TD 2 : Tableaux & Complexité

Info1.Algo1

2022-2023 Semestre Impair

Rappels

Tableaux

Un **tableau** permet de mémoriser une séquence d'éléments :

- Le **nombre d'éléments** du tableau est **fixe**.
- Tous les éléments du tableau sont de **même type**.
- Chaque élément est repéré par sa position dans le tableau, son indice.

```
# Creation d'un tableau d'entiers de taille 20 :
tableau = [0]*20
# Ecriture dans le tableau :
tableau[0] = int(input())
# Lecture et ecriture
tableau[1] = 2*tableau[0]
```

Matrices

Rappel 1 : Une matrice est représentée par un tableau de lignes (de longueur `nb_lignes`). Chaque ligne est elle-même un tableau (de longueur `nb_colonnes`). Toutes les lignes ont le même nombre de colonnes. Ainsi la variable :

```
matrice = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
```

représente la matrice :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

Rappel 2 : Afin de respecter les abstractions tableau et matrice, il est interdit d'utiliser les fonctions `append`, `insert`, `del`... au cours de cette séance.

Exercices : Matrices

Exercice 1 : parcours doubles

Qu'affichent les programmes suivant ?
1) a)

```
for i in range(3):
  for j in range(2):
      print(i,j)
```

b)

```
for i in range(3):
  for j in range(i,3):
      print(i,j)
```

c)

```
for i in range(3):
  for j in range(i):
      print(i,j)
```

2) a)

```
i,j = 0,0
while i<3:
  while j<2:
    print(i,j)
    j += 1
  i += 1
```

b)

```
i = 0
while i<3:
  j = 0
  while j<2:
    print(i,j)
    j += 1
  i += 1
```

Exercice 2 : représentation de matrices

1) On donne le code suivant :

```
matrice = [[1,6,5],[9,4,1],[7,0,3],[4,5,8],[9,1,0]]
print(matrice[0])
print(matrice[3])
print(matrice[0][2])
print(matrice[3][0])
```

a) Représenter graphiquement la variable `matrice`.
b) Quel est l'affichage obtenu ?
2) a) Représenter en Python la matrice suivante :

$$\begin{pmatrix} 7 & 5 & 3 \\ 2 & 9 & 11 \\ 6 & 8 & 1 \\ 10 & 12 & 4 \end{pmatrix}$$

b) Quelle expression Python permet d'accéder à l'élément égal à `8` ?
c) Quelle instruction Python permet de remplacer l'élément égal à `10` par la valeur `13` ?

Exercice 3 : dimensions

1) Pourquoi les variables

```
m1 = [[6,6,7],[7,8,9,0]]
m2 = [[6,8.0],[5,4]]
```

ne peuvent-elles pas représenter une matrice ?
2) Étant donnée une variable `matrice` respectant l'abstraction matrice, écrire les instructions permettant de connaître le nombre de ses lignes et de ses colonnes.
3) Étant une liste (non vide, de type `list`) de listes (non vides, de type `list`) d'entiers (de type `int`), écrire la fonction `est_matrice` qui vérifie si une telle liste de listes est effectivement une matrice.

Exercice 4 : création de matrice

On a exécuté le code suivant :

```
nb_lignes,nb_colonnes = 4,3
matrice = [[0]*nb_colonnes]*nb_lignes
matrice[3][2] = 567
print(matrice[3][1])
print(matrice[1][2])
```

Et on a obtenu l'affichage suivant :

```
0
567
```

a) Expliquer pourquoi on obtient cet affichage (on pourra s'aider du site **pythontutor.com)**.
b) Modifier le code donné de telle sorte que la variable matrice représente bien une matrice.

Exercice 5 : sommes

1) Écrire la fonction `sommes_lignes` qui accepte en paramètre une matrice et retourne un tableau composé des sommes de chacune de ses lignes.
Exemple : Si la matrice en paramètre est : $\begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{pmatrix}$
alors le tableau retourné est : $\begin{pmatrix} 6 & 18 & 30 \end{pmatrix}$
2) Écrire la fonction `sommes_colonnes` qui accepte en paramètre une matrice et retourne un tableau composé des sommes de chacune de ses colonnes.
Exemple : Si la matrice en paramètre est : $\begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{pmatrix}$
alors le tableau retourné est : $\begin{pmatrix} 15 & 18 & 21 \end{pmatrix}$

Exercices : Complexité

Exercice 6 : 3 fonctions...

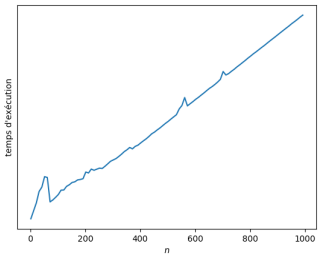
On donne 3 fonctions :

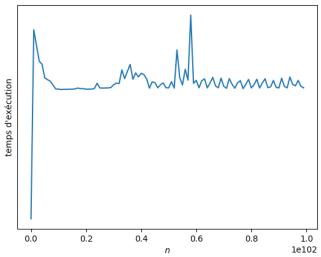
```
def f1(n):
  s = n
  for i in range(n):
    for j in range(n):
      s += i-j
  return s

def f2(n):
  s = 0
  while n>0:
    s += n
    n //= 2
  return s

def f3(n):
  s = 0
  while n>0:
    s += n
    n -= 1
  return s
```

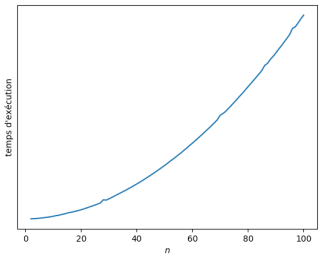
On donne 3 séries de mesures expérimentales des temps d'exécution (chaque série correspond à une fonction) :

```
n=10^1000 : 0.0027 sec
n=10^2000 : 0.0105 sec
n=10^3000 : 0.0240 sec

n=10^2 : 0.0005 sec
n=10^3 : 0.0549 sec
n=10^4 : 6.1432 sec

n=10^4 : 0.0006 sec
n=10^5 : 0.0067 sec
n=10^6 : 0.0660 sec
```

On donne enfin 3 courbes du temps d'exécution en fonction de n (les échelles sont linéaires) :







Associer à chaque fonction sa mesure expérimentale et la courbe représentative correspondante.

Exercice 7 : exponentiation rapide

Problème posé : On donne deux entiers naturels `a` et `n` et l'on souhaite `a**n` sans utiliser l'opérateur `**`.
On propose les algorithmes suivants :

```
def puissance_A(a,n):
  p = 1
  while n>0:
    p,n = p*a,n-1
  return p
```

et

```
def puissance_B(a,n):
  p = 1
  while n>0:
    if n%2==1:
      p,n = p*a,n-1
    a,n = a*a,n//2
  return p
```

On considère que l'entier `a` est une constante et on souhaite analyser la complexité de chacun des algorithmes en fonction de `n`.
1) Exprimer, en fonction de `n`, le nombre de boucles effectuées pour chacun de ces deux algorithmes.
2) Donner un ordre de grandeur du nombre de boucles effectuées dans les deux cas lorsque `n` vaut `1000000`.

Exercice 8 : plus grande somme

1) La fonction `somme_maximale` définie ci-dessous accepte en paramètre un tableau `tab` de taille `n` au moins égale à 2 et détermine la **plus grande somme de deux éléments d'indices distincts** de `tab`.

```
def somme_maximale(tab):
  n = len(tab)
  maximum = tab[0]+tab[1]
  for i in range(n):
    for j in range(i+1,n):
      if tab[i]+tab[j]>maximum:
        maximum = tab[i]+tab[j]
  return maximum
```

a) Exprimer en fonction de `n` le nombre de comparaisons effectuées dans cette fonction.
b) En déduire la complexité en temps de cet algorithme.
c) Une mesure expérimentale a donné les résultats suivants :

```
n=10^1 : 1.049041748046875e-05 sec
n=10^2 : 0.00037169456481933594 sec
n=10^3 : 0.03876376152038574 sec
n=10^4 : 3.3503222465515137 sec
n=10^5 : 336.06458735466003 sec
```

Comment évolue le temps d'exécution de l'appel `somme_maximale(tab)` lorsque la nombre `n` passé en paramètre est multiplié par `10` ? **Interpréter.**
2) On se propose d'améliorer la complexité en temps **en ne parcourant qu'une seule fois le tableau** `tab`. L'idée qui permet de réaliser cette amélioration est de déterminer au fur et à mesure de ce parcours la première et la deuxième plus grandes valeurs du tableau.
Modèle de solution : On effectue un parcours gauche-droite. À chaque nouvelle valeur rencontrée :

— Si elle remplace le maximum actuel alors le maximum actuel prend la place de la deuxième plus grande valeur.
— Sinon elle peut éventuellement prendre la place de la deuxième plus grande valeur.

a) Écrire la fonction `deux_maxima` qui accepte en paramètre un tableau `tab` et retourne un tuple `(max1,max2)`, où `max1` est la plus grande valeur du tableau (c'est-à-dire le maximum) et `max2` la deuxième plus grande valeur du tableau.
b) Ré-écrire la fonction `somme_maximale` avec un appel de la fonction `deux_maxima`.
Contraintes :
— N'effectuer qu'un seul parcours du tableau.
— Respecter l'abstraction tableau.
— Ne pas utiliser de fonctions élaborées de Python pour déterminer le maximum (`max`, `sorted` , ...).

Exercice 9 : complexité cachée

La fonction `calculer_indices` définie ci-dessous accepte en paramètre un tableau `valeurs` de taille `n` et contenant en un seul exemplaire toutes les valeurs entières de `0` à `n-1` (dans un ordre quelconque) et retourne le tableau `indices` des indices auxquels chaque valeur est située dans le tableau initial `valeurs`.

```
def calculer_indices(valeurs):
  n = len(valeurs)
  indices = [0]*n
  for valeur in range(n):
    indices[valeur] = valeurs.index(valeur)
  return indices
```

Exemple : si `valeurs=[4,3,0,5,2,1]` (ici `n=6`), le `0` est à l'indice 2, le `1` est à l'indice 5, etc. donc le tableau `indices` retourné est `[2,5,4,1,0,3]`.
1) Quel est, en fonction de `n`, le nombre d'itérations ?
2) On a mesuré expérimentalement les temps d'exécution pour différentes valeurs de `n` et l'on a obtenu les résultats suivants :

```
n=10^2 : 7.557868957519531e-05 sec
n=10^3 : 0.0061457157135009766 sec
n=10^4 : 0.6966583728790283 sec
```

Expliquer pourquoi ce résultat peut sembler incohérent.
3) Quel élément du code de la fonction pose ici problème ? Expliquer.
4) Ré-écrire la fonction `calculer_indices` afin de corriger ce problème.