アルゴリズム入門　共通問題（2020年度Aセメスター試験）

[試験日時：2020年1月26日第4限（60分），答案用紙：1部，計算用紙：1枚，持ち込み一切不可]

内容に関する質問は受け付けない．問題の記述が曖昧な場合は，適切な仮定をおいて回答し，どのような仮定をおいたのか明記せよ．

なお，`ita.array.make1d(n)` は長さ `n` で全要素が 0 の配列を，`ita.array.make2d(m, n)` は `m` 行 `n` 列で全要素が 0 の 2 次元配列を，それぞれ返す関数である．問題中のいずれのプログラムについても，`ita` ライブラリは既に読み込まれているものとする．

問題 1

カントール集合は，線分の中央の 1/3 を取り除き 2 つの線分を得ることを繰り返して得られる．「中央を取り除く」操作を `n` 段階繰り返して得られる集合を生成する関数 `cantor(n)` について考える．図 1 の上は `cantor(2)` の，下は `cantor(3)` の実行結果を図示したもので，白い部分が残った線分に対応している．以下のプログラムでは，白い線分から中央の 1/3 を取り除くのではなく，最初に黒い線分を用意し，取り除かれない可能性のある部分はどこかを調べてゆき，最後に残った部分を 1 にして白く塗っている．以上を参考に以下の問に答えよ．

```
def cantor(n):
  a = ita.array.make1d(3 ** n)
  # 黒い線分を用意
  subcantor(a, n, 0)
  return a

def subcantor(a, level, x):
  if level == 0:
    a[x] = 1 # 残った部分を白く塗る
  else:
    subcantor(a, level - 1, x)
    subcantor(a, level - 1, x + 2 * 3 ** (level - 1))
```

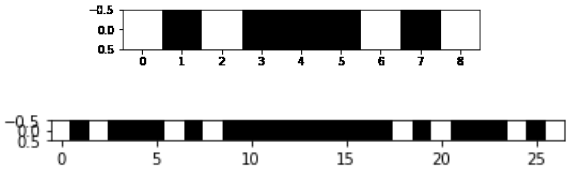

図 1: カントール集合

(1) シェルピンスキーカーペットは，正方形を 3×3 の 9 つの正方形に分解し中央を取り除くことを繰り返して得られる．以下の関数 `carpet(n)` はこの操作を `n` 段階繰り返した結果を生成する．図 2 は `carpet(3)` の実行結果の図示である．空欄 A ～ C を埋めてプログラムを完成させよ．

```
def carpet(n):
  a = ita.array.make2d(3 ** n, 3 ** n)
  # 真っ黒な正方形を用意
  subcarpet(a, n, 0, 0)
  return a

def subcarpet(a, level, y, x):
  if level == 0:
    a[y][x] = 1  # 残った部分を白く塗る
  else:
    subcarpet(a, level - 1, y, x)
    subcarpet(a, level - 1, y, [ A ])
    subcarpet(a, level - 1, y, x + 2 * 3 ** (level - 1))
    subcarpet(a, level - 1, y + 1 * 3 ** (level - 1), x)
    subcarpet(a, level - 1, y + 1 * 3 ** (level - 1), [ B ])
    subcarpet(a, level - 1, y + 2 * 3 ** (level - 1), [ C ])
    subcarpet(a, level - 1, y + 2 * 3 ** (level - 1), x + 1 * 3 ** (level - 1))
    subcarpet(a, level - 1, y + 2 * 3 ** (level - 1), x + 2 * 3 ** (level - 1))
```

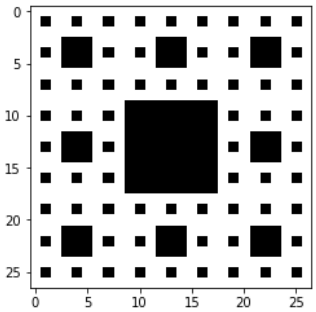

図 2: シェルピンスキーカーペット

(2) 以下の関数 pascal(n) はパスカル三角形を生成する．図 3 は pascal(3) の実行結果の図示である．空欄 D ～ H を埋めてプログラムを完成させよ．

```
def pascal(n):
  a = ita.array.make2d(3 ** n, 3 ** n)
  # 真っ黒な正方形を用意
  subpascal(a, n, 0, 0)
  return a

def subpascal(a, level, y, x):
  if level == 0:
    a[y][x] = 1  # 白く塗る
  else:
    subpascal(a, level - 1, y, x)
    subpascal(a, level - 1, y + 1 * 3 ** (level - 1), [D])
    subpascal(a, level - 1, y + 1 * 3 ** (level - 1), [E])
    subpascal(a, level - 1, y + 2 * 3 ** (level - 1), [F])
    subpascal(a, level - 1, y + 2 * 3 ** (level - 1), [G])
    subpascal(a, level - 1, y + 2 * 3 ** (level - 1), [H])
```

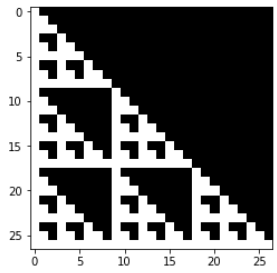

図 3: パスカル三角形

(3) 以下の関数 gasket(n) はシェルピンスキーガスケットを生成する．図 4 は gasket(5) の実行結果の図示である．空欄 I ・ J を埋めてプログラムを完成させよ．

```
def gasket(n):
  a = ita.array.make2d(2 ** n, 2 ** n)
  # 真っ黒な正方形を用意
  subgasket(a, n, 0, 0)
  return a

def subgasket(a, level, y, x):
  if level == 0:
    a[y][x] = 1  # 白く塗る
  else:
    subgasket(a, level - 1, y, x)
    subgasket(a, level - 1, y + 1 * 2 ** (level - 1), [I])
    subgasket(a, level - 1, y + 1 * 2 ** (level - 1), [J])
```

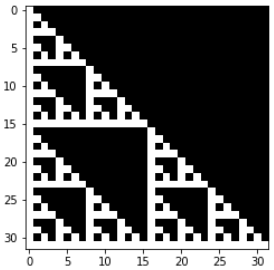

図 4: シェルピンスキーガスケット

問題 2

xy 平面上での放物運動のシミュレーションに関連して，以下の問に答えよ．なお，重力加速度は g = 9.8 とし，y 座標が減少する方向に働くものとする．

(1) 以下の関数 parabolic_m_step(y, x, vel_y, vel_x, stride) は，物体が位置 (x, y) から速度 (vel_x, vel_y) で運動するとき，stride 秒後の位置と速度を，stride が十分短い時間だと仮定して求め，配列 [y, x, vel_y, vel_x] として返す．さらに，関数 parabolic_motion_steps(y, x, vel_y, vel_x, stride, steps) は，関数 parabolic_m_step を steps 回適用した後の [y, x, vel_y, vel_x] を返す．空欄 A ～ F を埋めプログラムを完成させよ．

```
def parabolic_m_step(y, x, vel_y, vel_x, stride):
  g = 9.8
  x = x + vel_x * stride
  y = y + vel_y * stride
  vel_y = vel_y - [ A ]
  return [y, x, vel_y, vel_x]

def parabolic_motion_steps(y, x, vel_y, vel_x, stride, steps):
  cur = [ B ]
  for i in range(0, steps):
    cur = parabolic_m_step([ C ], [ D ], [ E ], [ F ], stride)
  return cur
```

(2) parabolic_motion_steps(10, 10, 2, -1, 0.1 ** 17, 100000) を実行したところ，結果は [10.0, 10.0, 2.0, -1] となり，vel_x だけでなく x や y や vel_y も変化しなかった．この原因として考えられることを 2 行程度で述べよ．

(3) 関数 parabolic_motion_steps によるシミュレーション結果は，stride が十分短いと仮定した差分化に基づいているため，正確ではない．計算結果の y 座標の誤差を stride と steps の式として表わせ．導出過程の概略も示すこと．丸め誤差の影響は無視してよい．なお，$T = \texttt{stride} \times \texttt{steps}$ 秒後の正確な y 座標は $\texttt{y} + \texttt{vel\_y} \times T - \frac{1}{2}\texttt{g}T^2$ である．

(4) このシミュレーションのプログラムを行列計算を用いることで改善したい．その準備としてまず，m 行 n 列の行列（2 次元配列）matrix と n 次元ベクトル（配列）vector の積を計算する関数 mMv(matrix, vector) を作成した．この関数は以下を満たす長さ m の配列 result を返す．

$$\texttt{result}[i] = \sum_{j=0}^{n-1} \texttt{matrix}[i][j] \times \texttt{vector}[j]$$

空欄 [G] ～ [I] を埋めプログラムを完成させよ．

```
def mMv(matrix, vector):
  result = ita.array.make1d(len(matrix))
  for i in [ G ]:
    for j in [ H ]:
      result[i] = [ I ]
  return result
```

(5) 関数 mMv を用いて関数 parabolic_motion_steps_matrix を作成した．この関数の計算結果が関数 parabolic_motion_steps と同じとなるよう，空欄 [J] ～ [L] を埋めよ．

```
def parabolic_motion_steps_matrix(y, x, vel_y, vel_x, stride, steps):
  g = 9.8
  vector = [ J ]
  matrix = [[1, 0, stride, 0, 0],
            [ K ],
            [0, 0, 1, 0, -stride],
            [0, 0, 0, 1, 0],
            [ L ]]
  for i in range(0, steps):
    vector = mMv(matrix, vector)
  return [vector[0], vector[1], vector[2], vector[3]]
```

(6) 行列 A に対し，A のべき乗を行列積を用いて $A^1 = A$ および $A^{n+1} = A^n \cdot A$ $(n \geq 1)$ と定義する．このとき，$A^{2n} = (A^2)^n$ および $A^{2n+1} = ((A^2)^n) \cdot A$ という関係がなり立つ．以下の関数 `mPn(matrix, n)` はこの関係を使い行列 `matrix` の `n` 乗 $(\texttt{n} \geq 1)$ を計算する．空欄 M ～ O を埋めプログラムを完成させよ．必要があれば，2 つの 2 次元配列を入力とし，両者の行列積に対応する 2 次元配列を返す関数 `mMm` を用いてよい．

```
def mPn(matrix, n):
  if n == 1:
    return matrix
  m = mPn( M , N )
  if n % 2 == 0:
    return m
  else:
    return O
```

(7) 以下の関数 `parabolic_motion_steps_matrix2` は，関数 `parabolic_motion_steps_matrix` と同じ結果を関数 `mPn` を用いて求める．空欄 J ～ L には関数 `parabolic_motion_steps_matrix` のものと同じものが入る．関数 `parabolic_motion_steps_matrix2` の `steps` に関する計算量のオーダーを示せ．導出過程の概略も示すこと．

```
def parabolic_motion_steps_matrix2(y, x, vel_y, vel_x, stride, steps):
  g = 9.8
  vector = J
  matrix = [[1, 0, stride, 0, 0],
            K ,
            [0, 0, 1, 0, -stride],
            [0, 0, 0, 1, 0],
            L ]
  matrix = mPn(matrix, steps)
  vector = mMv(matrix, vector)
  return [vector[0], vector[1], vector[2], vector[3]]
```