TP2 - Programmation dynamique (suite)

DM facultatif

La suite de ce TP fait l'objet d'un DM facultatif : aucun rendu (copie ou fichier Python) n'est attendu. En revanche, il est vivement conseillé de le travailler sérieusement pour réinvestir les stratégies de programmation dynamique, mais aussi les algorithmes gloutons. Le corrigé sera disponible dans une dizaine de jours.

Problème 2 : Location de skis

Récupérer le fichier `TP2_skis.py` mis à votre disposition dans votre dossier Commun ou sur la Dropbox. Il contient les variables globales et les entêtes des fonctions à écrire dans cet exercice.

Un magasin de ski possède en stock un ensemble de n paires de skis de différentes longueurs ℓ_i, $0 \leqslant i < n$. Il reçoit une demande de location de m paires pour des skieurs ayant donné leur taille t_j, $0 \leqslant j < m$, avec $m \leqslant n$. On dispose des listes **globales** `longueur_skis` et `taille_skieur` correspondantes, en centimètres, triées par ordre **croissant**. `n` et `m` sont aussi des variables globales.

Idéalement, on considère que dans un compromis plaisir-performance, l'optimal est atteint lorsque la taille du skieur égale la longueur de ses skis. Cependant le stock est fini et chaque longueur de ski est un multiple de 10 cm, la correspondance est donc rarement parfaite. De ce fait, on introduit une fonction $\varepsilon(i, j) = |\ell_i - t_j|$, mesurant l'erreur commise en donnant la paire i au skieur j.

☐ **Question 1.** Écrire une fonction `eps(i:int, j:int) -> float` qui calcule $\varepsilon(i, j)$.

Une **attribution** a est une liste d'entiers de taille m, telle que $a[j]$ soit l'indice dans `longueur_skis` de la paire attribuée au skieur j. Le propriétaire du magasin souhaite donc choisir l'attribution a_{opt} qui minimise le **score** S défini par

$$S(a) = \sum_{j=0}^{m-1} \varepsilon(a[j], j)$$

☐ **Question 2.** Écrire une fonction `score(a:[int]) -> float` qui calcule le score d'une attribution. Vérifier que le score de l'attribution empirique a_e : `[n*j//m for j in range (m)]` vaut 170.

1 - Un premier algorithme glouton

Une première idée est d'attribuer les paires de skis par taille croissante des skieurs, en distribuant au skieur i la paire j minimisant $\varepsilon(i, j)$ parmi les paires restantes.

☐ **Question 3.** Écrire une fonction `meilleure(j:int) -> int` prenant en argument l'indice j d'un skieur, et qui renvoie l'indice i dans `longueur_skis` d'une paire de ski la mieux adaptée à ce skieur.

☐ **Question 4.** On ne peut pas distribuer 2 fois la même paire de skis. on modifie donc `meilleure` en lui ajoutant un paramètre `disponible`, une liste de booléens de taille n, telle que `disponible[i]` vaut `True` si la paire `i` est disponible.

Écrire une fonction `meilleure_dispo(j:int, disponible:[bool]) -> int` qui renvoie l'indice i d'une paire de ski **disponible** optimale. On partira d'une copie du code de `meilleure` que l'on adaptera.

☐ **Question 5.** Écrire une fonction `glouton() -> [int]` qui renvoie l'attribution gloutonne, notée a_g. Justifier l'appelation d'algorithme glouton, et calculer le score de a_g. La solution a_g est-elle optimale ?

2 - Résolution par la programmation dynamique

Question P'1 (en DM, on admettra le résultat). On suppose que lors d'une attribution a, deux skieurs de tailles t et T, avec $t < T$, se voient attribués respectivement des skis de tailles L et ℓ, avec $L > \ell$. Montrer que si les skieurs échangent leurs skis, le score de l'attribution résultante ne peut être pire. On pourra faire une disjonction de cas, certains laissant le score inchangé.

On déduit de ce résultat que toute attribution optimale peut se mettre sous la forme d'une liste croissante d'indices, c'est-à-dire que $j < j' \Rightarrow a[j] < a[j']$. On limite donc la recherche de l'attribution optimale dans l'ensemble les listes croissantes.

Pour tout $i \in [\![0\,;n]\!]$ et tout $j \in [\![0\,;m]\!]$, on note $S[i,j]$ le score optimal associé au sous-problème réduit aux i plus petites paires de ski (selon l'ordre imposé dans `longueur_skis`) et aux j plus petits skieurs (selon l'ordre imposé dans `taille_skieur`). Les relations suivantes permettent de déterminer S :

$$S[i,0] = 0 \qquad \forall i \geqslant 0 \qquad (2)$$

$$S[0,j] = +\infty \qquad \forall j > 0 \qquad (3)$$

$$S[i,j] = \min\{\, S[i-1,j],\, S[i-1,j-1] + \varepsilon(i-1,j-1) \,\} \qquad \forall i > 0, \forall j > 0 \qquad (4)$$

Question P'2 Remplir la table des scores S suivante, pour $n = 3$ (petites) paires de skis et $m = 2$ (grands) skieurs.

	j	0	1	2
i	$\downarrow \ell_{i-1} \backslash t_{j-1} \rightarrow$		185	195
0				
1	160			
2	170			
3	180			

Question P'3 (en DM) Justifier, par une phrase en français, chacune des trois relations précédentes.

P'3

a) Approche itérative : calcul de bas en haut

On part d'un tableau Numpy global `S`, initialisé par `S = np.zeros(shape=(n+1,m+1))`. La syntaxe `S[i][j]` permet d'accéder au score $S[i,j]$.

Question 6. Écrire une fonction `remplissage() -> None` qui remplit le tableau `S` à l'aide des relations précédentes et ne renvoie rien. La valeur $+\infty$ est donnée par `np.inf`.

Question 7. En déduire le score optimal S_{opt}.

Question 8. Observer le contenu du tableau `S` après remplissage. En déduire un chemin à travers `S` permettant de reconstituer une attribution optimale.

Question 9. Écrire une fonction `attribution() -> [int]` qui renvoie une attribution optimale a_{o}.

b) Approche récursive naïve, puis par mémoïsation

Question 10. Écrire une fonction <u>récursive</u> `Sr(i:int, j:int) -> float` qui calcule naïvement $S[i,j]$ par les relations (1), (2) et (3).

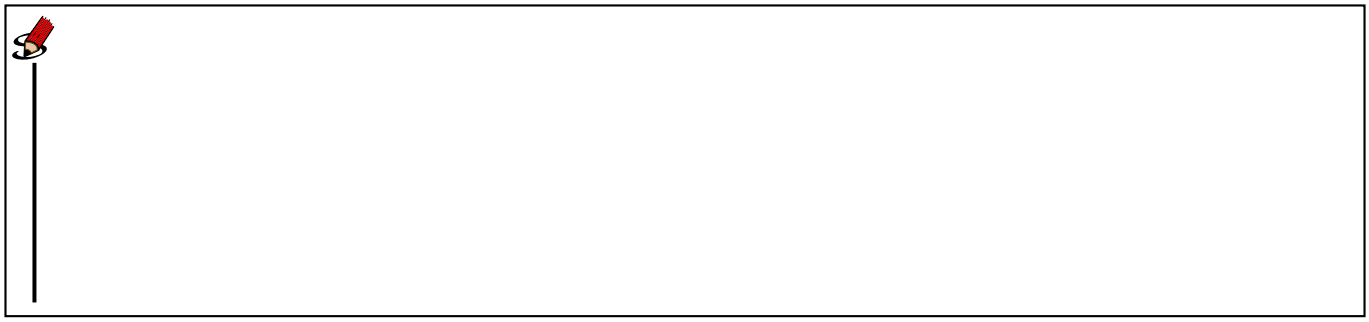
Question 11. Évaluer `Sr(i,j)` pour `i` $\in \{21, 23, 25\}$ et `j` $= 10$ dans la console, par la commande `%time Sr(i,j)`. Celle-ci évalue en parallèle le temps d'exécution. Que constate-t-on ? Peut-on accéder ainsi à $S[n,m]$?

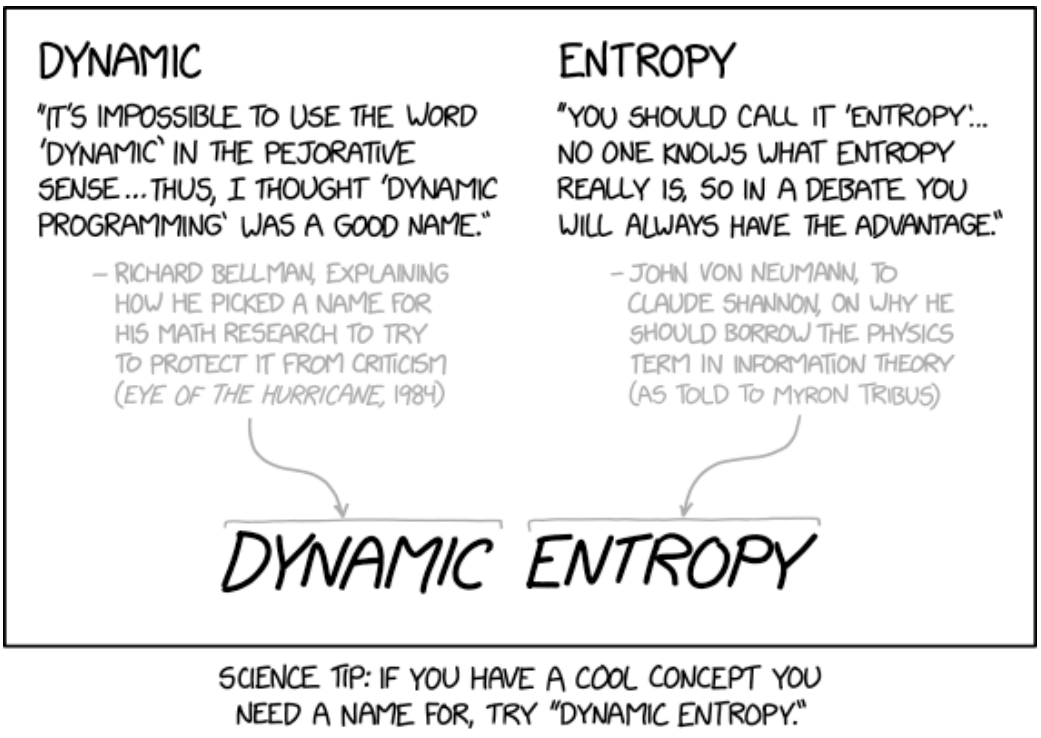
Pour pallier ce problème, il suffit de stocker pour chaque couple (i,j) la valeur de $S[i,j]$ dans un dictionnaire global `SR`, et de lancer un calcul récursif **uniquement** si elle ne s'y trouve pas déjà (mémoïsation). On initialisera ce dictionnaire par `SR = {}`.

Question 12. À partir d'une copie de la fonction `Sr`, écrire une fonction `Sr_mémo` qui met en œuvre cette mémoïsation. Retrouver $S[n,m]$ et conclure.

3 - Conclusion

Question 13. Comparer les solutions gloutonne a_g, empirique a_e et optimale a_o, en terme de complexité temporelle, de temps de développement, et de précision. Conclure.





SCIENCE TIP: IF YOU HAVE A COOL CONCEPT YOU NEED A NAME FOR, TRY "DYNAMIC ENTROPY."