Lecture 10: Representing Numbers, Gray Codes

Base Basics

- For a base b, the highest value of any digit is $b - 1$
- Computer scientists often use non-decimal bases

$\text{binary}_2 \rightarrow$ Storing flags in a byte: 01100101
$\text{octal}_8 \rightarrow$ Unix permission bits: 0755
$\text{hexadecimal}_{16} \rightarrow$ RGB color codes: #FF751A

  - Hexadecimal uses 0–9 and A–F, where $A = 10$ and $F = 16$

Base Basics

- For a base b, the highest value of any digit is $b - 1$
- Computer scientists often use non-decimal bases

$\text{binary}_2 \rightarrow$ Storing flags in a byte: 01100101
$\text{octal}_8 \rightarrow$ Unix permission bits: 0755
$\text{hexadecimal}_{16} \rightarrow$ RGB color codes: #FF751A

  - Hexadecimal uses 0–9 and A–F, where $A = 10$ and $F = 16$

Question: Why might we favor these particular bases in computing?

Polynomial Expansions

- In decimal, we can represent:
  - 10 different numbers using one digit: (0–9)
  - 100 different numbers with two digits: (00–99)
  - 1000 different numbers with three digits: (000–999)
  - 10^n unique numbers with $n \geq 1$ digits
- To understand why, consider the polynomial expansion of 1993

$$(1 \times 10^3) + (9 \times 10^2) + (9 \times 10^1) + (3 \times 10^0).$$

Polynomial Expansions Using Powers of 2

- We can do the same polynomial expansions using other powers.
- $22 = 10110$ in binary$_2$

$$(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

- $23 = 10111$ in binary$_2$

$$(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

Polynomial Expansions Using Powers of 2

- We can do the same polynomial expansions using other powers.
- $22 = 10110$ in binary_2

$$(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

- $23 = 10111$ in binary_2

$$(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

Insight: A number is odd if and only if the last binary bit is a 1

Converting to Binary

- Checking for even/odd values tells us the last bit of a number's binary representation

Task: Implement the function `num_to_binary(num)`, which takes a number and returns its binary representation as a list of bits.

Hint: What happens if we divide a number by 2 using integer division?

Converting to Binary

```
def num_to_binary(num):
    """
    return the binary representation of num as a list of bits (i.e., the
      integers 0 and 1)
    """
    if num == 0:
        return [0]

    bits = []
    while num > 0:
        if num % 2 == 0:
            bits.append(0)
        else:
            bits.append(1)
        num = num // 2
    bits.reverse()
    return bits
```

Generalized Conversion

How would we generalize this function to other bases?

Hint: for any base b, the largest value of any digit is $b - 1$

Generalized Conversion

```
def num_to_baseb(num, b) :
    """
    return the b-ary representation of num as a list of base-b integers
    """
    if num == 0 :
        return [0]

    digits = []
    while num > 0:
        digits.append(num % b)
        num = num // b
    digits.reverse()
    return digits
```

Generalized Conversion

```python
def num_to_baseb(num, b) :
    """
    return the b-ary representation of num as a list of base-b integers
    """
    if num == 0 :
        return [0]

    digits = []
    while num > 0:
        digits.append(num % b)
        num = num // b
    digits.reverse()
    return digits
```

This function works well, but it uses the minimum number of digits possible. What if we wanted a consistent width to our numbers?

Fixed Width Binary Lists

```
def num_to_padded_base(num, b, width) :
    digits = []
    while num > 0:
        digits.append(num % b)
        num = num // b

    digits.extend([0]*(width - len(digits)))
    digits.reverse()
    return digits
```

This also simplifies our code, since we don't have to check for 0!

Printing Fixed Width Binary Lists

```
import sys
from math import log, ceil

n = int(sys.argv[1])
b = int(sys.argv[2])
width = ceil(log(n-1, b))
for i in range(n):
    print(num_to_padded_base(i, b, width))
```

Printing Fixed Width Binary Lists

```
import sys
from math import log, ceil

n = int(sys.argv[1])
b = int(sys.argv[2])
width = ceil(log(n-1, b))
for i in range(n):
    print(num_to_padded_base(i, b, width))
```

```
$ python3 printbinary.py 8 2
[0, 0, 0]
[0, 0, 1]
[0, 1, 0]
[0, 1, 1]
[1, 0, 0]
[1, 0, 1]
[1, 1, 0]
[1, 1, 1]
```