DS1 /36

Exercice : Cours /14

1. Écrire une fonction **Python** calculant la première position du maximum d'une liste `L`.

- **1 pt : structure de fonction**
- **1 pt : initialisation**
- **1 pt : structure itérative**
- **2 pts : structure conditionnelle**

```
def maximum(A) :
    M = A[0]
    ind = 0
    for i in range(len(A)) :
        if A[i] > M :
            M = A[i]
            ind = i
    return ind
```

2. Tri fusion

a) Implémenter la fonction `fusion` qui prend en paramètre deux listes triées `L1` et `L2` et renvoie la liste `L` qui réalise la fusion des deux listes précédentes.

- **1 pt : initialisation**
- **1 pt : structure itérative**
- **2 pts : condition de la structure conditionnelle (dont 1 pt si condition pertinente mais cas extrêmaux omis)**
- **2 pts : contenu de la structure conditionnelle**

```
def fusion(L1, L2) :
    L = []
    i,j = 0,0
    m,n = len(L1), len(L2)
    for k in range(m + n) :
        if i < m and (j==n or L1[i] <= L2[j]):
            L = L + [L1[i]]
            i = i + 1
        else :
            L = L + [L2[j]]
            j = j + 1
    return L
```

b) Implémenter la fonction `tri_fusion` qui prend en paramètre une liste `L`, trie cette liste selon le principe du tri fusion et renvoie la liste obtenue.

- **1 pt : condition d'arrêt**
- **2 pts : récursivité**

```
def tri_fusion(L) :
    n = len(L)
    if n <= 1 :
        return L
    else :
        m = n // 2
        return fusion(tri_fusion(L[:m]), tri_fusion(L[m:]))
```

Problème : Trafic routier /22

Partie 1 : Préliminaires

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure **??**. Une file de longueur n est représentée par n cases. Une case peut contenir au plus une voiture. Les voitures présentes dans une file circulent toutes dans la même direction (sens des indices croissants, désigné par les flèches sur la Figure **??**) et sont indifférenciées.

1. Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.

- **1 pt :** `L[i]` $=$ $\begin{cases} \texttt{True} & \textbf{si la position i est occupée par une voiture} \\ \texttt{False} & \textbf{sinon} \end{cases}$

2. Donner une instruction **Python** permettant de définir une liste `A` représentant la file de voitures illustrée par la Figure **??**.

- **1 pt :** `A = [True, False] + 2 * [True] + 6 * [False] + [True]`

3. Soit `L` une liste représentant une file de longueur n. Soit `i` $\in [\![0, n-1]\!]$. Définir en **Python** la fonction `occupe` de paramètres `L` et `i` qui renvoie `True` lorsque la case d'indice `i` de la file est occupée par une voiture et `False` sinon.

- **1 pt**

```
def occupe(L, i) :
    return L[i]
```

4. Combien existe-t-il de files différentes de longueur n ? Justifier votre réponse.

- **1 pt : Il y a 2^n files de longueur n différentes.**
  0 pt en cas d'absence de justification

5. Écrire une fonction `egal` de paramètres `L1` et `L2` retournant un booléen permettant de savoir si deux listes `L1` et `L2` sont égales.

- **1 pt**

```
def egal(L1, L2) :
    return L1 == L2
```

6. Quelle est la complexité, dans le pire cas, de la fonction `egal` ? Autrement dit, combien de comparaisons de booléens cette fonction effectue-t-elle, au maximum ?

- **2 pts : La complexité, dans le pire cas, de la fonction `egal` est linéaire. Plus précisément, on effectue au plus n comparaisons pour des listes de longueurs n.**
  0 pt en cas d'absence de justification

7. Préciser le type d'objet renvoyé la fonction `egal`.

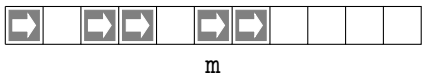
- **1 pt : La fonction `egal` renvoie un booléen (une variable de type `bool`).**

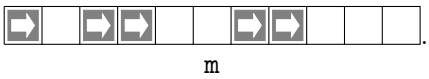
Partie 2 : Déplacement de voitures dans la file

8. Pour la liste `A` définie à la question ***2.***, que renvoie `avancer(avancer(A, False), True)` ?

- **1 pt : La liste renvoyée par `avancer(avancer(A, False), True)` sera donc :**
  `[True, False, True, False, True, True, False, False, False, False, False]`

9. On considère `L` une liste et `m` l'indice d'une case de cette liste ($\texttt{m} \in [\![0, \texttt{len(L)} - 1]\!]$). On s'intéresse à une étape partielle où seules les voitures situées sur la case d'indice `m` ou à droite de cette case peuvent avancer normalement, les autres voitures ne se déplaçant pas.

Par exemple, la file 

devient 

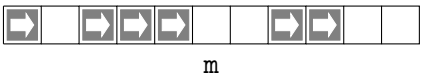
.

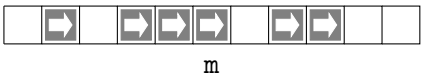
Définir en **Python** la fonction `avancer_fin` de paramètres `L` et `m` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier `L`.

- **1 pt : script**
- **1 pt : explications**

```
1  def avancer_fin(L, m) :
2      n = len(L)
3      return L[:m] + [False] + L[m:]
```

10. Soient `L` une liste, `b` un booléen et `m` l'indice d'une case inoccupée de cette liste. On considère une étape partielle où seules les voitures situées à gauche de la case d'indice `m` se déplacent, les autres voitures ne se déplacent pas. Le booléen `b` indique si une nouvelle voiture est introduite sur la case la plus à gauche.

Par exemple, la file 

devient 

lorsque aucune nouvelle voiture n'est introduite.

Définir en **Python** la fonction `avancer_debut` de paramètres `L`, `b` et `m` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier `L`.

- **1 pt : script**
- **1 pt : explications**

```
1  def avancer_debut(L, b, m) :
2      n = len(L)
3      return [b] + L[:m] + L[(m+1):]
```

11. On considère une liste `L` dont la case d'indice `m` > 0 est temporairement inaccessible et bloque l'avancée des voitures. Une voiture située immédiatement à gauche de la case d'indice `m` ne peut pas avancer. Les voitures situées sur les cases plus à gauche peuvent avancer, à moins d'être bloquées par une case occupée, les autres voitures ne se déplacent pas. Un booléen `b` indique si une nouvelle voiture est introduite lorsque cela est possible.

Par exemple, la file  devient  lorsque aucune nouvelle voiture n'est introduite.

Définir en **Python** la fonction `avancer_debut_bloque` de paramètres `L`, `b` et `m` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

- **4 pts : peu importe la version de l'algorithme**

```
def avancer_debut_bloque(L, b, m) :
    if m == 0 :
        return L
    elif occupe(L, m-1) :
        return avancer_debut_bloque(L, b, m-1)
    else :
        avancer_debut(L, b, m-1)
```

```
def avancer_debut_bloque(L, b, m) :
    i = m - 1
    while i >= 0 and occupe(L, i) :
        i = i - 1
    if i < 0 :
        return L
    else :
        return avancer_debut(L, b, i)
```

Partie 3 : Une étape de simulation à deux files

12. En utilisant le langage **Python**, définir la fonction `avancer_files` de paramètres `L1`, `b1`, `L2` et `b2` qui renvoie le résultat d'une étape de simulation sous la forme d'une liste de deux éléments notée [`R1`, `R2`] sans changer les listes `L1` et `L2`. Les booléens `b1` et `b2` indiquent respectivement si une nouvelle voiture est introduite dans les files `L1` et `L2`. Les listes `R1` et `R2` correspondent aux listes après déplacement.

- **1 pt : explications**
- **1 pt : initialisation**
- **2 pts : structure conditionnelle**

```
def avancer_files(L1, L2, b1, b2) :
    n = len(L1)
    m = (n - 1) // 2
    R1 = avancer(L1, b1)
    if occupe(R1, m) :
        L2bis = avancer_fin(L2, m)
        R2 = avancer_debut_bloque(L2bis, b2, m)
    else :
        R2 = avancer(L2, b2)
    return [R1, R2]
```

13. On considère les listes

$$\texttt{D} = [\texttt{False}, \texttt{True}, \texttt{False}, \texttt{True}, \texttt{False}]$$

$$\texttt{E} = [\texttt{False}, \texttt{True}, \texttt{True}, \texttt{False}, \texttt{False}]$$

Que renvoie l'appel `avancer_files(D, False, E, False)` ?

- **1 pt : L'appel `avancer_files(D, False, E, False)` renvoie donc la liste de listes :**

`[[False, False, True, False, True], [False, True, False, True, False]]`