Programmation 2 Spring, 2026

TD 8: Recursion

1. Écrivez une fonction récursive `mypower` qui calcule a^b pour a et b donnés, où b est un entier positif.

   N'utilisez pas l'opérateur `**`. Vous n'avez pas le droit d'utiliser des boucles ou toute autre fonction.

2. Écrivez une fonction récursive `mymax` qui renvoie le maximum d'une liste.

   Vous n'avez pas le droit d'utiliser des boucles ou toute autre fonction.

3. Écrivez une fonction récursive `evenbeforeodd1` qui prend en paramètre une liste d'entiers `A` et retourne une nouvelle liste contenant les mêmes éléments que `A`, réorganisés de telle sorte que :

   - tous les nombres pairs apparaissent avant tous les nombres impairs,

   La fonction ne doit pas modifier la liste originale `A` et doit être implémentée de manière récursive.

   Vous n'avez pas le droit d'utiliser des boucles.

4. Écrivez une fonction récursive `evenbeforeodd2` qui prend en paramètre une liste d'entiers `A` et retourne deux listes `B` et `C` contenant respectivement les nombres pairs et impairs de `A`, réorganisés de telle sorte que :

   - l'ordre relatif des nombres pairs entre eux soit préservé dans `B`, et
   - l'ordre relatif des nombres impairs entre eux soit également préservé dans `C`.

   La fonction ne doit pas modifier la liste originale `A` et doit être implémentée de manière récursive.

   Vous n'avez pas le droit d'utiliser des boucles.

5. Écrivez une fonction Python récursive `reverse` qui prend une chaîne de caractères `s` et retourne son inverse.

   Vous n'avez pas le droit d'utiliser des boucles ou toute autre fonction.

   Exemple: Pour `s = "hello"`, il renvoie `"olleh"`.

6. Écrivez deux fonctions :

   - l'une appelée `findeven_loops` qui utilise l'itération, et
   - l'autre appelée `findeven_recursion` qui utilise la récursivité.

   Ces fonctions réalisent l'opération suivante :

   L'entrée de la fonction est une liste d'entiers `A`. La fonction renvoie une (nouvelle) liste contenant uniquement les entiers pairs.

   Exemple: Pour `A = [3,1,5,4,4,2,1,22,7]`, il renvoie `[4, 4, 2, 22]`.

7. Écrivez une fonction récursive `flattenAll` qui prend une liste `A` qui contient des entiers (qui pourrait être à l'intérieur d'autres listes contenues dans `A`), et renvoie une **liste aplatie** contenant tous les entiers de `A`.

   Notez que `A` peut contenir une liste de listes de listes, et ainsi de suite ...

Exemple: Pour `A = [2, [ [ [3,4], [4] ], 1 ], [6] ]`, il renvoie `[2,3,4,4,1,6]`.

8. Écrivez une fonction récursive `allSubsets` qui renvoie la liste de tous les sous-ensembles d'une liste d'entrée `A` (sans répéter aucun sous-ensemble).

   Rappelons que le nombre total de sous-ensembles de `A` est $2^{\texttt{len}(A)}$.

   Exemple: Pour `A = [2,5,8]`, il renvoie `[[], [2], [5], [8], [2,5], [2,8], [5,8], [2,5,8]]`.

9. Écrivez une fonction récursive `allPermutations` qui renverra une liste de toutes les permutations d'une liste d'entrée `A`.

   Rappelons que le nombre total des permutations de `A` est `len(A)!`.

   Exemple: Pour `A = [2,5,8]`, il renvoie `[[2,5,8], [2,8,5], [5,2,8], [5,8,2], [8,2,5], [8,5,2]]`.

10. (**TP question.**) On considère une piste circulaire comportant n stations-service disposées tout autour.

    Chaque station i, où $i \in [0, n-1]$, dispose d'une certaine quantité d'essence, notée `gas[i]`.

    Cette quantité peut varier d'une station à l'autre—certaines peuvent avoir très peu, voire pas d'essence du tout, tandis que d'autres peuvent en avoir un large excédent.

    Soit `cost[i]` la quantité d'essence nécessaire pour se rendre de la station i à la station $(i+1) \bmod n$.

    La quantité totale d'essence disponible, toutes stations confondues, est exactement suffisante pour effectuer un tour complet de la piste. Autrement dit,

    $$\sum_{i=0}^{n-1} \texttt{gas}[i] = \sum_{i=0}^{n-1} \texttt{cost}[i].$$

    Vous commencez votre trajet avec le réservoir vide. À chaque fois que vous atteignez une station, vous pouvez récupérer toute l'essence qui s'y trouve et l'ajouter à votre réservoir.

    Votre objectif est de choisir une station de départ telle qu'en suivant la piste dans l'ordre, vous ne tombiez jamais en panne d'essence et puissiez ainsi effectuer un tour complet (c'est-à-dire revenir à votre point de départ).

    Montrer qu'il existe toujours au moins une station de départ permettant d'effectuer le circuit complet sans jamais tomber en panne d'essence. Cette fois, faites une preuve par induction.

    Donnez aussi l'algorithme récursif correspondant, qui suit la preuve.

11. **(difficile)** Étant donnée une chaîne `s`, écrivez une fonction récursive `findPalindrome` qui renvoie le plus grand palindrome de `s`, où nous sommes autorisés à supprimer des caractères de `s`.

    Exemple: Pour `s = "abcdba"`, il renvoie `"abcba"` ou `"abdba"`.

12. **(difficile)** On vous donne une liste `A` de entiers non négatifs, pour un jeu suivante.

    Joueur 1 choisit un entier parmi les extrémités de la liste, suivi de Joueur 2, puis de Joueur 1, et ainsi de suite.

    Chaque fois qu'un joueur choisit un parmi, celui-ci ne sera plus disponible pour le joueur suivant et est supprimé de `A`.

    Le jeux continue jusqu'à ce que tous les entiers aient été choisis.

    Le joueur avec le score maximum gagne.

    Soit une liste d'entiers `A`, écrivez une fonction récursive `listGame` qui calcule la valeur du jeu pour Joueur 1. Autrement dit, elle renvoie le score de `Joueur 1` si `Joueur 1` joue parfaitement.