Interrogation d'informatique n°4. *Barème sur 23 pts*

La fonction `min(a,b)` qui prend deux arguments (flottants ou entiers) peut être utilisée.

Exercice A. Plus courts-chemins dans un graphe pondéré : algorithme Floyd-Warshall

On considère un graphe orienté pondéré $G = (S, A, \omega)$, où S est l'ensemble des sommets, A l'ensemble (x, y) des arêtes $x \to y$ et ω la fonction donnant **le poids de chaque arête**.

On a ainsi $A \subset S \times S$ et $\omega : A \to \mathbb{R}$. On note $n = \operatorname{card} S$ et $m = \operatorname{card} A$.
Lorsque $(x, y) \in S \times S$ et que (x, y) n'est pas une arête, on pose $\omega(x, y) = +\infty$.

Un chemin Γ est une suite d'arêtes $x_0 \to x_1 \to ... \to x_p$ et le poids d'un chemin est la somme des poids des arêtes composant ce chemin, c'est-à-dire $\sum_{k=1}^{p} \omega(x_{k-1}, x_k)$.

Sous certaines conditions, l'algorithme Floyd-Warshall calcule, pour chaque paire de sommets d'un graphe, la distance entre les deux sommets. La distance entre deux sommets est définie par le poids minimal d'un chemin entre les deux sommets, s'il en existe au moins un, et $+\infty$ sinon.

La propriété de sous-optimalité est vérifée : si $x_0 \to x_1 \to ... \to x_{p-1} \to x_p$ est un chemin entre x_0 et x_p de coût minimal, alors chaque sous-chemin $x_i \to x_{i+1} \to ... \to x_{j-1} \to x_j$ est optimal, c'est-à-dire un chemin entre x_i et x_j de coût minimal.

On suppose que le graphe n'a **aucune boucle de poids strictement négatif** :
Autrement dit, pour toute boucle $x_0 \to x_1 \to ... \to x_p = x_0$, on a $\sum_{k=1}^{p} \omega(x_{k-1}, x_k) \geq 0$

1) [1.5 pt] Justifier brièvement que la distance $d(x, y)$ est bien définie, à valeurs dans $\mathbb{R} \cup \{+\infty\}$.

Indication : Faire intervenir l'ensemble des chemins **réduits**, c'est-à-dire les chemins ne passant pas deux fois par un même sommet (autrement dit, les chemins réduits sont les chemins sans boucle).

Dans la suite, on suppose les sommets du graphe numérotés de 0 **à** $n-1$**,** c'est-à-dire $S = \{0, 1, 2, ..., n-1\}$, et le graphe est défini par la liste `G` des listes d'adjacences pondérées : pour tout $0 \leq i < n$,

`G[i]` est la liste des couples $(j, \omega(i, j))$, où j décrit l'ensemble des successeurs de i.

Pour tout $0 \leq k < n$, on note D_k le tableau double telle que $D_k[i][j]$ est la longueur du plus court chemin entre les sommets i et j parmi les chemins **dont les sommets intermédiaires appartiennent à** $\{0, 1, ..., k-1\}$.

En particulier, on a $D_0[i, j] = \omega(i, j)$ si $(i, j) \in A$ et $D_0[i, j] = +\infty$ sinon.

2) [2 pts] Soit $0 \leq k < n$. Justifier que $D_{k+1}[i][j] = \min(\ D_k[i][j]\ ,\ D_k[i][k] + D_k[k][j]\ )$

3) [3 pts] Écrire une fonction `distances(G)` qui prend un graphe représenté par la liste `G` des listes d'adjacences et renvoie le tableau double D des distances entre les sommets.

On utilisera ici une méthode **itérative** (de "bas en haut"), c'est-à-dire une boucle `for` sur k.

Indication : La matrice D est ici définie par une liste de listes, qu'on pourra initialiser par :

```
D = [ [ float("inf") for i in range(n)] for j in range(n) ]
```

On utilisera une boucle `for` de sorte que `D` soit égal à D_k lors de l'itération d'indice k.
On supposera que la commande `D.copy()` effectue une copie profonde de `D`.

Rappels : On utilise `float("inf")` pour coder $+\infty$, pour lequel les opérations `+` et `min` sont bien définies.

4) [2 pts] Préciser la complexité de cet algorithme.

5) [3 pts] Écrire une fonction `chemin(D,i,j)` qui étant donnée la matrice des distances D obtenue au 3) renvoie en temps $O(n^2)$ un chemin de coût minimal reliant i à j (codé par une liste).

Exercice B. Optimisation du produit matriciel

Une matrice de type (n,p) est une matrice admettant n lignes et p colonnes. **Le calcul d'un produit AB de deux matrices A et B de types respectifs (n,p) et (p,q) nécessite npq opérations élémentaires.**

1) [1 pt] Écrire une fonction `ind_min(L)` qui étant donnée une liste L d'entiers non vide renvoie le plus petit indice i tel que $L[i] = \min(L)$.

2) [1.5 pt] Pour calculer le produit de matrices A, B et C, on peut effectuer $(AB)C$ ou $A(BC)$.

Montrer par un exemple que le nombre d'opérations utilisées n'est pas le même selon le choix effectué.

3) [3.5 pts] Soit $P = [p_0, p_1, ..., p_n]$ une liste de $(n+1)$ entiers non nuls.

Soit $(A_1, ..., A_n)$ une suite de matrices telles que A_k est de type (p_{k-1}, p_k) pour tout $k \in [\![1, n]\!]$.

On souhaite calculer le produit $A_1A_2....A_n$ en effectuant $(n-1)$ produits de deux matrices.

On cherche à effectuer le minimum d'opérations élémentaires possibles.

Écrire une fonction `minimum(P)` qui prend en argument la liste P et renvoie le nombre minimum d'opérations.

On utilisera le principe de la programmation dynamique :

Le calcul de $A_i...A_j$ se ramène à effectuer un produit MN, où $M = A_i...A_k$ et $N = A_{k+1}...A_j$ sont supposés connus, avec $i \leq k \leq j-1$. On utilisera **une fonction auxiliaire récursive** `aux` qui prend en arguments des entiers i et j, avec $1 \leq i \leq j \leq n$, et
- **d'une part renvoie** le nombre minimum d'opérations nécessaires pour calculer $A_i...A_j$
- **d'autre part met à jour** un dictionnaire `d` de mémoïsation en définissant `d[(i,j)]` s'il ne l'est déjà**.**

Ainsi, un appel à `aux(i,j)` permet à la fois d'obtenir la valeur souhaitée et de mettre à jour le dictionnaire.

4) [2 pts] Préciser la complexité de `minimum(P)` exprimée en fonction de n.

5) [2.5 pts] Pour indiquer les opérations effectuées, on utilise une formule parenthésée.

On représente chaque parenthèse ouvrante par un `"1"` et une parenthèse fermante par un `"-1"`.

Exemples : Le produit $((A_1A_2)(A_3(A_4A_5)))$ est codé par la liste d'entiers `[1,1,-1,1,1,-1,-1,-1]`.
Le produit $(A_1(A_2(A_3(A_4A_5))))$ est codé par la liste `[1,1,1,1,-1,-1,-1,-1]`.
(A_1A_2) et A_1 sont codés par `[1,-1]` et `[]`. Il y a toujours $(n-1)$ couples de parenthèses.

Écrire une fonction `solution(P)` qui renvoie une liste représentant une configuration optimale.

Suggestion : On pourra utiliser une fonction auxiliaire qui renvoie un couple.

6) [1 pt] Montrer que ce problème d'optimisation se ramène à la recherche d'un plus-court chemin dans un graphe (où chaque arête est pondérée positivement), qu'on explicitera sans justification.

Suggestion : Associer à chaque sommet une certaine liste $[i_1, ..., i_p]$, où $0 < i_1 < i_2 < ... < i_p < n$.

7) *Question supplémentaire* ; Écrire une fonction `c(n)` qui renvoie le nombre c_n de parenthésages contenant exactement n parenthèses. Par exemple, $c_2 = 2$ car les 2-parenthésages sont `[1,1,-1,-1]` et `[1,-1,1,-1]`.