seti

December 4, 2018

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import math
        import itertools
        import pandas as pd

In [2]: #

        def optiSizeVec(vec):
            optiVec = []
            if len(vec) == 0:
                return optiVec
            if len(vec) == 1:
                if vec[0] == 1:
                    return vec
                else:
                    return optiVec
            for i in range(1, len(vec) + 1):
                if vec[-i] == 0:
                    continue
                else:
                    for j in range(-len(vec), -i + 1):
                        optiVec.append(vec[j])
                    break;
            return optiVec


        def BinPolySum(b1, b2):
            result = []
            shortest = None
            longest = None
            if (len(b1) == len(b2)):
                for i in range(0, len(b1)):
                    result.append((int(b1[i]) + int(b2[i])) % 2)
                result = optiSizeVec(result)
                return result
            elif (len(b1) > len(b2)):
                shortest = b2
        longest = b1
    else:
        shortest = b1
        longest = b2
    for i in range(0, len(shortest)):
        result.append((b1[i] + b2[i]) % 2)
    for j in range(len(shortest), len(longest)):
        result.append(longest[j])
    result = optiSizeVec(result)
    return result


def BinPolyMul(b1, b2):
    result = []
    for i in range(0, len(b1) + len(b2) - 1):
        result.append(0)
    for i in range(0, len(b1)):
        if b1[i] == 0:
            continue;
        for j in range(0, len(b2)):
            if b2[j] == 0:
                continue
            sumDegree = i + j
            result[sumDegree] = 1
    result = optiSizeVec(result)
    return result


def BinPolyDiv(b1, b2):
    quotient = []
    for i in range(0, len(b1)):
        quotient.append(0)
    remainder = b1
    while (len(remainder) >= len(b2)):
        # print 'Iteration'
        dividendDegree = len(remainder) - 1
        dividerDegree = len(b2) - 1
        degreeDelta = dividendDegree - dividerDegree
        quotient[degreeDelta] = 1
        subQuotient = []
        for i in range(0, degreeDelta + 1):
            if i == degreeDelta:
                subQuotient.append(1)
            else:
                subQuotient.append(0)
        summer = BinPolyMul(b2, subQuotient)
        # print 'quotient = '+str(quotient)
        # print 'subQuotient = '+str(subQuotient)
```

```
                # print 'summer = '+str(summer)
                # print 'before sum remainder = '+str(remainder)
                remainder = BinPolySum(remainder, summer)
                # print 'before optiSizeVec remainder = '+str(remainder)
                remainder = optiSizeVec(remainder)
            # print 'remainder = '+str(remainder)
            quotient = optiSizeVec(quotient)
            remainder = optiSizeVec(remainder)
            return (quotient, remainder)
```

```
In [3]: #   .
        # my_m -
        xnk = [0, 0, 0, 1]
        g = [1, 1, 0, 1]
        my_m = [1, 1, 1, 0]
```

```
In [4]: #

        def code(m):
        #       3
            mx = BinPolyMul(m, xnk)
        #
            p = BinPolyDiv(mx, g)
        #
            v = BinPolySum(mx, p[1])
        #         ,       7
            while len(v) < 7:
                v.append(0)
            return v
```

```
In [5]: #
        code(my_m)
```

```
Out[5]: [0, 1, 0, 1, 1, 1, 0]
```

```
In [6]: #    i   n
        def comb(i, n):
            return int(math.factorial(n) / (math.factorial(i) * math.factorial(n - i)))
```

```
In [7]: #
        def gen_ex(i):
            assert i <= 7, 'i must be < 7'
            ex = i * '1' + '0' * (7 - i)
            ex = list(set(itertools.permutations(ex)))
            return list(map(lambda x: [int(i) for i in x], ex))
        # gen_ex(2)
```

```
In [8]: #   ,
        def decode(v):
```

```
    n = 7
    res = {'i': [], 'C': [], 'NO': [], 'CO': [], '': []}
#
    for i in range(0, n + 1):
#
        N = 0
#
        ex_list = gen_ex(i)
#
        for a in range(0, comb(i, n)):
#
            rx = BinPolySum(v, ex_list[a])
#
            sm = BinPolyDiv(rx, g)[1]
#               ,
            if sm:
                N = N + 1
#
        res['i'].append(i)
        res['C'].append(comb(i, n))
        res['NO'].append(N)
        res['CO'].append(N / comb(i, n))
#
        if float(N) == float(comb(i, n)):
                res[''].append('   ')
        else:
            if N == 0 and comb(i, n) == 1:
                res[''].append('')
            else:
                res[''].append('  : {}'.format(100 - 100 * (N / comb(i, n))))
    return res
```

```
In [9]: # ,
        def table_data(d):
            df = pd.DataFrame(data=d)
            return df
```

```
In [10]: #
         decode(code(my_m))
```

```
Out[10]: {'i': [0, 1, 2, 3, 4, 5, 6, 7],
          'C': [1, 7, 21, 35, 35, 21, 7, 1],
          'NO': [0, 7, 21, 28, 28, 21, 7, 0],
          'CO': [0.0, 1.0, 1.0, 0.8, 0.8, 1.0, 1.0, 0.0],
          '': ['',
           '   ',
           '   ',
           '  : 20.0',
           '  : 20.0',
           '   ',
           '   ',
           '']}

In [11]: #
         table_data(decode(code(my_m)))

Out[11]:    i   C  N0   C0
         0  0   1   0  0.0
         1  1   7   7  1.0
         2  2  21  21  1.0
         3  3  35  28  0.8    : 20.0
         4  4  35  28  0.8    : 20.0
         5  5  21  21  1.0
         6  6   7   7  1.0
         7  7   1   0  0.0
```