2023 年度 大問 5

hari64boli64 (hari64boli64@gmail.com)

2025 年 4 月 23 日

1　問題

疎な三値表現

2　解答

(1)

貪欲に 1 を割り当てていくのが最善。$2^n + 2^{n-2} + \cdots$ が答え。

$$L_n = \begin{cases} \frac{2^{n+1}-1}{3} & n \text{ が奇数} \\ 2\frac{2^n-1}{3} & n \text{ が偶数} \end{cases}$$

(2)

存在性は (3) より言えるので、一意性のみ言えばよい。相異なる疎な表現 $\{d_i\}$ と $\{e_i\}$ が存在したとする。これらの内、相異なる添え字 i の内、最小のものを考える。

d_i, e_i の内、片方が 0 の場合、この桁において 2^i のズレが生じるが、これ以降の桁において生成できるズレは、2^{i+1} の倍数のみ。よって、d と e で表す数が異なり矛盾。

d_i, e_i が、共に 1 か-1 の場合、この桁において 2^{i+1} のズレが生じるが、これ以降の桁において生成できるズレは、疎性に注意すると、2^{i+2} の倍数のみ。よって、d と e で表す数が異なり矛盾。

以上より、示された。

(3)

下位のビットから見ていって、二進数の 0111(8*0+4*1+2*1+1*1) を (1,0,0,-1)(8*1+4*0+2*0+1*(-1)) に変換していくのが主な方針である。詳細は以下。計算量は自明に $O(\log d)$ である。

油断した実装だと、十進法における 11(二進表現で 1011) が三進表現で (1,1,0,-1) になって、疎で

なくなるので注意 (一敗)。正しくは、(1,0,-1,0,-1)=16-4-1=11 である。実装では、右から左でなく、左から右の向きに記しているので注意。

なお、Slack 上では、上位のビットから貪欲に見ていくという方針もあった。コード 1 において、anotherSolution という関数で実装している。尤も、二進表現を疎な三進表現に変換せよという問題だったので、あまり想定解ではないかも知れない。

Listing 1　toSparseTernaryRepresentation

```python
import math
from collections import defaultdict

from numba import jit
from tqdm import tqdm


@jit(cache=True)
def toSparseTernaryRepresentation(S: str):
    ret = ""
    i = 0
    while i < len(S):
        if S[i] == "0":
            ret += "0,"
            i += 1
        elif S[i] == "1":
            cnt = 0
            while i < len(S) and S[i] == "1":
                cnt += 1
                i += 1
            if cnt == 1:
                ret += "1,"
            else:
                ret += "-1," + "0," * (cnt - 1)
                # originally, the following
                # must be inplace operation
                # for the sake of efficiency
                S = S[:i] + "1" + S[i + 1 :]
    return ret[:-1]


def anotherSolution(n: int):
    def L(i):
        if i <= 0:
            return 0
        if i % 2 == 1:
            return (2 ** (i + 1) - 1) // 3
        else:
            return 2 * (2**i - 1) // 3
```

```
40
41      d = []
42      for i in range(math.ceil(math.log2(n)) + 3)[::-1]:
43          if abs(n - (2**i)) <= L(i - 1):
44              n -= 2**i
45              d.append(1)
46          elif abs(n + (2**i)) <= L(i - 1):
47              n += 2**i
48              d.append(-1)
49          else:
50              d.append(0)
51
52      d = d[::-1]
53      while d[-1] == 0:
54          d.pop()
55      return ",".join(map(str, d))
56
57
58  def problem5():
59      maxN = 16
60      zeroCnt = 0
61      cntPer3 = defaultdict(int)
62      for n in tqdm(range(1, (1 << maxN) + 1)):
63          S = bin(n)[2:][::-1]
64          T = toSparseTernaryRepresentation(S)
65          S += "0" * (maxN - len(S))
66          listT = list(map(int, T.split(",")))
67          listT += [0] * (maxN - len(listT))
68          if listT[maxN // 2] == 0:
69              zeroCnt += 1
70              cntPer3[tuple(S[maxN // 2 - 1 : maxN // 2 + 2][::-1])
                    ] += 1
71      print(f"result: {zeroCnt/(1<<maxN)}")
72      print(f"cntPer3: {sorted(cntPer3.items())}")
73
74
75  def main():
76      maxIntPerDigit = defaultdict(int)
77      for n in range(1, 1000 + 1):
78          S = bin(n)[2:][::-1]
79
80          # ternary representation
81          T = toSparseTernaryRepresentation(S)
82          print(f"binary: {S} | ternary: {T}")
83
84          # another solution
85          T2 = anotherSolution(n)
```

```
86            assert T == T2, f"{T} != {T2}"
87
88            # assertion
89            TasInt = sum([c * (2**i) for i, c in enumerate(list(map(
                  int, T.split(","))))])
90            assert TasInt == n
91
92            # count
93            maxIntPerDigit[len(T.split(","))] = max(maxIntPerDigit[
                  len(T.split(","))], n)
94
95        print(f"maxIntPerDigit: {maxIntPerDigit}")
96
97
98    if __name__ == "__main__":
99        main()
100       # problem5()
```

Listing 2　result

```
binary: 1 | ternary: 1
binary: 01 | ternary: 0,1
binary: 11 | ternary: -1,0,1
binary: 001 | ternary: 0,0,1
binary: 101 | ternary: 1,0,1
binary: 011 | ternary: 0,-1,0,1
binary: 111 | ternary: -1,0,0,1
binary: 0001 | ternary: 0,0,0,1
binary: 1001 | ternary: 1,0,0,1
binary: 0101 | ternary: 0,1,0,1
binary: 1101 | ternary: -1,0,-1,0,1
binary: 0011 | ternary: 0,0,-1,0,1
binary: 1011 | ternary: 1,0,-1,0,1
binary: 0111 | ternary: 0,-1,0,0,1
binary: 1111 | ternary: -1,0,0,0,1
binary: 00001 | ternary: 0,0,0,0,1
```

(4)

題意が成立しないと仮定すると、ある自然数 d に対して、疎な三値表現以外の表現によって、w の最小値は達成される。そのような最小値を達成する表現に対して、(3) と似たような手順を踏むことによって、さらに疎な表現が得られることを示し、最小値である事に対する矛盾か、疎な三値表現の一意性に対する矛盾を導く。

下位ビットから上位ビットに向かって対象としている疎でない表現に対する文字列検索を走査していく。

まず、表現に $(1,-1)$ か $(-1,1)$ が出現する場合、それぞれ $(0,1)$ と $(0,-1)$ で置き換えればより疎

な表現が得られるので、w が減少している。その時点で操作を終了する。

次に、表現に $(1,1,\cdots)$ や $(-1,-1,\cdots)$ が出現する場合について考える。そのようなまとまり (ラン) を (3) と同様に最後まで考えて、$(0,1,1,\cdots,1)$ や $(0,-1,-1,\cdots,-1)$ が出現しているとしてよい。それぞれ $(1,0,\cdots,0,-1)$ と $(-1,0,\cdots,0,1)$ でそれぞれ置き換えるという操作を行う。w が減少すれば、終了する。そうでない場合、すなわち、まとまりの長さが 2 の場合、w は変化こそしていないが、その時点までの疎性は保証されることに注意する。

以上の操作を最上位ビットになるまで繰り返す。

これが w の減少如何を問わず、最後のビットまで見た、という条件によって終了することは、3 の操作で高々 1 つしか桁が増えないこと、および、最後のビットが定義より 1 で、その先は 0 だけであることから従う。この時点で w は変化していないが、先述の通り、また、操作の内容より、必ず疎な表現になっている。そして、仮定より、この表現の w は、仮定の時点で登場している疎な三値表現の w よりも小さい。よって、自然数 d に対して二つの異なる疎な三値表現が得られるので、疎な三値表現の一意性に矛盾する。

以上より、疎な三値表現以外の表現によって、w の最小値は達成されないことが示された。

(5)

あまり想定解な気はしないが、一応 1/3 の値が出てきたので、書いておく。

ある桁に注目した際に、その桁が 0 になる確率を求める。ただし、左右の方向に無限に列が続くとする。n が無限であることから、そのような値と求めるべき値は一致する。

(3) のアルゴリズムにおける、変換前の 2 進数における状態で場合分けをして考える。すると、以下のような表が得られる。

表 1　変換前の 2 進数における状態と、その桁が 0 になる確率

変換前の 2 進数 (当該桁+前後)	その桁が 0 になる確率
000	1
001	1/2(後続が 0)×2/3(繰り上がりなし)
010	1/4(後続が 11)+1/4(後続が 10)×1/3(繰り上がり)
011	1
100	1
101	1/2(後続が 0)×2/3(繰り上がりなし)
110	1/4(後続が 11)+1/4(後続が 10)×1/3(繰り上がり)
111	1

この表における確率を合計すると、$(16/3)\times(1/8)=2/3$ となるが、これは、求めるべき確率が 1/3 であることと一致する。

この表の意味を説明する。まず、前準備として、変換前の 2 進数において、0 になっている桁が、変換により繰り上がって 1 になる確率を考える。例えば、変換前に “0”11 だった場合、変

換によって、(“1”,0,-1) と 1 になる。このような確率を求める。これは、この確率を p と置くと、$p = 0$(後続が 0)$+1/4 \times p$(後続が 10、かつ、ここでも繰り上がりが起きる場合)$+1/4$(後続が 11... の場合) であるので、$p = 1/3$ となる。

次に、実際に表を埋める。

000 の場合に当該桁 (つまり、真ん中の桁) が必ず 0 になることは、その変換アルゴリズムから明らか。

001 の場合に当該桁が 0 になる確率を考える。後続が 0 の場合、変換前の二進数は 0“0”10 となり、変換によって、(0,“0”,1,0) となるので、0 となる。しかし、先述の繰り上がりによって、後続が 0“0”11 となってしまうと、変換によって、(0,“1”,0,-1) となり、1 となる。よって、繰り上がりが起きない確率を求めなければならず、これは、$1/2 \times 2/3$ である。

010 の場合に当該桁が 0 になる確率を考える。これも殆ど同様で、繰り上がりが起きるかどうかに注意すると、これは $1/4 + 1/4 \times 1/3$ となる。

他も同様。

よって、期待値は 1/3 と示された。

3 知識

平衡三進法と呼ばれる有名な記法が元ネタ?

https://ja.wikipedia.org/wiki/%E4%B8%89%E9%80%B2%E6%B3%95#%E5%B9%B3%E8%A1%A1%E4%B8%89%E9%80%B2%E6%B3%95