Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

5.5 DATA COMPRESSION

- *introduction*
- *run-length coding*
- *Huffman compression*
- *LZW compression*

Last updated on 4/25/16 1:40 PM

http://algs4.cs.princeton.edu

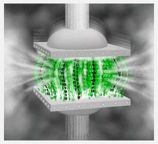
5.5 DATA COMPRESSION

- *introduction*
- *run-length coding*
- *Huffman compression*
- *LZW compression*

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.



> *" Everyday, we create 2.5 quintillion bytes of data—so much that 90% of the data in the world today has been created in the last two years alone. "* *— IBM report on big data (2011)*

Applications

Generic file compression (always lossless).

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

Multimedia (usually lossy).

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

Communication.

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

Databases. Google, Facebook, NSA, ...

Lossless compression and expansion

Message. Bitstream 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.

Compression before computers

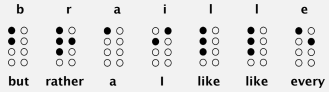
Data compression has been omnipresent since antiquity:

- Number systems.
- Natural languages.
- Mathematical notation.

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

It played a central role in communications technology:

- Grade 2 Braille.
- Morse code.
- Telephone system.



Data representation: genomic code

Genome. String over the alphabet { `A`, `T`, `C`, `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.
- $8N$ bits.

char	hex	binary
'A'	41	01000001
'T'	54	01010100
'C'	43	01000011
'G'	47	01000111

Two-bit encoding.

- 2 bits per char.
- $2N$ bits (25% compression ratio).

char	binary
'A'	00
'T'	01
'C'	10
'G'	11

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

Reading and writing binary data

Binary standard input. Read bits from standard input.

`public class BinaryStdIn`

`boolean readBoolean()`	*read 1 bit of data and return as a* `boolean` *value*
`char readChar()`	*read 8 bits of data and return as a* `char` *value*
`char readChar(int r)`	*read r bits of data and return as a* `char` *value*
[similar methods for `byte` *(8 bits);* `short` *(16 bits);* `int` *(32 bits);* `long` *and* `double` *(64 bits)]*	
`boolean isEmpty()`	*is the bitstream empty?*
`void close()`	*close the bitstream*

Binary standard output. Write bits to standard output

`public class BinaryStdOut`

`void write(boolean b)`	*write the specified bit*
`void write(char c)`	*write the specified 8-bit char*
`void write(char c, int r)`	*write the r least significant bits of the specified char*
[similar methods for `byte` *(8 bits);* `short` *(16 bits);* `int` *(32 bits);* `long` *and* `double` *(64 bits)]*	
`void close()`	*close the bitstream*

Writing binary data

Date representation. Three different ways to represent 12/31/1999.

A character stream (StdOut)

```
StdOut.print(month + "/" + day + "/" + year);
```

```
00110001|00110010|00101111|00110111|00110001|00101111|00110001|00111001|00111001|00111001
   1        2        /        3        1        /        1        9        9        9
```

80 bits

Three ints (BinaryStdOut)

```
BinaryStdOut.write(month);
BinaryStdOut.write(day);
BinaryStdOut.write(year);
```

```
00000000|00000000|00000000|00001100|00000000|00000000|00000000|00011111|00000000|00000000|00000111|11001111
                  12                                  31                                 1999
```

96 bits

A 4-bit field, a 5-bit field, and a 12-bit field (BinaryStdOut)

```
BinaryStdOut.write(month, 4);
BinaryStdOut.write(day, 5);
BinaryStdOut.write(year, 12);
```

```
11001111|10111110|01111000
 12  31     1999
```

21 bits (+ 3 bits for byte alignment at close)

Binary dumps

Q. How to examine the contents of a bitstream?

Standard character stream

```
% more abra.txt
ABRACADABRA!
```

Bitstream represented as 0 and 1 characters

```
% java BinaryDump 16 < abra.txt
0100000101000010
0101001001000001
0100001101000001
0100010001000001
0100000101010010
0100000100100001
96 bits
```

Bitstream represented with hex digits

```
%  java HexDump 4 < abra.txt
41 42 52 41
43 41 44 41
42 52 41 21
12 bytes
```

Bitstream represented as pixels in a Picture

```
% java PictureDump 16 6 < abra.txt
```

16-by-6 pixel window, magnified

```
96 bits
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Hexadecimal to ASCII conversion table

Which of these formats are text-based, and which are binary?

HTML

GIF

MPEG

PDF

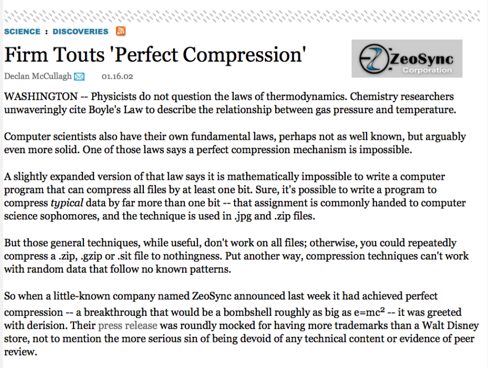
SVG

Java source code

Java bytecode

Universal data compression

ZeoSync. Announced 100:1 lossless compression of random data using Zero Space Tuner™ and BinaryAccelerator™ technology.



SCIENCE : DISCOVERIES

Firm Touts 'Perfect Compression'

Declan McCullagh 01.16.02

WASHINGTON -- Physicists do not question the laws of thermodynamics. Chemistry researchers unwaveringly cite Boyle's Law to describe the relationship between gas pressure and temperature.

Computer scientists also have their own fundamental laws, perhaps not as well known, but arguably even more solid. One of those laws says a perfect compression mechanism is impossible.

A slightly expanded version of that law says it is mathematically impossible to write a computer program that can compress all files by at least one bit. Sure, it's possible to write a program to compress *typical* data by far more than one bit -- that assignment is commonly handed to computer science sophomores, and the technique is used in .jpg and .zip files.

But those general techniques, while useful, don't work on all files; otherwise, you could repeatedly compress a .zip, .gzip or .sit file to nothingness. Put another way, compression techniques can't work with random data that follow no known patterns.

So when a little-known company named ZeoSync announced last week it had achieved perfect compression -- a breakthrough that would be a bombshell roughly as big as $e=mc^2$ -- it was greeted with derision. Their press release was roundly mocked for having more trademarks than a Walt Disney store, not to mention the more serious sin of being devoid of any technical content or evidence of peer review.

ZeoSync corporation folds after issuing $40 million in private stock

Quotes from this interview

Wired News: When did you start working on this technology?
Peter St. George: I started developing the technology about a dozen years ago. I worked on this one problem for 12 years consecutively. This is a project that I dedicated my life to a dozen years ago.

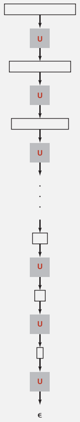
WN: Let's go into the details. Tell me how it works. It can compress random data?
PSG: If you say absolutely random, it's going to be very hard to agree what absolutely random is.

WN: How do you get around the conventional wisdom that says simple mathematics says it's impossible?
PSG: We plan to attack that issue head on. What hasn't been previously proven, we're proving.

I have one quote I'd like to share with you: "The person who says it cannot be done should not interrupt the person doing it."

Universal data compression

Proposition. No algorithm can compress every bitstring.



Universal data compression?

Pf 1. [by contradiction]

Pf 2. [by counting]

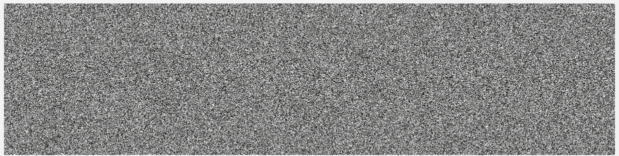
- Suppose your algorithm that can compress all $1{,}000$-bit strings.
- 2^{1000} possible bitstrings with $1{,}000$ bits.
- Only $1 + 2 + 4 + \ldots + 2^{998} + 2^{999}$ can be encoded with ≤ 999 bits.
- Similarly, only 1 in 2^{499} bitstrings can be encoded with ≤ 500 bits!

Can you compress this string of decimal digits?

```
1415926535897932384626433832795028841971693993751O
5820974944592307816406286208998628034825342117O679
8214808651328230664709384460955058223172535940812８
4811174502841027019385211055596446229489549303819６
```

It's the first 1000 digits of pi after the decimal point. (But how to compress?)

Undecidability



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

```
public class RandomBits
{
   public static void main(String[] args)
   {
      int x = 11111;
      for (int i = 0; i < 1000000; i++)
      {
         x = x * 314159 + 218281;
         BinaryStdOut.write(x > 0);
      }
      BinaryStdOut.close();
   }
}
```

Rdenudcany in Enlgsih lnagugae

Q. How much redundancy in the English language?
A. Quite a bit.

> *" ... 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. "* — Graham Rawlinson

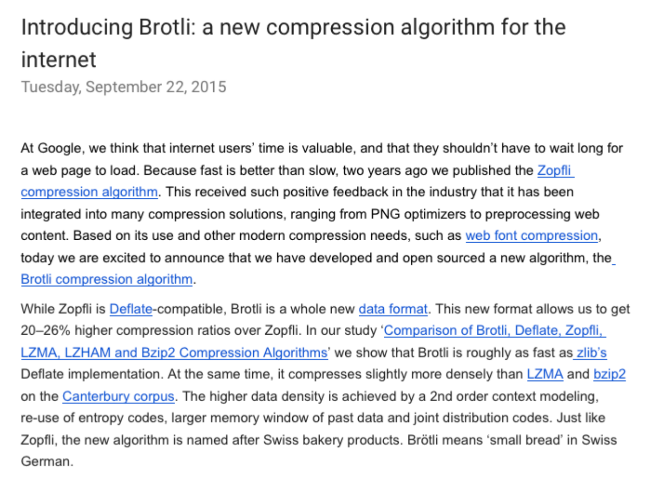
The gaol of data cmperisoson is to inetdify rdenudcany and epxloit it.
Aside. Design an algorithm to correct text with letters permuted.

Data compression: quiz 1

Rank these in the order of compressibility:
1. An ASCII text file of Shakespeare's works
2. A bitmap image of this slide
3. An mp3 file of Justin Bieber's "Baby"

A. 3 > 2 > 1
B. 3 > 1 > 2
C. 2 > 1 > 3
D. 2 > 3 > 1
E. *I don't know.*

Compression still active area of research, big improvements possible



Introducing Brotli: a new compression algorithm for the internet

Tuesday, September 22, 2015

At Google, we think that internet users' time is valuable, and that they shouldn't have to wait long for a web page to load. Because fast is better than slow, two years ago we published the Zopfli compression algorithm. This received such positive feedback in the industry that it has been integrated into many compression solutions, ranging from PNG optimizers to preprocessing web content. Based on its use and other modern compression needs, such as web font compression, today we are excited to announce that we have developed and open sourced a new algorithm, the Brotli compression algorithm.

While Zopfli is Deflate-compatible, Brotli is a whole new data format. This new format allows us to get 20–26% higher compression ratios over Zopfli. In our study 'Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms' we show that Brotli is roughly as fast as zlib's Deflate implementation. At the same time, it compresses slightly more densely than LZMA and bzip2 on the Canterbury corpus. The higher data density is achieved by a 2nd order context modeling, re-use of entropy codes, larger memory window of past data and joint distribution codes. Just like Zopfli, the new algorithm is named after Swiss bakery products. Brötli means 'small bread' in Swiss German.

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

5.5 DATA COMPRESSION

- *introduction*
- *run-length coding*
- *Huffman compression*
- *LZW compression*

Run-length encoding

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

0000000000000001111111000000011111111111

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.

$\underbrace{1111}_{15}\underbrace{0111}_{7}\underbrace{0111}_{7}\underbrace{1011}_{11}$ ← 16 bits (instead of 40)

Q. How many bits to store the counts?
A. We typically use 8 (but 4 in the example above for brevity).

Q. What to do when run length exceeds max count?
A. Intersperse runs of length 0.

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

Data compression: quiz 2

What is the best compression ratio achievable from run-length coding when using 8-bit counts?

A. 1 / 256
B. 1 / 16
C. 8 / 255
D. 24 / 510 = 4 / 85
E. *I don't know.*

http://algs4.cs.princeton.edu

5.5 DATA COMPRESSION

David Huffman

Variable-length codes

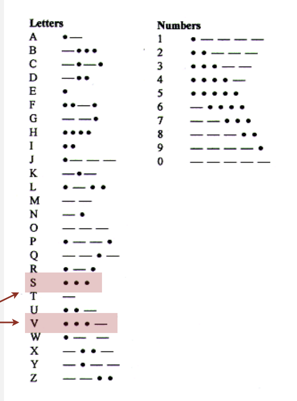
Use different number of bits to encode different chars.
Assign shorter codes to more common chars.

Ex. Morse code: • • • – – – • • •

Issue. Ambiguity.

SOS ?
V7 ?
IAMIE ?
EEWNI ?

In practice. Use a medium gap to separate codewords.



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	— —••		

codeword for S is a prefix of codeword for V

Variable-length codes

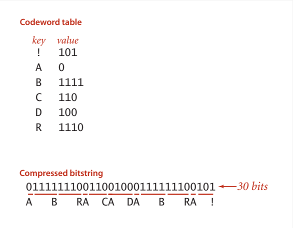
Q. How do we avoid ambiguity?

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

Ex 1. Fixed-length code.

Ex 2. Append special stop character to each codeword.

Ex 3. General prefix-free code.



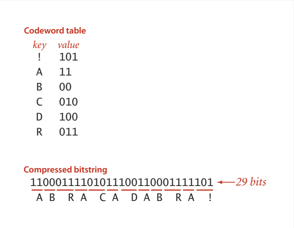
Codeword table

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

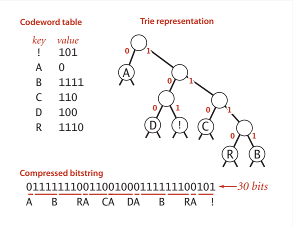
A B R A C A D A B R A !

Prefix-free codes: trie representation

Q. How to represent the prefix-free code?

A. A binary trie!

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



Codeword table

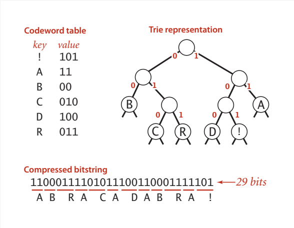
key	value
!	101
A	0
B	1111
C	110
D	100
R	1110

Trie representation

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

Trie representation

Compressed bitstring

11000111101011100110001111101 ← 29 bits

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

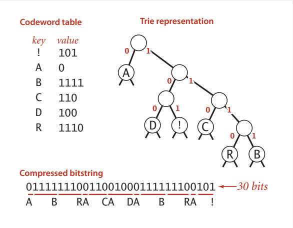
Prefix-free codes: expansion

Expansion.

- Start at root.
- Go left if bit is 0; go right if 1.
- If leaf node, write character; return to root node; repeat.

Q. Why would this fail if the code isn't prefix-free?

A. Internal nodes also have chars, but decompressor will never output them.



Codeword table

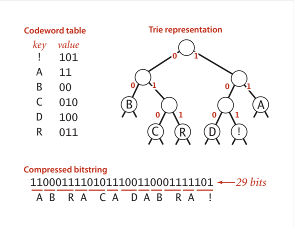
key	value
!	101
A	0
B	1111
C	110
D	100
R	1110

Trie representation

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

Trie representation

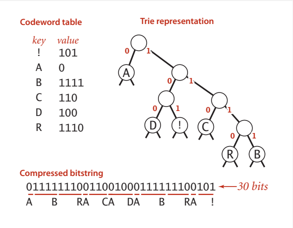
Compressed bitstring

11000111101011100110001111101 ← 29 bits

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

Prefix-free codes: compression

Compression: create ST of key-value pairs.



Codeword table

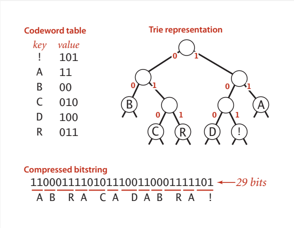
key	value
!	101
A	0
B	1111
C	110
D	100
R	1110

Trie representation

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

Trie representation

Compressed bitstring

11000111101011100110001111101 ← 29 bits

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

Data compression: quiz 3

Consider the following trie representation of a prefix-free code.
Expand the compressed bitstring `100101000111011`.

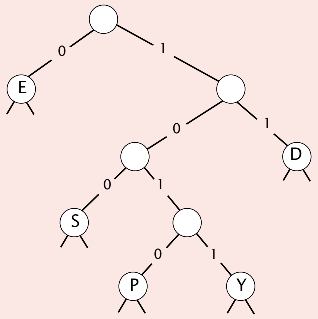
A. PEED

B. PESDEY

C. SPED

D. SPEEDY

E. *I don't know.*



Huffman coding overview

Static model. Use the same prefix-free code for all messages.
Dynamic model. Use a custom prefix-free code for each message.

Compression.

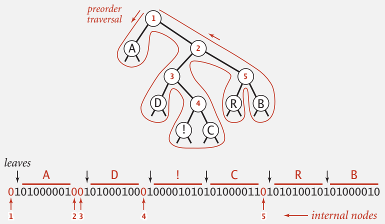
- Read message.
- Build best prefix-free code for message. How? [ahead]
- Write prefix-free code (as a trie).
- Compress message using prefix-free code.

Expansion.

- Read prefix-free code (as a trie) from file.
- Read compressed message and expand using trie.

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.

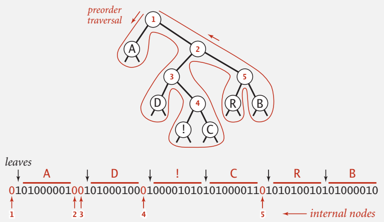


Using preorder traversal to encode a trie as a bitstream

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

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

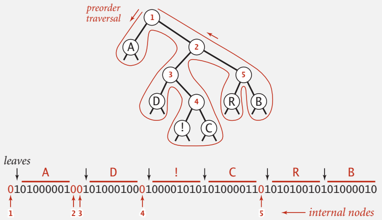
```
private static void writeTrie(Node x)
{
   if (x.isLeaf())
   {
      BinaryStdOut.write(true);
      BinaryStdOut.write(???);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(???);
   writeTrie(???);
}
```

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

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

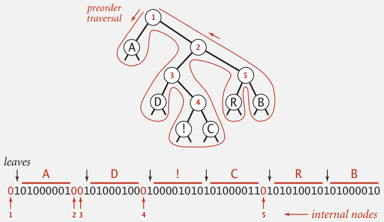
```
private static void writeTrie(Node x)
{
   if (x.isLeaf())
   {
      BinaryStdOut.write(true);
      BinaryStdOut.write(x.ch, 8);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(x.left);
   writeTrie(x.right);
}
```

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

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);
}
```

arbitrary value
(value not used with internal nodes)

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:

JPEG PDF Adobe mp3 DivX MPEG4 gzip

Huffman coding demo

- Count frequency for each character in input.

char	freq	encoding
A		
B		
C		
D		
R		
!		

input

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

Huffman coding demo

- Count frequency for each character in input.

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

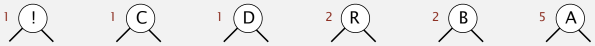
input

ABRACADABRA!

Huffman coding demo

- Start with one node corresponding to each character with weight equal to frequency.

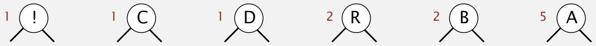
char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

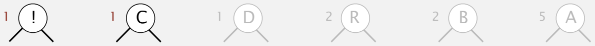
char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

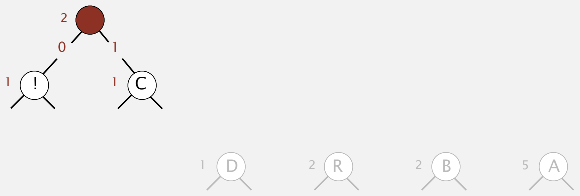
char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

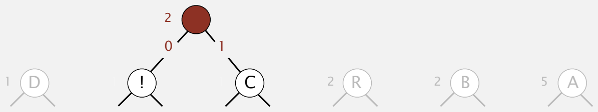
char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

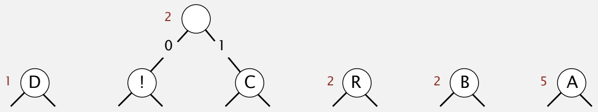
char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

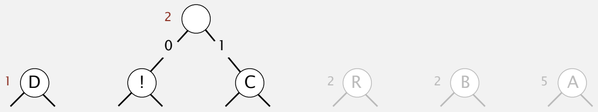
char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

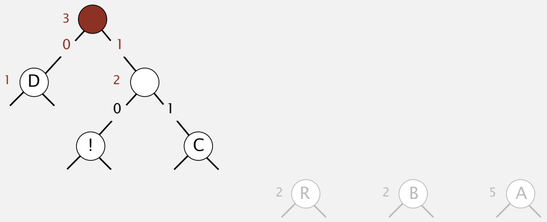
char	freq	encoding
A	5	
B	2	
C	1	1
D	1	
R	2	
!	1	0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

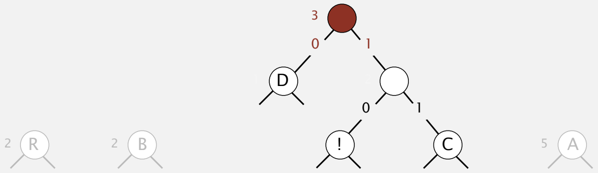
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

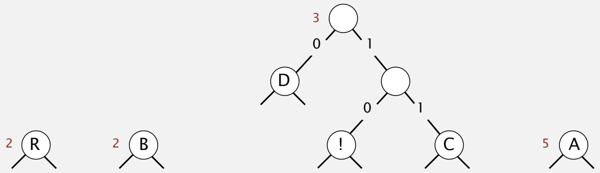
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

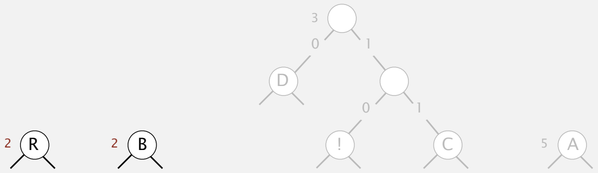
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

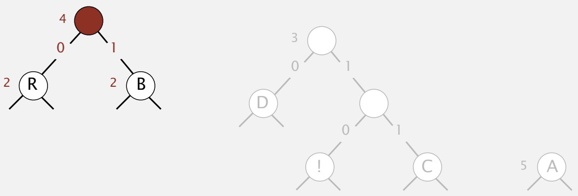
char	freq	encoding
A	5	
B	2	
C	1	1 1
D	1	0
R	2	
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

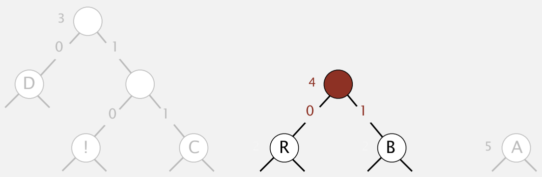
char	freq	encoding
A	5	
B	2	1
C	1	1 1
D	1	0
R	2	0
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

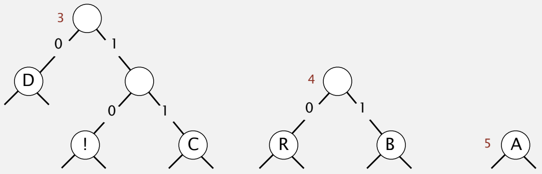
char	freq	encoding
A	5	
B	2	1
C	1	1 1
D	1	0
R	2	0
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

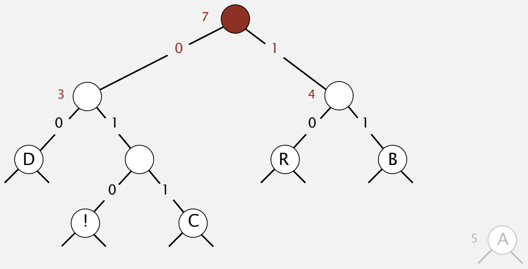
char	freq	encoding
A	5	
B	2	1
C	1	1 1
D	1	0
R	2	0
!	1	1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

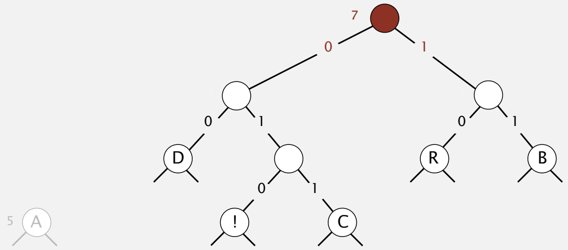
char	freq	encoding
A	5	
B	2	1 1
C	1	0 1 1
D	1	0 0
R	2	1 0
!	1	0 1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

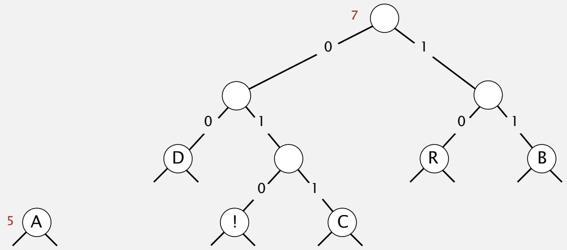
char	freq	encoding
A	5	
B	2	1 1
C	1	0 1 1
D	1	0 0
R	2	1 0
!	1	0 1 0



Huffman coding demo

- Select two tries with min weight.
- Merge into single trie with cumulative weight.

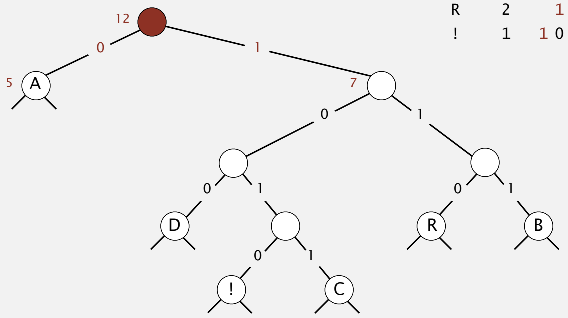
char	freq	encoding
A	5	
B	2	1 1
C	1	0 1 1
D	1	0 0
R	2	1 0
!	1	0 1 0



Huffman coding demo

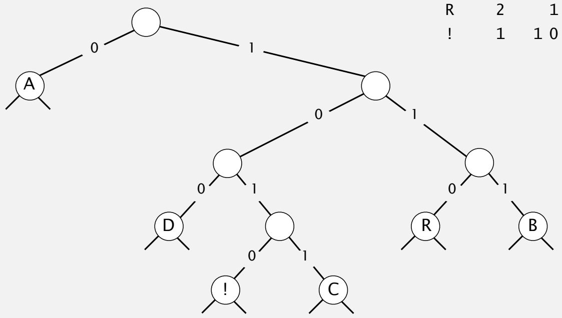
- Select two tries with min weight.
- Merge into single trie with cumulative weight.

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 coding 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



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

Practice

Construct the Huffman code for the following strings:

aababcabcdabcde

abcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcd

Practice

Construct the Huffman code for the following strings:

aababcabcdabcde

a 11
b 10
c 01
d 001
e 000

abcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcd

a 00
b 01
c 10
d 11

Each codeword uses 2 bits, so no compression (or expansion) of input. Small overhead due to need to store trie.

Huffman coding: overview

Compression: high-level steps:

- Build prefix-free code for message:
  - Tabulate character frequencies.
  - Recursively merge two min weight tries.
- Write prefix-free code (as a trie).
- Compress message using prefix-free code:
  - Build symbol table from characters to codewords.
  - Output codeword for each character in input.

Expansion: high-level steps:

- Read and decode prefix-free code (as a trie) from file.
- Expand compressed message using trie:
  - Repeatedly find path from root to leaf in trie using bit sequence.

Huffman compression summary

Proposition. Huffman's algorithm produces an optimal prefix-free code.
Pf. See textbook.

no prefix-free code uses fewer bits

Two-pass implementation (for compression).

- Pass 1: tabulate character frequencies; build trie.
- Pass 2: encode file by traversing trie (or symbol table).

Running time (for compression). Using a binary heap ⇒ $N + R \log R$.
Running time (for expansion). Using a binary trie ⇒ N.

input size

alphabet size

Q. Can we do better? [stay tuned]

Lossy vs. lossless compression

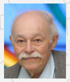
This lecture: lossless compression

Images, music, videos, ... :
lossy compression dramatically more effective

http://algs4.cs.princeton.edu

5.5 Data Compression

- *introduction*
- *run-length coding*
- *Huffman compression*
- *LZW compression*


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 compression demo

input	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A	
matches	A	B	R	A	C	A	D	A B		R A		B R		A B R			A	
value	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
⋮	⋮
A	41
B	42
C	43
D	44
⋮	⋮

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

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

codeword table
stop char: 80

Lempel-Ziv-Welch compression

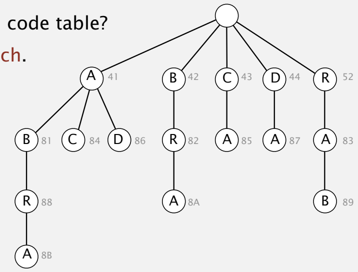
LZW compression.

- Create ST mapping string keys to W-bit codewords.
- Initialize ST with codewords for single-character 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 character in the input.

longest prefix match

Q. How to represent LZW compression code table?

A. A trie to support longest prefix match.



LZW expansion demo

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.

- Create ST mapping W-bit keys to string values.
- Initialize ST to contain single-character 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 length 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
⋮	⋮

Data compression: quiz 4

What is the LZW compression of ABABABA ?

A. 41 42 41 42 41 42 80

B. 41 42 41 81 81

C. 41 42 81 81 41

D. 41 42 81 83 80

E. *I don't know.*

LZW tricky case: compression

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

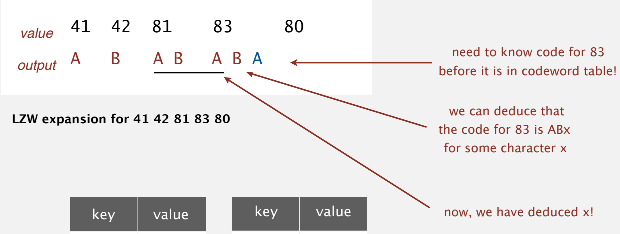
LZW compression for ABABABA

key	value
⋮	⋮
A	41
B	42
C	43
D	44
⋮	⋮

key	value
AB	81
BA	82
ABA	83

codeword table

LZW tricky case: expansion



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

need to know code for 83 before it is in codeword table!

we can deduce that the code for 83 is ABx for some character x

now, we have deduced x!

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.

Unix compress, GIF, TIFF, V.42bis modem: LZW. ← previously under patent

zip, 7zip, gzip, jar, png, pdf: deflate / zlib.
iPhone, Wii, Apache HTTP server: deflate / zlib. ← not patented (widely used in open source)

Apache HTTP SERVER PROJECT

Lossless data compression benchmarks

year	scheme	bits / char
1967	**ASCII**	7
1950	**Huffman**	4.7
1977	**LZ77**	3.94
1984	**LZMW**	3.32
1987	**LZH**	3.3
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

← next programming assignment

data compression using Calgary corpus

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/DCT, wavelets, fractals, …

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right]$$

Theoretical limits on compression. Shannon entropy: $H(X) = -\sum_{i}^{n} p(x_i) \lg p(x_i)$

Practical compression. Exploit extra knowledge whenever possible.