Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

https://algs4.cs.princeton.edu

5.5 DATA COMPRESSION

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

Last updated on 4/24/23 2:57 PM


Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

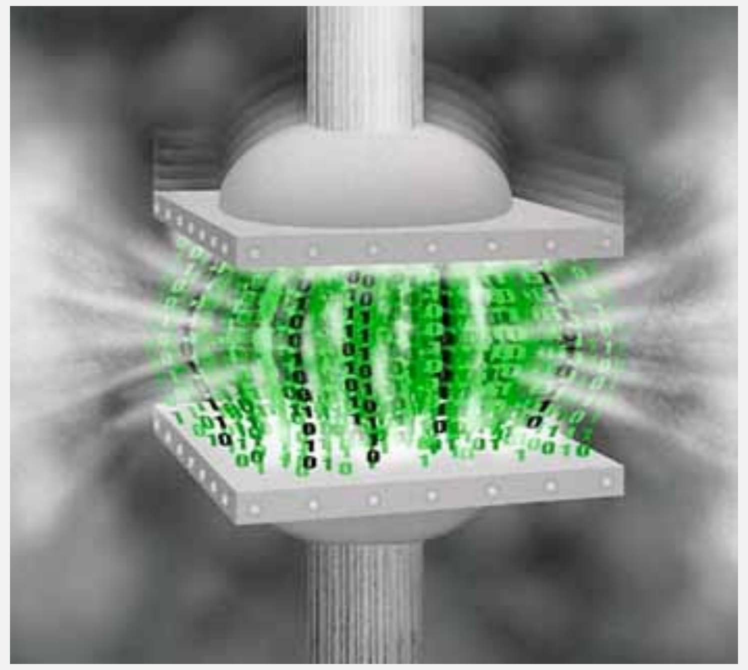
5.5 DATA COMPRESSION

- ▸ *introduction*
- ▸ *run-length encoding*
- ▸ *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.



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, sensors, …

Every day, we create:

- 900 million Tweets.
- 300 billion emails.
- 100 million Instagram photos.
- 750,000 hours YouTube video.

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

Applications

Generic file compression.

- Files: Gzip, bzip2, 7z, PKZIP, ...
- File systems: ZFS, HFS+, ReFS, GFS, APFS, ...

Multimedia.

- Images: GIF, JPEG, PNG, RAW, ...
- Sound: MP3, AAC, Ogg Vorbis, ...
- Video: MPEG, HDTV, H.264, HEVC, ...

Communication. Fax, Skype, WeChat, Zoom, ...

Databases. SQL, Google, Facebook, NSA, ...

Smart sensors. Phone, watch, car, health, ...

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
(we hope)

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

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

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.
- $8\,n$ bits.

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

Two-bit encoding.

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

compression ratio = 25%
(compared to ASCII)

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

	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

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

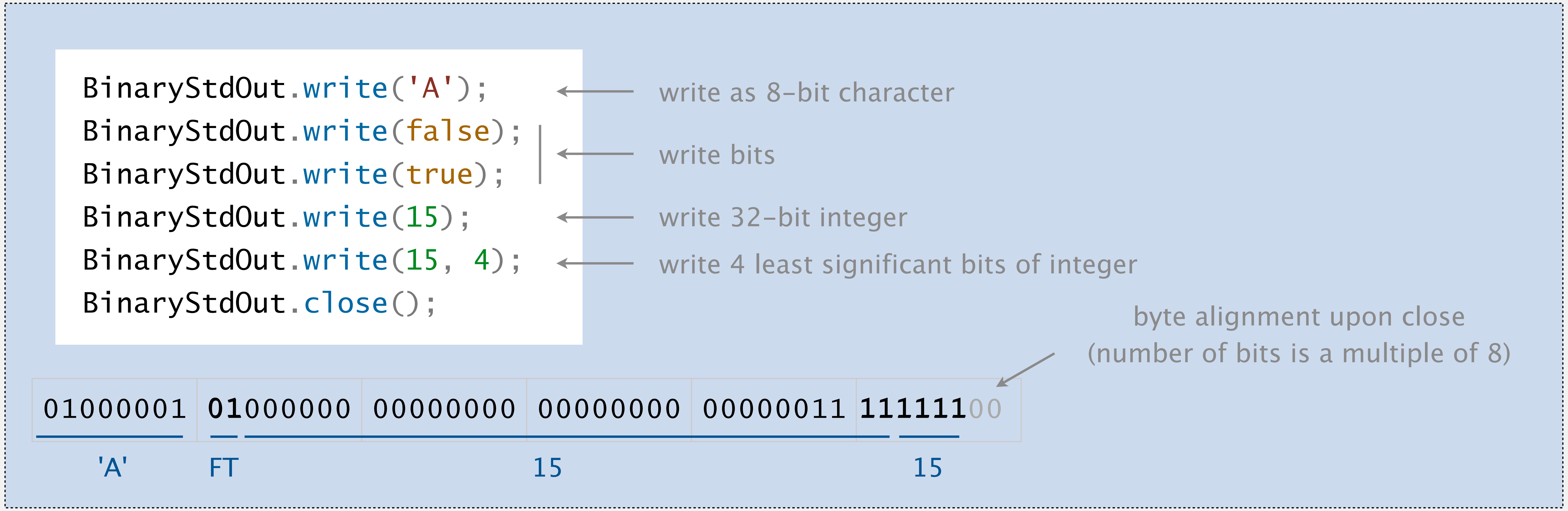
Amazing but true. Some genomic databases in 1990s used ASCII.

Writing binary data

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*



```
BinaryStdOut.write('A');
BinaryStdOut.write(false);
BinaryStdOut.write(true);
BinaryStdOut.write(15);
BinaryStdOut.write(15, 4);
BinaryStdOut.close();
```

write as 8-bit character

write bits

write 32-bit integer

write 4 least significant bits of integer

byte alignment upon close
(number of bits is a multiple of 8)

Reading 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?*

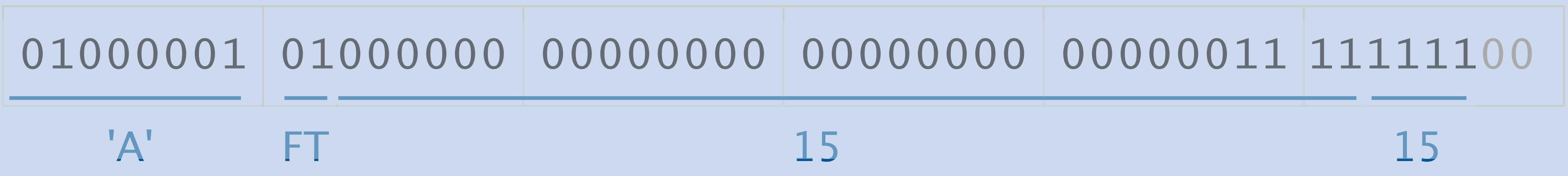
```
char c = BinaryStdIn.readChar();
boolean b1 = BinaryStdIn.readBoolean();
boolean b2 = BinaryStdIn.readBoolean();
int x = BinaryStdIn.readInt();
int y = BinaryStdIn.readInt(4);
```

read 8-bit character

read bits

read 32-bit integer

read 4-bit integer



Binary representation

Q. How to examine the contents of a bitstream (e.g., when debugging)?

standard character stream

```
~> more dna.txt
ATCGCA
```

bitstream represented with hex digits

```
~> java HexDump < dna.txt
41 54 43 47 43 41
48 bits
```

bitstream of binary file represented with hex digits

```
~> java HexDump < us.gif
47 49 46 38 39 61 8e 01 01 01 d5 00 00 94 18 29
06 02 03 84 29 4a 6b 18 4a 73 29 6b 6e 5d 6e 4a
45 4a f7 ef f7 42 00 52 1f 00 37 42 10 6b 31 00
63 31 08 5a 7e 70 8d 36 08 6b 4a 21 7b 5b 36 86
...
b7 de 7b f3 dd b7 df 7f 03 1e 38 cc 41 00 00 3b
99200 bits
```

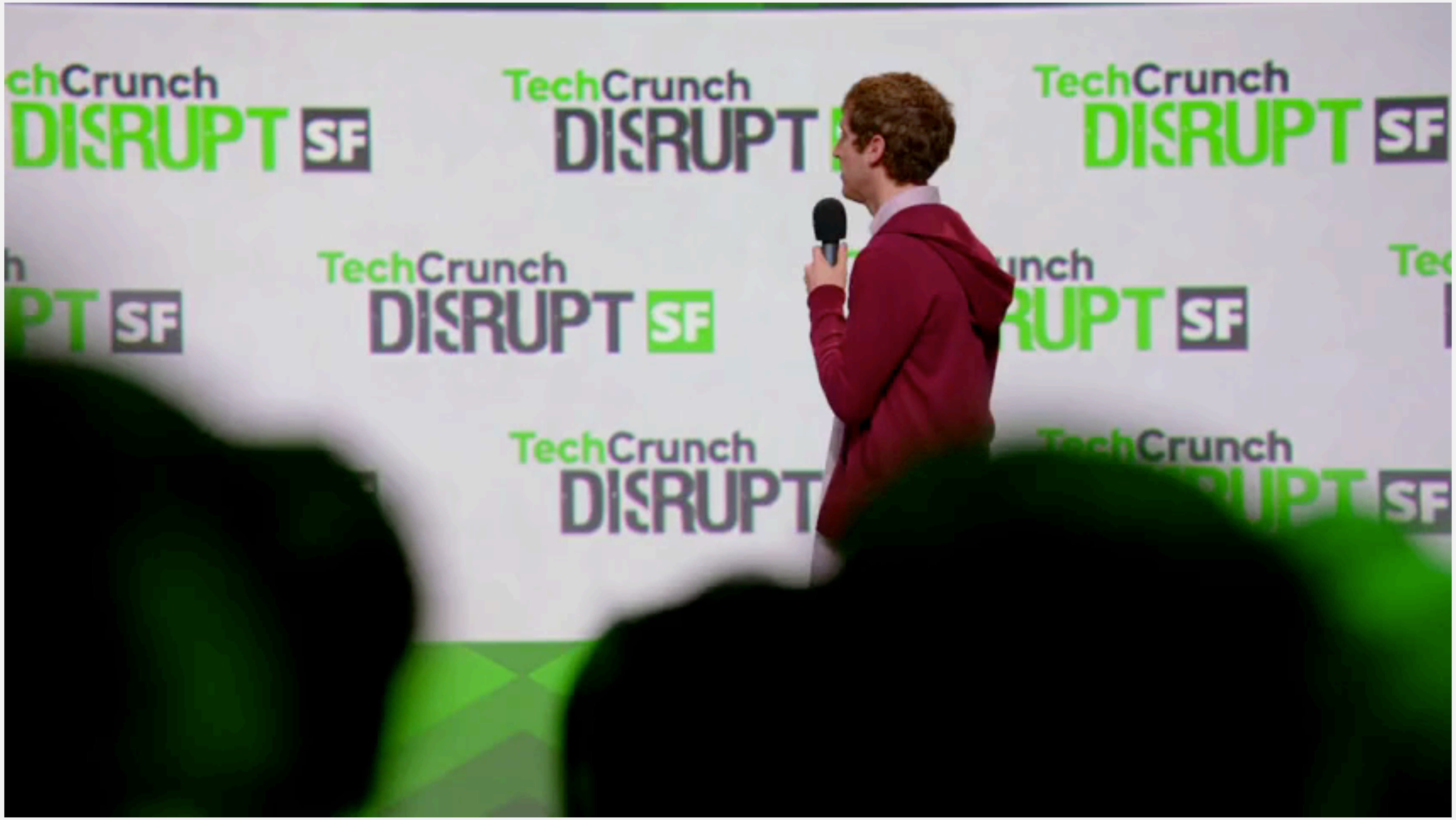
lots of unprintable characters
(don't `System.out.print()` binary data)

	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

Universal data compression

Pied Piper. Claims lossless compression ratio of 1 : 3.8 for arbitrary files.



Data compression: quiz 1

Should you invest in Pied Piper?

- **A.** Yes.
- **B.** No.

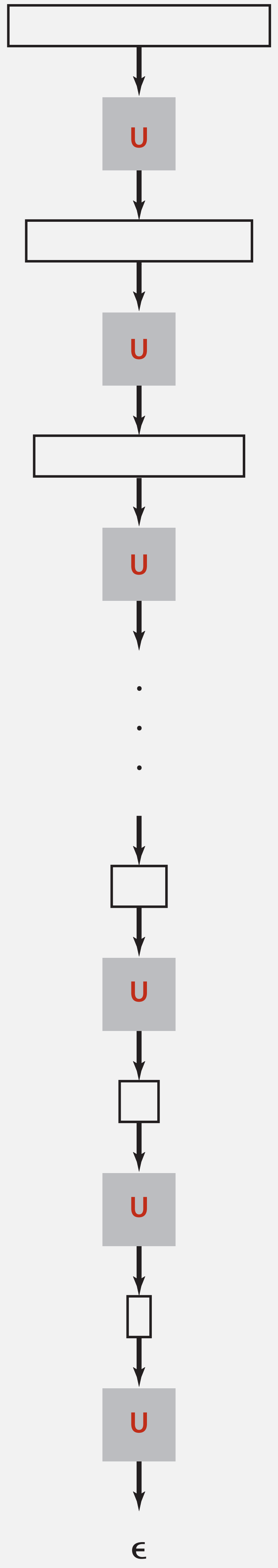
Universal data compression

Proposition. No algorithm can compress every bitstring.

Pied Piper

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 a shorter bitstring B_1.
- Compress B_1 to get a shorter bitstring B_2.
- Continue until reaching bitstring of length 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 $1 + 2 + 4 + \ldots + 2^{998} + 2^{999} \ = \ 2^{1000} - 1$ can be encoded with ≤ 999 bits.
- Similarly, only 1 in 2^{499} bitstrings can be encoded with ≤ 500 bits!

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.*” — Graham Rawlinson

Bottom line. The gaol of data cmperisoson is to inetdify rdenudcany and epxloit it.

https://algs4.cs.princeton.edu

5.5 DATA COMPRESSION

- ▸ *introduction*
- ▸ *run-length encoding*
- ▸ *Huffman compression*
- ▸ *LZW compression*

Run-length encoding (RLE)

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

$$\underbrace{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0}_{\text{run of length 15}}\;\underbrace{1\,1\,1\,1\,1\,1\,1}_{\text{run of length 7}}\;\underbrace{0\,0\,0\,0\,0\,0\,0}_{\text{run of length 7}}\;\underbrace{1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1}_{\text{run of length 11}} \longleftarrow \text{40 bits}$$

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

$$\underbrace{1\,1\,1\,1}_{15}\;\underbrace{0\,1\,1\,1}_{7}\;\underbrace{0\,1\,1\,1}_{7}\;\underbrace{1\,0\,1\,1}_{11} \longleftarrow \text{now only 16 bits}$$

Q. How many bits r to use to store each run length?

A. Typically 8 bits (but 4 on this slide for brevity).

Q. What to do when run length exceeds max count $2^r - 1$?

A. Intersperse runs of length 0.

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

Data compression: quiz 2

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

- **A.** 1 / 256
- **B.** 1 / 16
- **C.** 8 / 255
- **D.** 1 / 8
- **E.** 16 / 255

Run-length encoding: Java implementation

```
public class RunLength
{

   public static void compress()
   {  /* see textbook */  }

   public static void expand()
   {
      boolean bit = false;
      while (!BinaryStdIn.isEmpty())
      {
         int run = BinaryStdIn.readInt(8);
         for (int i = 0; i < run; i++)
            BinaryStdOut.write(bit);
         bit = !bit;
      }
      BinaryStdOut.close();
   }

}
```

initial run is of 0s

read 8-bit count from standard input

write run of 0s or 1s to standard output

flip bit (for next run)

pad last byte with 0s (if needed)
and close output stream

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

5.5 DATA COMPRESSION

- ‣ *introduction*
- ‣ *run-length encoding*
- ‣ *Huffman compression*
- ‣ *LZW compression*

Variable-length codes

Key idea. Use different number of bits to encode different characters.

Ex. Morse code: ● ● ● — — — ● ● ●

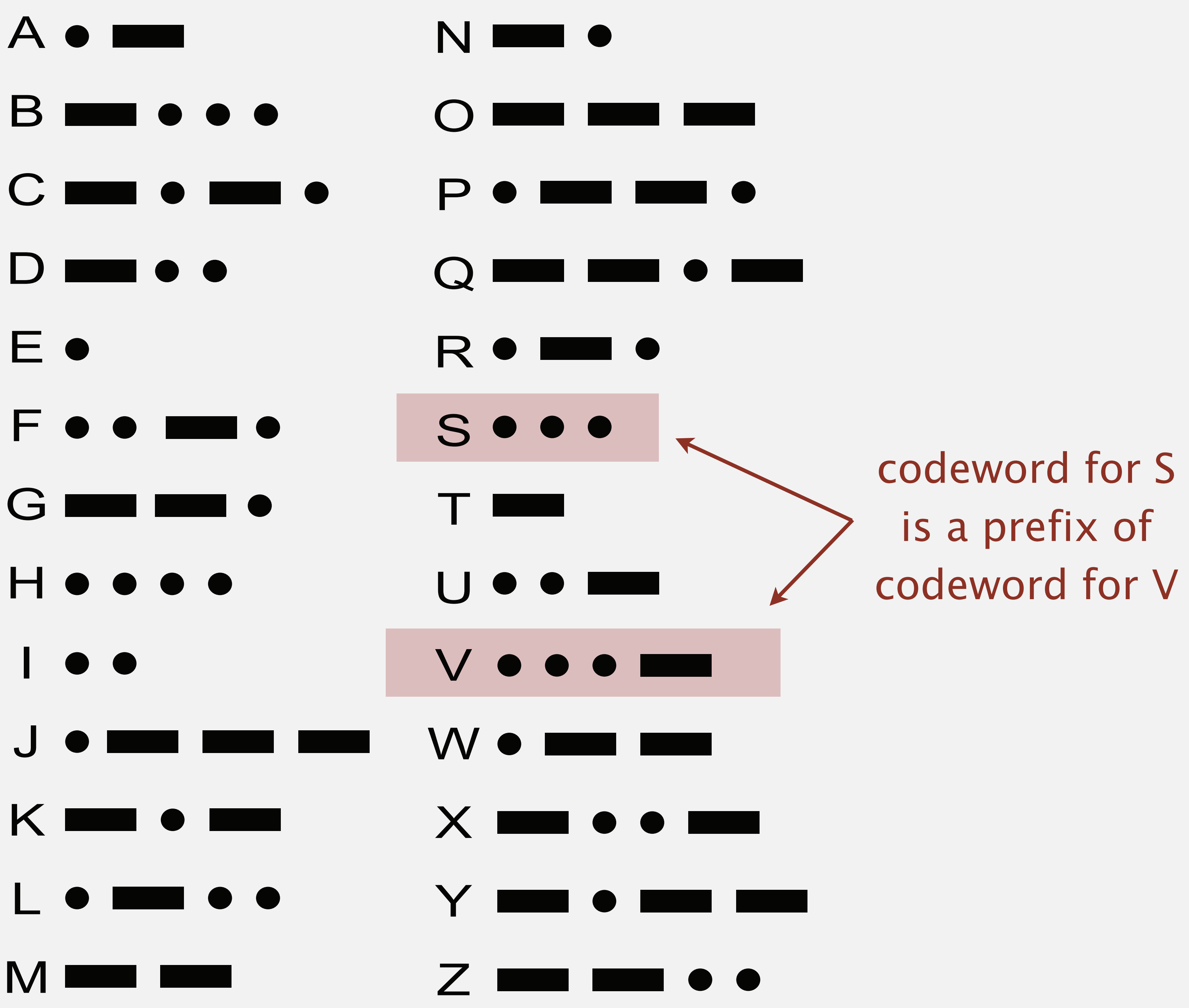
Issue. Ambiguity.

SOS ?

VZE ?

EEJIE ?

EEWNI ?



A	● ▬	N	▬ ●
B	▬ ● ● ●	O	▬ ▬ ▬
C	▬ ● ▬ ●	P	● ▬ ▬ ●
D	▬ ● ●	Q	▬ ▬ ● ▬
E	●	R	● ▬ ●
F	● ● ▬ ●	S	● ● ●
G	▬ ▬ ●	T	▬
H	● ● ● ●	U	● ● ▬
I	● ●	V	● ● ● ▬
J	● ▬ ▬ ▬	W	● ▬ ▬
K	▬ ● ▬	X	▬ ● ● ▬
L	● ▬ ● ●	Y	▬ ● ▬ ▬
M	▬ ▬	Z	▬ ▬ ● ●

codeword for S is a prefix of codeword for V

In practice. Use a short gap to separate characters.

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.

char	*codeword*
!	101
A	11
B	00
C	010
D	100
R	011

no codeword
is a prefix of another

$$\underbrace{\overset{\downarrow}{1}\ 1}_{A}\ \underbrace{0\ 0}_{B}\ \underbrace{0\ 1\ 1}_{R}\ \underbrace{1\ 1}_{A}$$

Prefix-free codes: compression

Q. How to represent the prefix-free code for compression?

A. A symbol table or array.

char	*codeword*
!	101
A	11
B	00
C	010
D	100
R	011

key/index value

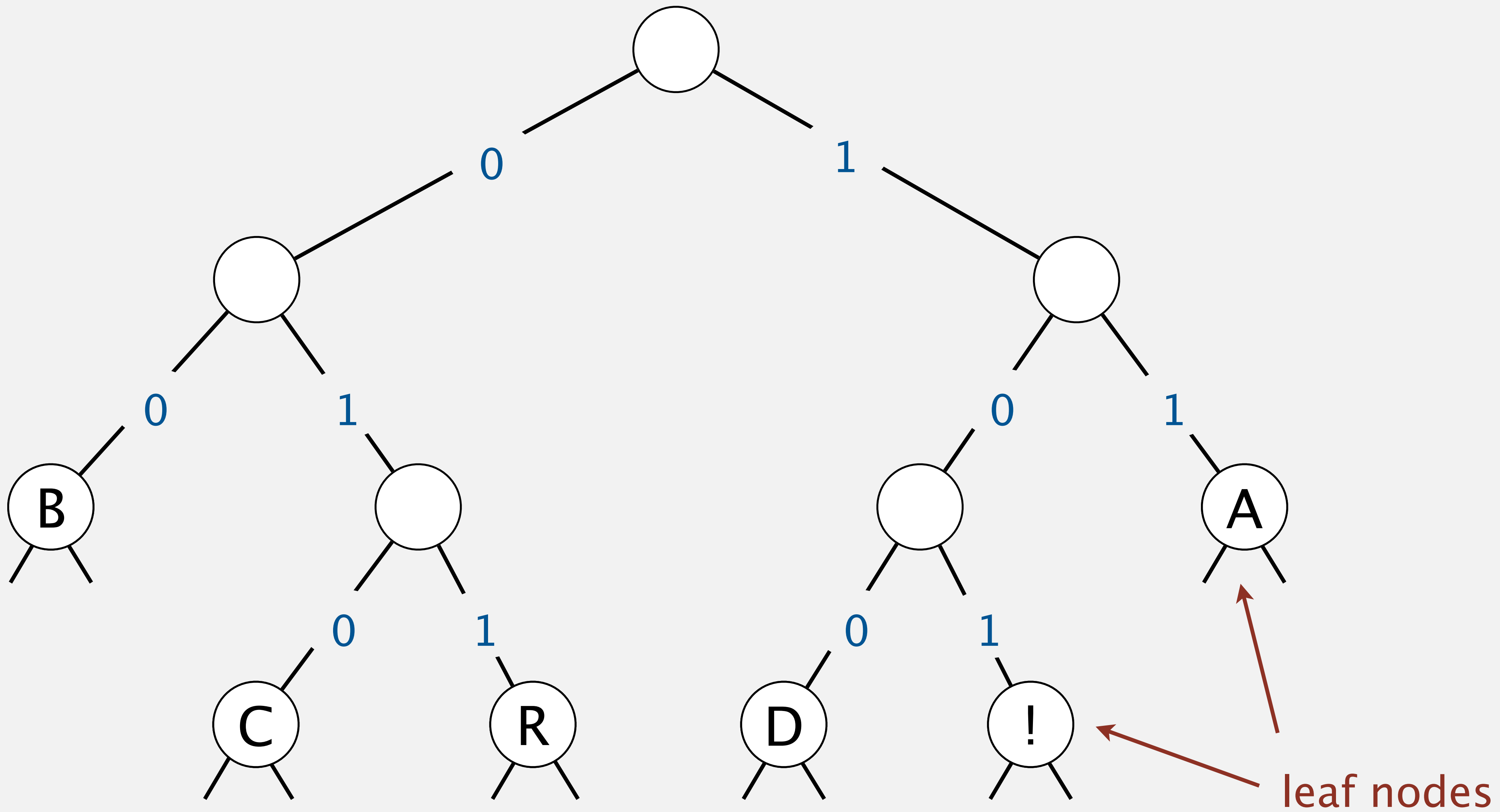
Prefix-free codes: trie representation

Q. How to represent the prefix-free code for expansion?

A. A binary trie.

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

char	*codeword*
!	101
A	11
B	00
C	010
D	100
R	011



Prefix-free codes: expansion

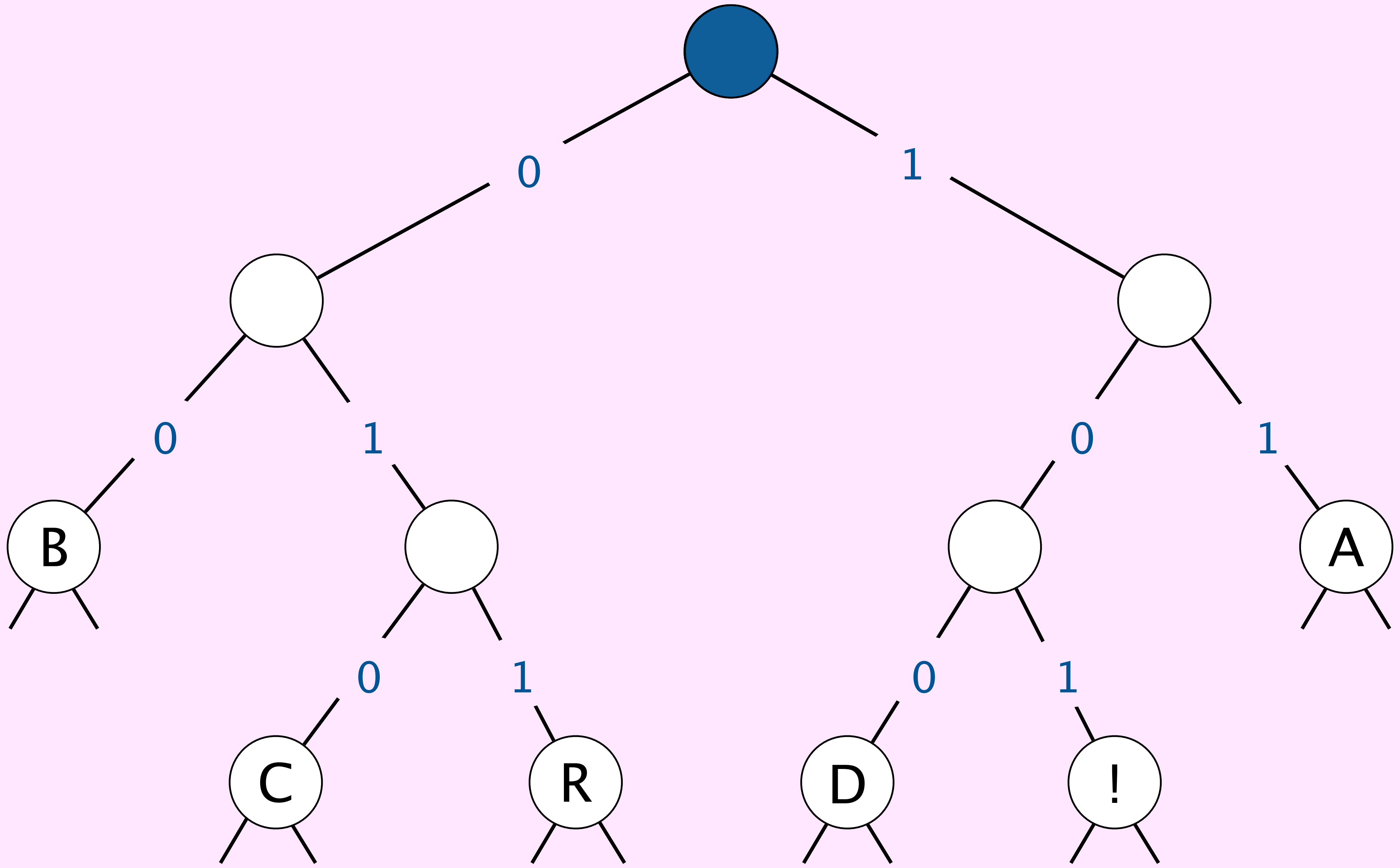
Expansion.

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

↓

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

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

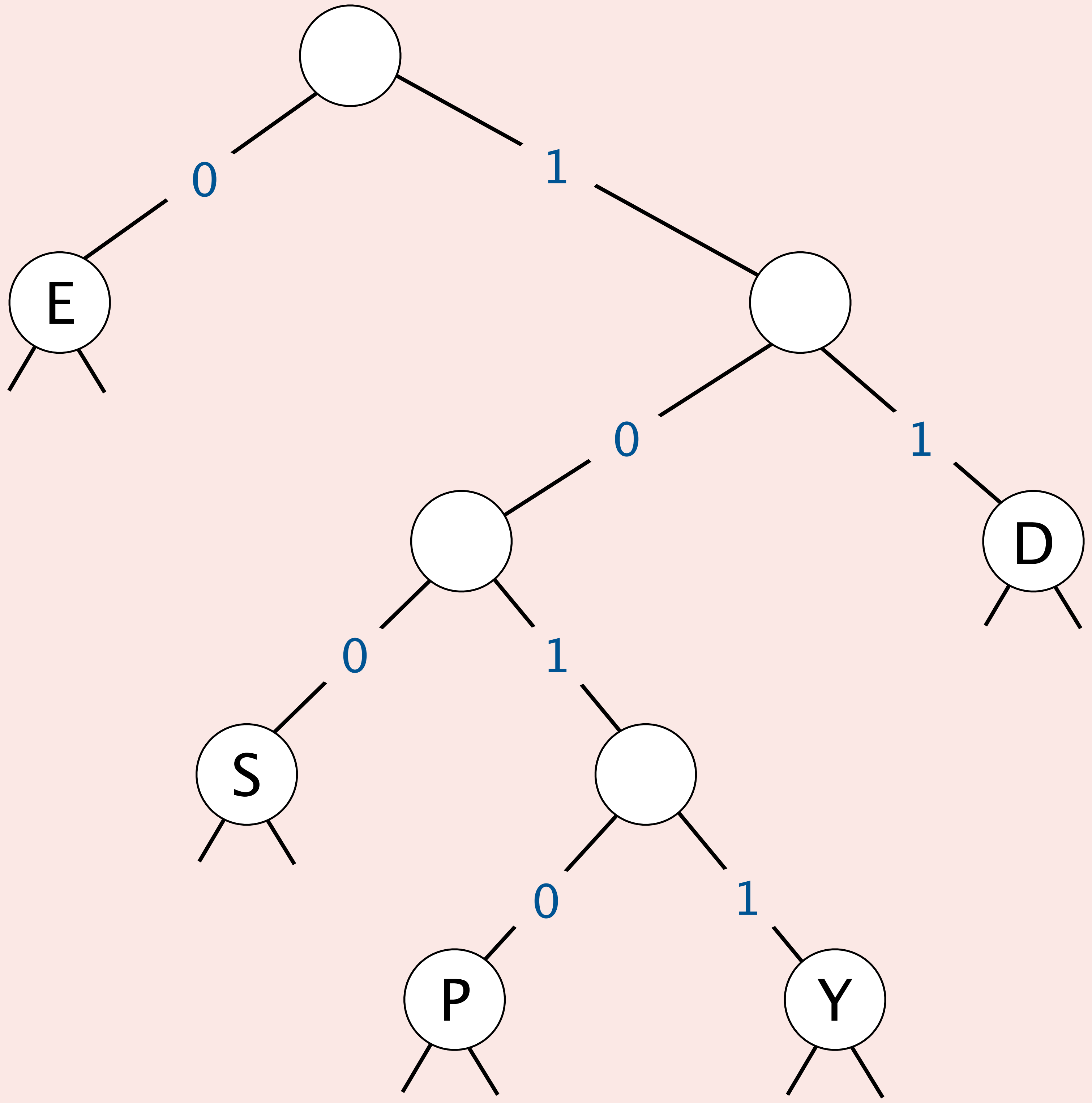


Data compression: quiz 3

Consider the following trie representation of a prefix-free code.

Expand the compressed bitstring 1 0 0 1 0 1 0 0 0 1 1 1 0 1 1 ?

- **A.** PEED
- **B.** PESDEY
- **C.** SPED
- **D.** SPEEDY



Prefix-free codes: expansion

```
public void expand()
{
   Node root = readTrie();
   int n = BinaryStdIn.readInt();

   for (int i = 0; i < n; i++)
   {

      Node x = root;
      while (!x.isLeaf())
      {
         if (!BinaryStdIn.readBoolean())
            x = x.left;
         else
            x = x.right;
      }
      BinaryStdOut.write(x.ch, 8);

   }
   BinaryStdOut.close();
}
```

read encoding trie

read number of encoded characters (32 bits)

for each encoded character *i*

follow path from root to leaf
to determine character

read 0 or 1 (1 bit)

write character (8 bits)

don't forget this!

Running time. Linear in input size (number of bits).

Huffman compression overview

Static model. Use the same prefix-free code for all messages.

Dynamic model. Use a custom prefix-free code for each message.

Compression.

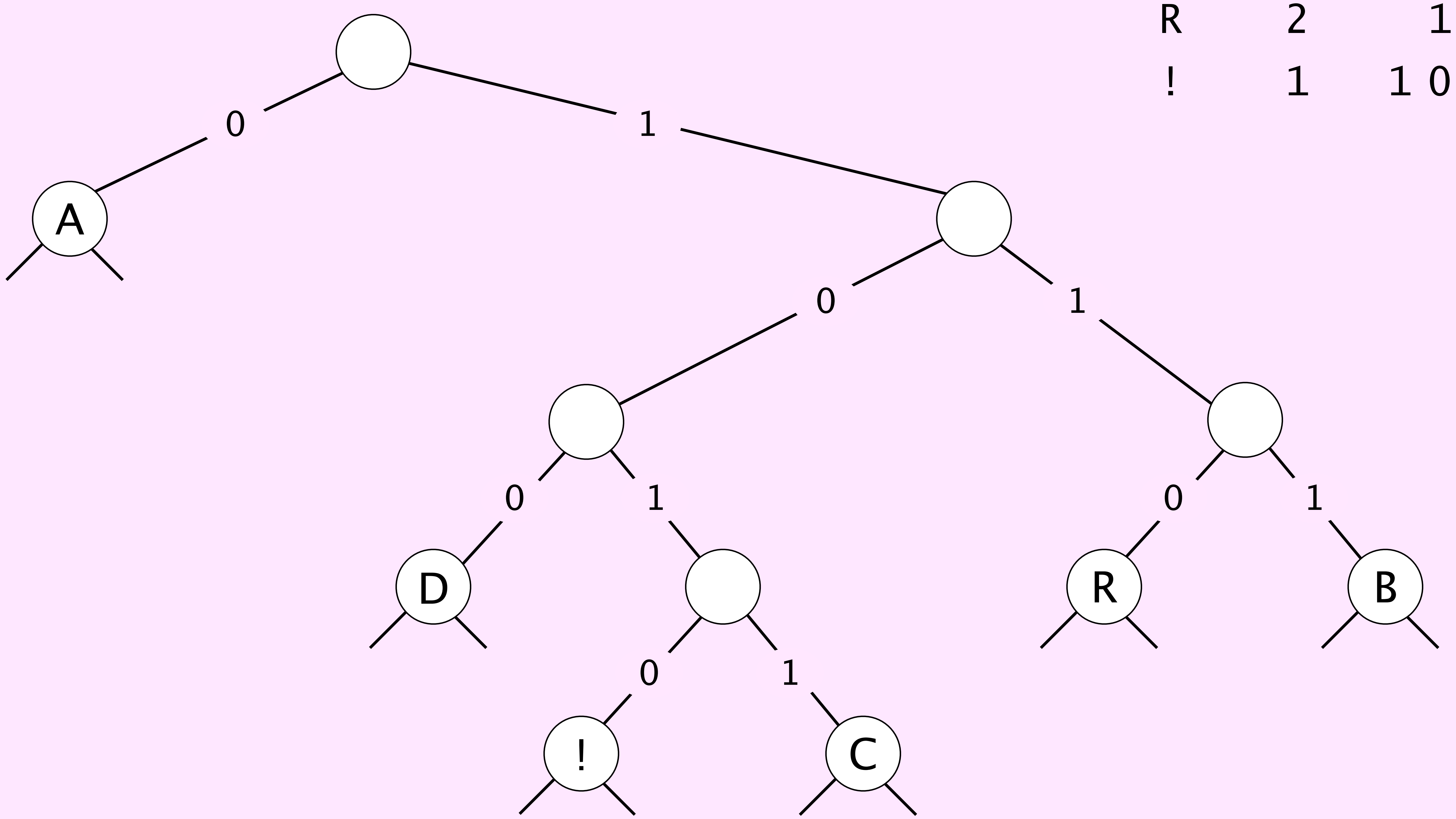
- Read message.
- Build best prefix-free code for message using Huffman's algorithm. [next]
- Write prefix-free code.
- Compress message using prefix-free code.

Expansion.

- Read prefix-free code.
- Read compressed message and expand using prefix-free code.

Huffman's 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's algorithm

Huffman's algorithm:

- Count frequency `freq[c]` of each character `c` in input.
- Start with one node corresponding to each character `c` (with weight `freq[c]`).
- 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]`

Proposition. Huffman's algorithm computes an optimal prefix-free code for a given message.

Pf. See textbook.

no prefix-free code
uses fewer bits

Applications:

Constructing a Huffman trie: Java implementation

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

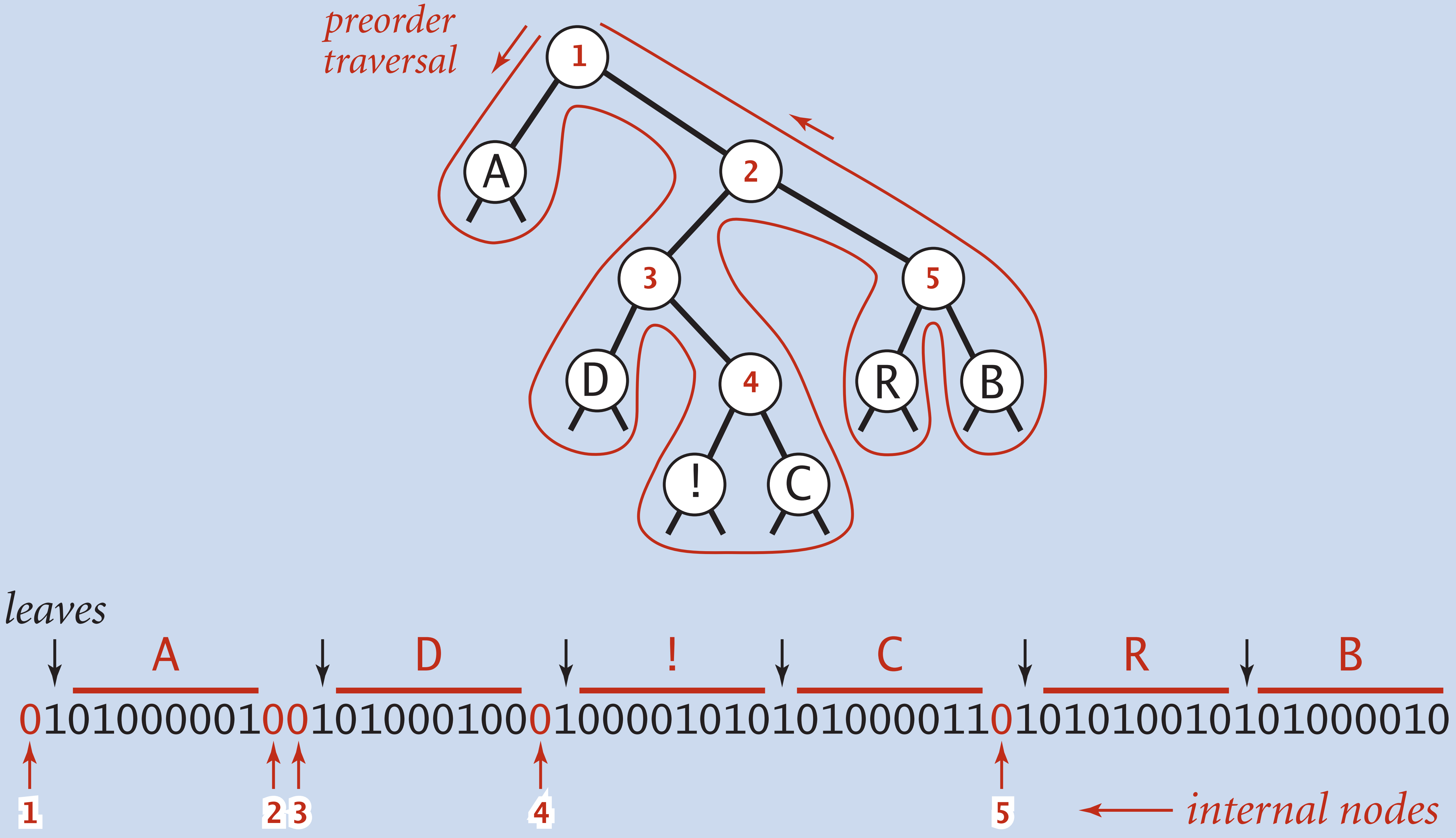
two subtries

SERIALIZING THE BINARY TRIE

Q. How to transmit the binary trie?

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

0 for internal nodes
1 for leaf nodes



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

https://algs4.cs.princeton.edu

5.5 DATA COMPRESSION

- ▸ *introduction*
- ▸ *run-length encoding*
- ▸ *Huffman compression*
- ▸ *LZW compression*

Statistical methods

Static model. Same model for all messages.

- Fast but not optimal: different messages have different statistical properties.
- Ex: ASCII, Morse code.

Dynamic model. Generate model based on message.

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

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

- More refined modeling can produces better compression.
- Ex: LZW.

LZW compression demo (for 7-bit chars and 8-bit codewords)

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
⋮	⋮
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A
ABRA	8B

codeword table

Input.

- 7-bit ASCII chars.
- ASCII 'A' is 41_{16}.

Codeword table.

- 8-bit codewords: 00_{16} to FF_{16}.
- Codewords for single chars are ASCII values.
- Codewords 81_{16} to FF_{16} for multiple chars.
- Stop symbol = 80_{16}.

LZW expansion demo (for 7-bit chars and 8-bit codewords)

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
⋮	⋮
81	AB
82	BR
83	RA
84	AC
85	CA
86	AD
87	DA
88	ABR
89	RAB
8A	BRA
8B	ABRA

codeword table

Input.
- 7-bit ASCII chars.
- ASCII 'A' is 41_{16}.

Codeword table.
- 8-bit codewords: 00_{16} to FF_{16}.
- Codewords for single chars are ASCII values.
- Codewords 81_{16} to FF_{16} for multiple chars.
- Stop symbol = 80_{16}.

Data compression: quiz 4

Which is the LZW compression for ABABABA ?

A. 41 42 41 42 41 42 80

B. 41 42 41 81 81 80

C. 41 42 81 81 41 80

D. 41 42 81 83 80

Data compression: quiz 5

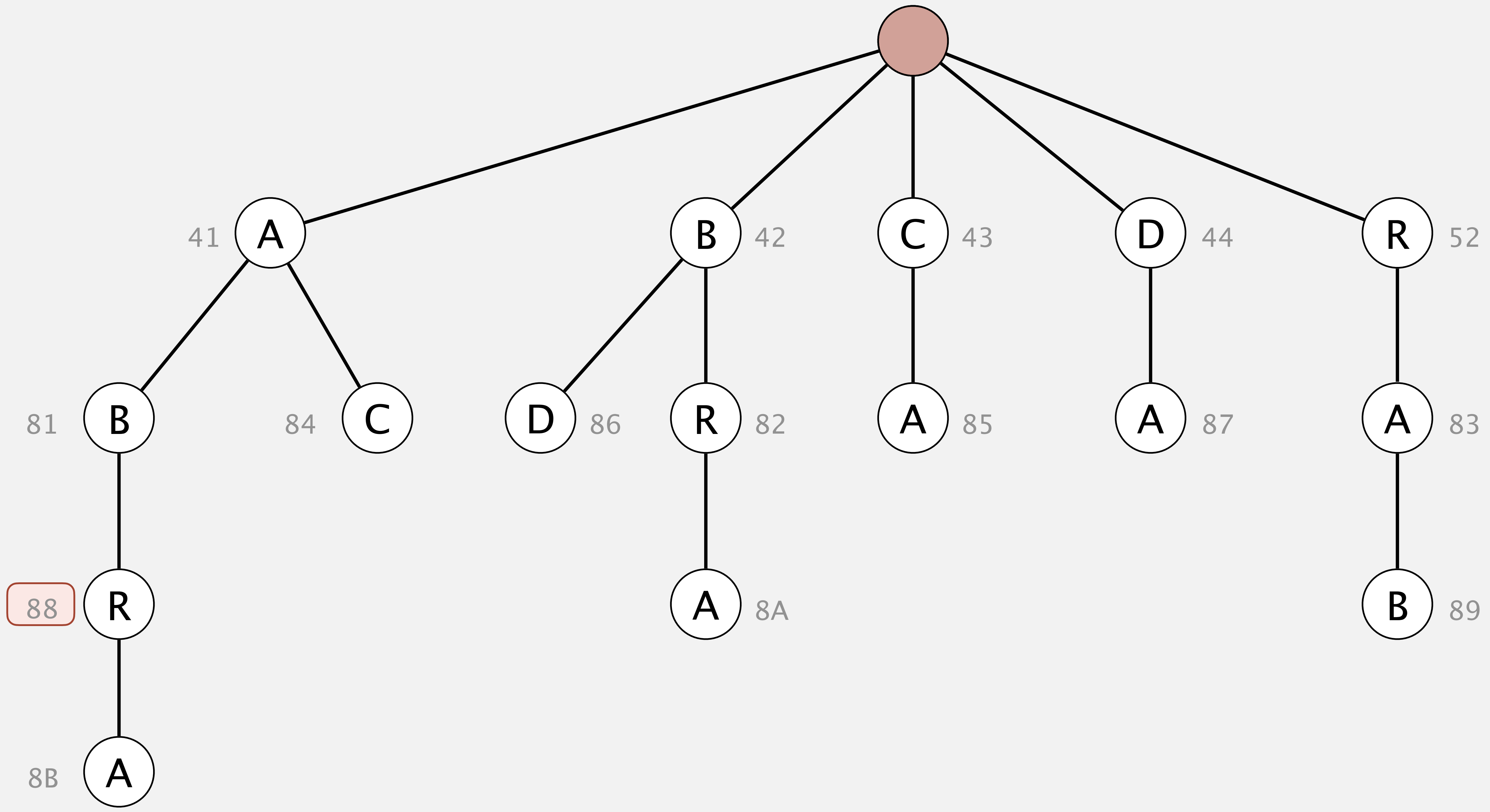
Which is the key data structure to implement LZW compression efficiently?

- **A.** Array
- **B.** Red–black BST
- **C.** Hash table
- **D.** Trie

Implementing LZW compression: longest prefix match

Find longest key in symbol table that is a prefix of query string.

input	…	A	B	R	A	C	A	B	R	O	C	C	O	L	I	…
value		88			41	43	88									



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 Web Server

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, …

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

Theoretical limits on compression. Shannon entropy: $H(X) = -\sum_{i}^{n} p(x_i) \log_2 p(x_i)$

Practical compression. Exploit extra knowledge whenever possible.