INFORMATIQUE

Devoir surveillé n° 2

Durée 2h — sans ordinateur ni calculatrice

I Exercices sur la récursivité

Exercice 1 La suite (u_n) est définie par $u_0 = a$ et $\forall n \in \mathbb{N}$, $u_{n+1} = 4 * u_n + 2$.
Écrire une fonction récursive `suite(n,a)` qui renvoie u_n.

Exercice 2 Écrire une fonction récursive `est_element(L,x)` qui renvoie `True` si `x` est élément de la liste `L` et qui renvoie `False` sinon.

II Le recuit simulé appliqué au problème du voyageur de commerce

On rappelle les instructions suivantes des modules random, math et numpy :

- La fonction `randint(a,b)` renvoie un entier tiré au sort (loi uniforme) entre les entiers `a` et `b` (`a` et `b` compris)
- La fonction `random()` renvoie un flottant tiré au sort (loi uniforme) dans [0,1[, c'est-à-dire, que si $0 \leq a \leq b \leq 1$, la probabilité qu'un résultat de la fonction `random` appartienne à l'intervalle $[a, b[$ est égale à $b - a$.
- La fonction `sqrt` est la racine carrée ; la fonction `exp` est l'exponentielle.
- La fonction `zeros([n,p])` renvoie un tableau numpy `n` lignes et `p` colonnes.

II.1 Principe général

Un voyageur de commerce doit visiter n villes une et une seule fois et revenir à son point de départ (on dit qu'il fait un circuit). Son but est de minimiser la distance totale parcourue. Le nombre de circuits passant par les n villes une et une seule fois est $n!$, il est donc illusoire d'examiner tous les circuits. La méthode du recuit simulé, inspirée du recuit des matériaux consiste à partir d'un premier circuit, effectuer une légère modification aléatoire de ce circuit et comparer la distance parcourue à la distance parcourue avec le premier circuit. Si elle est inférieure, le nouveau circuit est retenu. Si elle est supérieur, le nouveau circuit est retenu avec une probabilité p et n'est pas retenu avec la probabilité $1 - p$. Ce processus est itéré avec une diminution du paramètre p au cours du temps.

II.2 Pourquoi retenir des circuits moins bons?

L'espace des circuits possibles étant très grand, ne retenir que les circuits meilleurs ne permettrait pas d'explorer cet espace des circuits de manière satisfaisante. On risquerait d'arriver à un "minimum local" éloigné du minimum global. Comme dans le recuit des matériaux, lors du lent refroidissement, l'energie est décroissante globalement mais il y a des recombinaisons intermédiaires des atomes qui augmentent l'énergie et permettent un meilleur agencement final des atomes. Ces recombinaisons intermédiaires "moins bonnes" sont de moins en moins probables au fur et à mesure que la température diminue.

II.3 Liaisons de ville à ville

Dans une situation réelle, il faudrait partir d'une carte routière pour en déduire les distances mutuelles des villes considérées. Cela donnerait un graphe valué. Nous adoptons ici un autre modèle, plus simple.

Nous allons supposer dans la suite que les villes sont numérotées de 0 à $n - 1$ et que la position de la ville (numérotée) i est donnée par deux coordonnées x_i et y_i. La distance entre la ville i et la ville j est la distance euclidienne usuelle entre les points de coordonnées (x_i, y_i) et (x_j, y_j), c'est-à-dire le réel $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

II.4 Matrice des distances mutuelles

On suppose données les coordonnées des n villes à l'aide d'une liste `L` de longueur n où `L[i]` est la liste $[x_i, y_i]$ des coordonnées de la ville numérotée i.

Q1 Un point `P` est une liste `[x,y]` où `x` et `y` sont des flottants, coordonnées du points `P`. Écrire une fonction `distance(P1,P2)` d'arguments des points `P1` et `P2` et qui renvoie la distance de `P1` à `P1`.

Q2 Écrire une fonction `distances_mutuelles(L)` d'argument une liste `L` de la forme $[[x_0, y_0], [x_1, y_1], \ldots, [x_{n-1}, y_{n-1}]]$. Cette fonction renvoie la matrice `M` à `n` lignes et `n` colonnes — représentée, selon votre choix, par une liste de listes ou un tableau numpy — telle que `M[i][j]` soit la distance de la ville `i` à la ville `j`.

II.5 Longueur d'un circuit

Un circuit est donné par une liste d'entiers deux à deux distincts $C = [v_0, v_1, \ldots, v_{n-1}]$, $v_i \in [[0, n-1]]$ désignant un numéro de ville. La longueur du circuit C est la somme des distances des villes successives de ce circuit soit $d(v_0, v_1) + d(v_1, v_2) + \cdots + d(v_{n-2}, v_{n-1}) + d(v_{n-1}, v_0)$, où $d(v_i, v_j)$ représente la distance de la ville v_i à la ville v_j.

Q3 Écrire une fonction `longueur(M,C)` qui renvoie la longueur du circuit `C` connaissant la matrice `M` des distances mutuelles.

Q4 On suppose que l'argument `L` de la fonction suivante est une liste de circuits `L=[C0,...,Ck]`.
Écrire une fonction `meilleur_circuit(M,L)` qui renvoie la liste `[Ci,li]` ou `li` est la longueur minimale d'un circuit de la liste `L` et `Ci` un circuit de longueur `li` (en cas d'ex-aequo, on renverra un quelconque de ces circuits).

II.6 Modification d'un circuit

À chaque itération de l'algorithme du recuit simulé, une modification aléatoire du circuit étudié se produit :
on tire au sort deux entiers i et j (avec $0 \leq i < j \leq n-1$) (voir l'instruction randint en annexe).
Le circuit $[v_0, v_1, \ldots v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_{n-1}]$ est modifié en inversant l'ordre des villes situées entre les indices i et j ; le circuit modifié est donc $[v_0, v_1, \ldots v_{i-1}, v_j, v_{j-1}, \ldots, v_{i+1}, v_i, v_{j+1}, \ldots, v_{n-1}]$.

Q5 Écrire une fonction `modif_alea(C)` qui renvoie un circuit modifié de cette manière à partir du circuit `C`.

II.7 Mise en oeuvre de l'algorithme

Si `l` est la longueur du circuit modifié et `l0` la longueur de l'ancien circuit, la probabilité que le circuit modifié soit retenu comme nouveau circuit est $\begin{cases} p = 1 \text{ si } \texttt{l} < \texttt{l0} \\ p = e^{\frac{\texttt{l0}-\texttt{l}}{kT}} \text{ sinon} \end{cases}$ (relation $(\mathcal{R})$), où k une constante strictement positive et où T décroît géométriquement en fonction de l'itération de l'algorithme, c'est-à-dire que, à l'itération $i+1$ de l'algorithme, T vaut $T_{i+1} = (1-x)\,T_i$ où T_i est la valeur de T à l'itération i de l'algorithme et $x \in]0, 1[$ est fixé.

Q6 Écrire une fonction python `proba(l0,l,k,T)` qui renvoie le nombre réel `p` défini par la relation $(\mathcal{R})$ ci-dessus.

Q7 En utilisant `random` décrit en préambule, écrire une fonction `bernoulli(p)` qui simule la loi de Bernoulli de paramètre `p,` c'est-à dire qui renvoie 1 avec la probabilité `p` et qui renvoie 0 avec la probabilité `1-p`.

Q8 Écrire une fonction `main(L,T0,k,x,N)` qui applique `N` fois le recuit simulé décrit en I1 et I2 où l'argument `L` est la liste des coordonnées des villes à visiter, `T0,k,x` permettant de calculer `p` à chaque itération. Cette fonction renverra le derner circuit. On choisira comme premier circuit celui ou les numéros de villes sont rangés dans l'ordre croissant.

On voudrait afficher le circuit $C = [v_0, v_1, \ldots, v_{n-1}]$, c'est-à-dire tracer la ligne polygonale rejoignant par un segment la ville v_0 à v_1, puis v_1 à v_2, ainsi de suite jusqu'au segments de v_{n-2} à v_{n-1} et de v_{n-1} à v_0.

Q9 En utilisant les fonctions du module matplotlib.pyplot, écrire une fonction `affiche(C,L)` qui produit cet affichage (l'argument `L` est la liste des coordonnées des villes comme en I4).

Q10 Quelle est la complexité de la fonction `longueur(M,C)`. Expliquer comment on pourrait la remplacer dans le programme `main` (sauf pour le premier appel de la fonction longueur) par une ou plusieurs instructions s'exécutant en temps constant. On ne détaillera pas tous les changements devant être apportés à la fonction en conséquence.

III Base de données d'un magasin

On suppose qu'une base de données contient 3 tables:
La table clients ayant 3 attributs:

- id: identifiant du client, de type Integer
- Nom: nom du client, de type Text
- Ville: nom de la ville de résidence du client, de type Text

La table produits ayant 4 attributs:

- id: identifiant du produit, de type Integer
- nom: nom du produit, de type Text
- prix: prix en euros d'une unité du produit, de type Decimal
- stock: nombre d'exemplaires en stock de type Integer

La table vente ayant 4 attributs:

- idCli: identifiant du client acheteur
- idProd: identifiant du produit vendu
- date: date de la vente, de type Date
- quantite: quantité de produits vendus, de type Integer

Q11 Écrire une requète donnant le nom des produits dont il reste moins de 5 exemplaires en stock.

Q12 Écrire une requète donnant la moyenne des prix des produits enregistrés dans la table produits (on ne demande pas la moyenne pondérée par le nombre dd'exemplaires en stock).

Q13 Écrire une requète qui donne le nom des clients ayant effectué en une fois l'achat d'un ou plusieurs exemplaires d'un même produit pour un montant total supérieur à 10 000 euros.

Q14 Écrire une requète qui donne le nom des clients ayant acheté au moins une fois du 'AAA' et au moins une fois du 'BBB'.

Q15 Écrire une requète qui donne pour chaque client, son nom et le nombre de ventes dans lesquelles il est acheteur.

Q16: Écrire une requète qui donne pour chaque client, son nom et le nombre de ventes dans lesquelles il est acheteur uniquement si le nombre de ces ventes est supérieur ou égal à 100.

Q17: Écrire une requète qui donne le nom du produit (ou des produits en cas d'ex-eaquos) qui sont les plus chers.

Q18: Écrire la table des nom1,nom2 de deux produits différents ayant le même prix, nom1 étant classé avant nom2 dans l'ordre alphabétique (le symbole $<$ permet de le vérifier).

Exercice 3 On considère un tableau à deux dimensions `T` (liste de `n` listes de longueurs `p`). On suppose que `T[i][j]` est entier. Écrire une fonction qui renvoie le maximum de $\sum_{i=0}^{n-1} T[i][f(i)]$ où f est une fonction de $[[0, n-1]]$ dans $[[0, p-1]]$ telle que :

$$\forall i \in [[1, n-1]],\ f(i) \in \{f(i-1)-1, f(i-1), f(i-1)+1\}.$$

INFORMATIQUE

Corrig du devoir surveillé n° 2

Exercice 1:

```
def suite(n,a):
    if n==0:
        return a
    else:
        return 4*suite(n-1,a)+2
```

Exercice 2:

```
def est_element(L,x):
    if len(L)=0:
        return False
    if L[0]==x or est_element(L[1:],x):
        return True
    else:
        return False

#ou
def est_element(L,x):
    if len(L)=0:
        return False
    elif L[0]==x:
        return True
    else:
        return est_element(L[1:],x)
```

Problème:

```
import numpy as np
#Q1
from math import sqrt
def distance(P1,P2):
    return sqrt((P2[0]-P1[0])**2+(P2[1]-P1[1])**2)

#Q2
def distances_mutuelles(L):
    n=len(L)
    M=np.zeros([n,n])
    for i in range(n):
        for j in range(n):
            M[i,j]=distance(L[i],L[j])
    return M
#Q3
def longueur(M,C):
    n,long=len(C),0
    for i in range(n-1):
        long=long+M[C[i],C[i+1]]
    long=long+M[C[n-1],C[0]]
    return long
#Q4
def meilleur_circuit(M,L):
    n,min,indice_min=len(L),0,-1
```

```
    for i in range(n):
        long=longueur(M,L[i])
        if long<min:
            min,indice_min=i,long
    return [L[indice_min],min]
#Q5
from random import randint
def modif_alea(C):
    n=len(C)
    # pour tirer au hasard i,j tels que i<=j, on les tire au hasard
    # et si on obtient i>j, on  les echange
    i,j=randint(0,n-1),randint(0,n-1)
    if j<i:
        i,j=j,i
    CC=[C[k] for k in range(j,i-1,-1)] #circuits de Cj a Ci dans l'ordre inverse
    return C[0:i]+CC+C[j+1:n]
#Q6
from math import exp
def proba(l0,l,k,T):
    if l<l0:
        return 1
    else:
        return 1-exp((l0-l)/k*T)
#Q7
def bernoulli(p):
    x=random()
    if x<p:
        return 1
    else:
        return 0
#Q8
def main(L,T0,k,x,N):
    M=distances_mutuelles(L)
    C=[i for i in range(len(L))]
    l0=longueur(M,C)
    T=T0
    for i in range(N):
        NC=modif_alea(C)
        l=longueur(M,NC)
        p=proba(l0,l,k,T)
        if bernoulli(p)==1:
            C,l0=NC,l
        T=(1-x)*T
    return C
#Q9
import matplotlib.pyplot as plt
def affiche_circuit(L,C):
    liste_x=[L[C[i][0] for i in range(len(C))]
    liste_y=[L[C[i][1] for i in range(len(C))]
    plt.plot(liste_x,liste_y]
```

Pour voir l'évolution du circuit, on insèrerait l'instruction

```
affiche circuit(L,C)
```

en dernière ligne de la boucle for dans `main`.

Q10 La complexité de la fonction longueur est O(n) où n est la longueur du circuit. Dans la fonction main, lorsqu'on passe d'un circuit C au circuit modifié NC, seuls deux liasons entre villes sont présentes dans C et ne sont pas dans NC: Si i et j sont les valeurs tirées au sort, la liaison entre les villes C[i-1] et C[i] dans C est remplacée dans NC par la liaison C[i-1],C[j] et la liaison entre les villes C[j] et C[j+1] dans C est remplacée dans NC par la liaison C[i],C[j+1]. On a donc NL=L-d(C[i-1],C[i])+d(C[i-1],C[j])-d(C[j],C[j+1])+d(C[i],C[j+1]). Ce qui donnera un calcul en temps constant (il faut réorganiser l'ensemble de la fonction si on veut profiter de cette amélioration).

Base de données:

Q11 Écrire une requète donnant les noms des produits dont il reste moins de 5 exemplaires en stock.
SELECT nom FROM produits WHERE stock>=5

Q12 Écrire une requète donnant la moyenne des prix des produits enregistrés dans la table produits.
SELECT AVG(prix) FROM produits

Q13 Écrire une requète qui donne le nom des clients ayant fait un achat supérieur à 10000 euros.
SELECT C.nom FROM clients AS C JOIN ventes AS V ON C.id=idCli JOIN produits ASP ON P.id=idProd WHERE prix*quantite>=10000

Q14 Écrire une requète qui donne le nom des clients ayant acheté au moins une fois du 'AAA' et au moins une fois du 'BBB'.
SELECT C.nom FROM clients AS C JOIN ventes AS V ON C.id=idCli JOIN produits ASP ON P.id=idProd WHERE P.nom='AAA'
INTERSECT
SELECT C.nom FROM clients AS C JOIN ventes AS V ON C.id=idCli JOIN produits ASP ON P.id=idProd WHERE P.nom='BBB'

Q15 Écrire une requète qui donne pour chaque client, son nom et le nombre de ventes dans lesquelles il est acheteur.
SELECT nom, COUNT(*) FROM clients JOIN ventes ON id=idCli GROUP BY id

Q16: Écrire une requète qui donne pour chaque client, son nom et le nombre de ventes dans lesquelles il est acheteur uniquement si le nombre de ces ventes est supérieur ou égal à 100.
SELECT nom, COUNT(*)AS NB FROM clients JOIN ventes ON id=idCli GROUP BY id HAVING NB>=100
Ou bien, sans HAVING:
SELECT nom,NB FROM (SELECT nom, COUNT(*)AS NB FROM clients JOIN ventes ON id=idCli GROUP BY id) WHERE NB>=100

Q17: Écrire une requète qui donne le nom du produit (ou des produits en cas d'ex-eaquos) qui sont les plus chers.
SELECT nom FROM produits WHERE prix=(SELECT MAX(prix) FROM produits)

Q18: Écrire une requète qui donne la table des noms de deux produits différents ayant le même prix, le premier nom étant classé avant le deuxième dans l'ordre alphabétique (le symbole < permet de le vérifier).
SELECT P1.nom,P2.nom FROM produits AS P1 JOIN produits AS P2 ON P1.prix=P2.prix WHERE P1.nom<P2.nom

Exercice 3: on utilise la fonction `max` de python (qu'on peut aussi programmer a part directement).

Le principe du programme est de calculer de proche en proche le maximum, pour $f(k)$ fixé, de $\sum_{i=k}^{n-1} T[i][f(i)]$ pour toutes les valeurs de $j = f(k)$ possibles en commençant par $k = n-2$ et en remontant les lignes une à une. Le tableau A contient toutes les valeurs calculées et sa première ligne contient les maximum, pour $f(0)$ fixé, de $\sum_{i=0}^{n-1} T[i][f(i)]$ pour toutes les valeurs de $j = f(k)$. Il suffit donc d'en prendre le maximum.

```
import copy
def maxi(T):
    n,p=len(T),len(T[0])
    A=copy.deepcopy(T)
    for i in range(n-2,-1,-1):
        for j in range(p):
            if j==0:
                x2,x3=T[i][j]+A[i+1][j],T[i][j]+A[i+1][j+1]
                A[i][j]=max([x2,x3])
            elif j==p-1:
                x1,x2=T[i][j]+A[i+1][j-1],T[i][j]+A[i+1][j]
                A[i][j]=max([x1,x2])
            else:
                x1,x2,x3=T[i][j]+A[i+1][j-1],T[i][j]+A[i+1][j],T[i][j]+A[i+1][j+1]
                A[i][j]=max([x1,x2,x3])
    return max(A[0])
```