Algorithm redundancy

Real-life representations are often prone to corruption. Biological codes, like RNA, may mutate naturally[1] and during measurement; cosmic radiation and other ambient noise can flip bits in computer storage[2]. One way to recover from corrupted data is to introduce or exploit redundancy.

Consider the following algorithm to introduce redundancy in a string of 0s and 1s.

Create redundancy by repeating each bit three times

```
1  procedure redun3(a_{k-1} ··· a_0: a nonempty bitstring)
2  for i := 0 to k − 1
3    c_{3i} := a_i
4    c_{3i+1} := a_i
5    c_{3i+2} := a_i
6  return c_{3k-1} ··· c_0
```

Decode sequence of bits using majority rule on consecutive three bit sequences

```
1  procedure decode3(c_{3k-1} ··· c_0: a nonempty bitstring whose length is an integer multiple of 3)
2  for i := 0 to k − 1
3    if exactly two or three of c_{3i}, c_{3i+1}, c_{3i+2} are set to 1
4      a_i := 1
5    else
6      a_i := 0
7  return a_{k-1} ··· a_0
```

Give a recursive definition of the set of outputs of the *redun*3 procedure, *Out*,

Consider the message $m = 0001$ so that the sender calculates $redun3(m) = redun3(0001) = 000000000111$.

Introduce ___ errors into the message so that the signal received by the receiver is _____ but the receiver is still able to decode the original message.

Challenge: what is the biggest number of errors you can introduce?

Building a circuit for lines 3-6 in *decode* procedure: given three input bits, we need to determine whether the majority is a 0 or a 1.

c_{3i}	c_{3i+1}	c_{3i+2}	a_i
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

Circuit

[1] Mutations of specific RNA codons have been linked to many disorders and cancers.

[2] This RadioLab podcast episode goes into more detail on bit flips: `https://www.wnycstudios.org/story/bit-flip`

Algorithm rna mutation insertion deletion

Recall that S is defined as the set of all RNA strands, nonempty strings made of the bases in $B = \{\mathtt{A}, \mathtt{U}, \mathtt{G}, \mathtt{C}\}$. We define the functions

$$\mathit{mutation} : S \times \mathbb{Z}^+ \times B \to S \qquad \mathit{insertion} : S \times \mathbb{Z}^+ \times B \to S$$

$$\mathit{deletion} : \{s \in S \mid \mathit{rnalen}(s) > 1\} \times \mathbb{Z}^+ \to S \qquad \text{with rules}$$

```
1  procedure mutation(b_1 ··· b_n : a RNA strand, k : a positive integer, b : an element of B)
2  for i := 1 to n
3    if i = k
4      c_i := b
5    else
6      c_i := b_i
7  return c_1 ··· c_n {The return value is a RNA strand made of the c_i values}
```

```
1  procedure insertion(b_1 ··· b_n : a RNA strand, k : a positive integer, b : an element of B)
2  if k > n
3    for i := 1 to n
4      c_i := b_i
5    c_{n+1} := b
6  else
7    for i := 1 to k − 1
8      c_i := b_i
9    c_k := b
10   for i := k + 1 to n + 1
11     c_i := b_{i−1}
12 return c_1 ··· c_{n+1} {The return value is a RNA strand made of the c_i values}
```

```
1  procedure deletion(b_1 ··· b_n : a RNA strand with n > 1, k : a positive integer)
2  if k > n
3    m := n
4    for i := 1 to n
5      c_i := b_i
6  else
7    m := n − 1
8    for i := 1 to k − 1
9      c_i := b_i
10   for i := k to n − 1
11     c_i := b_{i+1}
12 return c_1 ··· c_m {The return value is a RNA strand made of the c_i values}
```

Algorithm definition

New! An algorithm is a finite sequence of precise instructions for solving a problem.

Algorithms can be expressed in English or in more formalized descriptions like pseudocode or fully executable programs.

Sometimes, we can define algorithms whose output matches the rule for a function we already care about. Consider the (integer) logarithm function

$$logb : \{b \in \mathbb{Z} \mid b > 1\} \times \mathbb{Z}^+ \quad \rightarrow \quad \mathbb{N}$$

defined by

$$logb(\ (b, n)\ ) = \text{greatest integer } y \text{ so that } b^y \text{ is less than or equal to } n$$

Calculating integer part of base b logarithm

```
1   procedure logb(b,n: positive integers with b>1)
2   i := 0
3   while n > b-1
4     i := i+1
5     n := n div b
6   return i {i holds the integer part of the base b logarithm of n}
```

Trace this algorithm with inputs $b = 3$ and $n = 17$

	b	n	i	$n > b - 1$?
Initial value	3	17		
After 1 iteration				
After 2 iterations				
After 3 iterations				

Compare: does the output match the rule for the (integer) logarithm function?

Base expansion algorithms

Two algorithms for constructing base b expansion from decimal representation

Most significant first: Start with left-most coefficient of expansion (highest value)

Informally: Build up to the value we need to represent in "greedy" approach, using units determined by base.

Calculating base b expansion, from left

```
1  procedure baseb1(n,b: positive integers with b>1)
2  v := n
3  k := 1+ output of logb algorithm with inputs b and n
4  for i := 1 to k
5     a_{k-i} := 0
6     while v ≥ b^{k-i}
7        a_{k-i} := a_{k-i} + 1
8        v := v - b^{k-i}
9  return (a_{k-1},...,a_0){(a_{k-1}...a_0)_b is the base b expansion of n}
```

Least significant first: Start with right-most coefficient of expansion (lowest value)

Idea: (when $k > 1$)

$$\begin{aligned} n &= a_{k-1}b^{k-1} + \cdots + a_1 b + a_0 \\ &= b(a_{k-1}b^{k-2} + \cdots + a_1) + a_0 \end{aligned}$$

so $a_0 = n$ **mod** b and $a_{k-1}b^{k-2} + \cdots + a_1 = n$ **div** b.

Calculating base b expansion, from right

```
1  procedure baseb2(n, b: positive integers with b > 1)
2  q := n
3  k := 0
4  while q ≠ 0
5     a_k := q mod b
6     q := q div b
7     k := k + 1
8  return (a_{k-1}, ..., a_0){(a_{k-1} ... a_0)_b is the base b expansion of n}
```

Base conversion algorithm

Practice: write an algorithm for converting from base b_1 expansion to base b_2 expansion: