Interrogation d'informatique n°1

Exercice. Codage des matrices creuses par des dictionnaires

Les matrices creuses sont des matrices où la proportion de coefficients non nuls est très faible.

L'idée est alors de stocker seulement les coefficients non nuls.

Une matrice carrée creuse $A \in \mathcal{M}_n(\mathbb{R})$ est définie par le dictionnaire dont les couples (*clé,valeur*) sont d'une part (" dim ", n) et d'autre part les $((i,j), A[i,j])$ pour lesquels $A[i,j] \neq 0$.

Par exemple, `A = {'dim':10,(0,1):5,(3,2):1,(4,6):-1}`

Écrire une fonction `somme(A,B)` qui renvoie la somme de deux matrices creuses de même dimension.

On proposera une fonction de complexité linéaire en les tailles des dictionnaires A et B.

Problème. Chemins auto-évitants

Un point entier du plan est un couple d'entiers $P = (x, y)$. Un chemin auto-évitant (CAE) de longueur n est une liste de $n+1$ points entiers $[P_0, ., ..., P_n]$ telle que $\begin{cases} \forall i \in [\![0, n-1]\!],\ \|P_{i+1} - P_i\| = 1 \\ \forall i, j \in [\![0, n]\!]^2,\ (i \neq j) \Rightarrow (P_i \neq P_j) \end{cases}$.

Par exemple, $[(0,0),(0,1),(0,2),(1,2),(1,1)]$ est un CAE de longueur 4.

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) *Initialisation* : 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.

Le module `random` fournit la fonction `randrange(a, b)` qui renvoie un entier compris entre a et $b-1$ inclus. L'expression `x in L` est une expression booléenne qui vaut `True` si x est l'un des éléments de L et `False` sinon. On supposera que la méthode employée pour évaluer cette expression sur une liste est une recherche séquentielle. La valeur spéciale `None` est utilisée en PYTHON pour représenter une valeur invalide, inconnue ou indéfinie. L'expression booléenne `v is None` indique si la valeur de v est cette valeur spéciale.

Q.1) Écrire une fonction `choice(L)` qui étant donnée une liste non vide, renvoie un de ses éléments choisis aléatoirement selon la loi uniforme.

Q.2) Écrire une fonction `positions_possibles(p,atteints)` qui construit la liste des positions suivantes possibles à partir du point `p`. La liste `atteints` contient les points déjà atteints par le chemin.

Q.2) bis) *Question supplémentaire hors-interrogation.* Donner graphiquement un exemple de CAE le plus court possible pour lequel, à une étape donnée, la fonction `positions_possibles` va renvoyer une liste vide. Combien de tels chemins distincts existent pour cette longueur minimale ?

Q.3) Écrire une fonction fonction `genere_chemin_naif(n)` qui construit la liste des points représentant un chemin auto-évitant de longueur n et renvoie le résultat, ou bien renvoie la valeur spéciale `None` si à une étape `positions_possibles` renvoie une liste vide.

Q.4) Évaluer avec soin la complexité temporelle asymptotique dans le pire des cas de la fonction `genere_chemin_naif(n)` en fonction de n, en supposant que la fonction ne renvoie pas `None`.

Q.5) On considère le code suivant :

```
01.     N, M, P = 10000, 301, []
02.     for n in range(1, M) :
03.         nb = 0
04.         for i in range(N) :
05.             chemin = genere_chemin_naif(n)
06.             if chemin is None :
07.                 nb += 1
08.         P.append(nb/N)
```

Que représentent les éléments de P ?
Les éléments de P forment une suite croissante s'approchant rapidement de 1. Que peut-on en déduire ?

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 $\left\{\pi, \frac{\pi}{2}, -\frac{\pi}{2}\right\}$.

(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 conviennent.

(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.

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 plus efficace en utilisant les propriétés des dictionnaires.

Q.6) Implémenter la fonction booléenne `est_CAE(chemin)` qui vérifie si un chemin (de longueur n) est auto-évitant en utilisant un dictionnaire. On demande une complexité linéaire, c'est-à-dire en $O(n)$.

Q.7) *Remarque* : Cette question essentiellement mathématique peut être admise dans un premier temps.

Implémenter la fonction `rot(p,q,a)` qui renvoie le point image du point q par la rotation r de centre p et d'angle θ défini par la valeur de a : 0 pour $\theta = \pi$, 1 pour $\theta = \frac{\pi}{2}$, et 2 pour $\theta = -\frac{\pi}{2}$.

Suggestion : Pour trouver la formule mathématique, utiliser les complexes : $r(q) = p + e^{i\theta}(q - p)$.

Q.8) Implémenter la fonction `rotation(chemin,i_piv,a)` 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.

Q.9) Implémenter la fonction `genere_chemin_pivot(n,n_rot)` permettant de générer un chemin auto-évitant de longueur n en appliquant `n_rot` rotations. L'application d'une rotation peut nécessiter plusieurs tentatives.