DIU Enseigner l'informatique au lycée
lundi 19 octobre (matin)

Complexité

Exercice 1 Le problème du *sous-tableau de somme maximale* consiste, étant donné un tableau d'entiers, à déterminer la somme maximale que l'on peut obtenir en ajoutant des éléments consécutifs de ce tableau. Ainsi, pour le tableau

```
[2, -7, 4, 3, -1, 5, -2]
```

la somme maximale est 11, obtenue pour le sous-tableau `[4, 3, -1, 5]`. Si le tableau ne contient que des entiers négatifs, la réponse est 0. Le fichier `stm.py` fourni contient quatre fonctions Python pour résoudre ce problème, appelées `stm1`, `stm2`, `stm3` et `stm4`.

Dans cet exercice, on se propose de comparer les performances de ces quatre solutions — sans nécessairement chercher à comprendre ces solutions. Dans ce qui suit, `n` désigne le nombre d'éléments du tableau.

1. En se servant de `time.perf_counter`, mesurer les performances de la fonction `stm1` sur des tableaux de différentes tailles, par exemple 100, 200, 400 et 800. Pour chaque taille `n`, on pourra lancer `stm1` sur un tableau aléatoire construit avec

   ```
   [randint(-n, n) for _ in range(n)]
   ```

2. Tracer la courbe de ces temps de calcul avec `matplotlib` (comme illustré dans les transparents).
3. Les résultats observés sont-ils cohérents avec ce que fait le code Python de la fonction `stm1` ?
4. De la même façon, mesurer les performances de la fonction `stm2` sur les mêmes valeurs de `n`. Tracer les performances de `stm1` et `stm2`, lancés sur les mêmes tableaux pour chaque valeur de `n`, sur un même graphique. (Deux appels à `plot` vont donner deux courbes de deux couleurs différentes.)
5. Qu'observe-t-on ? Quelle fonction paraît être la plus efficace ? Ceci est-il cohérent avec ce que font les codes Python des fonctions `stm1` et `stm2` ?
6. Procéder de même pour comparer les fonctions `stm2` et `stm3`. Cette fois, on pourra prendre des valeurs de `n` plus grandes, par exemple 100, 200, 400, ..., 6400 (on double à chaque fois, sans dépasser 10 000).
7. Là encore, qu'observe-t-on ? Cette fois, on ne cherchera pas à lire le code de la fonction `stm3` mais seulement à comparer empiriquement les performances de `stm3` par rapport à `stm2`.

8. Enfin, procéder de même pour comparer les fonctions `stm3` et `stm4`. Cette fois, on pourra prendre des valeurs de `n` jusqu'à 1 000 000.
9. À la lecture du code Python de `stm4`, que peut-on dire de sa complexité ?

□

Exercice 2 En supposant qu'on a écrit un tri par sélection qui prend un temps directement proportionnel à N^2 pour trier N valeurs et qu'il prend 6,8 secondes pour trier 16 000 valeurs, calculer le temps qu'il faudrait pour trier un million de valeurs avec ce même tri par sélection. □

Exercice 3 On considère ces deux versions d'une fonction Python qui teste si un tableau est trié :

```
def est_trie1(t):
    """teste si le tableau t est trié"""
    for i in range(0, len(t)):
        for j in range(i, len(t)):
            if t[i] > t[j]:
                return False
    return True

def est_trie2(t):
    """teste si le tableau t est trié"""
    for i in range(0, len(t) - 1):
        if t[i] > t[i+1]:
            return False
    return True
```

1. Sont-elles toutes les deux correctes ?
2. Si on double la taille du tableau, comment est modifié le temps de calcul pour chacune de ces deux fonctions ? Plus généralement, comparer les performances de ces deux fonctions et conclure.

□

Calculabilité

Exercice 4 (Simulation d'une machine de Turing) On se propose d'écrire un programme simulant l'exécution d'une machine de Turing. Fixons tout d'abord certains choix de représentation.

- Un état de la machine est identifié par une chaîne de caractères.
- Le symbole écrit dans une case du ruban est représenté par une chaîne de caractères.

- L'ensemble des règles de transition de la machine est matérialisé par un dictionnaire dont les clés sont les états. Consulter ce dictionnaire pour un état donné fournit l'ensemble des règles applicables à partir de cet état, à nouveau sous la forme d'un dictionnaire, dont les clés sont les symboles manipulés par la machine et les valeurs sont les actions associées.
- Les actions à effectuer à la lecture d'un symbole donné dans un état donné sont représentées par un triplet (e, d, s) où :
  - e est le symbole à écrire (une chaîne de caractères) ;
  - d est le déplacement éventuel : -1 pour aller à gauche, 0 pour ne pas bouger, 1 pour aller à droite ;
  - s est l'état suivant (une chaîne de caractères).

Une machine de Turing est donnée par un état de départ, un état final et un dictionnaire de règles de transition.

Lors de l'exécution de la machine, on fait évoluer une *configuration* formée d'un ruban mémoire, d'une identification de la position et de l'état courant de la machine. Pour représenter le ruban, on utilisera la classe `TaBiDir` (contenue dans le fichier `tabidir.py` fourni) qui réalise un tableau bidirectionnel. Ce tableau ne pouvant pas être infini, il ne contiendra que les cases effectivement utilisées, toutes les positions au-delà de ce qui est représenté dans le tableau étant supposées contenir le symbole •. Pour cela, le ruban est initialisé au début de l'exécution avec l'ensemble des cases représentant l'entrée, puis il est étendu à gauche ou à droite à chaque fois que la machine se déplace en dehors des limites courantes.

Écrire une fonction prenant en paramètres la description d'une machine de Turing M et une entrée e pour cette machine, et simulant l'exécution de la machine M sur l'entrée e. Il est possible d'utiliser une ou plusieurs classes pour structurer les données manipulées et le code. Tester ce simulateur avec la machine d'incrément de 1 donnée en exemple dans le cours. □

Exercice 5 Définir une machine de Turing qui, partant de l'extrémité gauche de son entrée, parcourt cette entrée à la recherche du symbole 0. Si le symbole est trouvé elle devra écrire 1 dans la case précédent l'entrée, et sinon elle devra y écrire 0. On supposera que l'entrée est composée de symboles 0 et 1 et bordée de symboles blancs •. □

Exercice 6 Supposons un ruban de machine de Turing dont une case contient le symbole 0, et tout le reste le symbole blanc •. Supposons que la machine se trouve à une certaine position du ruban, dont on ne sait pas si elle est à gauche ou à droite du symbole 0. Donner un ensemble de règles permettant, à coup sûr, d'arriver en temps fini à la position contenant un 0.

On pourra utiliser un symbole au choix, par exemple l'astérisque *, pour laisser une marque sur une case. Un objectif facultatif consiste à nettoyer le ruban de toutes les marques * utilisées avant de s'arrêter. □