第10回
科学の甲子園 全国大会

実技競技①

Challenge-18

解答例と解説

（解答例）

問題の名称	配点	解答例
【素数】問１	100	9
【素数】問２	200	78
【竹内関数】問	100	588802012
【2 進法】問	100	2020
【和が決められた数列】問１	100	3658
【和が決められた数列】問２	100	204226
【和が決められた数列】問３	100	3658
【約数】問１	200	128
【約数】問２	200	2020
【部分文字列】問	300	1304
【整数】問１	300	4104
【整数】問２	300	2463
【整数】問３	300	3392
【ドレミ暗号】問１	100	3
【ドレミ暗号】問２	200	0re040do020mi010do031fa00
【文字列操作】問	300	chopw;sort;uniqc;sort
【数列操作】問１	100	57
【数列操作】問２	200	2097130

（解説）

【素数】

問１

最初の 20 個の素数（2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71）を見つけて，これらを印刷した場合に「1」が何回印刷されるか数える。

問２

「エラトステネスの篩（ふるい）」のような素数を求めるアルゴリズムのプログラムを用い「2 から 997 までの最初の 168 個の素数」を求める。それらを印刷した場合に「1」は何回印刷されるかについてもプログラムにより求める。

（プログラム例）

```
def furui(n):
  nums = [i for i in range(2, n+1)]
  for m in nums:
    nums = [n for n in nums if (n==m or n%m!=0)]
  return nums

def countOne(s):
  count=0
  for c in str(s):
    if c=='1':
      count = count+1
  return count

count=0
for x in furui(1000):
  count = count + countOne(x)
print(count)
```

【竹内関数】

問

関数 countUp を他の関数内で呼び出すと countUp が呼び出された回数を記録することができる。最後に 2 を引いているのは tarai の最初の呼び出しと表示のために関数 countUp を呼び出している分を引くためである。

（プログラム例）

```
def countUp( ):
  countUp.cnt += 1
  return countUp.cnt

def tarai(x, y, z):
  countUp( )
  if x <= y:
    return y
  else:
    return tarai(tarai(x-1, y, z),
                 tarai(y-1, z, x),
                 tarai(z-1, x, y))

countUp.cnt = 0
tarai(14, 7, 0)
print(countUp( )-2)
```

【2 進法】

問

例えば $11111100100=(111111\times1000+1)\times100=((1000000-1)\times1000)+1)\times100$

（ここまで 2 進法，この先 10 進法）$=((2^6-1)\times2^3+1)\times2^2=((64-1)\times8+1)\times4=2020$

と手計算で計算できる。

【和が決められた数列】

問１

例えば次のような Python プログラムで求めることができる。このプログラム中の num_strict_parts(n, m)は，後の数が前の数より大きくなるように n 以上の整数を並べた列で，合計が m になるようなものの個数を求めている。

```
def num_strict_parts(n, m):
  if n > m:
    return 0
  else:
    return sum(num_strict_parts(k+1, m-k)
               for k in range(n, (m+1)//2))+1

print(num_strict_parts(1, 50))
```

プログラムで用いた考え方の解説は**問３**のところでまとめて行う。

問２

例えば次のような Python プログラムで求めることができる。このプログラム中の num_parts(n, m)は，後の数が前の数以上になるように n 以上の整数を並べた列で，合計が m になるようなものの個数を求めている。

```
def num_parts(n, m):
  if n > m:
    return 0
  else:
    return sum(num_parts(k, m-k)
               for k in range(n, m//2+1))+1

print(num_parts(1, 50))
```

プログラムで用いた考え方の解説は**問３**のところでまとめて行う。

問３

例えば次のような Python プログラムで求めることができる。このプログラム中の num_odd_parts(n, m)は，後の数が前の数以上になるように n 以上の奇数を並べた列で，合計が m になるものの個数を求めている。なお，num_odd_parts に与える最初の引数は正の奇数でなければならない。

```
def num_odd_parts(n, m):
  if n > m:
    return 0
  else:
    return sum(num_odd_parts(k, m-k)
               for k in range(n, m//2+1, 2))+m%2

print(num_odd_parts(1, 50))
```

問１から**問３**のプログラムで用いた考え方をまとめてここで述べる。

問１のように後の数が前の数より大きくなるように自然数を並べた列を「狭義増加列」，**問２**のように後の数が前の数以上になるように自然数を並べた列を「広義増加列」と呼ぶことにする。

問１は「1 以上の自然数を用いて合計が 50 になる狭義増加列」の個数を求める問題ということができる。このような列は 25 から 49 で始まることはありえない。なぜなら，50 とは異なりそこで列を終わることができず，一方で次の数を続けようとしても前の数より大きくすることができないからである。したがって最初の数として考えられるのは 1 から 24 と 50 のいずれかである。1 から 24 についてはその後を続けなければならないが，例えば 2 の後は「3 以上の自然数を用いて合計が 48 になる狭義増加列」となり，同様に考えると 24 から 47 で始まることはありえない。

つまり，「n 以上の自然数を用いて合計が m になる狭義増加列」($n \leqq m$)は，

1．n 以上 $m/2$ 未満の自然数 k で始まり，その後に「$k+1$ 以上の自然数を用いて合計が $m-k$ になる狭義増加列」が続くもの
2．m で始まり，そこで終わるもの

のいずれかとなる。**問１**はこの考え方を用いて $n=1$, $m=50$ の場合について網羅すればよい。

問２，**問３**の広義増加列に関しても同じような考え方が使える。**問３**については m が偶数の場合は上記２．に対応する場合がないことに注意が必要である。

【約数】

問１

例えば次のような Python プログラムで求めることができる。

```
def num_divisors(n):
  nd = [0] * (n+1)
  for d in range(1, len(nd)):
    for i in range(d, len(nd), d):
      nd[i] += 1
  return nd

print(max(num_divisors(100000)))
```

num_divisors(n)は添字が 0 から n で，添字 i の要素が i の正の約数の個数となるような配列（ただし添字 0 の要素は 0）を返す。自然数 d について，d の倍数は d から始めて d 個ごとに現れることを利用している。この方法だと除算を用いずに約数の個数を数えることが可能である。

問２

例えば次のような Python プログラムで求めることができる。

```
def sum_divisors(n):
  sd = [0] * (n+1)
  for d in range(1, len(sd)):
    for i in range(d, len(sd), d):
      sd[i] += d
  return sd

sd = sum_divisors(100000)
print(len([i for i in range(1, 100000+1) if sd[i] > 3 * i]))
```

基本的な考え方は**問１**の num_divisors と同じ。

【部分文字列】

問

例えば次のような Python プログラムで求めることができる。

```
def substrings(string):
  if string == '':
    return{''}
  else:
    ss = substrings(string[1:])
    s0 = string[0:1]
    return{s0 + s for s in ss}|ss

print(len(substrings('abracadabra')))
```

まず，空文字列 '' の部分文字列は空文字列のみである。空でない文字列を先頭の 1 文字 s_0 と残りの文字列 s に分けると，もとの文字列の部分文字列は，1) 文字列 s の部分文字列か，2) 文字 s_0 の後に文字列 s の部分文字列をつなげたものの形をしている。1) と 2) で網羅されるすべての文字列の間には重複があるかもしれないので，同じものは取り除きながら（上記プログラムでは集合を用いている）部分文字列を網羅していく。

なお，動的計画法と呼ばれる方法を用いるとより長い文字列についても効率的に部分文字列の個数を数えることが可能であるが，今回の問題のサイズであれば上記のような比較的簡単な方法でも現実的な時間で解くことができる。

【整数】

問 1

例えば次のプログラムによって求めることができる。

```
def taxi(n):
  m = 0
  while (m ** 3 < n):
    m = m + 1
  for a in range(1, m):
    for b in range(a, m):
      if a ** 3 + b ** 3 == n:
        for c in range(a + 1, m):
          for d in range(c, m):
            if c ** 3 + d ** 3 == n:
              return True
  return False
n = 1730
while not taxi(n):
  n = n + 1
print(n)
```

手順 taxi(n) は，正整数 n の 3 乗根を整数に切り上げたもの m を求めてから，$a \leqq b$ と $a < c \leqq d$ を満たす m 未満の正整数の組（a, b, c, d）であって $a^3 + b^3 = n$ かつ $c^3 + d^3 = n$ を満たすものがあるか調べている。これにより n が正整数の 3 乗の和として 2 通りに表されるかどうか判定できる。例えば taxi(1729) の結果は True となる。

これを使って，taxi(n) が True となるような最小の整数 $n \geqq 1730$ を探している。

問2

例えば次のプログラムによって求めることができる。

```
def c(k):
  if k % 2 == 0:
    return k // 2
  else:
    return k * 3 + 1
def collatz(n):
  if n == 1:
    return 0
  else:
    return collatz(c(n)) + 1
n = 1
while collatz(n) <= 200:
  n = n + 1
print(n)
```

手順 c(*k*) は，整数 $k \geqq 2$ に操作を 1 回施した結果を求める。例えば c(3) の結果は 10 となり，c(10) の結果は 5 となる。

手順 collatz(*n*) は，*n* にこの操作 c を何回施すと 1 になるか求める。例えば collatz(27) の結果は 111 となる。

これを使って，collatz(*n*) <= 200 が成り立たない最小の正整数 *n* を探している。

問3

例えば次のプログラムによって求めることができる。

```
def goldbach(n):
  prime = [True] * (n + 1)
  prime[0] = prime[1] = False
  for i in range(2, n + 1):
    if prime[i] ! = 0:
      for j in range(i, n // i + 1):
        prime[i * j] = False
  count = 0
  for a in range(1, n):
    if prime[a]:
      for b in range(a, n – a):
        if prime[b]:
          c = n – a – b
          if c >= b and prime[c]:
            count = count + 1
  return count
print(goldbach(2021))
```

手順 goldbach(n)は，正整数 n が 3 つの素数の和として何通りに表されるかを求める。例えば goldbach(47)の結果は 13 となる。このためにまずリスト prime を作り，n 以下の各非負整数 i について，i が素数であるか否かを prime[i]に記録している。その後，$a+b+c=n$ かつ $a \leqq b \leqq c$ を満たす正整数の組（a, b, c）であって prime[a], prime[b], prime[c]がすべて成り立つものの個数を数えている。

【ドレミ暗号】

問１

もとの文「おかしとぱんをたべたい」は 11 文字，ドレミ暗号に含まれる音は 11 個なので，ひらがな 1 文字に音が 1 つ対応すると考えられる。それを踏まえて暗号文に現れる音と数字の並びを見てみると，数字 音 数字 数字 の並びが 11 回繰り返されていることがわかる。したがって，ひらがな 1 文字を，1 つの音と 3 つの数字で表していると考えられる。

問２

問１の答えをもとに，暗号文とそれが表すひらがなの列の対応を調べると，以下のようになる。

0do04	0re00	0mi01	0fa04	0la20	1fa00	1mi04	0fa00	0la13	0fa00	0do01
お	か	し	と	ぱ	ん	を	た	べ	た	い

1 つのひらがなを構成する 数字 音 数字 数字 の組を，今，ａｂｃｄ と置き，ａｂｃｄ の特徴を調べると，以下であることがわかる。

- a：0 か 1。
- b：do から la のいずれかの音。
- c：0 か 1 か 2。
- d：0 から 4 のいずれか。

さらに，d の数字とひらがなの対応を詳しくみると，以下であることがわかる。

- 「か」「ぱ」「ん」「た」は 0
- 「し」「い」は 1
- 「べ」は 3
- 「お」「と」「を」は 4

同じ数字になっているひらがなは，同じ母音であることがわかり，d は 50 音図の「あ段」から「お段」に対応すると推測される。

すると，ａｂｃ で 50 音図の「あ行」，「か行」，…を表すと推測される。この推測の下にそれぞれのひらがなに相当する ａｂｃ を見てみると，以下であることがわかる。

- 「お」，「い」（「あ行」）は 0do0
- 「か」（「か行」）は 0re0
- 「し」（「さ行」）は 0mi0
- 「と」，「た」（「た行」）は 0fa0

「あ行」,「か行」, … が 0do0, 0re0, … となっているので, b のドレミの音の順番が 50 音図の行に対応しそうであることがわかるが, 音は 7 つしかないので, 50 音図の行を表すには足りない。そこで, b の音が同じで違う行のものがあるかを見ると,「し」と「を」がそれぞれ 0mi0, 1mi0 となっている。このことから,「あ行」から「ま行」までは a を 0 として do, re, mi, fa, so, la, si を順に並べ,「や行」以降は a を 1 としてまた do から並べていると推測される。この推測に基づくと,「や行」は 1do,「ら行」は 1re,「わ行」は 1mi,「ん」は 1fa のはずで,「を」が 1mi であることに一致する。c については, ほとんどの文字が 0 であるが,「ぱ」は 2,「べ」は 1 となっているので, 清音（「゛」や「゜」がつかないもの）は 0, 濁音（「゛」がつくもの）が 1, 半濁音（「゜」がつくもの）が 2 として, 清音, 濁音, 半濁音の区別をしていると考えられる。

以上から,「こ」は 0re04,「う」は 0do02,「し」は 0mi01,「え」は 0do03,「ん」は 1fa00 となり, 答えは 0re040do020mi010do031fa00 となる。

50 音図（清音）

	わ行	ら行	や行	ま行	は行	な行	た行	さ行	か行	あ行	
ん 1fa00	わ 1mi00	ら 1re00	や 1do00	ま 0si00	は 0la00	な 0so00	た 0fa00	さ 0mi00	か 0re00	あ 0do00	あ段
		り 1re01		み 0si01	ひ 0la01	に 0so01	ち 0fa01	し 0mi01	き 0re01	い 0do01	い段
		る 1re02	ゆ 1do02	む 0si02	ふ 0la02	ぬ 0so02	つ 0fa02	す 0mi02	く 0re02	う 0do02	う段
		れ 1re03		め 0si03	へ 0la03	ね 0so03	て 0fa03	せ 0mi03	け 0re03	え 0do03	え段
	を 1mi04	ろ 1re04	よ 1do04	も 0si04	ほ 0la04	の 0so04	と 0fa04	そ 0mi04	こ 0re04	お 0do04	お段

【文字列操作】

問

各行に複数ある単語を１行ずつに分け，同じ単語が隣り合うようにソートし，出現回数をカウントしてから，回数の少ない順に再度ソートすればよい。

【数列操作】

問１

手計算またはプログラムで $t_n=2t_{n-1}+n$ の関係を用いて $t_1, t_2, \cdots, t_5$ を順に求めればよい。

問２

プログラムで計算することもできるが，$2^{n+1}-(n+2)$ を求めて計算することもできる。$t_1, t_2, \cdots$ を順に求めたあと，各項から 2^{n+1} を引けば，この式は容易に見つけられる。