4-floating-point-decimal-to-binary-PL

September 8, 2022

0.1 PrairieLearn HW problem

Write the function `decimal_to_floating_point`

[1]:
```
import math
```

0.2 Simple Solution

We will first obtain the value of the exponent, and then we will get the fractional part of the significand.

We know that:

$1 \leq \frac{x}{2^m} = (1.f)_2 < 2$

$\frac{1}{2} < \frac{2^m}{x} \leq 1$

$\frac{x}{2} < 2^m \leq x$

$\log\_2(\frac{x}{2}) < m \leq \log\_2(x)$

Compute the boundaries for the example above, and then determine the value of m:

[2]:
```
x = 19.25
n = 8
p = 8

# Write the boundaries and determine m

#clear
print("bounds for m = ", math.log(x/2,2),math.log(x,2) )

m = math.floor(math.log(x,2))
print("m = ", m)
```

```
bounds for m =  3.2667865406949015 4.266786540694902
m =  4
```

We now need to compute the fractional part of the significand. We know that:

$(1.f)\_2 = \frac{x}{2^m}$

Compute the value of f as a decimal number:

[3]:
```
#clear
fdec = x/2**m - 1
print("f in decimal = ", fdec)
```

```
f in decimal =  0.203125
```

Convert f to a binary number with n bits. Recall that f is now a number < 1.

[4]:
```
#clear
f = ''
temp = fdec
for i in range(n):
    temp = temp*2
    temp_int = int(temp)
    temp = temp - temp_int
    f += str(temp_int)
f
```

[4]:
```
'00110100'
```

You are ready to define the function `decimal_to_floating_point`

[5]:
```
def decimal_to_floating_point(x, n, p):
    f = ...
    m = ...
    #clear

    m = math.floor(math.log(x,2))
    temp = x/2**m - 1
    f = ''
    for i in range(n):
        temp = temp*2
        temp_int = int(temp)
        temp = temp - temp_int
        f += str(temp_int)

    #clear
    return f,m
```

[6]:
```
decimal_to_floating_point(19.25, 8, 8)
```

[6]:
```
('00110100', 4)
```

[7]:
```
decimal_to_floating_point(0.004245, n, p)
```

[7]:
```
('00010110', -8)
```

0.3 Not as simple...

Some students use the approach of splitting the decimal and integer parts. In this case, we need to make sure we are considering cases of positive and negative exponents, and also that we have the correct padding of zeros when the binary representation is less than n. Let's take a look at another possible solution (this was inpired by other student's solutions during class)

Integer part if the number x is greater than 1, we can compute the binary representation for the integer part:

[8]:
```
x = 19.25 #0.00424
n = 8
p = 8

a = int(x)
print('integer part is = ', a)

# we will solve this problem using lists
# to store the binary numbers at first
binary_integer = []

for i in range(100):
    rem = a % 2
    a = int(a/2)
    binary_integer.append(rem)
    if a == 0:
        break

# recall that we need to invert the binary list
binary_integer = binary_integer[::-1]
print('binary for integer number is = ', binary_integer)
```

```
integer part is =  19
binary for integer number is =  [1, 0, 0, 1, 1]
```

Since we want a normalized floating point number, we can get the exponent by checking how many times we need to shift the binary point from the integer number.

[9]:
```
m = len(binary_integer) - 1
print("m = ", m)
```

```
m =  4
```

Notice that this is not going to work for small numbers. Change x above to 0.0625 and see what happens. If you use this method, you need to make sure the computation above is only performed when $x > 1$. We will use that when we are puttig all the code together.

Fractional part Once we remove the integer part computed above, we can compute the binary representation for the fractional part. This will also be performed in the cases for numbers smaller than one (without an integer part).

[10]:
```
fdec = x - int(x)
print('fractional part is = ', fdec)
```

```
fractional part is =  0.25
```

[11]:
```
binary_fractional = []

temp = fdec
for i in range(100):
    temp = temp*2
    temp_int = int(temp)
    temp = temp - temp_int
    binary_fractional.append(temp_int)
    if temp == 0:
        break
```

Combining both parts We need to now put these things together into the function, and make sure we check for edge cases:

[12]:
```
def decimal_to_floating_point(x, n, p):


    a = int(x)
    binary_integer = []

    if x > 1:

        # we will solve this problem using lists
        # to store the binary numbers at first

        for i in range(100):
            rem = a % 2
            a = int(a/2)
            binary_integer.append(rem)
            if a == 0:
                break

        # recall that we need to invert the binary list
        binary_integer = binary_integer[::-1]
        m = len(binary_integer) - 1


    fdec = x - int(x)
    binary_fractional = []

    #clear
    temp = fdec
```

```
    for i in range(50):
        temp = temp*2
        temp_int = int(temp)
        temp = temp - temp_int
        binary_fractional.append(temp_int)

    if x <= 1:
        for i,b in enumerate(binary_fractional):
            if b == 1:
                break
        m = -(i+1)
        binary_fractional = binary_fractional[i:]

    print("m = ", m)
    binary_number = binary_integer + binary_fractional

    f = ''.join(map(str,binary_number[1:1+n]))
    print("f = ", f)


    return f,m
```

[13]:
```
decimal_to_floating_point(x, n, p)
```

```
m =  4
f =  00110100
```

[13]: ('00110100', 4)

1 In case you find these helpful...

1.0.1 Decimal Fraction (less than 1) to Binary

[14]:
```
def decfrac_to_binary(fdec,n=23):
    f = ''
    dig = 0
    while ((fdec > 0) & (dig<n)):
        fdec = fdec*2
        temp_int = int(fdec)
        fdec = fdec - temp_int
        f += str(temp_int)
        dig += 1
    return f
decfrac_to_binary(0.42891717890,n=6)
```

[14]: '011011'

1.0.2 Decimal Integer to Binary

[15]:
```
bin(2341)[2:]
```

[15]:
```
'100100100101'
```

1.0.3 Binary to Decimal

binary is normalized

[16]:
```
def binary_to_decimal(f,m):
    decimal = 1
    for i,digit in enumerate(f):
        decimal += int(digit)*2**(-(i+1))
    decimal *= 2**(m)
    return decimal
```

[17]:
```
binary_to_decimal('00010101',8)
```

[17]:
```
277.0
```