```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 21 08:26:58 2022

@author: stephane
"""

from math import inf
from numpy.random import randint

"""
PARTITIONS D'ENTIERS
"""
# exos 2 et 3; Project Euler 76

def partitions1_maxi(n: int, k: int) -> int:  # full récursif
    if k == 1:
        return 1
    elif n < k:
        return partitions1_maxi(n, k-1)
    elif n == k:
        return 1 + partitions1_maxi(n, k-1)
    else:
        return partitions1_maxi(n, k-1) + partitions1_maxi(n-k, k)

def partitions1(n: int) -> int:
    return partitions1_maxi(n, n) - 1
"""
>>> partitions1(100)
190569291
# mais ça prend des plombes (de l'ordre de la minute !)
"""

def partitions2(n0: int) -> int:
    remember = {}
    def part_rec(n: int, k: int) -> int:
        if k == 1:
            return 1
        elif not ((n, k) in remember):
            if n < k:
                remember[(n, k)] = part_rec(n, k-1)
            elif n == k:
                remember[(n, k)] = part_rec(n, k-1) + 1
            else:
                remember[(n, k)] = part_rec(n, k-1) + part_rec(n-k, k)
        return remember[(n, k)]
    return part_rec(n0, n0) -1
"""
>>> partitions2(100)
190569291
# une fraction de seconde !
"""


"""
SOMMES SUR UN CHEMIN
"""

# Exos 4, 5, 6, 7 et 8; Project Euler 18 et 67

exple1 = [[3], [7, 4], [2, 4, 6], [8, 5, 9, 3]]
```



```
exple2 = [[int(x) for x in ligne.split(',')] for ligne in open('pe18.txt')]

exple3 = [[int(x) for x in ligne.split(' ')] for ligne in open('pe67.txt')]

def d_m_depuis(T:list, i: int, j: int) -> int: # full récursif
    if i == len(T):
        return 0
    else:
        return T[i][j] + max(d_m_depuis(T, i+1, j), d_m_depuis(T, i+1, j+1))

def descente_max1(T: list) -> int:
    return d_m_depuis(T, 0, 0)

"""
>>> descente_max1(exple1)
23

>>> descente_max1(exple1[:3])
14

>>> descente_max1(exple2)
1074
"""

# impossible à traiter en full récursif !

def descente_max2(T: list) -> int:
    remember = {}
    def d_m_dep(i: int, j: int) -> int:
        if i == len(T):
            return 0
        if not ((i, j) in remember):
            remember[(i, j)] = T[i][j] + max(d_m_dep(i+1, j), d_m_dep(i+1, j+1))
        return remember[(i, j)]
    return d_m_dep(0, 0)

"""
>>> descente_max1(exple3[:20])
1407
# Long !

>>> descente_max2(exple3[:20])
1407
# instantanné

>>> descente_max2(exple3)
7273
# idem
"""

# Au tableur :
#  - importer
#  - Placer à droite ou en bas une copie de la case en bas à gauche "=A20"
#  - faire glisser à droite
#  - écrire une formule pour la case dessus : "=A19+max(...;...)
#  - faire glisser à droite, puis en haut : c'est gagné !

def descente_max3(T: list) -> int:
    remember = {}
    def d_m_dep(i: int, j: int) -> int:
        if i == len(T):
            return [0, None]
```

```
        if not ((i, j) in remember):
            (g, sg) = d_m_dep(i+1, j)
            (d, sd) = d_m_dep(i+1, j+1)
            if g >= d:
                remember[(i, j)] = (T[i][j] + g, j)
            else:
                remember[(i, j)] = (T[i][j] + d, j+1)
        return remember[(i, j)]
    _ = d_m_dep(0, 0)
    chemin, suivant = [], 0
    for h in range(len(T)):
        chemin.append(T[h][suivant])
        suivant = d_m_dep(h, suivant)[1]
    return sum(chemin), chemin

"""
>>> descente_max3(exple1)
(23, [3, 7, 4, 9])

>>> descente_max3(exple2)
(1074, [75, 64, 82, 87, 82, 75, 73, 28, 83, 32, 91, 78, 58, 73, 93])
"""

"""
PRODUIT DE MATRICES
"""

# Exos 9, 10, 11 et 12

matrices1 = [[10, 1], [1, 10], [10, 1]]

def produit_optimal(T: list) -> int:  # T est la liste des (hauteur, largeur)
    rem = {}
    def opt_entre(i: int, j: int) -> int:
        if i == j:
            return 0
        if not ((i, j) in rem):
            rem[(i, j)] = min(opt_entre(i, k) + opt_entre(k+1, j)
                              +T[i][0]*T[k][1]*T[j][1] for k in range(i, j))
        return rem[(i, j)]
    return opt_entre(0, len(T)-1)

"""
>>> produit_optimal(matrices1)
20
"""

def produit_optimal_bis(T: list) -> (int, str):
    rem = {}
    def opt_entre(i: int, j: int) -> int:
        if i == j:
            return 0, None
        if not ((i, j) in rem):
            maxi = inf
            for k in range(i, j):
                nv = opt_entre(i, k)[0] + opt_entre(k+1, j)[0] +\
                                T[i][0]*T[k][1]*T[j][1]
                if maxi > nv:
                    maxi = nv
                    par = k
                rem[(i, j)] = maxi, par
        return rem[(i, j)]
```



```
    _ = opt_entre(0, len(T)-1)
    def chemin(i, j):
        if i == j:
            return str(i)
        k = opt_entre(i, j)[1]
        return "(%s)(%s)" %  (chemin(i, k), chemin(k+1, j))
    return opt_entre(0, len(T)-1)[0], chemin(0, len(T)-1)
"""
>>> produit_optimal_bis(matrices1)
(20, '(0)((1)(2))')
"""

def ecrire(mats: list, nom: str) -> ():
    out = open(nom, 'w')
    for ligne in mats:
        out.write('%i,%i\n' % (ligne[0], ligne[1]))
    out.close()

ecrire(matrices1, 'matrices1.txt')

def lire(nom: str) -> list:
    return [list(map(int, L.strip().split(','))) for L in open(nom, 'r')]

def rdmat(Nb: int) -> list:
    L = []
    n = randint(1, 101)
    for _ in range(Nb):
        p = randint(1, 101)
        L.append([n, p])
        n = p
    return L
"""
>>> rdmat(5)
[[69, 94], [94, 7], [7, 13], [13, 9], [9, 78]]

>>> ecrire(rdmat(100), 'matrices2.txt')

>>> lire('matrices2.txt')
[[19, 47], [47, 76],
 ...
 [64, 37], [37, 84]]

>>> produit_optimal_bis(lire('matrices2.txt'))
(238618,
 '((0)((1)((2)((3)((4)((5)  ...   (96))(97))(98))(99))')
"""

def cout_gd(L):
    if len(L) == 1:
        return 0
    else:
        (a, b), (_, c) = L[0], L[1]
        return a*b*c + cout_gd([(a,c)] + L[2:])  #Bravo...
"""
>>> cout_gd(matrices1)
200

>>> cout_gd(lire('matrices2.txt'))
4488598
"""
```