Interrogation d'informatique n°6. *Barème sur 24 pts*

Exercice A. Partitionnement optimal en dimension 1

Soit un ensemble $E = \{x_0, ..., x_{N-1}\}$ de N réels codé en PYTHON par la **liste triée** **de ses éléments**.

On cherche à déterminer une partition $\mathcal{P} = \{C_0, ..., C_{K-1}\}$ de $[\![0, N-1]\!]$ en $K \leq N$ parties non vides (appelées classes) de sorte à minimiser le score de la partition défini par

$$S(\mathcal{P}) = \sum_{i=0}^{K-1} \sum_{j \in C_i} (x_j - \mu_i)^2$$

où μ_i est la valeur moyenne des réels appartenant à la classe C_i.

Par exemple, pour $E = \{1, 2, 3, 5, 8, 10, 14, 15, 18\}$ et $K = 3$, la partition optimale est définie par les classes $C_0 = \{0, 1, 2, 3\}$, $C_1 = \{4, 5\}$ et $C_2 = \{6, 7, 8\}$.

On peut naturellement **se limiter aux partitions formées de classes formées d'éléments consécutifs** et on peut prendre les classes par ordre croissant : on suppose donc

$$\forall i < i' \ , \ \forall (j, j') \in C_i \times C_{i'} \ , \ x_j \leq x'_j$$

On représente une partition $\mathcal{P} = \{C_0, ..., C_{K-1}\}$ de $[\![0, N-1]\!]$ en K classes par une liste P de taille K telle que $P[i]$ est le plus petit élément de C_i.
Avec l'hypothèse faite précédemment sur les C_i, la liste P est triée, et $P[0]$ vaut toujours 0.

Par exemple, la partition $\{\{0, 1, 2, 3\}, \{4, 5\}, \{6, 7, 8\}\}$ est codée par `[0,4,6]`.

Partie I. Fonctions élémentaires

Pour $E = [x_0, ..., x_{N-1}]$ et $0 \leq i < j \leq N$, on définit la distance quadratique à la moyenne par

$$D(i, j) = \sum_{k=i}^{j-1} (x_k - \mu)^2 \text{ , où } \mu = \frac{1}{j-i} \sum_{k=i}^{j-1} x_k$$

1) [0.5 pt] Écrire une fonction `min(L)` qui étant donnée une liste non vide L de réels renvoie sa valeur minimale.

2) [0.5 pt] Écrire une fonction `moyenne(E,i,j)` qui étant donnés une liste E de N valeurs et deux indices $0 \leq i < j \leq N$, renvoie la moyenne des éléments x_k où $i \leq k < j$.

3) [1 pt] Écrire une fonction `distance(E,i,j)` qui étant donnés une liste E de N valeurs et deux indices $0 \leq i < j \leq N$, calcule et renvoie $D(i, j)$.

On demande une fonction de complexité $O(j - i)$.

4) [1.5 pt] Écrire une fonction `score(E,P)` qui étant données une liste triée E de longueur N et une partition P de $[\![0, N-1]\!]$, renvoie le score $S(P)$.

Partie II. Méthode gloutonne de "clustering" hiérarchique ascendante

On cherche dans cette sous-partie à calculer une solution non nécessairement optimale par une approche gloutonne dite « hiérarchique ascendante » ou CHA.

Pour partitionner E en K classes, l'idée est la suivante :

– on crée d'abord N classes, chacune correspondant à un singleton d'un élément de E

– tant que le nombre de classes est $> K$, on fusionne les deux classes les plus proches, c'est-à-dire celles dont les moyennes sont les plus proches. En cas d'égalité, on fusionne celles qui ont les moyennes les plus basses.

5) [1.5 pt] Écrire une fonction `classes_plus_proches(E,P)` qui étant donnés un ensemble E et une partition P de taille K, renvoie un indice i_{opt} compris entre 0 et $K-2$ tel que les classes $C_{i_{opt}}$ et $C_{i_{opt}+1}$ sont les classes les plus proches au sens défini ci-dessus.

6) [1 pt] Écrire une fonction `fusion_classes(P,iopt)` qui renvoie une partition de taille $K-1$ obtenue en fusionnant les classes d'indices i_{opt} et i_{opt+1}.

7) [1.5 pt] En déduire une fonction `CHA(E,K)` qui calcule et renvoie une partition de taille K d'un ensemble E selon l'algorithme de clustering hiérarchique ascendant.

Partie III. Solution optimale en programmation dynamique

Pour $0 \leq n \leq N$ et $1 \leq k \leq K$, on note $s(n,k)$ le score minimal possible d'une partition de $\{x_0, ..., x_{n-1}\}$ en k classes (non vides).

8) [0.5 pt] Que vaut $s(n,k)$ lorsque $k = 1$? On exprimera sans justification le résultat à l'aide de la fonction D.

9) [1 pt] On vérifie (*admis ici*) que pour tous $n > 0$ et $k \geq 2$, on a

$$s(n,k) = \min_{k-1 \leq i \leq n-1} (s(i,k-1) + D(i,n))$$

Indiquer seulement ce que représente i dans cette expression et la raison pour laquelle on prend $i \geq k-1$.

10) [2 pts] En déduire une fonction `clustering_dynamique(E,K)` qui calcule efficacement le score minimal possible d'une partition de E en K classes non vides. On utilisera :
- un dictionnaire `dico` initialement vide dont les clés sont les couples `(n,k)` et les valeurs les réels `s(n,k)`
- une procédure-fonction récursive auxiliaire `aux(E,n,k,dico)` qui met à jour `dico` et renvoie `s(n,k)`.

11) [1 pt] Donner sans justification, en fonction de N et K, la complexité temporelle de la fonction précédente.

Exercice B. Noyau et numérotation 0-1 de Sprague-Grundy

Partie I. Noyau

Considérons un graphe orienté acyclique $G = (S, A)$, c'est-à-dire sans cycle.

Remarque : Dans un graphe acyclique, il existe nécessairement des sommets sans successeur.

Définition :

- Une partie N de S est **stable** si tout sommet de N n'a aucun successeur dans N.

- Une partie N de S est **absorbante** si tout sommet de $S \setminus N$ possède au moins un successeur dans N.

Un noyau est une partie à la fois stable et absorbante. On se propose de prouver :

Théorème : **Tout graphe orienté et acyclique possède un unique noyau**.

Pour calculer le noyau N, on peut utiliser (*admis ici*) l'algorithme suivant :

- On part d'une liste vide représentant le noyau

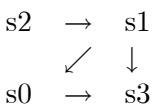
- On cherche **un sommet s sans successeur**, on l'ajoute au noyau

- On supprime dans le graphe **à la fois le sommet s et ses prédécesseurs** (et les arêtes associées)

- On **recommence** tant qu'il reste au moins un sommet, c'est-à-dire tant que le graphe n'est pas vide.

Ainsi, le noyau est défini comme la liste des sommets sans successeur obtenus au cours de l'algorithme.

On code en PYTHON un graphe par un dictionnaire `d` : les clés sont les sommets et la valeur d'une clé est la liste des successeurs de ce sommet. Par exemple, le graphe



est codé par `d = { s0 : [s3] , s1 : [s0,s3] , s2 : [s1] , s3 : [] }`

Le noyau est $N = \{s_2, s_3\}$. Les éléments s_0 et s_1 admettent au moins un successeur dans N.

1) [0.5 pt] Écrire une fonction `sansSuccesseur(d)` qui renvoie un sommet sans successeur.

2) [1 pt] Écrire une fonction `transpose(d)` qui étant donné un graphe codé par le dictionnaire d renvoie le dictionnaire codant le graphe transposé (obtenu en inversant les arêtes du graphe).

Par exemple, si `d` est le dictionnaire du graphe donné en exemple ci-dessus, `transpose(d)` renvoie le dictionnaire `{ s0 : [s1] , s1 : [s2] , s2 : [] , s3 : [s0,s1] }`.

3) [1.5 pt] Écrire une fonction `supprimeSommet(d,s)` qui étant donné un graphe codé par le dictionnaire d et un sommet s du graphe renvoie le dictionnaire codant le graphe obtenu **en supprimant dans le graphe le sommet s et ses prédécesseurs**, ainsi que les arêtes qui admettent un de ces sommets comme extrémité.

Avec l'exemple précédent, `supprimeSommet(d,s3)` renvoie le dictionnaire `{ s2 : [] }`.

Remarque : On pourra utiliser le graphe transposé pour construire la liste L comprenant s et ses prédecesseurs.
On pourra aussi utiliser `(s in L)` pour tester l'appartenance de s à la liste L :
On peut ainsi commencer par : `dT = transpose(d) ; L = dT[s] ; L.append(s)`

4) [1 pt] En déduire une fonction `noyau(d)` qui renvoie la liste des sommets constituant le noyau N en utilisant l'algorithme décrit précédemment.

Partie II. Nouvel An à Marienbad

Le jeu de Marienbad est un jeu avec plusieurs tas d'allumettes et deux joueurs jouant à tour de rôle. La configuration initiale est choisie aléatoirement. A chaque étape, le joueur choisit un tas et en retire tout ou partie, c'est-à-dire qu'il retire du tas au moins une allumette et qu'il peut retirer jusqu'à la totalité du tas choisi.

Le perdant est celui qui retire la dernière allumette du dernier tas.

Une position du jeu est codé par la liste L des **cardinaux des tas non vides classés par ordre décroissant**. On considère dans la suite qu'**une position est associée à une liste L non vide** (autrement dit, un joueur perd s'il ne peut plus jouer).

5) [1.5 pt] Écrire une fonction `retrait(L:[int],j:int)` qui étant donnée une liste L décroissante de n entiers non nuls et un entier $0 \leq j < n$, renvoie la liste obtenue en retranchant 1 à $L[j]$ et en reclassant la liste (si la nouvelle valeur de $L[j]$ n'est pas nulle) ou en supprimant $L[j]$ (si la nouvelle valeur de $L[j]$ est nulle).

Exemples : `retrait([5,2,2,1],1)` renvoie `[5,2,1,1]`, et `retrait([5,2,2,1],3)` renvoie `[5,2,2]`.

6) [1.5 pt] Écrire une fonction `successeurs(L:[int]) -> [[int]]` qui étant donné une position du jeu (c'est-à-dire codé par une liste d'entiers non nuls) renvoie la liste de ses successeurs, c'est-à-dire la liste de toutes les positions accessibles en un coup.

7) [1.5 pt] En déduire une fonction `marienbad(L)` qui étant donné une position initiale `L` renvoie le dictionnaire représentant le graphe du jeu : les sommets sont les positions possibles de jeu accessibles à partir de la position initiale et une arête relie deux états ssi l'un est un successeur de l'autre.

On utilisera une pile représentant la liste des positions à traiter. Par ailleurs, en principe, les clés d'un dictionnaire ne doivent pas être mutables, mais on s'autorise ici cette situation.

Partie III. Positions et stratégies gagnantes dans le jeu de Marienbad

On note N le noyau du graphe du jeu, et $R = S \setminus N$ le complémentaire.
On note T l'ensemble des sommets terminaux, c'est-à-dire sans successeur.

On a ainsi $\begin{cases} x \in N \Rightarrow \forall y \in G[x],\ y \in R \\ x \in R \Rightarrow \exists\, y \in G[x],\ y \in N \end{cases}$

Remarque : On a nécessairement $T \subset N$, car les sommets $x \in R$ admettent au moins un successeur.

Les joueurs A et B jouent à tour de rôle. Si le joueur A se trouve dans une position x, son coup correspond au choix d'une position $y \in G[x]$. Le joueur B se trouve alors en position y. Une partie est donc une succession de positions qui correspond à un chemin dans le graphe. Le joueur qui arrive dans une position terminale (c'est-à-dire qui ne peut pas jouer) a perdu.

Une stratégie pour un joueur est une application $\delta : S \setminus T \to S$ telle que $\delta(x) \in G[x]$ pour tout x.
Autrement dit, le joueur décide à l'avance quel coup $x \to \delta(x)$ il jouera selon la position x où il se trouve.

8) a) [1 pt] Montrer que les positions $x \in R$ sont gagnantes, c'est-à-dire qu'il existe pour le joueur A une stratégie δ telle que s'il se trouve en une position $x \in R$, il est sûr de gagner quels que soient les coups de l'autre joueur.

b) [0.5 pt] Montrer que les positions $x \in N$ sont perdantes : si le joueur A se trouve en une position $x \in N$, il est sûr de perdre si l'autre joueur adopte une bonne stratégie.

Partie IV. Numérotation 0-1 de Sprague-Grundy

Soit $G = (S, A)$ un graphe acyclique.

On définit une fonction $f : S \to \{0, 1\}$ définie par induction de la façon suivante :
$f(s)$ vaut 1 si et seulement si s admet au moins *un* successeur t vérifiant $f(t) = 0$.

En particulier, lorsque s n'a pas de successeur, alors $f(s)$ vaut 0.

9) [1 pt] Écrire une fonction `sprague(d)` qui étant un dictionnaire codant un graphe acylique G renvoie un dictionnaire `dS` dont les couples (*clé,valeur*) sont les $(s, f(s))$, où $s \in S$. On utilisera un parcours en profondeur.

10) [1 pt] Démontrer la propriété : *Tout graphe acyclique admet un et un seul noyau.*