E　サンプルプログラム

E.1　リスト／セット／辞書のアクセス速度の比較

■ スライスに整数のインデックスを与える形のアクセス

「n 番目の要素」という形で要素にアクセスする場合の速度比較を行うプログラムが spdTest00.py である．このプログラムは長い（10^6 個の要素を持つ）リスト L，辞書 D を作成し，それらの n 番目の要素にランダムにアクセスするものである．またランダムなアクセスを，その要素の個数と同じ回数（10^6 回）リスト，辞書それぞれに対して行い，実行時間を計測する．

プログラム：spdTest00.py

```
# coding: utf-8
import time
import secrets
#########################################################
# 実行速度テスト（インデックスによる）                  #
#########################################################

#--- インデックスのランダムアクセス ---
def spdTestIdx( Data ):
    n = len( Data )
    t1 = time.time()
    for c in range(n):
        I = secrets.randbelow(n)
        Data[I] = I
    t2 = time.time()
    return( t2 - t1 )

def spdTestIdxAvr( Data, n ):
    tL = []
    for c in range(n):
        t = spdTestIdx( Data )
        print( c+1,'回目:\t', t, '秒' )
        tL.append( t )
    avr = sum( tL ) / n
    print( '平均:\t', avr, '秒' )
    return( avr )

#--- 実行時間テスト ---

N = 1000000                     # データサイズ
L = list( range(N) )            # リスト
D = { x:x for x in range(N) }   # 辞書

print( 'リストの場合のテスト' )
print( '--------------------' )
aL = spdTestIdxAvr( L, 3 )

tL = []
print( '\n辞書の場合のテスト' )
print( '--------------------' )
aD = spdTestIdxAvr( D, 3 )
print( 'リストの場合に対する速度比:\t', aL/aD, '倍' )
```

このプログラムの実行例を次に示す．

例. spdTest00.py の実行例

```
C:￥Users￥katsu>py spdTest00.py  Enter      ←コマンドからスクリプトを起動
リストの場合のテスト
--------------------
1 回目:  2.092395782470703 秒
2 回目:  2.1306302547454834 秒
3 回目:  2.0907864570617676 秒
平均:  2.104604164759318 秒

辞書の場合のテスト
--------------------
1 回目:  2.2947611808776855 秒
2 回目:  2.278236150741577 秒
3 回目:  2.258124589920044 秒
平均:  2.277040640513102 秒
リストの場合に対する速度比:  0.9242716740817908 倍
```

※ **実行環境：** Python 3.6.7, Intel Corei7-5500U 2.4GHz, 8GB RAM, Windows10 Pro

（評価）

リストの方が辞書に比べて若干早いことがわかる．

■ メンバシップ検査に要する時間

データ構造の中に特定の要素があるかどうかを調べるのに要する時間を調べるプログラムが spdTest01.py である．このプログラムでは 30,000 個の要素を持つデータ構造に対して要素の含有検査（メンバシップ検査）を行う．リスト L，セット S，辞書 D はそれぞれ 0〜29,999 の整数を要素として持ち，発生した整数の乱数がそのデータ構造に含まれるかどうかを検査する．メンバシップ検査は要素の個数と同じ回数繰り返して，その実行に要した時間を計測する．

プログラム：spdTest01.py

```
# coding: utf-8
import time
import secrets
#########################################################
# 実行速度テスト(1)                                     #
#########################################################

#--- メンバシップ検査の速度テスト ---
def spdTest( Data ):
    n = len( Data )
    t1 = time.time()
    for c in range(n):
        chk = secrets.randbelow(n) in Data
    t2 = time.time()
    return( t2 - t1 )

def spdTestAvr( Data, n ):
    tL = []
    for c in range(n):
        t = spdTest( Data )
        print( c+1,'回目:\t', t, '秒' )
        tL.append( t )
    avr = sum( tL ) / n
    print( '平均:\t', avr, '秒' )
    return( avr )

#--- 実行時間テスト ---
N = 30000                           # 要素の個数
L   = list( range(N) )              # リスト
S   = set( L )                      # セット
D   = { x:x for x in range(N) }     # 辞書
i = 3                               # 検査実行回数

```

```
34 print( 'リストの場合のテスト' )
35 print( '--------------------' )
36 aL = spdTestAvr( L, 3 )
37
38 print( '\nセットの場合のテスト' )
39 print( '--------------------' )
40 aS = spdTestAvr( S, 3 )
41 print( 'リストの場合に対する速度比:\t', aL/aS, '倍' )
42
43 tL = []
44 print( '\n辞書の場合のテスト' )
45 print( '--------------------' )
46 aD = spdTestAvr( D, 3 )
47 print( 'リストの場合に対する速度比:\t', aL/aD, '倍' )
```

このプログラムの実行例を次に示す.

例. spdTest01.py の実行例

```
C:￥Users￥katsu>py spdTest01.py  Enter      ←コマンドからスクリプトを起動
リストの場合のテスト
--------------------
1 回目:  7.3886847496032715 秒
2 回目:  6.19019889831543 秒
3 回目:  6.297661542892456 秒
平均:  6.625515063603719 秒

セットの場合のテスト
--------------------
1 回目:  0.05897831916809082 秒
2 回目:  0.06302452087402344 秒
3 回目:  0.06297516822814941 秒
平均:  0.06165933609008789 秒
リストの場合に対する速度比:  107.45355827256158 倍

辞書の場合のテスト
--------------------
1 回目:  0.060981035232543945 秒
2 回目:  0.06202244758605957 秒
3 回目:  0.06197810173034668 秒
平均:  0.0616605281829834 秒
リストの場合に対する速度比:  107.45148085565991 倍
```

※ **実行環境：** Python 3.6.7, Intel Corei7-5500U 2.4GHz, 8GB RAM, Windows10 Pro

（評価）

セットと辞書は共に同等の実行速度であり，リストに比べて 100 倍以上早い ことがわかる.

次に，spdTest01.py で調べた実行時間が，データの個数が増えるのに対してどのように伸びてゆくかを調べる．次のサンプルプログラム spdTest02.py で調べる.（グラフ描画に matplotlib を要する）

プログラム： spdTest02.py

```
1  # coding: utf-8
2  import time
3  import secrets
4  import matplotlib.pyplot as plt
5  ##########################################################
6  # 実行速度テスト(2)                                      #
7  ##########################################################
8
9  #--- メンバシップ検査の速度テスト ---
10 def spdTest( Data ):
11     n = len( Data )
```

```
12      t1 = time.time()
13      for c in range(n):
14          chk = secrets.randbelow(n) in Data
15      t2 = time.time()
16      return( t2 - t1 )
17
18  def spdTestAvr( Data, n ):
19      tL = []
20      for c in range(n):
21          t = spdTest( Data )
22          tL.append( t )
23      avr = sum( tL ) / n
24      return( avr )
25
26  #--- 実行時間テスト ---
27
28  # リストのテスト
29  N = 10000                               # 要素の個数
30  X = range(0,N,100)
31  A1 = [];    A2 = [];    A3 = []         # 実行時間のリスト
32  for n in X:
33      D = list( range(n) )
34      A1.append( spdTestAvr( D, 3 ) )
35      D = set( D )
36      A2.append( spdTestAvr( D, 3 ) )
37      D = { x:x for x in range(n) }
38      A3.append( spdTestAvr( D, 3 ) )
39
40  plt.plot(list(X),A1,label='List')
41  plt.plot(list(X),A2,label='Set')
42  plt.plot(list(X),A3,label='Dict')
43  plt.legend()
44  plt.title('test for List/Set/Dict')
45  plt.show()
```

このプログラムの実行結果の例を図 63 に示す.（実行環境は先と同じ）

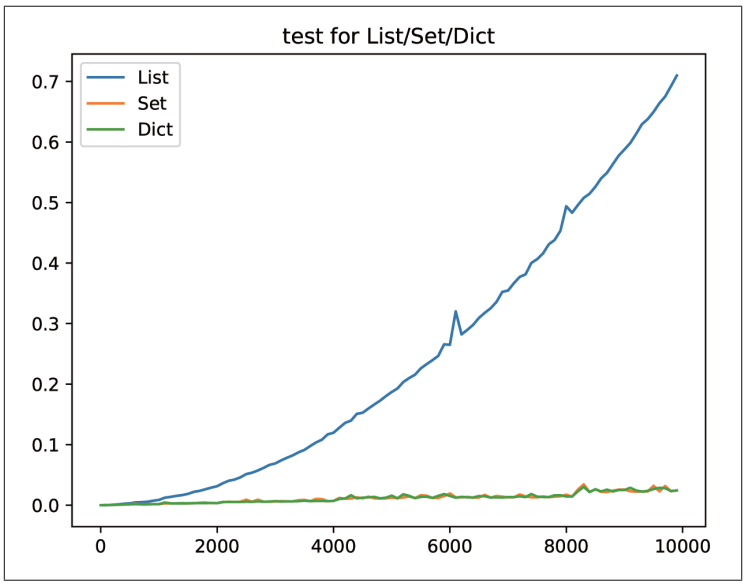

図 63: spdTest02_exe.py の実行結果

横軸はデータの個数，縦軸は実行時間

（評価）

データ個数の増加に対してセットと辞書では検査の実行時間の伸びが小さいのに対して，リストでは概ねデータ個数の 2 乗に比例する形で実行時間が大きくなる．リストに対する in 演算子による要素の探索では，線形探索アルゴリズムと同等の規模の計算時間であることがわかる．