UNIVERSITE DE KINSHASA

FACULTE POLYTECHNIQUE

COURS D'ALGORITHMIQUE ET PROGRAMMATION

PROJET SUR SUR LA STRUCTURE DES DONNES

Par :
Le groupe 4
MFWAMBA NDAYA 2GEI
SHIMBA ILUNGA 2GEI
BADIBANGA BADIBANGA 2GEI

Dirigé par l'assistant **MOBISA**

Année académique 2021-2022

STRUCTURES DES DONNEES

Liste des classes et fonctions :

1. CircularQueue.py
2. LinkedDeque.py
3. _DoublyLinked.py
4. ArrayStack.py
5. ArrayQueue.py
6. LinkedStack.py
7. LinkedQueue.py
8. is_matched_html.py
9. is_matched.py
10. reverse_file.py
11. DynamicArraysizeEvaluation.py

Tutotriel du module des classes

Classe ArrayStack

Objectif: utiliser une liste python pour le stockage d'elements.

```
  #Classe ArrayStack
  ArrStack=ArrayStack()
```

Methodes:

- **__len__() :** Retourne la longeur de la pile
- **is_empty() :** Si la classe est vide, la methode retourne false, dans la cas contraire True.
- **push(e) :** Pour inserer un element dans la pile.
- **top() :** Donne le dernier element(l'element en tête de la pile)
- **pop() :** Supprime le dernier element stocké

```
#-----ESSAI------------

Stack=[i for i in range(10)]
arStack=ArrayStack()
print(reverse_elem(Stack))
for i in range(10):
    arStack.push(i)
for i in range(3):
    arStack.pop()

print("taille",arStack.__len__())
```

Classe ArrayQueue

Objectif: File d'attente FIFO pour utiliser une liste python comme stockage d'elements ***Methodes:***

- **first() :Retourne le premier element de la file**
- **dequeue() :** Supprime et retourne le premier element de la file.

- ✢ **enqueue :** Ajoute un element à la fin de la file.
- ✢ **_resize() :** pour redimensionner à une nouvelle liste de capacité superieure ou égale à la taille de la pile

```
#---------On test la classe ArrayQueue----------
print("test de la classe ArrayQueue")
ArrQueue=ArrayQueue()
for i in range(20):
  ArrQueue.enqueue(i)

for i in range(5):
  ArrQueue.dequeue()

for i in range(100,112):
  ArrQueue.enqueue(i)

print("Taille:",ArrQueue.__len__())
print()
```

Classe ArrayDeque

Objectif: Herite de la classe ArrayQueue
Methodes:

- ✢ **add_first() :** Ajoute un element au debut du tableau.
- ✢ **add_last() :** Comme enqueue
- ✢ **delete_first() :** Similaire à enqueue mais à la fin de la file.
- ✢ **delete_last() :** Similaire à dequeue mais à la fin de la file.

```
#--------On test la classe ArrayDeque--------
arrDeque=ArrayDeque()

for i in range(10):
    arrDeque.add_last(i)

for i in range(11,15):
    arrDeque.add_first(i)

print("Taille:",len(arrDeque))
print("premier element",arrDeque.first())
print("Le dernier element est:",arrDeque.last())
```

Classe DynamicArray

Objectif: Tableau dynamic utilisé comme list, mais ici pour créer un tableau de grande capacité que le tableau précedent.
Methodes:

- ✢ **append(self,obj) :** Ajoute un élement à la fin du tableau.
- ✢ **_resize(self,c) :** Rédimensionne le tableau interne de capacité c.

```
  # -*- coding: utf-8 -*-

if __name__ == '__main__':
  x= DynamicArray()
  print("Le tableau actuel possede {} elements et une capacite elementaire de {}".
    format(x._n, x._capacity))
  #On peut definir alors une nouvelle liste  de 30 elements
  for i in range(30):
  x.append(i)
  print("Ce nouveau tableau possede {} elements et une capacite de {}".format(len(x)
    , x._capacity))
```

```
# En se servant de __getitem__ on cherche la position d'un element dans notre tableau

print("L'element qui occupe la position 15 est: {}".format(x.__getitem__(16)))

# Puisque notre tableau possede 30 element avec une capacite de 32
# Dans ce cas si nous ajoutons 1'element c'est seulement les nombres d'element qui changer et non la capacite
x.append(107)
print("Le nombre d'element du tableau est", x._n,"mais la capacite est toujours egale a", x._capacity )
# Notre capacite devrait double si on ajoute deux autres elements
x.append(0)
x.append(5)
print("Le nombre d'element du tableau est", x._n,"Mais la capacite devient egale a ", x._capacity )
#Essayons avec un tableau ayant una capacite de
x._resize(35)
print("Le nombre d'element du tableau est {} et  la capacite devient egale a {}".format(x._n,x._capacity))
#On cree un tableau avec une capacite quelconque
y = x._make_array(10)
print(y)
```

Classe PositionalList

Objectif: Un conteneur séquentiel d'élements permettant d'acceder à une position.
Methodes:

- **_validate(self,p) :** Retourne la position du noeud.
- **_make_position(self,node) :** Retourne une position d'instance pour un noeud donné.
- **first(self) :** Retourne la première position dans la liste.
- **last(self) :** Retourne la dernière position dans la liste.
- **before(self,p) :** Retourne la position juste avant la position avant la position p.
- **after(self,p) :** Retourne la position juste avant la postion p.
- **_insert_between(self, e, predecessor, sucessor) :** Ajoute un élement entre deux noeuds existant et retourne la nouvelle position.
- **add_first(self, e) :** Insère un élement e au debut de la liste et retourne la nouvelle position.
- **add_last(self, e) :** Insère un élement à la fin de la liste et retourne la nouvelle position.
- **add_before(self, e) :** Insère un élement e dans la liste avant avant une position p et retourne la nouvelle position.

```
#-------Essai

if __name__ == '__main__':


  r = PositionalList()
  r.add_last(4)
  r.add_first(5)

  r._insert_between(10,r.first(),r.last())

  print("taille:",r.__len__())
```