SÉANCE 5

QUELQUES EXEMPLES DE PROBLÈMES D'OPTIMISATION

I - Acheter puis vendre !

On cherche à calculer le gain maximum possible à la Bourse sur une action pendant une période de n jours, en ne faisant qu'une opération d'achat ou de vente d'une seule action. On suppose que les cours quotidiens de cette action sont enregistrés dans une liste d'entiers naturels ($a_i \in \mathbb{N}$) de n éléments ($0 \leqslant i < n$).

Le *gain maximum* est la quantité suivante :

$$\text{gain} = \max_{0 \leqslant i \leqslant j < n} (a_j - a_i).$$

Question 1

En suivant textuellement la définition du gain, écrire une fonction `gain1(a)` qui renvoie, avec une complexité quadratique (par rapport à n), le gain maximal possible sur le cours représenté par le tableau a.

Pour tout $i \in [\![0, n-1]\!]$, on définit le gain courant maximum GC_i comme le gain maximum possible obtenu en vendant son action au temps i, c'est-à-dire :

$$\mathrm{GC}_i = \max_{0 \leqslant k \leqslant i} (a_i - a_k).$$

On vérifie alors que l'on a la relation de récurrence :

$$\mathrm{GC}_i = \max\left\{0 \,;\, \mathrm{GC}_{i-1} + a_i - a_{i-1}\right\}.$$

Question 2

Écrire une fonction `gain2(a)` qui renvoie, avec une complexité linéaire (par rapport à n), le gain maximum possible sur le cours représenté par le tableau a.

Exercice

Modifier les deux fonctions précédentes pour qu'elles renvoient également les deux dates i et j d'achat et de vente de l'action permettant d'obtenir le gain maximum sur le tableau a (avec $j - i$ minimum).

II - Rendu de monnaie et stratégie glouton

Un *algorithme glouton* est un algorithme qui suit le principe de faire, étape par étape, un choix localement optimal, dans l'espoir d'obtenir un résultat global optimal. Dans les cas où l'algorithme ne fournit pas systématiquement la solution optimale, il est appelé une heuristique gloutonne.

La méthode « usuelle »pour rendre la monnaie est celle de l'algorithme glouton : tant qu'il reste quelque chose à rendre, choisir la plus grosse pièce qu'on peut rendre (sans rendre trop). C'est un algorithme très simple et rapide, et on appelle canonique un système de pièces pour lequel cet algorithme donne une solution optimale quelle que soit la valeur à rendre.

On note S = $[p_0, p_1, \cdots p_i, \cdots p_{n-1}]$ un système de n pièces ou billets de valeurs p_i rangées par ordre croissant. On considère une somme r à rendre à l'aide du système S (il n'y a pas de limitation du nombre de chaque pièce). Une façon de constituer cette somme est donnée par le n-uplet $(m_0, m_1, \cdots m_i, \cdots m_{n-1})$ tel que :

$$r = \sum_{i=0}^{n-1} m_i \cdot p_i$$

On cherche à minimiser le nombre total $\sum_{i=0}^{n-1} m_i$ de pièces rendues.

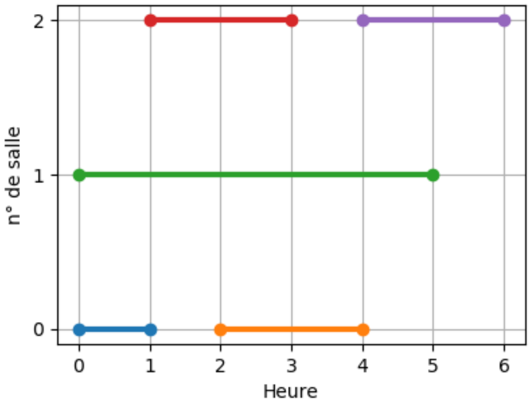
Exercice

Écrire la fonction `monnaie_a_rendre (somme_a_rendre, systeme)` qui renvoie une liste des nombres de pièces à rendre la plus proche de `somme_a_rendre` sans la dépasser, dans le système de monnaie `systeme`.
Par exemple, on doit avoir :

```
>>> monnaie_a_rendre(8, [1, 4, 6])
[2, 0, 1]
>>> monnaie_a_rendre(49, [1, 3, 6, 12, 24, 30])
[1, 0, 1, 1, 0, 1]
>>> monnaie_a_rendre(11, [2, 4, 6])
[0, 1, 1]
```

III - Gestion d'un emploi du temps de salles

L'objectif est de construire un emploi du temps d'allocation de salles.
Un emploi du temps est ici une liste de plannings de salles, chaque planning étant lui-même une liste de plages horaires, chaque plage étant une liste `[debut,fin]` de deux entiers naturels tels que debut < fin.



→ voir le document projeté (d'après le cours de M. Grenet de l'Université de Montpellier).

IV - Problème du sac à dos

On dispose d'un sac à dos dont la charge utile est limitée à un poids maximal $p_{\max}$ et de n objets $x_0, x_1, \dots, x_{n-1}$ possédant chacun un poids p_i et une valeur v_i. Le but est de remplir le sac en emportant la valeur maximale sans dépasser le poids limite.

⋄ Une première possibilité consiste à utiliser une stratégie de «force brute» : on liste toutes les possibilités et on prend la meilleure. S'il y a n objets, il y a alors 2^n possibilité. C'est inutilisable en pratique.

⋄ Une deuxième possibilité consiste à utiliser une stratégie «glouton». Par exemple en classant les objets selon le rapport $\frac{\text{valeur}}{\text{poids}}$ et en donnant systématiquement la priorité à l'objet présentant ce meilleur rapport.

⋄ Une autre stratégie, dite de «programmation dynamique» consiste à chercher une relation de récurrence comme dans le premier exemple.

On note $f(k,p)$ la valeur maximale obtenue avec les objets $x_0, \dots, x_k$ ne dépassant pas le poids p (on recherche donc ici $f(n-1, p_{\max})$. On cherche une relation de récurrence vérifiée par la fonction f.

- Si $k = 0$ alors il y a deux cas à distinguer pour $f(0,p)$:
  - ⋄ si $p_0 > p$ alors $f(0,p) = 0$,
  - ⋄ si $p_0 \leqslant p$ alors $f(0,p) = v_0$;
- pour $k \neq 0$, l'idée est de relier $f(k,p)$ à $f(k-1,p')$:
  - ⋄ si $p_k > p$ alors $f(k,p) = f(k-1,p)$,
  - ⋄ si $p_k \leqslant p$ alors $f(k,p) = \max\big(f(k-1,p), v_k + f(k-1,p-p_k)\big)$.