Problem 169: DNA Storage

Difficulty: Medium

Author: Brett Reynolds, Annapolis Junction, Maryland, United States

Originally Published: Code Quest 2022

Problem Background

The ability to store and retrieve data is a critical part of any computing system. As we develop more and more powerful computer systems, however, this becomes an even greater challenge. In order to keep up with the rate at which our computers are able to process data, we have to develop systems that can store very large amounts of data very quickly. So far, we've made significant progress - the amount of data that can be stored on a single MicroSD card would require entire rooms full of magnetic tapes just half a century ago. However, we have to continue to find more efficient storage methods, and recent studies have found that we may have had the solution inside us all along.

DNA, or deoxyribonucleic acid, is a complex molecule found in the cells of every living thing. DNA contains instructions on how our cells should function; it's effectively life's "programming language." It takes a lot of DNA to describe how a human, plant, or animal should function, and that correlates to a lot of data. For this reason, recent research has tried to investigate ways to use DNA molecules to store and retrieve computing data. Despite the amount of information DNA is able to store, it is still a molecule, and therefore presents an extremely space-efficient means of storing data.

Lockheed Martin is working with a startup biotech company to develop a device that can read (or "sequence") DNA molecules to convert data into binary computer data. Given the structure of a DNA molecule, your device will need to translate and display the information it contains.

Problem Description

DNA consists of two twisted chains of molecules called "nucleotides." When reading information from a DNA molecule, special enzymes are used to unravel a DNA molecule to read the nucleotides contained in a single chain. There are four kinds of nucleotides found in DNA, each typically represented by a single letter: Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). These nucleotides create pairs which join the two chains together to form the complete DNA molecule. A always pairs with T, and G always pairs with C. Since only two pairs are possible, this allows us to create a DNA molecule that represents a binary string, without being concerned about which chain in the molecule gets read.

For this problem, you will be given the nucleotide sequence of one chain of a DNA molecule that contains computing data. You will need to translate the nucleotide bases into binary digits as shown:

Adenine (A) or Thymine (T) = 0

Guanine (G) or Cytosine (C) = 1

Once translated into a binary string, you will need to convert the binary values to ASCII characters. Each character will be represented by seven bits (seven nucleotides), and all characters will be printable. For example, see the DNA sequence below:

G A T C A T A	G C A T C A G	C G A G C T A	G C A C G T A	C G A G C G C
1 0 0 1 0 0 0	1 1 0 0 1 0 1	1 1 0 1 1 0 0	1 1 0 1 1 0 0	1 1 0 1 1 1 1
H	e	l	l	o

A full listing of all US ASCII characters and their binary equivalents is provided at the end of this document.

Sample Input

The first line of your program's input, received from the standard input channel, will contain a positive integer representing the number of test cases. Each test case will consist of a single line of text containing only the characters A, C, G, and/or T. The number of characters in each line will be evenly divisible by seven.

```
2
GATCATAGCATCAGCGAGCTAGCACGTACGAGCGC
CTTATGGCTCATTCTGGATGTACGAATTACCATGTAGCATTATCTAATG
```

Sample Output

For each test case, your program must print the text represented by the binary string contained within the given DNA sequence.

```
Hello
CQ2020!
```

US ASCII Table

The inputs for all problems make use of printable US ASCII characters. Non-printable or control characters will not be used in any problem or appear in the outputs for any problem unless explicitly noted otherwise within the problem description. In some cases, you may be asked to convert characters to or from their numeric equivalents, shown in the table below.

Binary	Decimal	Character	Binary	Decimal	Character	Binary	Decimal	Character
0100000	32	(space)	1000000	64	@	1100000	96	`
0100001	33	!	1000001	65	A	1100001	97	a
0100010	34	"	1000010	66	B	1100010	98	b
0100011	35	#	1000011	67	C	1100011	99	c
0100100	36	$	1000100	68	D	1100100	100	d
0100101	37	%	1000101	69	E	1100101	101	e
0100110	38	&	1000110	70	F	1100110	102	f
0100111	39	'	1000111	71	G	1100111	103	g
0101000	40	(	1001000	72	H	1101000	104	h
0101001	41	)	1001001	73	I	1101001	105	i
0101010	42	*	1001010	74	J	1101010	106	j
0101011	43	+	1001011	75	K	1101011	107	k
0101100	44	,	1001100	76	L	1101100	108	l
0101101	45	-	1001101	77	M	1101101	109	m
0101110	46	.	1001110	78	N	1101110	110	n
0101111	47	/	1001111	79	O	1101111	111	o
0110000	48	0	1010000	80	P	1110000	112	p
0110001	49	1	1010001	81	Q	1110001	113	q
0110010	50	2	1010010	82	R	1110010	114	r
0110011	51	3	1010011	83	S	1110011	115	s
0110100	52	4	1010100	84	T	1110100	116	t
0110101	53	5	1010101	85	U	1110101	117	u
0110110	54	6	1010110	86	V	1110110	118	v
0110111	55	7	1010111	87	W	1110111	119	w
0111000	56	8	1011000	88	X	1111000	120	x
0111001	57	9	1011001	89	Y	1111001	121	y
0111010	58	:	1011010	90	Z	1111010	122	z
0111011	59	;	1011011	91	[	1111011	123	{
0111100	60	<	1011100	92	\	1111100	124	|
0111101	61	=	1011101	93	]	1111101	125	}
0111110	62	>	1011110	94	^	1111110	126	~
0111111	63	?	1011111	95	_			