CS2010: Data Structures and Algorithms II

Compression algorithms

Ivana.Dusparic@scss.tcd.ie

Compression algorithms

- Intro to compression algorithms – Sillicon Valley style (http://www.imdb.com/title/tt2575988/)

- https://www.youtube.com/watch?v=l49MHwooaVQ

- Shannon coding
- Huffman
- Lempel-Ziv
- Burrows-Wheeler

- Weissman score?

Data compression

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, …

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

Applications

Generic file compression.

- Files: GZIP, BZIP, 7z.
- Archivers: PKZIP.
- File systems: NTFS, HFS+, ZFS.

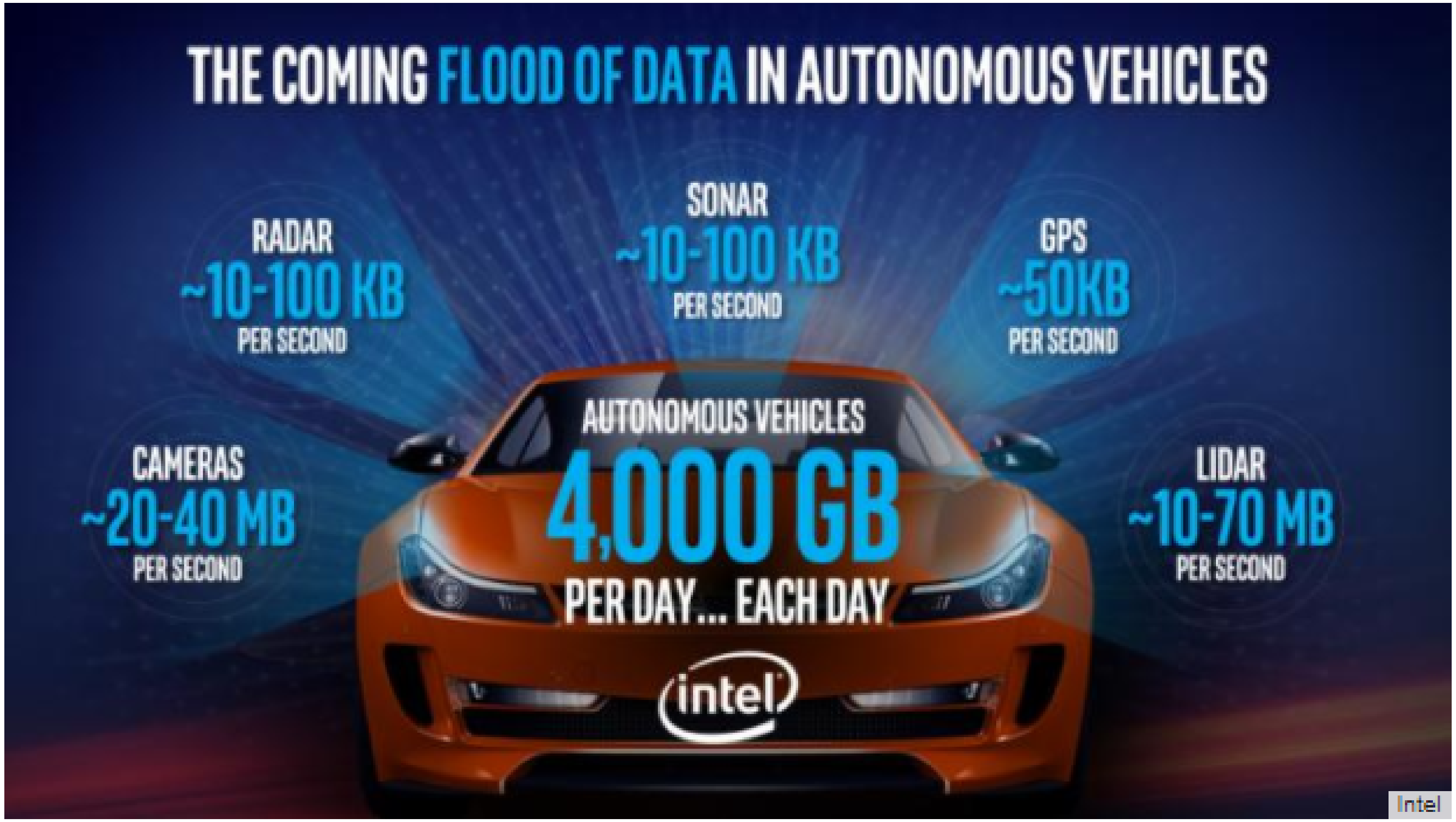
Multimedia.

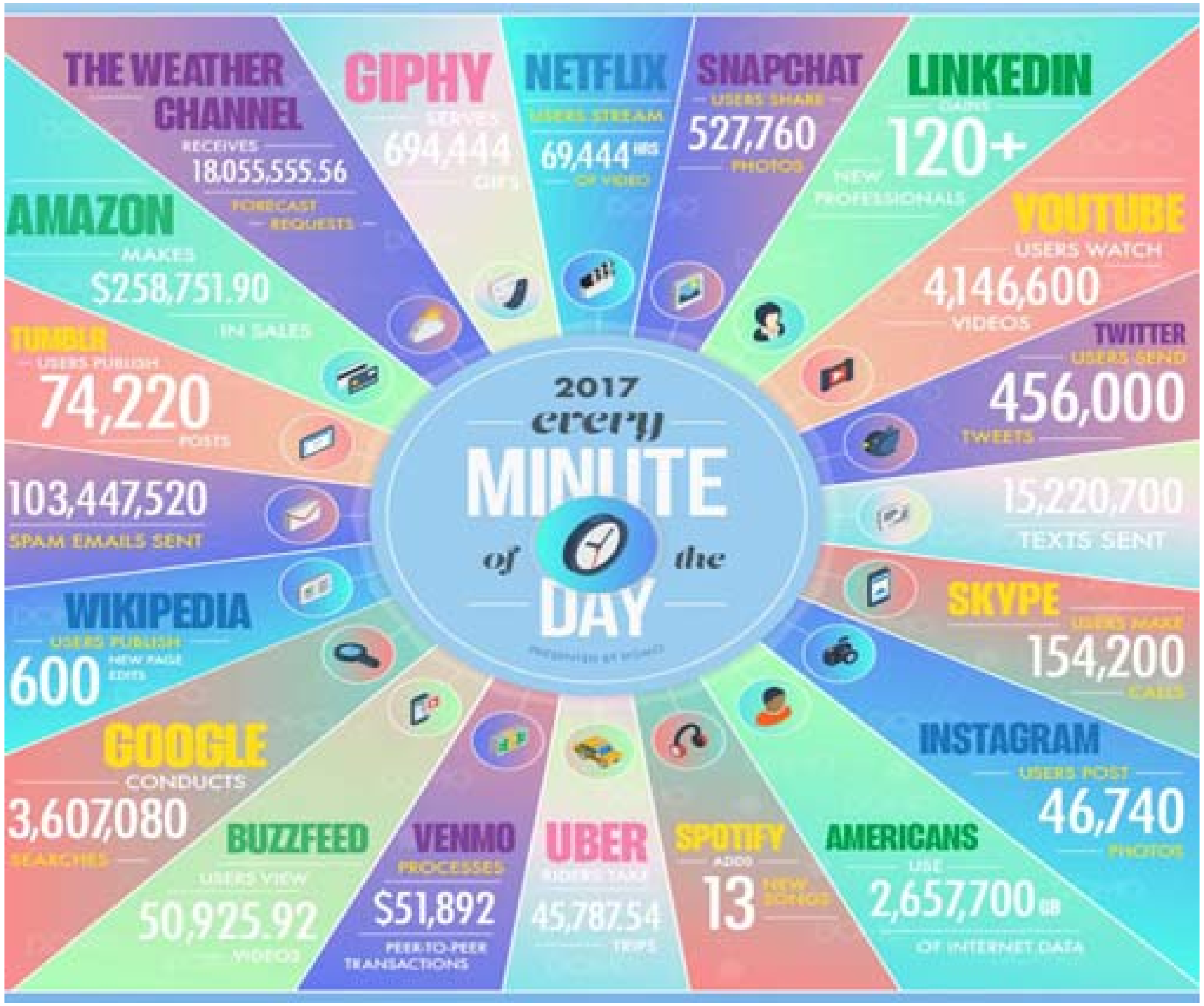
- Images: GIF, JPEG.
- Sound: MP3.
- Video: MPEG, DivX™, HDTV.

Communication.

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

Databases. Google, Facebook, ...


THE COMING FLOOD OF DATA IN AUTONOMOUS VEHICLES
RADAR
~10-100 KB
PER SECOND
SONAR
~10-100 KB
PER SECOND
GPS
~50KB
PER SECOND
CAMERAS
~20-40 MB
PER SECOND
AUTONOMOUS VEHICLES
4,000 GB
PER DAY... EACH DAY
LIDAR
~10-70 MB
PER SECOND
intel
Intel



https://www.domo.com/ Data never sleeps 5.0

Lossy compression

- Essential data removed so impossible to restore full original data
- Used for images, audio, video
  - Video up to 100:1, with little visible quality loss
  - Audio 10:1 with imperceptible loss of quality
  - Images can go up to 10:1 but the quality loss is more noticeable

Lossless compression and expansion

Message. Binary data B we want to compress.

Compress. Generates a "compressed" representation $C(B)$.

Expand. Reconstructs original bitstream B.

uses fewer bits (you hope)

Basic model for data compression

Compression ratio. Bits in $C(B)$ / bits in B.

Ex. 50–75% or better compression ratio for natural language.

Brief History

- http://ethw.org/History_of_Lossless_Data_Compression_Algorithms
- Morse code - the most common letters in the English language such as "e" and "t" are given shorter Morse codes
- Shannon-Fano coding - codes to symbols in a given block of data based on the probability of the symbol occuring. The probability is of a symbol occuring is inversely proportional to the length of the code, resulting in a shorter way to represent the data
- Huffman - The key difference between Shannon-Fano coding and Huffman coding is that in the former the probability tree is built bottom-up, creating a suboptimal result, and in the latter it is built top-down.
- LZ77 – first to use a dictionary to compress data - substitutes a reference to the string's position in the data structure
  - LZ77, LZ78, LZW, LZC
- Deflate – combination of Huffman and LZ77 (gzip, zip)
- Burrows-Wheeler Transform - transforms the input such that it can be more efficiently coded by a Run-Length Encoder or other secondary compression techniques

Data representation: genomic code

Genome. String over the alphabet `{ A, C, T, G }`.

Goal. Encode an N-character genome: `A T A G A T G C A T A G ...`

Standard ASCII encoding.

- 8 bits per char.
- $8\,N$ bits.

char	hex	binary
A	41	01000001
C	43	01000011
T	54	01010100
G	47	01000111

Two-bit encoding.

- 2 bits per char.
- $2\,N$ bits.

char	binary
A	00
C	01
T	10
G	11

Fixed-length code. k-bit code supports alphabet of size 2^k.

Amazing but true. Initial genomic databases in 1990s used ASCII.

Universal data compression

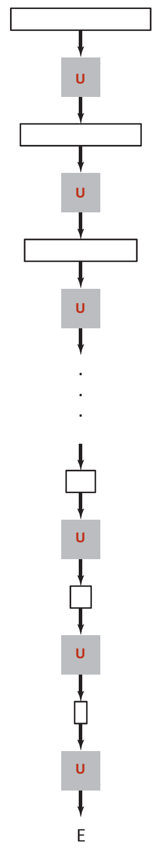
US Patent 5,533,051 on "Methods for Data Compression", which is capable of compression all files.

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.… "*

Universal data compression

Proposition. No algorithm can compress every bitstring.



Universal data compression?

Pf 1. [by contradiction]

- Suppose you have a universal data compression algorithm U that can compress every bitstream.
- Given bitstring B_0, compress it to get smaller bitstring B_1.
- Compress B_1 to get a smaller bitstring B_2.
- Continue until reaching bitstring of size 0.
- Implication: all bitstrings can be compressed to 0 bits!

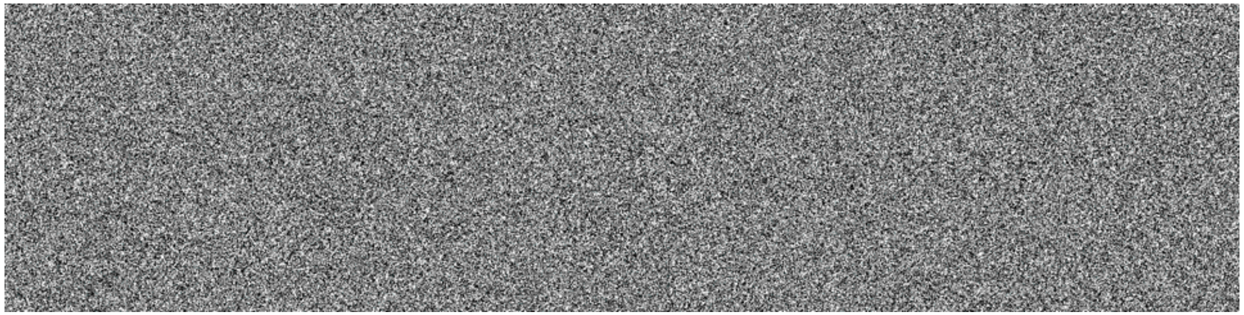
Pf 2. [by counting]

- Suppose your algorithm that can compress all $1{,}000$-bit strings.
- 2^{1000} possible bitstrings with $1{,}000$ bits.
- Only $1 + 2 + 4 + \ldots + 2^{998} + 2^{999}$ can be encoded with ≤ 999 bits.
- Similarly, only 1 in 2^{499} bitstrings can be encoded with ≤ 500 bits!

What can be compressed?

- Exploit redundancy/patterns in data
  - Eg colour repetition in images
  - Redundancy in language

```
% java RandomBits | java PictureDump 2000 500
```



```
1000000 bits
```

A difficult file to compress: one million (pseudo-) random bits

Run Length Encoding (RLE)

Run-length encoding

Simple type of redundancy in a bitstream. Long runs of repeated bits.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1

40 bits

Representation. 4-bit counts to represent alternating runs of 0s and 1s: 15 0s, then 7 1s, then 7 0s, then 11 1s.

$\underbrace{1\,1\,1\,1}_{15}\;\underbrace{0\,1\,1\,1}_{7}\;\underbrace{0\,1\,1\,1}_{7}\;\underbrace{1\,0\,1\,1}_{11}$ ← 16 bits (instead of 40)

Q. How many bits to store the counts?

A. We'll use 8 (but 4 in the example above).

Q. What to do when run length exceeds max count?

A. If longer than 255, intersperse runs of length 0.

Applications. JPEG, ITU-T T4 Group 3 Fax, ...

Example - colors

HTML / CSS Color Name	Hex Code #RRGGBB	Decimal Code (R,G,B)	binary
black	#000000	rgb(0,0,0)	0000 0000 0000 0000 0000 0000
white	#ffffff	Rgb(255,255,255)	1111 1111 1111 1111 1111 1111

An application: compress a bitmap

Typical black-and-white-scanned image.

- 300 pixels/inch.
- 8.5-by-11 inches.
- $300 \times 8.5 \times 300 \times 11 = 8.415$ million bits.

Observation. Bits are mostly white.

Typical amount of text on a page.

40 lines $\times$ 75 chars per line = 3,000 chars.

7 1s

```
% java BinaryDump 32 < q32x48.bin
00000000000000000000000000000000     32
00000000000000000000000000000000     32
00000000000000011111110000000000     15  7 10
00000000000011111111111111100000     12 15  5
00000000001111000011111111100000     10  4  4  9  5
00000000111100000000011111100000      8  4  9  6  5
00000001110000000000001111100000      7  3 12  5  5
00000011110000000000001111100000      6  4 12  5  5
00000111100000000000001111100000      5  4 13  5  5
00001111000000000000001111100000      4  4 14  5  5
00001111000000000000001111100000      4  4 14  5  5
00011110000000000000001111100000      3  4 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111110000000000000001111100000      2  5 15  5  5
00111111000000000000001111100000      2  6 14  5  5
00111111000000000000001111100000      2  6 14  5  5
00011111100000000000001111100000      3  6 13  5  5
00011111100000000000001111100000      3  6 13  5  5
00001111110000000000001111100000      4  6 12  5  5
00001111111000000000001111100000      4  7 11  5  5
00000111111100000000001111100000      5  7 10  5  5
00000011111111000000011111100000      6  8  7  6  5
00000001111111111111111111100000      7 20  5
00000000011111111111001111100000      9 11  2  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000001111100000     22  5  5
00000000000000000000011111110000     21  7  4
00000000000000000011111111111100     18 12  2
00000000000000000111111111111110     17 14  1
00000000000000000000000000000000     32
00000000000000000000000000000000     32

1536 bits
```

17 0s

A typical bitmap, with run lengths for each row

Huffman

ASCII table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Letter frequency in English Language

- Oxford concise dictionary
- The third column represents proportions, taking the least common letter (q) as equal to 1, eg E is 56.88 times more common than Q

E	11.1607%	56.88	M	3.0129%	15.36
A	8.4966%	43.31	H	3.0034%	15.31
R	7.5809%	38.64	G	2.4705%	12.59
I	7.5448%	38.45	B	2.0720%	10.56
O	7.1635%	36.51	F	1.8121%	9.24
T	6.9509%	35.43	Y	1.7779%	9.06
N	6.6544%	33.92	W	1.2899%	6.57
S	5.7351%	29.23	K	1.1016%	5.61
L	5.4893%	27.98	V	1.0074%	5.13
C	4.5388%	23.13	X	0.2902%	1.48
U	3.6308%	18.51	Z	0.2722%	1.39
D	3.3844%	17.25	J	0.1965%	1.00
P	3.1671%	16.14	Q	0.1962%	(1)

What about other languages?

› Eg Spanish

Character	Frequency
E	13,72%
A	11,72%
O	8,44%
S	7,20%
N	6,83%
R	6,41%
I	5,28%
L	5,24%
D	4,67%
T	4,60%
U	4,55%
C	3,87%
M	3,08%
P	2,89%
B	1,49%
H	1,18%
Q	1,11%
Y	1,09%
V	1,05%
G	1,00%
Ó	0,76%
Í	0,70%
F	0,69%
J	0,52%
Z	0,47%
Á	0,44%
É	0,36%
Ñ	0,17%
X	0,14%
Ú	0,12%
K	0,11%
W	0,04%
Ü	0,02%

Variable-length codes

Use different number of bits to encode different chars.

Ex. Morse code: • • • – – – • • •

Issue. Ambiguity.

SOS ?

V7 ?

IAMIE ?

EEWNI ?

In practice. Use a medium gap to separate codewords.

codeword for S is a prefix of codeword for V

Letters		Numbers	
A	• —	1	• — — — —
B	— • • •	2	• • — — —
C	— • — •	3	• • • — —
D	— • •	4	• • • • —
E	•	5	• • • • •
F	• • — •	6	— • • • •
G	— — •	7	— — • • •
H	• • • •	8	— — — • •
I	• •	9	— — — — •
J	• — — —	0	— — — — —
K	— • —		
L	• — • •		
M	— —		
N	— •		
O	— — —		
P	• — — •		
Q	— — • —		
R	• — •		
S	• • •		
T	—		
U	• • —		
V	• • • —		
W	• — —		
X	— • • —		
Y	— • — —		
Z	— — • •		

Variable-length codes

Q. How do we avoid ambiguity?

A. Ensure that no codeword is a prefix of another.

Ex 1. Fixed-length code.

Ex 2. Append special stop char to each codeword.

Ex 3. General prefix-free code.

Codeword table

key	*value*
!	101
A	0
B	1111
C	110
D	100
R	1110

Compressed bitstring

011111110011001000111111100101 ← *30 bits*

A B RA CA DA B RA !

Codeword table

key	*value*
!	101
A	11
B	00
C	010
D	100
R	011

Compressed bitstring

11000111101011100110001111101 ← *29 bits*

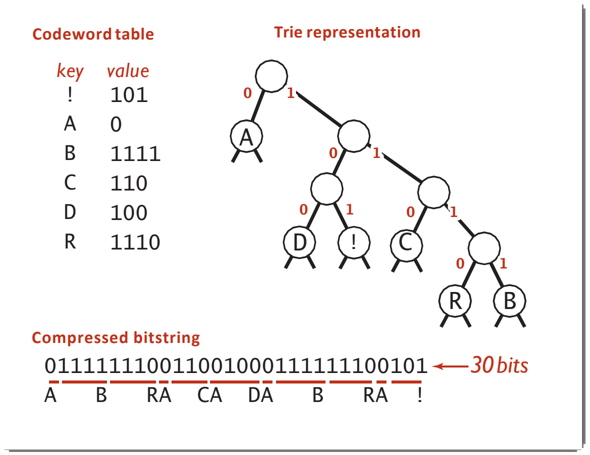
A B R A C A D A B R A !

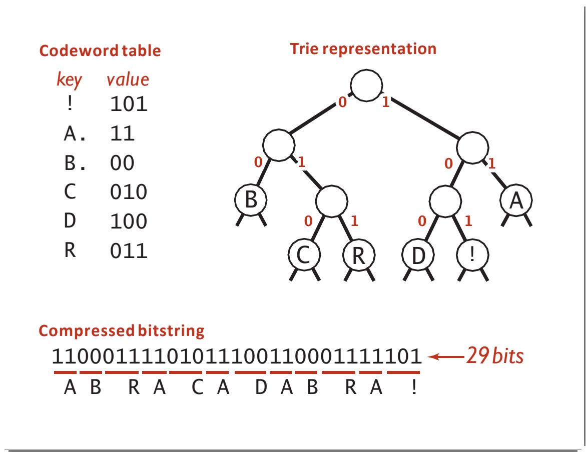
Prefix-free codes: trie representation

Q. How to represent the prefix-free code?

A. A binary trie!

- Chars in leaves.
- Codeword is path from root to leaf.





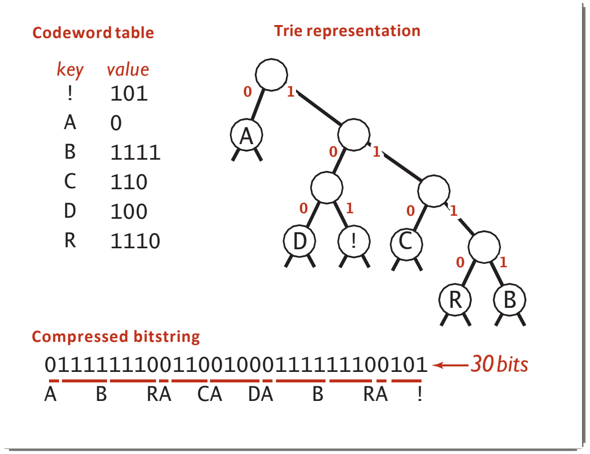
Prefix-free codes: compression and expansion

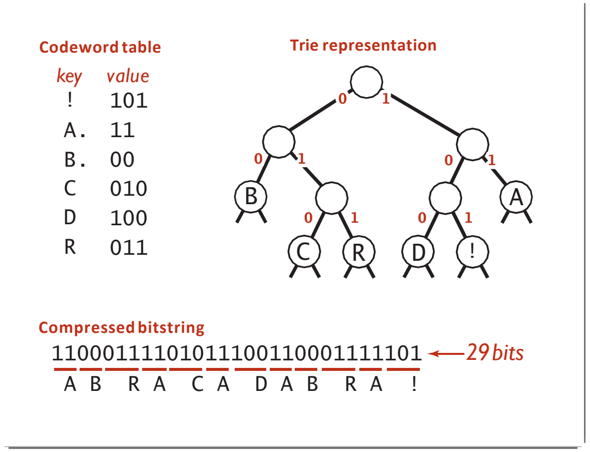
Compression.

- Method 1: start at leaf; follow path up to the root; print bits in reverse.
- Method 2: create ST of key-value pairs.

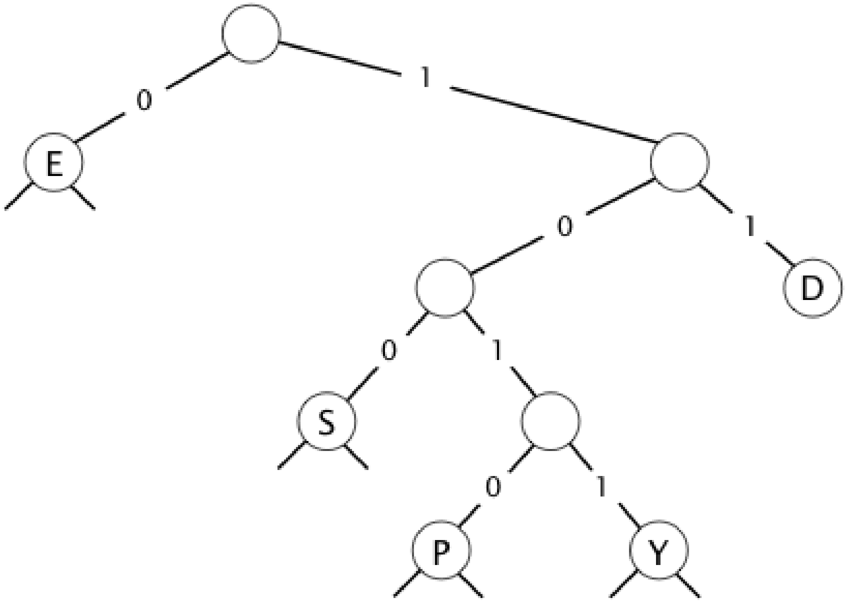
Expansion.

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





Consider the following trie representation of a prefix-free code. Which string is encoded by the compressed bit string 100101000111011?



Huffman trie node data type

```
private static class Node implements Comparable<Node>
{
   private final char ch;   // used only for leaf nodes
   private final int freq;  // used only for compress
   private final Node left, right;

   public Node(char ch, int freq, Node left, Node right)
   {
      this.ch    = ch;
      this.freq  = freq;
      this.left  = left;
      this.right = right;
   }

   public boolean isLeaf()
   {  return left == null && right == null; }

   public int compareTo(Node that)
   {  return this.freq - that.freq;   }
}
```

initializing constructor

is Node a leaf?

compare Nodes by frequency
(stay tuned)

Prefix-free codes: expansion

```
public void expand()
{
   Node root = readTrie();                    // read in encoding trie
   int N = BinaryStdIn.readInt();             // read in number of chars

   for (int i = 0; i < N; i++)
   {
      Node x = root;
      while (!x.isLeaf())                     // expand codeword for ith char
      {
         if (!BinaryStdIn.readBoolean())
            x = x.left;
         else
            x = x.right;
      }
      BinaryStdOut.write(x.ch, 8);
   }
   BinaryStdOut.close();
}
```

read in encoding trie

read in number of chars

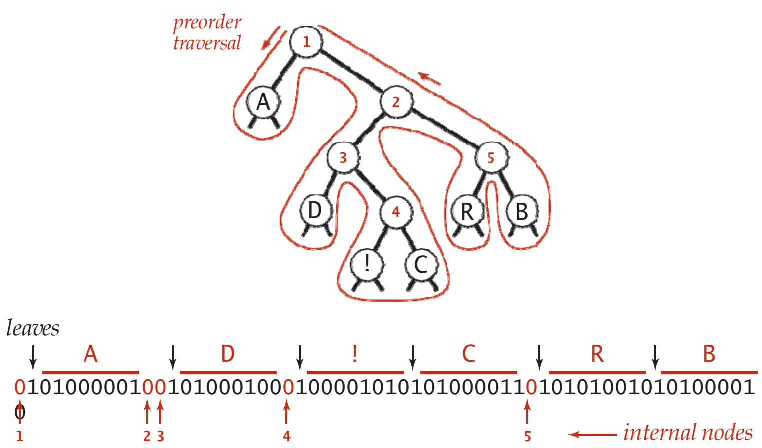
expand codeword for i^{th} char

Running time. Linear in input size N.

Prefix-free codes: how to transmit

Q. How to write the trie?

A. Write preorder traversal of trie; mark leaf and internal nodes with a bit.



Using preorder traversal to encode a trie as a bitstream

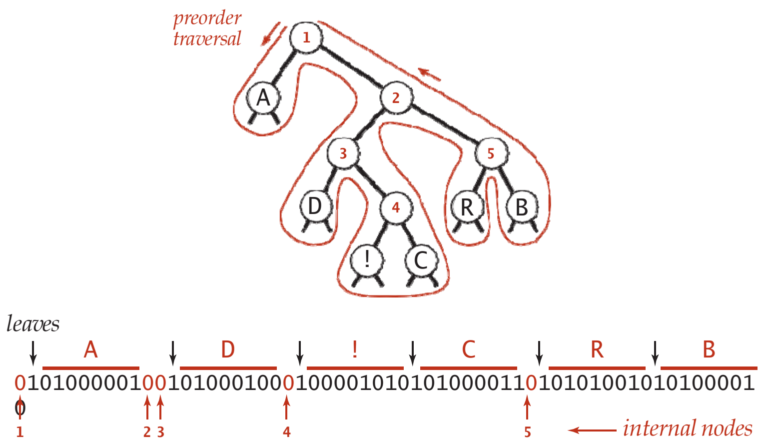
```
private static void writeTrie(Node x)
{
   if (x.isLeaf())
   {
      BinaryStdOut.write(true);
      BinaryStdOut.write(x.ch, 8);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(x.left);
   writeTrie(x.right);
}
```

Note. If message is long, overhead of transmitting trie is small.

Prefix-free codes: how to transmit

Q. How to read in the trie?

A. Reconstruct from preorder traversal of trie.



Using preorder traversal to encode a trie as a bitstream

```
private static Node readTrie()
{
   if (BinaryStdIn.readBoolean())
   {
      char c = BinaryStdIn.readChar(8);
      return new Node(c, 0, null, null);
   }
   Node x = readTrie();
   Node y = readTrie();
   return new Node('\0', 0, x, y);
}
```

used only for leaf nodes

Shannon-Fano codes

Q. How to find best prefix-free code?

Shannon-Fano algorithm:

- Partition symbols S into two subsets S_0 and S_1 of (roughly) equal freq.
- Codewords for symbols in S_0 start with 0; for symbols in S_1 start with 1.
- Recur in S_0 and S_1.

char	freq	encoding
A	5	0...
C	1	0...

S_0 = codewords starting with 0

char	freq	encoding
B	2	1...
D	1	1...
R	2	1...
!	1	1...

S_1 = codewords starting with 1

Problem 1. How to divide up symbols?

Problem 2. Not optimal!

Huffman algorithm demo

- Count frequency for each character in input.

char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	

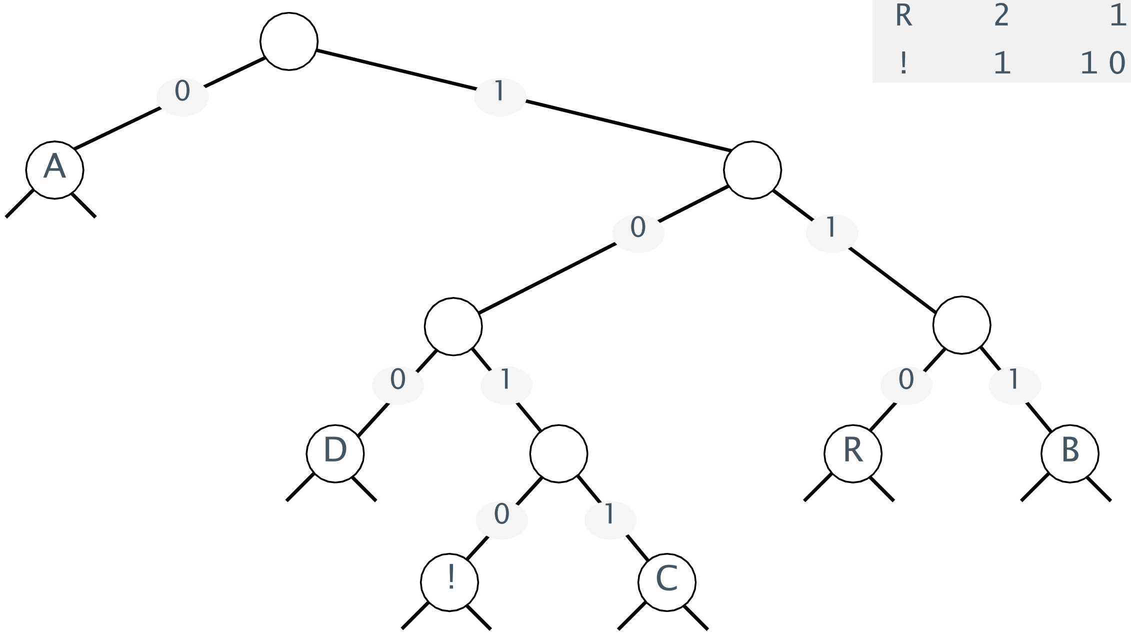
input

A B R A C A D A B R A !

Huffman demo

Huffman algorithm demo

char	freq	encoding
A	5	0
B	2	1 1 1
C	1	1 0 1 1
D	1	1 0 0
R	2	1 1 0
!	1	1 0 1 0



Huffman codes

Q. How to find best prefix-free code?

Huffman algorithm:

- Count frequency `freq[i]` for each char `i` in input.
- Start with one node corresponding to each char `i` (with weight `freq[i]`).
- Repeat until single trie formed:
  - select two tries with min weight `freq[i]` and `freq[j]`
  - merge into single trie with weight `freq[i] + freq[j]`

Applications:

Constructing a Huffman encoding trie: Java implementation

```
private static Node buildTrie(int[] freq)
{
   MinPQ<Node> pq = new MinPQ<Node>();
   for (char i = 0; i < R; i++)
      if (freq[i] > 0)
         pq.insert(new Node(i, freq[i], null, null));

   while (pq.size() > 1)
   {
      Node x = pq.delMin();
      Node y = pq.delMin();
      Node parent = new Node('\0', x.freq + y.freq, x, y);
      pq.insert(parent);
   }

   return pq.delMin();
}
```

initialize PQ with singleton tries

merge two smallest tries

not used for internal nodes

total frequency

two subtries

Huffman encoding summary

Proposition. [Huffman 1950s] Huffman algorithm produces an optimal prefix-free code.

no prefix-free code uses fewer bits

Pf. See textbook.

Implementation.

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

Running time. Using a binary heap $\Rightarrow$ $N + R \log R$.

input size

alphabet size

Q. Can we do better? [stay tuned]

Exercise

- Build a Huffman trie for encoding the following text

"I love data structures"

- Good demo/visualisation on https://people.ok.ubc.ca/ylucet/DS/Huffman.html

LZW

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 compression example

input											B	
matches		B	R	A	C	A	D	A B	R A	B R	A B R	A
value	A 41	42	52	41	43	41	44	81	83	82	88	41 80

LZW compression for A B R A C A D A B R A B R A B R A

key	value	key	value	key	value
⋮	⋮	AB	81	DA	87
A	41	BR	82	ABR	88
B	42	RA	83	RAB	89
C	43	AC	84	BRA	8A
D	44	CA	85	ABRA	8B
⋮	⋮	AD	86		

codeword table

Lempel-Ziv-Welch compression

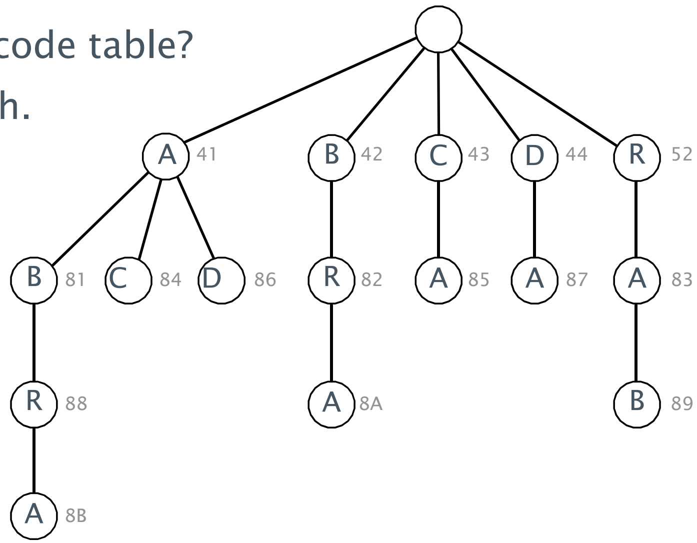
LZW compression.

- Create ST associating W-bit codewords with string keys.
- Initialize ST with codewords for single-char keys.
- Find longest string s in ST that is a prefix of unscanned part of input.
- Write the W-bit codeword associated with s.
- Add $s + c$ to ST, where c is next char in the input.

longest prefix match

Q. How to represent LZW compression code table?

A. A trie to support longest prefix match.



LZW compression: Java implementation

```
public static void compress()
{
   String input = BinaryStdIn.readString();          // read in input as a string

   TST<Integer> st = new TST<Integer>();
   for (int i = 0; i < R; i++)
       st.put("" + (char) i, i);                      // codewords for single-char, radix R keys
   int code = R+1;

   while (input.length() > 0)
   {
      String s = st.longestPrefixOf(input);           // find longest prefix match s
      BinaryStdOut.write(st.get(s), W);               // write W-bit codeword for s
      int t = s.length();
      if (t < input.length() && code < L)
         st.put(input.substring(0, t+1), code++);     // add new codeword
      input = input.substring(t);                     // scan past s in input
   }

   BinaryStdOut.write(R, W);                          // write "stop" codeword
   BinaryStdOut.close();                              // and close input stream
}
```

LZW expansion example

value	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A	D	A B	R A	B R	A B R	A	

LZW expansion for 41 42 52 41 43 41 44 81 83 82 88 41 80

key	value
⋮	⋮
41	A
42	B
43	C
44	D
⋮	⋮

key	value
81	AB
82	BR
83	RA
84	AC
85	CA
86	AD

key	value
87	DA
88	ABR
89	RAB
8A	BRA
8B	ABRA

codeword table

LZW expansion

LZW expansion

LZW expansion.

- Create ST associating string values with W-bit keys.
- Initialize ST to contain single-char values.
- Read a W-bit key.
- Find associated string value in ST and write it out.
- Update ST.

Q. How to represent LZW expansion code table?

A. An array of size 2^W.

key	value
⋮	⋮
65	A
66	B
67	C
68	D
⋮	⋮
129	AB
130	BR
131	RA
132	AC
133	CA
134	AD
135	DA
136	ABR
137	RAB
138	BRA
139	ABRA
⋮	⋮

LZW Exercise

- What is LZW compression of ABABABA?

LZW example: tricky case

input	A	B	A	B	A	B	A	
matches	A	B	A B		A B A			
value	41	42	81		83			80

LZW compression for ABABABA

key	value
⋮	⋮
A	41
B	42
C	43
D	44
⋮	⋮

key	value
AB	81
BA	82
ABA	83

codeword table

LZW example: tricky case

value	41	42	81	83	80
output	A	B	A B	A B A	

need to know which key has value 83 before it is in ST!

LZW expansion for 41 42 81 83 80

key	value
⋮	⋮
41	A
42	B
43	C
44	D
⋮	⋮

key	value
81	AB
82	BA
83	ABA

codeword table

LZW implementation details

How big to make ST?

- How long is message?
- Whole message similar model?
- [many other variations]

What to do when ST fills up?

- Throw away and start over. [GIF]
- Throw away when not effective. [Unix compress]
- [many other variations]

Why not put longer substrings in ST?

- [many variations have been developed]

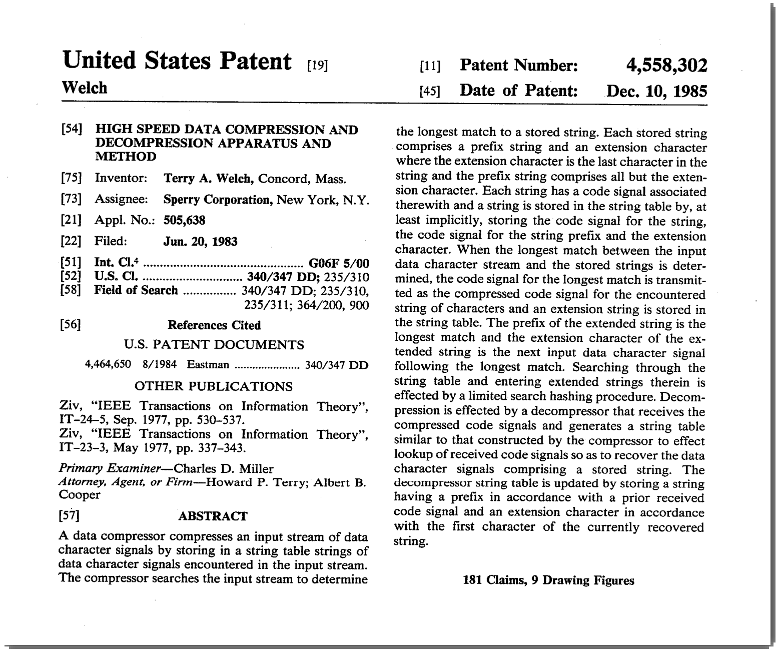
LZW in the real world

Lempel-Ziv and friends.

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

LZ77 not patented ⇒ widely used in open source

LZW patent #4,558,302 expired in U.S. on June 20, 2003



United States Patent [19]
Welch

[11] **Patent Number: 4,558,302**
[45] **Date of Patent: Dec. 10, 1985**

[54] **HIGH SPEED DATA COMPRESSION AND DECOMPRESSION APPARATUS AND METHOD**

[75] Inventor: **Terry A. Welch,** Concord, Mass.

[73] Assignee: **Sperry Corporation,** New York, N.Y.

[21] Appl. No.: **505,638**

[22] Filed: **Jun. 20, 1983**

[51] **Int. Cl.⁴** **G06F 5/00**
[52] **U.S. Cl.** .... **340/347 DD;** 235/310
[58] **Field of Search** .... 340/347 DD; 235/310, 235/311; 364/200, 900

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,464,650 8/1984 Eastman .... 340/347 DD

OTHER PUBLICATIONS

Ziv, "IEEE Transactions on Information Theory", IT-24-5, Sep. 1977, pp. 530-537.
Ziv, "IEEE Transactions on Information Theory", IT-23-3, May 1977, pp. 337-343.

Primary Examiner—Charles D. Miller
Attorney, Agent, or Firm—Howard P. Terry; Albert B. Cooper

[57] **ABSTRACT**

A data compressor compresses an input stream of data character signals by storing in a string table strings of data character signals encountered in the input stream. The compressor searches the input stream to determine the longest match to a stored string. Each stored string comprises a prefix string and an extension character where the extension character is the last character in the string and the prefix string comprises all but the extension character. Each string has a code signal associated therewith and a string is stored in the string table by, at least implicitly, storing the code signal for the string, the code signal for the string prefix and the extension character. When the longest match between the input data character stream and the stored strings is determined, the code signal for the longest match is transmitted as the compressed code signal for the encountered string of characters and an extension string is stored in the string table. The prefix of the extended string is the longest match and the extension character of the extended string is the next input data character signal following the longest match. Searching through the string table and entering extended strings therein is effected by a limited search hashing procedure. Decompression is effected by a decompressor that receives the compressed code signals and generates a string table similar to that constructed by the compressor to effect lookup of received code signals so as to recover the data character signals comprising a stored string. The decompressor string table is updated by storing a string having a prefix in accordance with a prior received code signal and an extension character in accordance with the first character of the currently recovered string.

181 Claims, 9 Drawing Figures

LZW in the real world

Lempel-Ziv and friends.

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

Unix compress, GIF, TIFF, V.42bis modem: LZW.

zip, 7zip, gzip, jar, png, pdf: deflate / zlib.

iPhone, Sony Playstation 3, Apache HTTP server: deflate / zlib.

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
1994	SAKDC	2.47
1994	PPM	2.34
1995	Burrows-Wheeler	2.29
1997	BOA	1.99
1999	RK	1.89

Lossless data compression benchmarks

The **Calgary corpus** is a collection of text and binary data files, commonly used for comparing data compression algorithms.

http://www.data-compression.info/Corpora/CalgaryCorpus/

Calgary Corpus

Size (bytes)	File name	Description
111,261	BIB	ASCII text in UNIX "refer" format – 725 bibliographic references.
768,771	BOOK1	unformatted ASCII text – Thomas Hardy: Far from the Madding Crowd.
610,856	BOOK2	ASCII text in UNIX "troff" format – Witten: Principles of Computer Speech.
102,400	GEO	32 bit numbers in IBM floating point format – seismic data.
377,109	NEWS	ASCII text – USENET batch file on a variety of topics.
21,504	OBJ1	VAX executable program – compilation of PROGP.
246,814	OBJ2	Macintosh executable program – "Knowledge Support System".
53,161	PAPER1	UNIX "troff" format – Witten, Neal, Cleary: Arithmetic Coding for Data Compression.
82,199	PAPER2	UNIX "troff" format – Witten: Computer (in)security.
513,216	PIC	1728 x 2376 bitmap image (MSB first): text in French and line diagrams.
39,611	PROGC	Source code in C – UNIX compress v4.0.
71,646	PROGL	Source code in Lisp – system software.
49,379	PROGP	Source code in Pascal – program to evaluate PPM compression.
93,695	TRANS	ASCII and control characters – transcript of a terminal session.

Canterbury corpus

Size (bytes)	File name	Description
152,089	alice29.txt	English text
125,179	asyoulik.txt	Shakespeare
24,603	cp.html	HTML source
11,150	fields.c	C source
3,721	grammar.lsp	LISP source
1,029,744	kennedy.xls	Excel spreadsheet
426,754	lcet10.txt	Technical writing
481,861	plrabn12.txt	Poetry
513,216	ptt5	CCITT test set
38,240	sum	SPARC executable
4,227	xargs.1	GNU manual page

Burrows - Wheeler

Move to front coding

- The goal of the Burrows-Wheeler transform is not to compress a message, but rather to transform it into a form that is more amenable to compression.
- used as a sub-step in several algorithm
- preprocess the message sequence by converting it into a sequence of integers
- each symbol in the data is replaced by its index in the stack of "recently used symbols"
- For example, long sequences of identical symbols are replaced by as many zeroes, whereas when a symbol that has not been used in a long time appears, it is replaced with a large number.

Example – compress bananaaa

Iteration	Sequence	List
bananaaa	1	(abcdefghijklmnopqrstuvwxyz)
b**a**nanaaa	1,1	(bacdefghijklmnopqrstuvwxyz)
ba**n**anaaa	1,1,13	(abcdefghijklmnopqrstuvwxyz)
ban**a**naaa	1,1,13,1	(nabcdefghijklmopqrstuvwxyz)
bana**n**aaa	1,1,13,1,1	(anbcdefghijklmopqrstuvwxyz)
banan**a**aa	1,1,13,1,1,1	(nabcdefghijklmopqrstuvwxyz)
banana**a**a	1,1,13,1,1,1,0	(anbcdefghijklmopqrstuvwxyz)
bananaa**a**	1,1,13,1,1,1,0,0	(anbcdefghijklmopqrstuvwxyz)
Final	1,1,13,1,1,1,0,0	(anbcdefghijklmopqrstuvwxyz)

Data compression summary

Lossless compression.

- 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, …

Practical compression. Use extra knowledge whenever possible.