TD 14 : les piles, les files et les files de priorité

Dans toute la suite, on cherchera à n'utiliser que les fonctions suivantes pour la manipulation des piles (qui seront modélisée par des listes)

In [1]:
```
#une fonction qui permet de savoir si la pile est vide
def pile_vide(p):
    """fonction qui renvoie True si la pile est vide, False dans le
    return p==[]
# fonction qui ajoute un élément sur la pile
def empile(p,x):
    "ajoute l'élément x à la fin"
    p.append(x)
# fonction qui retire l'élément en haut de la pile
def desempile(p):
    "renvoie le sommet de la pile p et retir ce sommet"
    if len(p)>0:
        return p.pop()
```

Pour les files, on utilisera la librairie collections

In [2]:
```
from collections import deque
#pour les files
file = deque([1,2,3])
file.append(4)
print(file)#deque([1,2,3,4])
file.popleft()
print(file)#deque([2,3,4])
```

```
deque([1, 2, 3, 4])
deque([2, 3, 4])
```

Pour les files de priorité, on rappelle les fonctions suivantes :

In [3]:
```
from queue import PriorityQueue

file_priorite=PriorityQueue()#initialisation
file_priorite.put((valeur,nom))#remplissage avec le 1e argument qui
print(file_priorite.empty())#renvoie True si vide
valeur,nom=file_priorite.get()#par défaut renvoie le couple associé
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-3-2da04c64ced1> in <module>
      2
      3 file_priorite=PriorityQueue()#initialisation
----> 4 file_priorite.put((valeur,nom))#remplissage avec le 1e argum
ent qui sera au classement prioritaire
      5 print(file_priorite.empty())#renvoie True si vide
      6 valeur,nom=file_priorite.get()#par défaut renvoie le couple
associé à la plus petite valeur val

NameError: name 'valeur' is not defined
```

Exercice 1: Inversion d'une pile

En utilisant les fonctions empile( ),pile_vide() et desempile( ), proposer une fonction inverser_pile( ) qui prend pour argument une pile p et qui renvoie une autre pile dont les éléments sont les éléments de p dans l'ordre inverse, p n'est pas modifiée à la fin.

In [3]:
```
def inverse(p):
    if pile_vide(p):
        return p
    pile_inter1=[]
    pile_inter2=[]
    while not pile_vide(p):
        valeur=desempile(p)
        empile(pile_inter1,valeur)
        empile(pile_inter2,valeur)
    while not pile_inter2 :
        empile(p,desempile(pile_inter2))
    return pile_inter1
```

In [4]:
```
p=[0,1,2,3,4]
print(inverse(p))
```

```
[4, 3, 2, 1, 0]
```

In [5]:
```
def inverse2(p):
    if pile_vide(p):
        return p
    pile_inter1=[]
    pile_inter2=p[:]
    while not pile_vide(pile_inter2):
        valeur=desempile(pile_inter2)
        empile(pile_inter1,valeur)
    return pile_inter1
p=[0,1,2,3,4]
inverse2(p)
```

Out[5]: [4, 3, 2, 1, 0]

Exercice 2 : désemplier

Ecrire une fonction *depileK* qui prend pour argument une pile p et un entier K et qui désempile p des K derniers éléments de p s'ils existent ou qui désempile entièrement p si len(p)<K. La fonction retourne la pile dépilée (en partie ou en totalité).

In [3]:
```
def depileK(p,K):
    if len(p)<=K :
        return []
    for i in range(K):
        desempile(p)
    return p
```

In [4]:
```
p=[9,8,7,6,5,4,3,2,1,0]
print(depileK(p,5))
```

```
[9, 8, 7, 6, 5]
```

Exercice 3 : Dépiler jusqu'à un élément E

Ecrire une fonction depileE( ) qui prend pour argument une pile p et un élément E et qui désempile p tant que l'élément E n'est pas rencontré ou que p n'est pas vide. La fonction retourne la pile dépilée avec E inclus

In [7]:
```
def depileE(p,E):
    while not pile_vide(p):
        valeur=desempile(p)
        if E==valeur:
            empile(p,E)
            return p
    return []
```

In [8]:
```
p=[9,8,7,6,5,4,3,2,1,0]
print(depileE(p,5))
```

```
[9, 8, 7, 6, 5]
```

Exercice 4 : Permutation

Ecrire une fonction permut( ) qui prend pour argument une pile p et qui permute le dernier élément avec l'avant dernier. La fonction retourne la pile ainsi modifiée (on suppose que p possède au moins 2 éléments).

In [20]:
```
def permut(p):
    der=desempile(p)
    avant_der=desempile(p)
    empile(p,der)
    empile(p,avant_der)
    return p

```

In [21]:
```
p=[9,8,7,6,5,4,3,2,1,0]
print(permut(p))
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 0, 1]
```

Exercice 5 : test de la parenthèse manquante

Ecrire un algorithme utilisant une pile et permettant de savoir si le nombre de parenthèses ouvrantes est égale au nombre de parenthèses fermantes. L'expression mathématique à tester est donnée dans une liste de chaîne de caractères. Par exemple, l'expression 3×(4+2) sera donnée sous la forme [" 3 "," x "," ("," 4 "," + "," 2 ",") "]. L'algorithme renvoie True, si la mise en paranthèse est correcte, False, sinon. L'idée est donc d'empiler à chaque parenthèse ouvrante rencontrée et de dépiler si une parenthèse fermante est rencontrée (en lisant l'expression de gauche à droite). Si l'écriture est bonne la pile est vide.

In [30]:
```
def test(liste):
    pile=[]
    for i in liste :
        if i =="(":
            empile(pile,1)
        if i==")" :
            if not pile_vide(pile):
                desempile(pile)
            else :
                return False
    if pile_vide(pile):
        return True
    else :
        return False
```

In [31]:
```
chaine =["3","x","(","4","+","2",")"]
print(test(chaine))
```

```
True
```

Exercice 6 : La calculette polonaise

L'écriture polonaise inversée ne nécessite pas de parenthèse, elle a été autrefois utilisée dans certaines calculatrices. Principe :

- Quand on rencontre un nombre, on ne fait rien
- Quand on rencontre un opérateur (+,-,*,/), on l'applique aux deux précédents nombres

$$7 + 6 \rightarrow 7\ 6 +$$

$$(10 + 5) * 3 \rightarrow 10\ 5 +\ 3 *$$

$$10 + 2 * 3 \rightarrow 10\ 2\ 3 *\ +$$

$$(2 + 8) * (6 + 11) \rightarrow 8\ 2 + 6\ 11 + *$$

Ecrire une fonction :

- qui prend en argument une liste contenant les éléments d'un calcul en écriture polonaise inversée. Par exemple $[2, 3, " + ", 4, " * "]$ pour (2+3)*4
- qui utilise une pile initialement vide (une liste vide).
- qui, pour chaque élément lue de la liste de gauche à droite :
  - si l'élément est un nombre, alors ce nombre est rajouté à la pile avec la fonction empile( )
  - si l'élément est une opération (*,/,+,-,), alors la pile est dépilée deux fois pour réaliser cette opération
- A la fin la pile ne contient plus qu'un seul élément, c'est le résultat du calcul

In [4]:
```
def calculette(L):
    pile=[]
    for i in L :
        if i == "+":
            a=desempile(pile)
            b=desempile(pile)
            resultat=a+b
            empile(pile,resultat)
        elif i == "-":
            a=desempile(pile)
            b=desempile(pile)
            resultat=b-a
            empile(pile,resultat)
        elif i == "*":
            a=desempile(pile)
            b=desempile(pile)
            resultat=a*b
            empile(pile,resultat)
        elif i == "/":
            a=desempile(pile)
            b=desempile(pile)
            resultat=b/a
            empile(pile,resultat)
        else :
            empile(pile,i)
    return pile
```

In [5]:
```
L=[2,3,"+",4,"*"]
print(calculette(L))

```

```
[20]
```

Exercice 8 : Création d'une file de priorité

Une file de priorité est une file qui donne accès au premier élément de la structure dont les éléments ont été classés selon un critère. Dans la suite, on va considérer des données du type tuple=(val,nom) dans la file de priorité (modélisée par une liste L). Nous allons chercher à classer ces tuples par ordre croissant de valeurs de val.

1)Ecrire une fonction insertion qui prend en arguments une liste L de nombres triés et un nombre x. Cette fonction renvoie une nouvelle liste en utilisant le mode suivant :

- Si x≤L[0] alors l'élément est placé en début de liste
- Si x≥L[-1] alors l'élément est placé en fin de liste
- Si x est dans la liste alors la dichotomie est utilisée pour insérer x dans L à la bonne position (juste après la valeur déjà présente).
- Si x n'est pas repéré dans la liste à l'issu de la recherche dichotomique, alors il est placé à la bonne position (entre deux éléments l'encadrant).

2) Ecrire une fonction put(t,L) qui prend en entrée un tuple t=(val,nom) et qui insert correctement t par une méthode dichotomique dans une file modélisée par une liste L.

3) Ecrire une fonction get(L) qui retourne et enlève le tuple prioritaire placé au début de L en utilisant la méthode pop().

In [8]:

```
def insertion(L,x):
    if x<=L[0]:
        L=[x]+L
        return L
    elif x>=L[-1]:
        L.append(x)
        return L
    else :
        d,f=0,len(L)-1
        while f>=d:
            m=(f+d)//2
            if L[m]==x:
                L=L[:m+1]+[x]+L[m+1:]
            elif L[m]>x:
                f=m-1
            else :
                d=m+1
    return L[:d]+[x]+L[d:]



def put(L,t):
    x=t[0]
    if x<=L[0][0]:
        L=[t]+L
        return L
    elif x>=L[-1][1]:
        L.append(t)
        return L
    else :
        d,f=0,len(L)-1
        while f>=d:
            m=(f+d)//2
            if L[m]==x:
                L=L[:m+1]+[t]+L[m+1:]
            elif L[m]>x:
                f=m-1
            else :
                d=m+1
    return L[:d]+[t]+L[d:]
L=[(0,"A"),(1,"B"),(3,'D')]
t=(3,"c")
put(t,L)
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-8-b6b0025d7b67> in <module>
     41 L=[(0,"A"),(1,"B"),(3,'D')]
     42 t=(3,"c")
---> 43 put(t,L)

<ipython-input-8-b6b0025d7b67> in put(L, t)
     22 def put(L,t):
     23     x=t[0]
---> 24     if x<=L[0][0]:
     25         L=[t]+L
     26         return L

TypeError: 'int' object is not subscriptable
```

In [9]:

```
def insertion(L,x):
    if x<=L[0]:
        L.append(x)
        L[1:],L[0]=L[0:-1],L[-1]#en place
        return L
    elif x>=L[-1]:
        L.append(x)
        return L
    else :
        d=0
        f=len(L)-1
        while f>=d:
            m=(f+d)//2
            if L[m][0]==x:#x est dans L
                L.append(x)
                L[m+1],L[m+2:]=x,L[m+1:-1]
                return L
            elif L[m]>x:
                f=m-1
            else :
                d=m+1
    L.append(x)
    L[d],L[d+1:]=x,L[d:-1]
    return L

```

In [11]:

```
def get(L):
    valeur=L[0]
    L[0:-1]=L[1:]
    L.pop()
    return valeur
get(L)
```

Out[11]: (0, 'A')

In [14]:
```
1 from collections import deque
2 def get2(L):
3     return deque(L).popleft()
4
5 get2(L)
```

Out[14]:
```
(1, 'B')
```

In [ ]:
```
1
```