```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 22 11:04:31 2022

@author: stephane
"""

from numpy.random import randint, permutation
from time import time

"""
Manipulation de dictionnaires
"""

# Exo 2

L = [i**2+1 for i in range(11)]
D = {i:i**2+1 for i in range(11)}
B = {j/10:(j/10)**2+1 for j in range(101)}

"""
>>> L[2], D[2], B[2]
(5, 5, 5.0)

>>> len(L)
11

>>> len(D.keys()), len(D)
11, 11

>>> len(B)
101

>>> B[0.4]
1.1600000000000001
# huhu !
"""

# Exo 3

# Tout d'abord : voila comment j'ai créé puis sauvé mon dictionnaire


"""
dico0 = {12: 'pif', 'douze': 'paf', '12': 42, 'plouf': 15}
for i in range(20, 500, 42):
    dico0[i/10] = 'pif'+str(i/10)
    dico0[10*i] = i % 4

def ecrire_dico(d: dict, name: str) -> None:
    f = open(name, 'w')
    for (k, v) in d.items():
        f.write('%s,%s,%s,%s\n' % (type(k), type(v), str(k), str(v)))
    f.close()
    pass

ecrire_dico(dico0, 'dico0.txt')
"""

def typer(x: str, t: str):
    if t == 'int':
```

```
        return int(x)
    elif t == 'float':
        return float(x)
    else:
        return x

def lecture_dico(fichier: str) -> dict:
    dico = {}
    for ligne in open(fichier):
        t1, t2, k0, v0 = ligne.strip().split(',')
        k, v = typer(k0, t1), typer(v0, t2)
        dico[k] = v
    return dico

dico1 = lecture_dico('dico0.txt')

"""
>>> dico0 == dico1
True
"""

"""
>>> dico1[12], dico1['12'], dico1['douze']
('pif', 42, 'paf')

>>> dico1['plouf']
15

>>> dico1['paf']
Traceback (most recent call last):

  File "<ipython-input-77-62196a735514>", line 1, in <module>
    dico1['paf']

KeyError: 'paf'

>>> len(dico1)
28
"""

i1, i2 = 0, 0
for (k,v) in dico1.items():
    if isinstance(k, int):
        i1 += 1
    if isinstance(v, int):
        i2 += 1
print('%i clés sont des entiers et %i valeurs sont des entiers' % (i1, i2))
"""
13 clés sont des entiers et 14 valeurs sont des entiers
"""


"""
Suite de Fibonacci
"""

# Exos 4, 5 et 6

def fibo1(n: int) -> int: # full-récursif
    assert n >= 0
    if n <= 1:
        return n
    return fibo1(n-1)+fibo1(n-2)

def fibo2(n: int) -> int: # Bottom-up
    assert n >= 0
    vals = [0] * (n+1)
    vals[1] = 1
    for i in range(2, n+1):
        vals[i] = vals[i-1]+vals[i-2]
    return vals[n]

def fibo3(n: int) -> int:  # Top-down mémoïzée
    assert n >= 0
    remember = {0 : 0, 1 : 1}
    def fibo_rem(k: int) -> int:
        if not(k in remember):
            remember[k] = fibo_rem(k-1)+fibo_rem(k-2)
        return remember[k]
    return fibo_rem(n)

"""
>>> fibo1(10), fibo2(10), fibo3(10)
(55, 55, 55)
"""

def chrono_fibo(n: int) -> list:
    t0 = time()
    _ = fibo1(n)
    t1 = time()
    _ = fibo2(n)
    t2 = time()
    _ = fibo3(n)
    t3 = time()
    return [t1-t0, t2-t1, t3-t2]

#for n in [10, 20, 30, 35, 40]:
#    print(n, chrono_fibo(n))

"""
10 [1.3589859008789062e-05, 3.0994415283203125e-06, 5.4836273193359375e-06]
20 [0.0013709068298339844, 3.0994415283203125e-06, 6.67572021484375e-06]
30 [0.16498970985412598, 7.62939453125e-06, 1.3828277587890625e-05]
35 [1.92362380027771, 9.059906005859375e-06, 1.52587890625e-05]
40 [21.89234471321106, 1.1205673217773438e-05, 2.0265579223632812e-05]
"""

#vals = [chrono_fibo(n)[0] for n in range(29, 36)]
#print([vals[i+1]/vals[i] for i in range(6)])
"""
[1.6413406480447938, 1.6246678971370871, 1.6164893815738977,
1.6262669768463052, 1.6286777447009309, 1.6303310843799739]

>>> (1+5**0.5)/2
1.618033988749895
"""



"""
Plus longue sous-suite croissante
"""

# Exos 7, 8 et 9
```

```
def plsscroi(L: list) -> int:
    phi = [1]
    for i in range(1, len(L)):
        # on regarde quels les j<i tels que L[j]<=L[i]
        avant = [phi[j] for j in range(i) if L[j] <= L[i]]
        if avant == []:
            phi.append(1)
        else:
            phi.append(1+max(avant))
    return max(phi)

"""
>>> plsscroi([10, 1, 5, 3, 4, 2, 3, 12, 1])
4
"""

def plsscroi_expl(L: list, debug = False) -> list:
    # Version feignasse : je stocke une sous-liste optimale
    phi = [(1, [L[0]])]
    for i in range(1, len(L)):
        avant = [phi[j] for j in range(i) if L[j] <= L[i]]
        if avant == []:
            phi.append((1, [L[i]]))
        else:
            maxi = max(c[0] for c in avant)
            for c in avant:
                 if c[0] == maxi: # c'est la longueur maximale
                     debut = c[1]  # la sous-suite correspondante
            phi.append((1+maxi, debut + [L[i]]))
    maxi = max(c[0] for c in phi)
    if debug:
        print(phi)
    for i in range(len(L)):
        if phi[i][0] == maxi:
            return phi[i][1]

"""
>>> plsscroi_expl([10, 1, 5, 3, 4, 2, 3, 12, 1])
[1, 2, 3, 12]

>>> plsscroi_expl([10, 1, 5, 3, 4, 2, 3, 12, 1], True)
[(1, [10]), (1, [1]), (2, [1, 5]), (2, [1, 3]), (3, [1, 3, 4]),
(2, [1, 2]), (3, [1, 2, 3]), (4, [1, 2, 3, 12]), (2, [1, 1])]
[1, 2, 3, 12]
"""

def esperance_longueur(n: int, Ne: int) -> float:
    return sum(plsscroi(permutation(list(range(n)))) for _ in range(Ne))/Ne

"""
>>> esperance_longueur(25, 10**3)
7.535

>>> esperance_longueur(100, 10**3)
16.739

>>> esperance_longueur(400, 10**3)
35.546

>>> esperance_longueur(1600, 10**3)
74.468
"""
```

```python
"""
Plus longue sous-suite commune
"""

# Exos 10, 11, 12 et 13

def randlist(N: int) -> list:
    return [randint(2*N) for _ in range(N)]
"""
>>> randlist(5)
[3, 6, 7, 6, 4]
"""

def plsscfins1(L1: list, L2: list, i1: int, i2: int) -> int:
    if i1 == -1 or i2 == -1:
        return 0
    elif L1[i1] == L2[i2]:
        return 1+plsscfins1(L1, L2, i1-1, i2-1)
    else:
        return max(plsscfins1(L1, L2, i1-1, i2), plsscfins1(L1, L2, i1, i2-1))

def plssc1(L1: list, L2: list) -> int:
    return plsscfins1(L1, L2, len(L1)-1, len(L2)-1)

"""
>>> plssc1([5, 3, 11, 18, 19, 11, 1, 7, 16, 14],
           [6, 16, 14, 5, 13, 12, 16, 18, 19, 7])
4
"""

def plssc2(L1: list, L2:list) -> int:
    vals = [[0] * len(L1) for _ in range(len(L2))]
    def calcul(i1: int, i2: int) -> int:
        if (i1, i2) == (0, 0):
            if L1[0] == L2[0]:
                return 1
            else:
                return 0
        if i1 == 0:
            if L1[0] == L2[i2]:
                return 1
            else:
                return vals[0][i2-1]
        if L1[i1] == L2[i2]:
            return 1 + vals[i1-1][i2-1]
        else:
            return max(vals[i1][i2-1], vals[i1-1][i2])
    for i2 in range(len(L2)):
        for i1 in range(len(L1)):
            vals[i1][i2] = calcul(i1, i2)
    return vals[-1][-1]

"""
>>> plssc2([5, 3, 11, 18, 19, 11, 1, 7, 16, 14],
           [6, 16, 14, 5, 13, 12, 16, 18, 19, 7])
4
"""

def plssc3(L1: list, L2: list) -> int:
    remember = {}
```

```python
    def plssc_rem(i1: int, i2: int) -> int:
        if not((i1, i2) in remember):
            if i1<0 or i2<0: # L'un des deux morceaux est vide
                remember[(i1, i2)] = 0
            else:
                if L1[i1] == L2[i2]:
                    remember[(i1, i2)] = 1 + plssc_rem(i1-1, i2-1)
                else:
                    remember[(i1, i2)] = max(plssc_rem(i1-1, i2),
                                             plssc_rem(i1, i2-1))
        return remember[(i1, i2)]
    return plssc_rem(len(L1)-1, len(L2)-1)



def chrono_plssc(L1: list, L2: list) -> list:
    t0 = time()
    if len(L1)+len(L2) <= 30:
        _ = plssc1(L1, L2)
    t1 = time()
    _ = plssc2(L1, L2)
    t2 = time()
    _ = plssc3(L1, L2)
    t3 = time()
    return [t1-t0, t2-t1, t3-t2]

#for n in range(10, 16):
#    print(n, chrono_plssc(randlist(n), randlist(n)))

"""
10 [0.028853178024291992, 3.24249267578125e-05, 6.079673767089844e-05]
11 [0.07154965400695801, 4.458427429199219e-05, 7.772445678710938e-05]
12 [0.6073739528656006, 5.030632019042969e-05, 9.751319885253906e-05]
13 [1.648728370666504, 5.793571472167969e-05, 0.000118255615234375]
14 [6.344505071640015, 6.604194641113281e-05, 0.0001397132873535156 2]
15 [25.344244718551636, 7.414817810058594e-05, 0.0001614093780517578]
"""
#for n in [100, 200, 400, 800]:
#    print(n, chrono_plssc(randlist(n), randlist(n))[1:])

"""
100 [0.0022749900817871094, 0.006298303604125976 6]
200 [0.00894474983215332, 0.0250701904296875]
400 [0.03813576698303223, 0.11228442192077637]
800 [0.16267752647399902, 0.5312995910644531]
"""

def plssc4(L1: list, L2: list) -> list:
    remember = {}
    suites = {}
    def plssc_rem(i1: int, i2: int) -> int:
        if not((i1, i2) in remember):
            if i1<0 or i2<0:
                remember[(i1, i2)] = 0
                suites[(i1, i2)] = []
            else:
                if L1[i1] == L2[i2]:
                    remember[(i1, i2)] = 1 + plssc_rem(i1-1, i2-1)
                    suites[(i1, i2)] = suites[(i1-1, i2-1)] + [L1[i1]]
                else:
                    v1, v2 = plssc_rem(i1-1, i2), plssc_rem(i1, i2-1)
                    if v1 <= v2:
                        remember[(i1, i2)] = v2
                        suites[(i1, i2)] = suites[(i1, i2-1)]
                    else:
                        remember[(i1, i2)] = v1
                        suites[(i1, i2)] = suites[(i1-1, i2)]
        return remember[(i1, i2)]
    foo = plssc_rem(len(L1)-1, len(L2)-1) # pour lancer/forcer le calcul !
    return foo, suites[(len(L1)-1, len(L2)-1)]


L1, L2 = randlist(10), randlist(10)
print(L1, L2, plssc1(L1, L2), plssc2(L1, L2), plssc3(L1, L2), plssc4(L1, L2))
"""
[19, 1, 6, 1, 14, 12, 15, 1, 9, 17]
[15, 14, 10, 9, 15, 11, 15, 15, 13, 4]
2 2 2 (2, [15, 9])
"""



"""
Problème du sac à dos
"""

# Exos 14 et 15

def knapsack(L: list, Pmax: int) -> int:
    # Il est important (et problématique !) que Pmax soit un entier
    remember = {}
    def knap_rem(i: int, P: int) -> int:
        # i est ici l'indice du dernier objet qu'on s'autorise à prendre
        if i < 0:
            return 0
        if not((i, P) in remember):
            if L[i][1] > P: # l'objet est trop lourd
                remember[(i, P)] = knap_rem(i-1, P)
            else:
                remember[(i, P)] = max(L[i][0] + knap_rem(i-1, P-L[i][1]),
                                       knap_rem(i-1, P))
        return remember[(i, P)]
    return knap_rem(len(L)-1, Pmax)


"""
>>> knapsack([(100,10), (60,5), (60,5)], 10)
120

>>> knapsack([(100,10), (60,9), (60,9)], 10)
100
"""

def knapsack_contenu(L: list, Pmax: list) -> list:
    remember = {} # remember[(i, P)] = (valeur max , [sc à dos correspondant])
    def knap_rem(i, P): # retourne le couple (valeur, contenu)
        if i < 0:
            return 0, []
        if not((i, P) in remember):
            if L[i][1] > P:
                remember[(i, P)] = knap_rem(i-1, P)

            else:
                foo, bar = knap_rem(i-1, P-L[i][1]), knap_rem(i-1, P)
                if L[i][0] + foo[0] > bar[0]: # je prends l'objet i
```

```
                    remember[(i, P)] = L[i][0] + foo[0], foo[1] + [L[i]]
                else:
                    remember[(i, P)] = bar
        return remember[(i, P)]
    return knap_rem(len(L)-1, Pmax)


"""
>>> knapsack_contenu([(100,10), (60,5), (60,5)], 10)
(120, [(60, 5), (60, 5)])

>>> knapsack_contenu([(100,10), (60,9), (60,9)], 10)
(100, [(100, 10)])
"""
```