TP 5
Création d'un chemin auto-évitant – Sujet

https://doi.org/10.1098/rspa.2018.0549

Une marche auto-évitante est un processus au cours duquel un point décrit un chemin auto-évitant, c'est-à-dire qui ne se recoupe jamais lui-même.

Dans ce sujet, on appellera chemin auto-évitant (CAE) de longueur n tout ensemble de $n+1$ points $P_i = (x_i, y_i) \in \mathbb{Z}^2$ pour $0 \leq i \leq n$ tels que :

- $\forall i, ||\overrightarrow{P_i P_{i+1}}|| = 1$;
- $\forall (i,j), i \neq j \Rightarrow P_i \neq P_j$.

Chaque point du chemin sera représenté par une liste à deux éléments entiers précisant les deux coordonnées (par exemple $P_i = (5, -4)$ est représenté par la liste `[5,-4]`). Le chemin lui-même est constitué de la liste des points, dans l'ordre dans lequel ils sont atteints. Les codes Python produits dans cette partie devront manipuler exclusivement des coordonnées entières. La figure 1 donne une représentation graphique de CAE de longueur 30 à partir de `[0, 0]`.

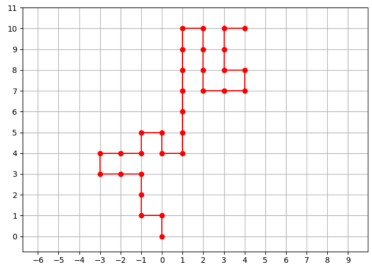

FIGURE 1 – CAE de longueur 30

Préambule - Tracé d'un chemin

On cherche d'abord à afficher un chemin quelquonque.

Le module `matplotlib` permet de réaliser des tracés.

```
# Import (et renommage) de la bibliothèque
import matplotlib.pyplot as plt     # À écrire en tout début de fichier

les_x = [0,1,1,2]           # Liste des abscisses
les_y = [0,0,1,1]           # Liste des ordonnées

plt.plot(les_x,les_y,'-or') # On affiche le tracé en rouge (r). Les points
                        # sont marqués par un cercle (o)
                        # et reliés par un trait (-).

plt.grid()
plt.axis('equal')
plt.show()                      # Affiche la courbe
```

On appelle `chemin` une liste de points, par exemple `[[0,0],[1,0],[1,1],[2,1]]`.

Question 1 Après avoir importé les bibliothèques nécessaires, implémenter la fonction `plot_chemin(chemin:[[int]]) -> None` qui trace le chemin `chemin`. Il faudra, à l'intérieur de cette fonction, créer les listes `les_x` et `les_y` des abscisses et des ordonnées de chacun des points .

Test

`plot_chemin([[0,0],[1,0],[1,1],[2,1]])` doit afficher la figure 2.

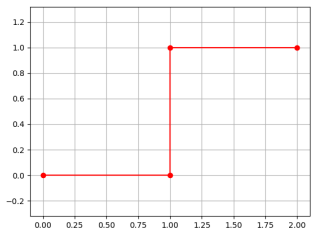

FIGURE 2 – Affichage d'un chemin

Chemin naïf

On s'intéresse dans un premier temps à une méthode naïve pour générer un chemin auto-évitant de longueur n sur une grille carrée. La méthode adoptée est une approche gloutonne :

1. le premier point est choisi à l'origine : $P_0 = (0, 0)$.
2. en chaque position atteinte par le chemin, on recense les positions voisines accessibles pour le pas suivant et on en sélectionne une au hasard. En l'absence de positions accessibles l'algorithme échoue.
3. on itère l'étape 2 jusqu'à ce que le chemin possède la longueur désirée ou échoue.

L'ordre des voisins n'est pas important.

Question 2 Implémenter la fonction `positions_voisines(p:[int]) -> [[int]]` qui construit la liste des positions voisines possibles à partir du point `p`.

Test

`positions_voisines([0,0])` doit renvoyer `[[0,1],[1,0],[0,-1],[-1,0]]`.

Question 3 Implémenter la fonction `positions_possibles(p:[int], chemin:[[int]]) -> [[int]]` qui construit la liste des positions suivantes possibles à partir du point `p`. La liste `chemin` contient les points déjà atteints.

L'ordre des positions possibles n'est pas important.

Test

`positions_possibles([0,0],[])` doit renvoyer `[[0, 1], [1, 0], [0, -1], [-1, 0]]`.
`positions_possibles([0,1],[[0,0]])` doit renvoyer `[[0, 2], [1, 1], [-1, 1]]`.

Question 4 (Sur feuille) Mettre en évidence graphiquement un exemple de CAE le plus court possible pour lequel, à une étape donnée, la fonction `positions_possibles` va renvoyer une liste vide. En prenant en compte les symétries, combien de tels chemins distincts existent pour cette longueur minimale ?

Le module `random` fournit la fonction `randrange(a, b)` qui renvoie un entier compris entre `a` et `b-1` inclus, ainsi que la fonction `choice(L)` qui renvoie l'un des éléments de la liste `L`, dans les deux cas choisis aléatoirement avec une probabilité uniforme.

```
# Import de la bibliothèque (et renommage en rd)
import random as rd             # À écrire en tout début de fichier

print(rd.randrange(1,5))        # À utiliser où on veut dans le fichier
                                # Affiche aléatoirement 1, 2, 3 ou 4
L = [1,2,3]
rd.choice(L)                # Renvoie aléatoirement 1, 2, ou 3
```

Question 5 Implémenter la fonction `genere_chemin_naif(n:int) -> [[int]]` qui construit la liste des points représentant le chemin auto-évitant de longueur `n` en partant de l'origine $(0, 0)$ et renvoie le résultat, ou bien renvoie la valeur spéciale `[[]]` si à une étape `positions_possibles` renvoie une liste vide.

Test

Tracer un chemin de longueur 20. Tracer un chemin de longueur 140. Que constatez-vous ?

On souhaite connaître la probabilité de pouvoir obtenir un chemin autoévitant de taille n. Pour cela, pour des longueurs de chemin comprises entre 0 et 350, on peut calculer le taux de chemins réalisables.

Question 6 En testant la possibilté de réaliser 1000 chemins pour chacune des longueurs comprises entre 1 et 350, tracer la probabilité qu'un chemin échoue en fonction de sa longueur (voir figure 3).

Question 7 Décrire ce que représente ce graphique et interpréter sa signification pour la méthode naïve.

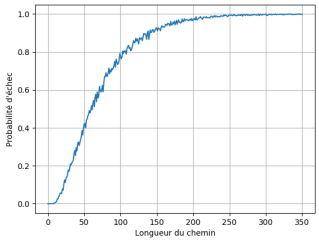

Figure 3 – Probabilité de pouvoir réaliser un CAE

Methode du pivot

Afin d'éviter les inconvénients de la méthode précédente, on s'intéresse à une solution différente nommée méthode du pivot, proposée par Moti Lal en 1969. Son principe est le suivant :

1. on part d'un chemin auto-évitant arbitraire de longueur n. Ici, on choisira une initialisation très simple ; le chemin droit `[[0, 0], [1, 0], [2, 0], ..., [n, 0]]` ;
2. on sélectionne au hasard un point, nommé pivot, entre le second et l'avant-dernier point du chemin, et un angle aléatoire de rotation parmi $\pi, \frac{\pi}{2}, -\frac{\pi}{2}$;
3. on laisse les points avant le pivot inchangés et on fait subir à l'ensemble des points situés strictement après le pivot une rotation ayant pour centre le pivot et pour angle, l'angle choisi à l'étape 2 ci-dessus ;
4. si le chemin ainsi obtenu est auto-évitant, on le garde. Sinon, on reprend à l'étape 2 la sélection d'un pivot et d'un angle, jusqu'à en trouver une paire qui convienne ;
5. on répète les étapes 2 à 4 un certain nombre de fois. Le choix du nombre minimal de rotations à effectuer pour obtenir un chemin non corrélé au chemin initial est laissé de côté dans ce sujet.

Cette méthode permet de générer de manière efficace des marches auto-évitantes de plusieurs milliers de points, comme la marche de longueur 2000 de la figure 4.

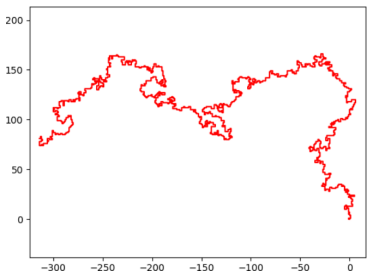

Figure 4 – Marche de longueur 2000

Dans cet algorithme, une étape importante est la vérification qu'un chemin donné est auto-évitant. Pour vérifier cela on pourrait bien sûr s'y prendre comme dans la fonction `positions_possibles`, mais on adopte une méthode différente avec une fonction `est_CAE(chemin)` qui trie les points du chemin puis met à profit ce tri pour vérifier efficacement que le chemin est auto-évitant.

On utilise pour cela la fonction **`sorted`** qui renvoie une nouvelle liste triée par ordre croissant. Elle fonctionne sur tout type d'éléments disposant d'une relation d'ordre, y compris des listes pour lesquelles l'ordre lexicographique (ordre du premier élément, en cas d'égalité du second, etc.) est appliqué.

Question 8 Implémenter la fonction `est_CAE(chemin:[[int]]) -> bool` qui vérifie si un chemin est auto-évitant en se basant sur **`sorted`**.

On rappelle qu'un chemin de longueur n est auto-évitant si pour $0 \leq i \leq n$:

- $\forall i, ||\overrightarrow{P_i P_{i+1}}|| = 1$;
- $\forall (i,j), i \neq j \Rightarrow P_i \neq P_j$.

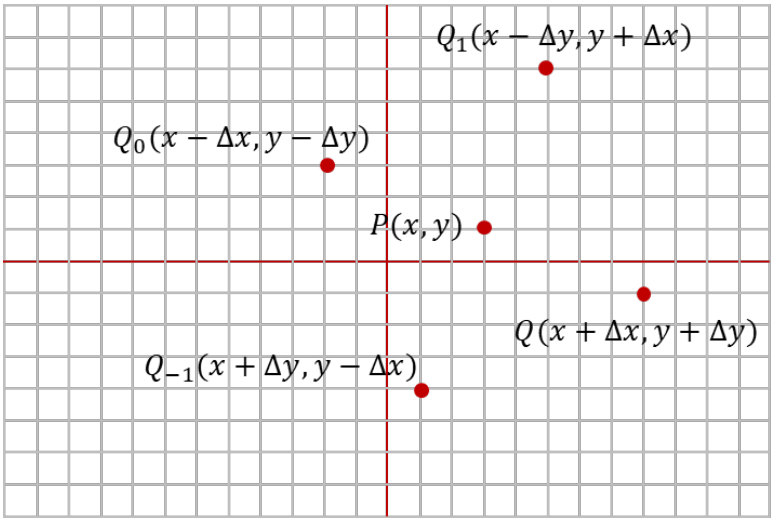

FIGURE 5 – Illustration de la rotation d'un point

Question 9 Implémenter la fonction `rot(p:[int], q:[int], a:int) -> [int]` qui renvoie le point image du point `q` par la rotation de centre `p` et d'angle défini par la valeur de `a` : `0` pour π, `1` pour $\frac{\pi}{2}$, `2` pour $-\frac{\pi}{2}$. On pourra pour cela utiliser la figure 5.

Question 10 Implémenter la fonction `rotation(chemin:[[int]], i_piv:int, a:int) -> [[int]]` qui renvoie un nouveau chemin identique à `chemin` jusqu'au pivot d'indice `i_piv`, et ayant subi une rotation de `a` autour du pivot (même codage que la fonction précédente) sur la partie strictement après le pivot.

Question 11 Implémenter la fonction `genere_chemin_pivot(n:int, n_rot:int) -> [[int]]` permettant de générer un chemin autoévitant de longueur `n` en appliquant `n_rot` rotations. L'application d'une rotation peut nécessiter plusieurs tentatives.