgedgdlgbioacflfjlafbcaemgpjlagbdgilhcfdcamhfmppfgohjphlmhegjechgdpkklj
pndphfcnnganmbmnggpphnckbieknjhilafkegboilajdppcodpeoddldjfcpialoalfeomjbphkmhnpdmcpgkgeaohfdm
cnegmibjkajcdcpjcpgjminhhakihfgiiachfepffnilcooiciepoapmdjniimfbolchkibkbmhbkgconimkdchahcnhap
fdkiapikencegcjapkjkfljgdlmgncpbakhjidapbldcgeekkjaoihbnbigmhboengpmedliofgioofdcphelapijcegej
gcldcfodikalehbccpbbcfakkblmoobdmdgdkafbbkjnidoikfakjclbchambcpaepfeinmenmpoodadoecbgbmfkkeabi
laoeoggghoekamaibhjibefmoppbhfbhffapjnodlofeihmjahmeipejlfhloefgmjhjnlomapjakhhjpncomippeanbik
khekpcfgbgkmklipfbiikdkdcbolofhelipbkbjmjfoempccneaebklibmcaddlmjdcajpmhhaeedbbfpjafcndianlfcj
mmbfncpdcccodeldhmnbdjmeajmboclkggojghlohlbhgjkhkmclohkgjamfmcchkchmiadjgjhjehflcbklfifackbecg
joggpbkhlcmfhipflhmnmifpjmcoldbeghpcekhgmnahijpabnomnokldjcpppbcpgcjofngmbdcpeeeiiiclmbbmfjkhl
anckidhmbeanmlabncnccpbhoafajjicnfeenppoekmlddholnbdjapbfcajblbooiaepfmmeoafedflmdcbaodgeahimc
gpcammjljoebpfmghogfckgmomecdipmodbcempidfnlcggpgbffoncajpncomalgoiikeolmigliikjkolgolfkdgiijj
iooiokdihjbbofiooibakadjnedlodeeiijkliicnioimablfdpjiafcfineecbafaamheiipegegibiooclmhjekfikf
effmddhoakllnifdhckmbonbchfhhclecjamjildonjjdpifngbojianpljahpkindkdoanlldcbmlmhjfomifhmncikol
jjhebidjdphpdepibfgdonjljfgifimniipogockpidamnkcpipglafmlmoacjibognbplejnikdoefccdpfkomkimffgj
gielocdemnblimfmbkfbhkelkpfoheokfofochbmifleecbglmnfbnfncjmefnihdcoeiefllemnohlfdcmbdfebdmbeeb
balggfbajdamplphdgiimehglpikbipnkkecekhilchhhfaeafbbfdmcjojfhpponglkfdmhjpcieofcnjgkpibcbiblfp
njlejkcppbhopohdghljlcokhdoahfmlglbdkliajbmnkkfcoklhlelhjhoiginaimgcabcfebmjdnbfhohkjphnklcbhc
jpgbadakoecbkjcaebbanhnfhpnfkfbfpohmnkligpgfkjadomdjjnhlnfailfpcmnololdjekeolhdkebiffebajjpclg
hllmemegnckmkkeoogilijmmkomllbkkabelmodcohdhppdakbelmlejdnmbfmcjdebefnjihnejmnogeeafldabjcgfo
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() * 16);
         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.

Q. How do we know if our algorithm is doing well?

A. Want lower bound on # bits required by any compression scheme.

Language Model

Q. How compression algorithms work?

A. Exploit statistical biases of input messages.

- White patches occur in typical images.
- Word `Princeton` occurs more frequently than `Yale`.

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.

Ex. Order 0 Markov model: each symbol s generated independently at random, with fixed probability p(s).

Entropy

Entropy. [Shannon, 1948] $H(S) = -\sum_{s \in S} p(s) \log_2 p(s)$

- Information content of symbol s is proportional to $-\log_2 p(s)$.
- 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)	p(f)	H(S)
Fair die	1/6	1/6	1/6	1/6	1/6	1/6	2.585

Entropy and Compression

Theorem. [Shannon, 1948] If data source is an order 0 Markov model, any compression scheme must use ≥ H(S) bits per symbol on average.

- Cornerstone result of information theory.
- Ex: to transmit results of fair die, need ≥ 2.58 bits per roll.

Theorem. [Huffman, 1952] If data source is an order 0 Markov model, Huffman code uses ≤ H(S) + 1 bits per symbol on average.

Q. Is there any hope of doing better than Huffman coding?

A. Yes. Huffman wastes up to 1 bit per symbol.
  - if H(S) is close to 0, this matters
  - can do better with "arithmetic coding"

A. 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

Q. How much information is in each character of English language?

Q. How can we measure it?

model = English text

A. [Shannon's 1951 experiment]

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

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

- Shannon's estimate = 0.6 - 1.3 bit/char.

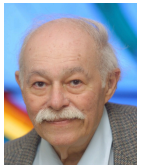
Lossless Compression Ratio for Calgary Corpus

Year	Scheme	Bits / char
1967	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

next assignment

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

Lempel-Ziv-Walsh



Abraham Lempel

Jacob Ziv

Statistical Methods

Static model. Same model for all texts.

- Fast.
- Not optimal: different texts have different statistical properties.
- Ex: ASCII, Morse code.

Dynamic model. Generate model based on text.

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

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

- More accurate modeling produces better compression.
- Decoding must start from beginning.
- Ex: LZW.

LZW Algorithm

Lempel-Ziv-Welch. [variant of LZ78]

- Create ST and associate an integer with each useful string.
- When input matches string in ST, output associated integer.

Encoding.

- Find longest string s in ST that is a prefix of remaining part of string to compress.
- Output integer associated with s.
- Add s · x to dictionary, where x is next char in string to compress.

Ex. Dictionary: a, aa, ab, aba, abb, abaa, abaab, abaaa,

- String to be compressed: abaababbb…
- s = abaab, x = a.
- Output integer associated with s; 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.

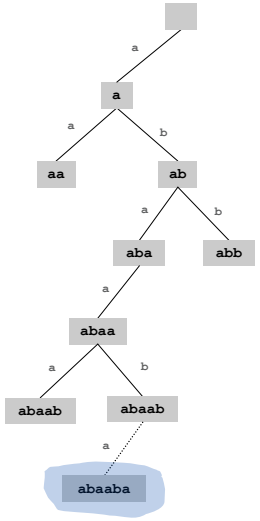
- Use trie to create symbol table on-the-fly.
- Note that prefix of every word is also in ST.

Encode.

- Lookup string suffix in trie.
- Output ST index at bottom.
- Add new node to bottom of trie.

Decode.

- Lookup index in array
- Output string
- Insert string + next letter.



abaaba…

LZW Encoder: Java Implementation

```
public class LZWEncoder {
   public static void main(String[] args) {
      String text = StdIn.readAll();
      StringST<Integer> st = new StringST<Integer>();
      int i;
      for (i = 0; i < 256; i++) {
         String s = Character.toString((char) i);
         st.put(s, i);
      }
      while (text.length() > 0) {
         String s = st.prefix(text);
         System.out.println(st.get(s));
         int length = s.length();
         if (length < text.length())
            st.put(text.substring(0, length + 1), i++);
         text = text.substring(length);
      }
   }
}
```

longest prefix match

in real applications, encode integers in binary

not the most efficient way

LZW Decoder: Java Implementation

```
public class LZWDecoder {
   public static void main(String[] args) {
      ST<Integer, String> st = new ST<Integer, String>();
      int i;
      for (i = 0; i < 256; i++) {
         String s = Character.toString((char) i);
         st.put(i, s);
      }

      int code = StdIn.readInt();
      String prev = st.get(code);
      System.out.print(prev);

      while (!StdIn.isEmpty()) {
         code = StdIn.readInt();
         String s = st.get(code);
         if (i == code) s = prev + prev.charAt(0);
         System.out.print(s);
         st.put(i++, prev + s.charAt(0));
         prev = s;
      }
   }
}
```

in real applications, integers will be encoded in binary

special case, e.g., for "abababababab"

LZW Implementation Details

What to do when ST gets too large?

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

LZW in the Real World

Lempel-Ziv and friends.

- LZ77.
- LZ78.
- LZW.
- Deflate = LZ77 variant + Huffman.

LZ77 not patented ⇒ widely used in open source
LZW patent #4,558,302 expired in US on June 20, 2003
some versions copyrighted

PNG: LZ77.
Winzip, gzip, jar: deflate.
Unix compress: LZW.
Pkzip: LZW + Shannon-Fano.
GIF, TIFF, V.42bis modem: LZW.
Google: 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. [not covered in this course]

- JPEG, MPEG, MP3.
- FFT, wavelets, fractals, SVD, …

Limits on compression. Shannon entropy.

Theoretical limits closely match what we can achieve in practice!