More **bits** of CS

*Too many bits?  Compress!*

Below binary: **physical circuits**

---

Circuit design, part 1

I'd call this a KNOT gate...

This circuit was NOT, in fact, designed!

---

**Hw #4 due Mon. 10/7**

- **pr0 (reading)** A bug and a crash!
- **pr1 (lab)** binary ~ decimal
- **pr2** conversion + compression
- **extra** image processing...

**Lots of tutoring hrs - join in... !**

**Office hours == Fri. aft.**
def numToBin( N ):
    """ converts a decimal int to a binary string """
    if N==0:
        return ''
    else:
        return numToBin( N//2 ) + str(N%2)

What if you wanted base-3 output?!  base-B output?
def binToNum( S ):
    """ converts a binary string to a decimal int """
    if S=='': return 0
    else: return 2*binToNum(S[:-1]) + int( S[-1] )

What if you wanted base-3 input?!  *base-B input?* saves the need for another *if*
# Bits & Binary

<table>
<thead>
<tr>
<th>value</th>
<th>representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
<tr>
<td>42</td>
<td>101010</td>
</tr>
<tr>
<td>127</td>
<td>11111111</td>
</tr>
</tbody>
</table>

- **shifting bits left** $\ll 1$
  - What's 101000?

- **shifting bits right** $\gg 1$
  - What's 1000?

- Maximum value of 4 bits?
- Maximum value of 7 bits?
- Maximum value of $N$ bits?!
How high can we count...?

<table>
<thead>
<tr>
<th>Bits</th>
<th>Count</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2 bits</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>3 bits</td>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>4 bits</td>
<td>1111</td>
<td>15</td>
</tr>
<tr>
<td>7 bits</td>
<td>1111111</td>
<td>127</td>
</tr>
<tr>
<td>8 bits</td>
<td>11111111</td>
<td>255</td>
</tr>
<tr>
<td>N bits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I can see some patterns here – even with one eye closed!
Counting sheep, xkcd style...

How many bits?
This week's reading: *bits can be vital*
How high can we count... \textit{in 2015?}

<table>
<thead>
<tr>
<th>Rank</th>
<th>Video name</th>
<th>Uploader / artist</th>
<th>Views (as of September 29, 2015)</th>
<th>Upload date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>&quot;Baby&quot;</td>
<td>Justin Bieber featuring Ludacris</td>
<td>1,216,729,955</td>
<td>February 19, 2010</td>
<td>[C]</td>
</tr>
<tr>
<td>2.</td>
<td>&quot;Blank Space&quot;</td>
<td>Taylor Swift</td>
<td>1,173,509,710</td>
<td>November 10, 2014</td>
<td>[D]</td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How high can we count... in 2015!

<table>
<thead>
<tr>
<th>Rank</th>
<th>Video name</th>
<th>Uploader / artist</th>
<th>Views (as of September 29, 2015)</th>
<th>Upload date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>&quot;Baby&quot;</td>
<td>Justin Bieber featuring Ludacris</td>
<td>1,216,729,955</td>
<td>February 19, 2010</td>
<td>[C]</td>
</tr>
</tbody>
</table>
Other overflow errors...

Less worrisome, perhaps...

The "sign bit" has flipped to one. Thus, the number has become negative...!
Hw4: images are just bits, too!

old pixel at 42,42 has

- red = 1 (out of 255)
- green = 36 (out of 255)
- blue = 117 (out of 255)

new pixel at 42,42 has

guesses as to what this transformation was?

how many bits represent each color channel?
Hw4: *images are just bits, too!*  

old pixel at 42,42 has:

- red = 1 (out of 255)
- green = 36 (out of 255)
- blue = 117 (out of 255)

new pixel at 42,42 has:

- red = 254 (out of 255)
- green = 219 (out of 255)
- blue = 138 (out of 255)

how many bits represent each color channel?
Hw4: images are just bits, too!

Binary Image

Encoding as raw bits
one big string of 64 characters

"10101010 01010101 10101010 01010101 10101010 01010101 10101010 01010101"

especially binary images
Too many pixels... too little time + space!

image compression is everywhere!
Too many pixels... too little time + space!

Image compression is everywhere!

How is it possible to throw away 98% of the image data!?
One solution!

How is it possible to throw away 98% of the image data!?  

We throw away 98% of the image area!

Looks like the right 2% to keep!
More often... what's done?

*compressed to 40kb*

*original: 2.3mb*
Hw4: **lossless** binary image compression

Binary Image

Encoding as raw bits

```
00000000
00000000
11111111
11111111
00000000
00000000
00000000
00011111
```

same-data streaks
a very compressible image...
Hw4: **lossless** binary image compression

If our images tend to have long streaks of unchanging data, how might we represent it more efficiently, *but still in binary*?

"0000000000000000111111111111111100000000000000000000000000000000"
Hw4: lossless image compression

One possible algorithm:

```
bit #repeats
```

Any problems with this?
Hw4: lossless image compression

It's ambiguous! this could just be a huge number of 0 pixels!

our algorithm:

bit #repeats

could be misinterpreted!
fixed-width compression

We need fixed-width blocks:

<table>
<thead>
<tr>
<th>bit</th>
<th>#repeats</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit fill</td>
<td>7 bits for the # of repeats</td>
</tr>
</tbody>
</table>

8-bits total
If you use **7 bits** to hold the # of consecutive repeats, what is the largest number of bits that *one block can represent*?

---

What if you need a **larger** # of repeats?
def compress(I):
    """ returns the RLE of the input binary image, I """

a binary image, I

"00000000000000000000000000000000000000001111111111111111111111111111111"

42 zeros

111111111111111111111111111111111111111111

31 ones

"00101010100011111"

42, in binary

31, in binary

the "compressed" output returned by compress(I)
```python
def compress(I):
    ""
    returns the RLE of the input binary image, I ""
    
    a 64-bit binary image, IQuiz

    "0000000000001111111111111111111110000000000000000000000000000000000000000011111111111"

    12 zeros     20 ones     21 zeros     11 ones

    compress(I)
```

Then, discuss...

What helper function would be useful for `compress`?

What's an image `I` whose compressed output gets larger, not smaller? (Aargh!)

- What are the BEST-compressible / WORST-compressible 64-bit images?
- How could you improve the algorithm so that it always compresses?!!
def compress(I):
    """ returns the RLE of the input binary image, I """

    a binary image, IQuiz

"""0000000000001111111111111111111100000000000000000000011111111111"

    12 zeros   20 ones   21 zeros   11 ones

    """000011001001010100000101011100010111"

    12    20    21    11

    the "compressed" image returned from compress(IQuiz)
Use this!

`frontNum(S)` returns the # of times the first element of the input `S` appears consecutively at the start of `S`:

```
def frontNum(S):
    if len(S) <= 1:
        return
    elif len(S)== 0:
        return
    else:
        return
```

- `frontNum('1111010')`: 4
- `frontNum('00110010')`: 2

The function `frontNum(S)` checks for two base cases:

1. If `S` is empty or contains only one element,
2. Or if the first two bits of `S` match.

If the first two bits DON'T match...

- If `S == ''` or `S == '1'` or `S == '0'`

- If the first two bits DO match....

- If the first two bits DON'T match....

BEST / WORST images?
What are the **BEST** and the **WORST** compression results you can get for an 8x8 image input (64 bits)?

How could we improve this compression algorithm so that all images compress to smaller than the originals? That is, how can we make compression always work?
What are the **BEST** and the **WORST** compression results you can get for an 8x8 image input (64 bits)?

- **BEST**: only 8 bits total!
- **WORST**: aargh! 512 bits!

How could we improve this compression algorithm so that *all images* compress to smaller than the originals? That is, how can we make compression always *work*?
What are the BEST and the WORST compression results you can get for an 8x8 image input (64 bits)?

**BEST**
only 8 bits total!

**WORST**
aargh! 512 bits!

How could we improve this compression algorithm so that all images compress to smaller than the originals? That is, how can we make compression always work?
What are the BEST and the WORST compression results you can get for an 8x8 image input (64 bits)?

- **BEST**: only 8 bits total!
- **WORST**: aargh! 512 bits!

Impossible! *Provably*!

How could we improve this compression algorithm so that *all images* compress to smaller than the originals? That is, how can we make compression always work?
It's **all** bits!

even the string 'forty*two' is represented as a sequence of bits...

\['\textbf{forty*two}'\]

0110011001101111011100100111010001111001001010100111011101101111

9 ASCII characters
8 bits each
9*8 == 72 bits total

\textit{All computation} boils down to manipulating bits!
All computation is simply functions of bits.
if $S[-1] == '0'$ and $T[-1] == '0'$:

return ______________ + '0'

<table>
<thead>
<tr>
<th>S</th>
<th>T</th>
<th>Output, $S+T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>001</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>010</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>011</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>001</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>010</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>011</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>010</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>011</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>101</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>011</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>110</td>
</tr>
</tbody>
</table>

This week: you'll build this in Python.

bitwise addition function

addB
if $S[-1] == '1'$ and $T[-1] == '1'$:
    return _____________________ + '0'
Adding strings?

Multiplying by machine:

Doing anything by machine...

is circuit addition!

is syntactic addition!

syntactic ～ meaning-free

is circuit multiplying!

is syntactic multiplying!

is circuit interaction!

is syntactic interaction!

means it can be done purely via surface syntax, which means it can be done without thinking...
In a computer, each bit is represented as a **voltage** (1 is +5v and 0 is 0v)

Computation is simply the **deliberate combination** of those voltages!

But what's this green thing?

---

42

101010

(1) *set input voltages*

---

ADDER circuit

001001
In a computer, each bit is represented as a voltage (1 is $+5\text{v}$ and 0 is $0\text{v}$)

Computation is simply the deliberate combination of those voltages!

But what's this green thing?

42

| 101010 |
|---|---|---|---|---|

(1) set input voltages

(2) perform computation

ADDER circuit

9

| 001001 |

110011
In a computer, each bit is represented as a voltage (1 is +5v and 0 is 0v)

Computation is simply the deliberate combination of those voltages!

Richard Feynman: "Computation is just a physics experiment that always works!"

---

**But what's this green thing?**

1. Set input voltages
2. Perform computation
3. Read output voltages

42

101010

42

ADDER circuit

001001

51

110011

9

Richard Feynman: "Computation is just a physics experiment that always works!"
Our building blocks: *logic gates*

**AND** outputs 1 only if **ALL** inputs are 1

**OR** outputs 1 if **ANY** input is 1

**NOT** reverses its input

These circuits are *physical* functions of bits...

... and *all* mathematical functions can be built from them!
Our building blocks: logic gates

**AND** outputs 1 only if **ALL** inputs are 1

**OR** outputs 1 if **ANY** input is 1

**NOT** reverses its input

... and *all* mathematical functions can be built from them!
From gates to **circuits**...

What inputs make this circuit output 1?

What inputs make this circuit output 0?
From gates to *circuits*...

What circuit outputs 1 for these four inputs?

... and outputs 0 for these four inputs?!
from circuit design...

next 2 weeks

...to a full computer!

Have an outstanding and fortunate weekend!

Why ?!