I. Distance de Levenshtein

```
def levenshtein(s,t):
        n,p = len(s), len(t)
        memo = [[None]*(p+1) for i in range(n+1)]
        for i in range(n+1) :
                memo[i][0] = i
        for j in range(p+1) :
                memo[0][j] = j
        def aux(i,j):
                if memo[i][j] == None and s[i-1] == t[j-1] :
                        aux(i-1,j-1)
                        memo[i][j] = memo[i-1][j-1]
                elif memo[i][j] == None :
                        aux(i-1,j-1)
                        aux(i-1,j)
                        aux(i,j-1)
                        memo[i][j] = min(memo[i-1][j-1],memo[i-1][j],memo[i][j-1])+1
                return None
        aux(n,p)
        return memo[n][p]

s = "exos"
t = "veto"

def reconstruction(memo,s,t,i,j,accu):
        if i == 0 and j == 0 :
                print("terminé : {}".format(accu))
                return None
        elif i == 0 :
                print("{} -> insertion du {}".format(accu,t[j-1]))
                reconstruction(memo,s,t,i,j-1,t[j-1]+accu)
        elif j == 0 :
                print("{} -> suppression du {}".format(accu,s[i-1]))
                reconstruction(memo,s,t,i-1,j,accu[:i-1]+accu[i:])
        elif memo[i][j]-1 == memo[i-1][j] :
                print("{} -> suppression du {}".format(accu,s[i-1]))
                reconstruction(memo,s,t,i-1,j,accu[:i-1]+accu[i:])
        elif memo[i][j]-1 == memo[i][j-1]:
                print("{} -> insertion du {}".format(accu,t[j-1]))
                reconstruction(memo,s,t,i,j-1,accu[:i]+t[j-1]+accu[i:])
        elif memo[i][j]-1 == memo[i-1][j-1]:
                print("{} -> remplacement de {} par {}".format(accu,s[i-1],t[j-1]))
                reconstruction(memo,s,t,i-1,j-1,accu[:i-1]+t[j-1]+accu[i:])
```

```
        else :
                reconstruction(memo,s,t,i-1,j-1,accu)

def levenshtein2(s,t):
        n,p = len(s), len(t)
        memo = [[None]*(p+1) for i in range(n+1)]
        for i in range(n+1) :
                memo[i][0] = i
        for j in range(p+1) :
                memo[0][j] = j
        def aux(i,j):
                if memo[i][j] == None and s[i-1] == t[j-1] :
                        aux(i-1,j-1)
                        memo[i][j] = memo[i-1][j-1]
                elif memo[i][j] == None :
                        aux(i-1,j-1)
                        aux(i-1,j)
                        aux(i,j-1)
                        memo[i][j] = min(memo[i-1][j-1],memo[i-1][j],memo[i][j-1])+1
                return None
        aux(n,p)
        reconstruction(memo,s,t,n,p,s)
        return memo[n][p]

def levenshtein3(s,t):
        n, p = len(s), len(t)
        L = [[0]*(p+1) for i in range(n+1)]
        for i in range(n+1):
                L[i][0] = i
        for j in range(p+1):
                L[0][j] = j
        for i in range(1,n+1):
                for j in range(1,p+1):
                        if s[i-1] == t[j-1]:
                                L[i][j] = L[i-1][j-1]
                        else :
                                L[i][j] = 1 + min(L[i-1][j-1],L[i-1][j],L[i][j-1])
        for ligne in L :
                print(" ".join([str(x) for x in ligne]))
        return L
s1 = "extraordinaire"
t1 = "excalibur"
```

II. Partition équilibrée

```
import random

def partition_equilibree(T):
        """Renvoie la somme d'un tableau"""
        S = sum(T)
        n = len(T)
        L = [False]*(S+1)
        prov = [False]*(S+1)
        L[0] = True
        prov[0] = []
        for i in range(n):
                for s in range(S,-1,-1):
                        if L[s] :
                                L[s+T[i]] = True
                                prov[s+T[i]] = prov[s] + [T[i]]
        for i in range((S+1)//2,S+1):
                if L[i]:
                        return i, prov[i]

T = [random.randrange(10) for i in range(10)]
```

III. Ordonnancement de tâches pondérées

```
def ordonnancement(d,f,p):
        n = len(d)
        S = [0]*n
        S[0] = p[0]
        for i in range(1,n):
                j = i
                while f[j] > d[i] and j >= 0 :
                        j = j-1
                if j == -1 :
                        S[i] = max(S[i-1],p[i])
                else :
                        S[i] = max(S[i-1], S[j] + p[i])
        return S[n-1]

d = [1,3,3,5,8,9]
f = [4,5,6,6.5,10,11]
p = [5,3,4,1,4,7]
```

```
T = [ (d[i],f[i],p[i]) for i in range(len(d))]

def ordonnancement2(T):
        d = [t[0] for t in T]
        f = [t[1] for t in T]
        p = [t[2] for t in T]
        n = len(T)
        # astuce : tâche vide en dernier
        S = [0]*(n+1)
        taches = [[] for i in range(n)]
        S[0] = p[0]
        taches[0] = [T[0]]
        for i in range(1,n):
                j = i
                while f[j] > d[i] and j >= 0 :
                        j = j-1
                s = S[j]
                t = taches[j].copy()
                if s+p[i] > S[i-1]:
                        S[i] = s + p[i]
                        taches[i] = t + [T[i]]
                else :
                        S[i] = S[i-1]
                        taches[i] = taches[i-1].copy()
        return S[n-1], taches[n-1]
```

IV. Plus longue sous-suite commune

```
def longueur_ss(s,t):
        n,p = len(s), len(t)
        memo = [[None]*(p+1) for i in range(n+1)]
        for i in range(n+1):
                memo[i][0] = 0
        for j in range(p+1) :
                memo[0][j] = 0
        def aux(i,j):
                if memo[i][j] == None :
                        if s[i-1] == t[j-1] :
                                aux(i-1,j-1)
                                memo[i][j] = memo[i-1][j-1] + 1
                        else :
                                aux(i-1,j)
                                aux(i,j-1)
```

```
                                memo[i][j] = max(memo[i-1][j],memo[i][j-1])
        aux(n,p)
        return memo[n][p]

s = "sesame ouvre toi"
t = "par la barbe de merlin"

def reconstruction(memo,s,t,i,j):
        if i == 0 or j == 0 :
                return ""
        elif memo[i][j] == memo[i][j-1] :
                return reconstruction(memo,s,t,i,j-1)
        elif memo[i][j] == memo[i-1][j]:
                return reconstruction(memo,s,t,i-1,j)
        else:
                return reconstruction(memo,s,t,i-1,j-1) + s[i-1]

def plus_longue_ss(s,t):
        n,p = len(s), len(t)
        memo = [[None]*(p+1) for i in range(n+1)]
        for i in range(n+1):
                memo[i][0] = 0
        for j in range(p+1) :
                memo[0][j] = 0
        def aux(i,j):
                if memo[i][j] == None :
                        if s[i-1] == t[j-1] :
                                aux(i-1,j-1)
                                memo[i][j] = memo[i-1][j-1] + 1
                        else :
                                aux(i-1,j)
                                aux(i,j-1)
                                memo[i][j] = max(memo[i-1][j],memo[i][j-1])
        aux(n,p)
        return memo[n][p], reconstruction(memo,s,t,n,p)
```

V. Floyd - Warshall

```
from copy import deepcopy
from math import inf

M = [[inf,1,inf,inf,inf,inf],
         [inf,inf,-1,inf,2,inf],
```

```
         [inf,9,inf,2,inf,inf],
         [inf,inf,inf,inf,2,inf],
         [1,inf,inf,inf,inf,inf],
         [inf,inf,1,inf,7,1]]

def floyd_warshall(M):
        n = len(M)
        pM = [0]*(n+1)
        pM[0] = deepcopy(M)
        for k in range(1,n+1):
                pM[k] = deepcopy(M)
                for i in range(n):
                        for j in range(n):
                                pM[k][i][j] = \
                                min(pM[k-1][i][j],pM[k-1][i][k-1] \
                                    +pM[k-1][k-1][j])
        return pM[n]

def derniers_init(M) :
        N = deepcopy(M)
        n = len(M)
        for i in range(n):
                for j in range(n):
                        if N[i][j] != inf :
                                N[i][j] = i
        return N

def chemin(N,i,j):
        C = deepcopy(N)
        if N[i][j] == inf :
                return []
        elif i == j :
                return [i]
        else :
                d = N[i][j]
                c = chemin(N,i,d)
                c.append(j)
                return c

def tous_les_chemins(N):
        C = deepcopy(N)
        n = len(C)
        for i in range(n):
                for j in range(n):
```

```
                        C[i][j] = chemin(N,i,j)
        return C

def floyd_warshall2(M):
        n = len(M)
        pM = [0]*(n+1)
        pM[0] = deepcopy(M)
        N = derniers_init(M)
        for k in range(1,n+1):
                pM[k] = deepcopy(pM[k-1])
                for i in range(n):
                        for j in range(n):
                                if pM[k-1][i][j] > pM[k-1][i][k-1]+pM[k-1][k-1][j] :
                                        pM[k][i][j] = pM[k-1][i][k-1]+pM[k-1][k-1][j]
                                        N[i][j] = N[k-1][j]
        return pM[n],tous_les_chemins(N)
```