

## TP3 : Algorithmes heuristiques pour le voyageur de commerce

Dans ce TP nous allons étudier en pratique le célèbre problème du *voyageur de commerce* (en anglais : *Travelling Salesman Problem*, généralement abrégé "TSP"). On a un ensemble de villes avec les distances qui les séparent. On veut trouver une tournée pour parcourir toutes les villes et revenir au point de départ, en minimisant la distance totale parcourue. Par exemple, avec les villes suivantes :

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon	Marseille
Clermont-Ferrand	0	376	377	167	475
Bordeaux	376	0	244	556	646
Toulouse	377	244	0	538	404
Lyon	167	556	538	0	314
Marseille	475	646	404	314	0

On peut faire par exemple la tournée Clermont → Toulouse → Bordeaux → Marseille → Lyon → Clermont pour  $377 + 244 + 646 + 314 + 167 = 1748\text{km}$ .

On va coder dans ce TP des algorithmes heuristiques pour calculer une tournée aussi courte que possible. La tournée sera représentée par une liste d'entiers (chacune des  $n$  villes étant représentée par un entier entre 0 et  $n - 1$ ). Les distances seront stockées dans une matrice (liste de listes)  $M$  où  $M[i][j]$  est la distance entre les villes  $n^{\circ}i$  et  $n^{\circ}j$ .

*L'utilisation de Python et Sagemath est optionnelle mais recommandée : il est proposé ci-dessous, une fonction écrite pour Sagemath qui affiche graphiquement une tournée. Vous pouvez utiliser un autre langage que Python si vous préférez, mais dans ce cas vous ne pourrez pas utiliser la fonction d'affichage proposée, et il n'est pas garanti que vos chargés de TP puissent vous aider au niveau du code.*

Exemple de données avec 5 villes (voir en bas du document pour des jeux de données plus importants) :

```
distances_5villes=[[0,376,377,167,475],[376,0,244,556,646],[377,244,0,538,404],  
[167,556,538,0,314],[475,646,404,314,0]]  
noms_5villes={0:'Clermont', 1:'Bordeaux', 2:'Toulouse', 3:'Lyon', 4:'Marseille'}  
coordonnees_5villes={0:(50,47), 1:(24,40), 2:(37,28), 3:(62,46), 4:(65,23)}
```

La fonction suivante permettra d'afficher une tournée dans Sagemath, étant donnée la tournée, les coordonnées des villes en 2D, et les noms des villes.

```
def afficher_tournee(T,coordonnees_villes,noms_villes,taille_texte):  
    L=[]  
    for i in T:  
        L.append(coordonnees_villes[i])  
    p=plot(line(L))  
    for i in T:  
        p += plot(text(noms_villes[i],coordonnees_villes[i],  
                        bounding_box={'boxstyle':'round', 'fc':'w'},  
                        fontsize=taille_texte))  
  
    p.show(axes=False)  
  
T=[0,3,1,2,4,0]  
afficher_tournee(T,coordonnees_5villes,noms_5villes,10) #test
```

**Exercice 1** (Algorithme glouton).

On peut générer une tournée de façon gloutonne, de la façon suivante :

1. Choisir une première ville  $i$  (par exemple  $i = 0$ , ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et  $n - 1$ )
2. initialiser la tournée :  $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans  $T$ ) :
  - choisir la ville  $j$  non encore visitée qui minimise la distance à la dernière ville visitée
  - ajouter  $j$  à  $T$
4. ajouter la ville de départ  $i$  à la fin de  $T$  pour clôturer la tournée
5. renvoyer  $T$

Coder une fonction `tournee_gloutonne(M)` qui prend en paramètre la matrice des distances  $M$  et calcule, puis renvoie, une tournée avec l'algorithme. (Par exemple, on testera la fonction en lançant

```
T=tournee_gloutonne(distances_5villes)
afficher_tournee(T,coordonnees_5villes,noms_5villes,10)
```

Le nombre de villes pourra être obtenu avec `len(M)`. La distance entre la ville  $i$  et la ville  $j$  est obtenue par `M[i][j]` ou bien `M[j][i]`.

## Exercice 2 (Descente de gradient).

On va utiliser maintenant la *descente de gradient*, algorithme méta-heuristique vu en cours.

On définit d'abord une opération de transformation d'une solution (ici, une tournée) en une autre, proche. Pour une solution  $T$ , soit  $N(T)$  l'ensemble des solutions "voisines", c'est-à-dire, qui peuvent être obtenues via une seule opération de transformation. Le principe est le suivant :

1. On génère une première solution  $T_0$   
 $T \leftarrow T_0$  # $T$  : solution courante
2. Tant que  $N(T)$  contient une solution meilleure que  $T$  :
  - choisir une nouvelle solution  $T'$  dans  $N(T)$
  - $T \leftarrow T'$
3. Retourner  $T$

Pour le voyageur de commerce, l'opération de transformation peut être définie de la façon suivante : étant donnée une tournée  $T$ , on prend deux villes et on échange leurs positions dans  $T$  pour obtenir une nouvelle tournée  $T'$ .

Coder l'algorithme de descente de gradient `DescenteGradient(M)` qui prend une matrice  $M$  des distances en entrée et renvoie une tournée. Vous pourrez d'abord :

- Coder une fonction `eval(T,M)` qui prend une solution  $T$  et la matrice de distances  $M$  et renvoie la longueur de la tournée qui correspond à  $T$ .
- Coder une fonction `echange(T,i,j)` qui prend une solution  $T$  et deux indices de villes  $i$  et  $j$ , et renvoie la tournée semblable à  $T$  mais où les villes en position  $i$  et  $j$  sont inversées.
- Coder une fonction `voisinage(T)` qui prend une solution  $T$  et renvoie la liste des solutions "voisines", en utilisant `echange(T,i,j)`.

Pour la solution initiale  $T_0$ , on pourra prendre au choix :

- une tournée calculée avec l'algorithme glouton de la question précédente.
- une tournée aléatoire (pour cela coder une fonction `tournee_aleatoire(M)` qui renvoie une tournée aléatoire).
- pour les paresseux ou les pressés, la tournée  $[0, \dots, n-1, 0]$  où  $n$  est le nombre de villes.

Testez l'algorithme sur les données présentées en fin du document, et essayez-le à partir de différentes tournées de départ.

### Exercice 3 (Algorithme génétique).

On rappelle le principe général des algorithmes génétiques :

- Population initiale : un ensemble de solutions (aléatoires)
- On réitère  $x$  fois ( $x$  peut être prédéfini, ou bien, on s'arrête lorsqu'on n'améliore pas la meilleure solution) :
  - On sélectionne les  $p$  solutions les meilleures de la population courante (cela représente par exemple les meilleurs 20%, ou 50%, etc, au choix) à garder dans la population suivante (pour éviter de perdre les bonnes solutions)
  - On "croise" un certain nombre de paires aléatoires de solutions à mettre dans la population suivante. Chaque paire pourra avoir un ou plusieurs "enfants".
  - Des "mutations" apparaissent aléatoirement parmi les nouvelles solutions
  - On garde les  $p$  meilleures solutions obtenues pour former la prochaine population

Chaque population successive (appelée génération) pourra être représentée par une liste de tournées.

Dans le cas du problème du voyageur de commerce, le "croisement" de deux solutions peut être fait de la façon suivante. Pour deux "parents"  $P = [p_1, \dots, p_n]$  et  $M = [m_1, \dots, m_n]$ , on choisit un nombre  $i$  entre 1 et  $n-1$ . Un enfant pourra être construit en prenant le début  $p_1, \dots, p_i$  de la tournée "père" puis, en considérant le reste des villes dans l'ordre de la tournée de la mère. Par exemple, avec les tournées  $P = [1, 3, 4, 0, 2, 5]$  et  $M = [0, 2, 5, 3, 1, 4]$ , avec  $i = 2$ , on obtiendra l'enfant  $[1, 3, 0, 2, 5, 4]$ . On peut avec une méthode similaire obtenir les enfants  $[3, 1, 4, 0, 2, 5]$  (où au contraire on a gardé la deuxième partie de la tournée paternelle et complété le début avec l'ordre de visite maternel), et  $[0, 2, 1, 3, 4, 5]$  (où le rôle du père et de la mère sont inversés). Il y a donc  $2(n-1)$  enfants possibles avec cette méthode.

Une "mutation" peut être faite en échangeant deux villes prises au hasard (mais pas la ville de départ) dans la tournée. On pourra fixer une probabilité de mutation (en principe, assez faible, par exemple 1 pour cent) des nouvelles solutions obtenues.

Les paramètres : nombre  $x$  de générations, nombre de paires croisées à chaque génération, taille  $p$  de la population, probabilité de mutation, devront être fixés en faisant des tests pour trouver de bonnes valeurs.

Coder l'algorithme génétique `AlgoGenetique(M)` qui prend une matrice  $M$  des distances en entrée et renvoie une tournée. On peut le découper selon les étapes suivantes :

1. Pour le début de l'algorithme, coder une fonction `tournee_aleatoire(M)` qui renvoie une tournée au hasard (c'est-à-dire une liste qui contient tous les nombres entre 0 et  $\text{len}(M)-1$ ), dans un ordre aléatoire. La première population sera créée par  $p$  appels à cette fonction.
2. Coder une fonction `selection_naturelle(S,p,M)` qui, étant donnée une liste  $S$  de tournées, un entier  $p$  et la matrice  $M$  des distances, renvoie une liste contenant les  $p$  meilleures tournées. Elle devra faire appel à la fonction `eval(T,M)` codée dans un des exercices précédents. (*Pour cela on pourra faire par exemple un tri par sélection : trouver d'abord l'indice dans  $S$  de la meilleure tournée, puis celui de la 2e meilleure, etc.*)
3. Coder une fonction `croisement(T1,T2)` qui renvoie la liste des tournées "filles" de  $T1$  et  $T2$ .
4. Coder une fonction `mutation(T,x)` qui génère une mutation aléatoire dans une tournée (et renvoie la tournée mutée) avec une certaine probabilité  $x$ , avec  $0 < x < 1$ . Utiliser la fonction `echange(T,i,j)` codée dans un exercice précédent pour faire la mutation. Pour cela, `float(randrange(0,10000)/10000)` permet d'obtenir un flottant aléatoire entre 0 et 1 parmi 10000 valeurs possibles.
5. Assembler toutes les fonctions dans l'algorithme génétique `AlgoGenetique(M)` qui renvoie la meilleure tournée obtenue par l'algorithme génétique décrit ci-dessus.

#### Exercice 4 (Recuit simulé).

On va revenir sur la descente de gradient. Cette méthode peut être affinée avec la méthode du recuit simulé vue en cours. Dans cette modification, on a une certaine probabilité de prendre une "mauvaise" solution. Cette probabilité baisse au fil du temps (ceci est modélisé par une valeur de "température" qui baisse à chaque itération).

1. On génère une première solution  $S_0$   
 $S \leftarrow S_0$  #solution courante  
 $i \leftarrow 0$  #compteur du nombre d'itérations  
 $T \leftarrow 100$  #température initiale
2. Tant que  $i < 1000$  :
  - $T \leftarrow 0.99T$  #la température baisse
  - $i \leftarrow i + 1$
  - choisir au hasard une nouvelle solution  $S'$  dans  $N(S)$
  - si  $S'$  est meilleure que  $S$  OU  $random(0, 1) < exp(-(|S'| - |S|)/T)$  # $|S|$  désigne la longueur de  $S$   
 $S \leftarrow S'$
3. Retourner  $S$

Dans cet algorithme on peut faire varier les paramètres pour obtenir des résultats différents (nombre d'itérations, fonction de probabilité, température initiale). Dans Sage, `float(randrange(0, 10000)/10000)` renverra un flottant au hasard entre 0 et 1 (parmi 10000 valeurs possibles).

Coder l'algorithme de descente de gradient avec recuit simulé `RecuitSimule(M)` qui prend une matrice  $M$  des distances en entrée et renvoie une tournée.

#### Exercice 5 (Bonus : colonie de fourmis).

Comprendre et implémenter un algorithme de colonie de fourmis qui résout le problème du voyageur de commerce.

## Données supplémentaires

Voir aussi le fichier *villes.txt* pour copier-coller plus facilement les données.

Exemple de données avec 20 villes françaises (distances par la route) :

```
distances_20villes=[
[0,182,830,593,918,822,841,974,393,719,705,884,520,863,738,895,627,781,1138,285],
[182,0,648,397,619,675,659,792,216,842,555,840,338,819,546,713,445,599,907,250],
[830,648,0,826,793,1051,536,719,610,866,1219,1379,310,1358,566,730,245,503,1495,855],
[593,397,826,0,327,266,587,637,259,163,475,650,531,630,419,555,584,507,627,403],
[918,619,793,327,0,279,510,466,402,197,523,597,586,876,300,276,562,425,304,679],
[822,675,1051,266,279,0,774,771,478,109,280,318,741,297,585,555,747,698,506,558],
[841,659,536,587,510,774,0,284,520,667,974,1092,398,1071,201,318,329,84,650,844],
[974,792,719,637,466,771,284,0,602,664,1005,1089,584,1068,215,192,506,216,498,926],
[393,216,610,259,402,478,520,602,0,366,609,797,300,776,369,514,372,444,690,324],
[719,842,866,163,197,109,667,664,366,0,326,425,598,404,462,473,640,591,434,506],
[705,555,1219,475,523,280,974,1005,609,326,0,206,909,185,772,799,958,898,769,407],
[884,840,1379,650,597,318,1092,1089,797,425,206,0,1069,21,915,873,1065,1015,824,586],
[520,338,310,531,586,741,398,584,300,598,909,1069,0,1048,379,549,106,357,795,540],
[863,819,1358,630,876,297,1071,1068,776,404,185,21,1048,0,894,852,1044,995,803,565],
[738,546,566,419,300,585,201,215,369,462,772,915,379,894,0,147,331,117,442,678],
[895,713,730,555,276,555,318,192,514,473,799,873,549,852,147,0,498,233,330,824],
[627,445,245,584,562,747,329,506,372,640,958,1065,106,1044,331,498,0,290,771,652],
[781,599,503,507,425,698,84,216,444,591,898,1015,357,995,117,233,290,0,566,768],
[1138,907,1495,627,304,506,650,498,690,434,769,824,795,803,442,330,771,566,0,940],
[285,250,855,403,679,558,844,926,324,506,407,586,540,565,678,824,652,768,940,0]]
noms_20villes={0:'Biarritz',1:'Bordeaux',2:'Brest',3:'Clermont-Fd',4:'Dijon',5:'Grenoble',
6:'Le Havre',7:'Lille',8:'Limoges',9:'Lyon',10:'Marseille',11:'Monaco',12:'Nantes',
13:'Nice',14:'Paris',15:'Reims',16:'Rennes',17:'Rouen',18:'Strasbourg',19:'Toulouse'}
coordonnees_20villes={0:(16,28),1:(24,40),2:(2,78),3:(50,47),4:(64,62),5:(68,40),6:(33,85),
7:(53,94),8:(37,49),9:(62,46),10:(65,23),11:(80,27),12:(20,64),13:(78,25),14:(47,77),
15:(59,81),16:(21,73),17:(39,84),18:(82,75),19:(37,28)}
```

Exemple de données avec les 48 capitales des états des USA métropolitains (ce jeu de données est très classique, et l'optimum est connu, c'est 33523) :

```
distances_48villes=[[0,4727,1205,6363,3657,3130,2414,563,463,5654,1713,1604,2368,2201,
1290,1004,3833,2258,3419,2267,2957,720,1700,5279,2578,6076,3465,2654,3625,3115,1574,
3951,1748,2142,6755,2383,3306,1029,3530,825,2188,4820,3489,1947,6835,1542,2379,3744],
[4727,0,3588,2012,1842,6977,6501,5187,5028,2327,4148,4723,3635,3125,4907,3930,7463,
6338,7243,5105,4043,4022,3677,2863,3106,1850,7173,6630,1204,6814,6001,3447,5253,2656,
3123,6274,7183,5622,3085,4564,2756,1591,7027,6186,3472,5461,4390,2088],
[1205,3588,0,5163,2458,3678,3071,1742,1444,4462,1184,1520,1498,1103,1501,951,4298,
2903,3967,2169,2209,652,828,4136,1518,4873,3954,3254,2446,3581,2441,2960,1966,950,
5564,2916,3878,2035,2482,1027,1395,3617,3891,2686,5661,2023,1867,2560],
[6363,2012,5163,0,2799,8064,7727,6878,6581,1402,5366,5946,4679,4378,6225,5709,8417,
7578,8296,6135,4802,5707,4982,2322,4178,320,8186,7800,2778,7859,7408,3763,6461,4223,
1427,7451,8263,7131,3669,6011,4638,1681,7987,7502,1877,6758,5360,2844],
[3657,1842,2458,2799,0,5330,4946,4200,3824,2012,2573,3157,1924,1580,3427,3179,5749,
4793,5577,3409,2223,3066,2185,1860,1401,2491,5486,5035,894,5141,4611,1669,3677,1590,
3113,4682,5533,4352,1252,3227,2426,1169,5313,4706,3241,3962,2651,304],
[3130,6977,3678,8064,5330,0,743,3209,2670,6929,2831,2266,3407,3854,2178,4076,727,881,
293,1930,3310,3672,3315,6199,3932,7745,365,482,5774,261,1659,4513,1746,4431,7910,769,
207,2225,4435,2681,5053,6384,550,1224,7805,1670,2704,5230],
```

(suite en page suivante)

[2414,6501,3071,7727,4946,743,0,2468,1952,6673,2380,1795,3051,3405,1604,3382,1469,168,1020,1681,3110,2993,2827,6009,3552,7412,1104,267,5300,821,916,4348,1270,3890,7698,332,900,1484,4185,2049,4415,6051,1219,482,7635,1054,2432,4884] ,  
[563,5187,1742,6878,4200,3209,2468,0,718,6203,2241,2051,2920,2762,1687,1304,3932,2331,3487,2669,3487,1175,2260,5840,3141,6596,3563,2728,4120,3240,1559,4507,2082,2658,7304,2512,3364,985,4091,1319,2544,5358,3632,1987,7391,1785,2879,4296] ,  
[463,5028,1444,6581,3824,2670,1952,718,0,5789,1602,1343,2330,2291,970,1451,3376,1796,2959,1951,2835,1112,1725,5346,2628,6285,3007,2193,3889,2661,1122,3920,1372,2391,6883,1927,2845,611,3543,676,2590,4993,3039,1486,6934,1112,2196,3876] ,  
[5654,2327,4462,1402,2012,6929,6673,6203,5789,0,4392,4947,3648,3501,5274,5183,7216,6535,7140,5022,3621,5077,4090,922,3207,1131,7014,6714,2437,6707,6477,2476,5432,3599,1102,6376,7121,6284,2497,5160,4318,937,6795,6507,1268,5773,4249,1914] ,  
[1713,4148,1184,5366,2573,2831,2380,2241,1602,4392,0,586,766,1029,883,2040,3353,2224,3100,1049,1246,1625,503,3841,1196,5054,3042,2488,2945,2676,2087,2331,1114,1650,5459,2132,3037,1958,1997,931,2513,3701,2923,2137,5459,1394,711,2534] ,  
[1604,4723,1520,5946,3157,2266,1795,2051,1343,4947,586,0,1299,1612,406,2208,2824,1639,2542,694,1586,1767,1050,4357,1770,5633,2498,1907,3520,2128,1558,2778,531,2171,6003,1552,2472,1538,2506,791,2912,4277,2403,1564,5983,827,892,3109] ,  
[2368,3635,1498,4679,1924,3407,3051,2920,2330,3648,766,1299,0,646,1642,2446,3840,2905,3655,1488,730,2096,697,3076,533,4363,3567,3122,2453,3219,2842,1592,1791,1480,4706,2772,3610,2721,1232,1656,2550,3001,3403,2860,4697,2126,756,1836] ,  
[2201,3125,1103,4378,1580,3854,3405,2762,2291,3501,1029,1612,646,0,1853,2026,4349,3247,4119,1997,1341,1753,606,3078,419,4070,4052,3517,1923,3690,3032,1866,2142,838,4593,3161,4060,2788,1380,1663,1932,2736,3915,3138,4647,2395,1351,1592] ,  
[1290,4907,1501,6225,3427,2178,1604,1687,970,5274,883,406,1642,1853,0,2029,2803,1438,2466,986,1987,1593,1253,4716,2072,5915,2454,1764,3710,2082,1204,3164,497,2287,6342,1419,2379,1134,2867,554,2885,4569,2405,1289,6338,555,1297,3406] ,  
[1004,3930,951,5709,3179,4076,3382,1304,1451,5183,2040,2208,2446,2026,2029,0,4759,3220,4368,2900,3151,442,1765,4960,2444,5443,4396,3610,2932,4034,2572,3891,2525,1590,6278,3313,4261,2033,3398,1476,1241,4287,4390,2928,6419,2428,2749,3337] ,  
[3833,7463,4298,8417,5749,727,1469,3932,3376,7216,3353,2824,3840,4349,2803,4759,0,1601,477,2359,3617,4345,3851,6433,4372,8098,370,1206,6267,726,2384,4754,2335,4991,8148,1452,609,2949,4752,3331,5687,6746,437,1948,8005,2334,3098,5618] ,  
[2258,6338,2903,7578,4793,881,168,2331,1796,6535,2224,1639,2905,3247,1438,3220,1601,0,1165,1563,2988,2829,2666,5882,3401,7263,1233,399,5138,923,794,4227,1117,3724,7565,286,1049,1348,4051,1881,4248,5903,1322,355,7508,887,2302,4736] ,  
[3419,7243,3967,8296,5577,293,1020,3487,2959,7140,3100,2542,3655,4119,2466,4368,477,1165,0,2170,3520,3965,3588,6393,4183,7977,202,767,6041,438,1932,4706,2027,4711,8107,1061,132,2503,4652,2972,5344,6617,486,1501,7989,1962,2939,5469] ,  
[2267,5105,2169,6135,3409,1930,1681,2669,1951,5022,1049,694,1488,1997,986,2900,2359,1563,2170,0,1430,2460,1547,4333,2019,5817,2079,1694,3910,1733,1813,2668,654,2694,6029,1366,2130,1991,2525,1474,3542,4455,1923,1641,5957,1071,777,3302] ,  
[2957,4043,2209,4802,2223,3310,3110,3487,2835,3621,1246,1586,730,1341,1987,3151,3617,2988,3520,1430,0,2779,1387,2905,1062,4482,3398,3119,2922,3087,3115,1240,1953,2175,4607,2796,3501,3119,1136,2173,3268,3136,3189,3029,4527,2355,711,2042] ,  
[720,4022,652,5707,3066,3672,2993,1175,1112,5077,1625,1767,2096,1753,1593,442,4345,2829,3965,2460,2779,0,1401,4781,2166,5427,3984,3212,2946,3620,2224,3603,2089,1496,6178,2906,3861,1719,3132,1040,1479,4211,3969,2553,6290,2012,2336,3189] ,  
[1700,3677,828,4982,2185,3315,2827,2260,1725,4090,503,1050,697,606,1253,1765,3851,2666,3588,1547,1387,1401,0,3621,903,4675,3537,2954,2475,3169,2427,2254,1578,1148,5177,2598,3521,2194,1833,1074,2054,3340,3423,2541,5213,1801,1077,2190] ,  
[5279,2863,4136,2322,1860,6199,6009,5840,5346,922,3841,4357,3076,3078,4716,4960,6433,5882,6393,4333,2905,4781,3621,0,2718,2042,6254,6024,2569,5966,5913,1687,4807,3384,1716,5699,6384,5787,1852,4687,4285,1272,6022,5892,1629,5178,3581,1639] ,

(suite en page suivante)

[2578,3106,1518,4178,1401,3932,3552,3141,2628,3207,1196,1770,533,419,2072,2444,4372,3401,4183,2019,1062,2166,903,2718,0,3864,4097,3635,1932,3748,3274,1448,2284,1164,4286,3283,4136,3086,967,1973,2285,2507,3935,3331,4312,2589,1284,1340] ,

[6076,1850,4873,320,2491,7745,7412,6596,6285,1131,5054,5633,4363,4070,5915,5443,8098,7263,7977,5817,4482,5427,4675,2042,3864,0,7866,7483,2515,7539,7101,3449,6146,3938,1375,7134,7944,6831,3349,5709,4397,1363,7667,7190,1798,6446,5041,2528] ,

[3465,7173,3954,8186,5486,365,1104,3563,3007,7014,3042,2498,3567,4052,2454,4396,370,1233,202,2079,3398,3984,3537,6254,4097,7866,0,839,5973,374,2019,4569,1996,4669,7970,1085,305,2581,4532,2976,5339,6509,287,1581,7844,1974,2838,5369] ,

[2654,6630,3254,7800,5035,482,267,2728,2193,6714,2488,1907,3122,3517,1764,3610,1206,399,767,1694,3119,3212,2954,6024,3635,7483,839,0,5427,558,1181,4349,1377,4044,7723,356,653,1744,4218,2241,4614,6121,955,743,7644,1231,2465,4957] ,

[3625,1204,2446,2778,894,5774,5300,4120,3889,2437,2945,3520,2453,1923,3710,2932,6267,5138,6041,3910,2922,2946,2475,2569,1932,2515,5973,5427,0,5612,4824,2550,4050,1498,3476,5071,5980,4470,2096,3388,1911,1501,5831,4994,3704,4264,3209,1196] ,

[3115,6814,3581,7859,5141,261,821,3240,2661,6707,2676,2128,3219,3690,2082,4034,726,923,438,1733,3087,3620,3169,5966,3748,7539,374,558,5612,0,1716,4280,1624,4298,7679,735,420,2263,4216,2606,4967,6179,400,1277,7567,1609,2501,5032] ,

[1574,6001,2441,7408,4611,1659,916,1559,1122,6477,2087,1558,2842,3032,1204,2572,2384,794,1932,1813,3115,2224,2427,5913,3274,7101,2019,1181,4824,1716,0,4330,1180,3346,7545,1023,1808,578,4062,1438,3693,5763,2115,440,7537,763,2404,4603] ,

[3951,3447,2960,3763,1669,4513,4348,4507,3920,2476,2331,2778,1592,1866,3164,3891,4754,4227,4706,2668,1240,3603,2254,1687,1448,3449,4569,4349,2550,4280,4330,0,3184,2510,3402,4031,4698,4281,533,3245,3612,2187,4339,4265,3296,3576,1941,1381] ,

[1748,5253,1966,6461,3677,1746,1270,2082,1372,5432,1114,531,1791,2142,497,2525,2335,1117,2027,654,1953,2089,1578,4807,2284,6146,1996,1377,4050,1624,1180,3184,0,2685,6475,1022,1952,1341,2963,1050,3358,4787,1926,1086,6436,422,1244,3619] ,

[2142,2656,950,4223,1590,4431,3890,2658,2391,3599,1650,2171,1480,838,2287,1590,4991,3724,4711,2694,2175,1496,1148,3384,1164,3938,4669,4044,1498,4298,3346,2510,2685,0,4697,3693,4636,2975,1981,1909,1124,2718,4565,3548,4830,2839,2140,1751] ,

[6755,3123,5564,1427,3113,7910,7698,7304,6883,1102,5459,6003,4706,4593,6342,6278,8148,7565,8107,6029,4607,6178,5177,1716,4286,1375,7970,7723,3476,7679,7545,3402,6475,4697,0,7393,8097,7370,3515,6249,5379,2001,7738,7556,461,6829,5267,3013] ,

[2383,6274,2916,7451,4682,769,332,2512,1927,6376,2132,1552,2772,3161,1419,3313,1452,286,1061,1366,2796,2906,2598,5699,3283,7134,1085,356,5071,735,1023,4031,1022,3693,7393,0,965,1542,3883,1913,4286,5772,1121,600,7322,902,2128,4608] ,

[3306,7183,3878,8263,5533,207,900,3364,2845,7121,3037,2472,3610,4060,2379,4261,609,1049,132,2130,3501,3861,3521,6384,4136,7944,305,653,5980,420,1808,4698,1952,4636,8097,965,0,2380,4629,2877,5250,6583,570,1380,7986,1866,2904,5432] ,

[1029,5622,2035,7131,4352,2225,1484,985,611,6284,1958,1538,2721,2788,1134,2033,2949,1348,2503,1991,3119,1719,2194,5787,3086,6831,2581,1744,4470,2263,578,4281,1341,2975,7370,1542,2380,0,3952,1127,3197,5518,2658,1002,7395,951,2429,4380] ,

[3530,3085,2482,3669,1252,4435,4185,4091,3543,2497,1997,2506,1232,1380,2867,3398,4752,4051,4652,2525,1136,3132,1833,1852,967,3349,4532,4218,2096,4216,4062,533,2963,1981,3515,3883,4629,3952,0,2873,3080,2012,4324,4046,3478,3328,1755,1000] ,

[825,4564,1027,6011,3227,2681,2049,1319,676,5160,931,791,1656,1663,554,1476,3331,1881,2972,1474,2173,1040,1074,4687,1973,5709,2976,2241,3388,2606,1438,3245,1050,1909,6249,1913,2877,1127,2873,0,2374,4392,2943,1659,6285,1012,1563,3254] ,

[2188,2756,1395,4638,2426,5053,4415,2544,2590,4318,2513,2912,2550,1932,2885,1241,5687,4248,5344,3542,3268,1479,2054,4285,2285,4397,5339,4614,1911,4967,3693,3612,3358,1124,5379,4286,5250,3197,3080,2374,0,3386,5284,3997,5585,3386,3125,2664] ,

[4820,1591,3617,1681,1169,6384,6051,5358,4993,937,3701,4277,3001,2736,4569,4287,6746,5903,6617,4455,3136,4211,3340,1272,2507,1363,6509,6121,1501,6179,5763,2187,4787,2718,2001,5772,6583,5518,2012,4392,3386,0,6314,5837,2205,5095,3680,1169] ,

[3489,7027,3891,7987,5313,550,1219,3632,3039,6795,2923,2403,3403,3915,2405,4390,437,1322,486,1923,3189,3969,3423,6022,3935,7667,287,955,5831,400,2115,4339,1926,4565,7738,1121,570,2658,4324,2943,5284,6314,0,1676,7603,1964,2662,5184] ,

(suite en page suivante)



```

[1947,6186,2686,7502,4706,1224,482,1987,1486,6507,2137,1564,2860,3138,1289,2928,1948,
355,1501,1641,3029,2553,2541,5892,3331,7190,1581,743,4994,1277,440,4265,1086,3548,
7556,600,1380,1002,4046,1659,3997,5837,1676,0,7521,744,2325,4670],
[6835,3472,5661,1877,3241,7805,7635,7391,6934,1268,5459,5983,4697,4647,6338,6419,8005,
7508,7989,5957,4527,6290,5213,1629,4312,1798,7844,7644,3704,7567,7537,3296,6436,4830,
461,7322,7986,7395,3478,6285,5585,2205,7603,7521,0,6805,5208,3102],
[1542,5461,2023,6758,3962,1670,1054,1785,1112,5773,1394,827,2126,2395,555,2428,2334,
887,1962,1071,2355,2012,1801,5178,2589,6446,1974,1231,4264,1609,763,3576,422,2839,
6829,902,1866,951,3328,1012,3386,5095,1964,744,6805,0,1644,3928],
[2379,4390,1867,5360,2651,2704,2432,2879,2196,4249,711,892,756,1351,1297,2749,3098,
2302,2939,777,711,2336,1077,3581,1284,5041,2838,2465,3209,2501,2404,1941,1244,2140,
5267,2128,2904,2429,1755,1563,3125,3680,2662,2325,5208,1644,0,2532],
[3744,2088,2560,2844,304,5230,4884,4296,3876,1914,2534,3109,1836,1592,3406,3337,5618,
4736,5469,3302,2042,3189,2190,1639,1340,2528,5369,4957,1196,5032,4603,1381,3619,1751,
3013,4608,5432,4380,1000,3254,2664,1169,5184,4670,3102,3928,2532,0]]
noms_48villes={0:'Montgomery (Alabama)',1:'Phoenix (Arizona)',2:'Little Rock (Arkansas)',
3:'Sacramento (California)',4:'Denver (Colorado)',5:'Hartford (Connecticut)',
6:'Dover (Delaware)',7:'Tallahassee (Florida)',8:'Atlanta (Georgia)',9:'Boise (Idaho)',
10:'Springfield (Illinois)',11:'Indianapolis (Indiana)',12:'Des Moines (Iowa)',
13:'Topeka (Kansas)',14:'Frankfort (Kentucky)',15:'Baton Rouge (Louisiana)',
16:'Augusta (Maine)',17:'Annapolis (Maryland)',18:'Boston (Massachusetts)',
19:'Lansing (Michigan)',20:'St Paul (Minnesota)',21:'Jackson (Mississippi)',
22:'Jefferson City (Missouri)',23:'Helena (Montana)',24:'Lincoln (Nebraska)',
25:'Carson City (Nevada)',26:'Concord (N Hampshire)',27:'Trenton (N Jersey)',
28:'Santa Fe (N Mexico)',29:'Albany (N York)',30:'Raleigh (N Carolina)',
31:'Bismarck (N Dakota)',32:'Columbus (Ohio)',33:'Oklahoma City (Oklahoma)',
34:'Salem (Oregon)',35:'Harrisburg (Pennsylvania)',36:'Providence (Rhode Is)',
37:'Columbia (S Carolina)',38:'Pierre (S Dakota)',39:'Nashville (Tennessee)',
40:'Austin (Texas)',41:'Salt Lake City (Utah)',42:'Montpelier (Vermont)',
43:'Richmond (Virginia)',44:'Olympia (Washington)',45:'Charleston (W Virginia)',
46:'Madison (Wisconsin)',47:'Cheyenne (Wyoming)'}
coordonnees_48villes={0:(6734,1453),1:(2233,10),2:(5530,1424),3:(401,841),
4:(3082,1644),5:(7608,4458),6:(7573,3716),7:(7265,1268),8:(6898,1885),9:(1112,2049),
10:(5468,2606),11:(5989,2873),12:(4706,2674),13:(4612,2035),14:(6347,2683),15:(6107,669),
16:(7611,5184),17:(7462,3590),18:(7732,4723),19:(5900,3561),20:(4483,3369),21:(6101,1110),
22:(5199,2182),23:(1633,2809),24:(4307,2322),25:(675,1006),26:(7555,4819),27:(7541,3981),
28:(3177,756),29:(7352,4506),30:(7545,2801),31:(3245,3305),32:(6426,3173),33:(4608,1198),
34:(23,2216),35:(7248,3779),36:(7762,4595),37:(7392,2244),38:(3484,2829),39:(6271,2135),
40:(4985,140),41:(1916,1569),42:(7280,4899),43:(7509,3239),44:(10,2676),45:(6807,2993),
46:(5185,3258),47:(3023,1942)}

```