**AXEL VAGANAY**

Fiche descriptive de l’activité

|  |  |  |
| --- | --- | --- |
| Enseignant : | Granjon david |  |
| Intitulé du thème : | Réaliser son propre conteneur linéaire légèrement original (priority queue) |  |
| Résumé : | A travers cette activité, les élèves vont implémenter une structure de données de type File, puis File de priorité, dans le but de réaliser un écran d’affichage dans une salle d’attente des urgences à l’hôpital. Le prochain patient pris en charge ainsi que les 9 suivants sont affichés sur l’écran.  Une petite interface permet à l’infirmier.ère qui accueille les patients de saisir leurs noms et la priorité de l’urgence.  Les structures File et File de priorité sont implémentées dans un premier temps par des tableaux dynamiques (list python) puis par des listes chainées.  La comparaison de leur efficacité temporelle est réalisée | |
| Cadre pédagogique (TD/TP/Projet, temporalité) : | Travail Dirigé ou Activité à réaliser en binôme, en fin de 1er trimestre ou début du deuxième trimestre | |
| Durée :  (temps élève) | 4 heures | |
| Prérequis : | Bases langage Python  Complexité d’un algorithme  Notions de programmation orientée objet  Structures linéaires de type Listes, File et Pile | |
| Compétences mobilisées : | Distinguer interface et implémentation.  Écrire plusieurs implémentations d’une même structure de données.  Spécifier une structure de données par son interface.  Distinguer des structures par le jeu des méthodes qui les caractérisent.  Choisir une structure de données adaptée à la situation à modéliser | |
| Matériel nécessaire : | Ordinateur, éditeur Python, librairie tkinter | |
| Mode d’évaluation : | Notation par binôme :   * Du sujet complété * Des fichiers .p*y* des implémentations des structures de données. | |

# Hôpital Médical Bâtiment Dessiné à La Main Peut être Un élément ...Mise en situation :

Vous vous êtes lancé dans une carrière de développeur informatique, et le premier projet qui vous a été confié est de participer au développement d’une application dans le milieu hospitalier. Cette application doit permettre à l’infirmier.ère à l’accueil des urgences d’enregistrer un patient, et afficher dans la salle d’attente le nom du prochain pris en charge ainsi que les 9 suivants. **L’ordre d’arrivée est, dans un premier temps, le seul critère de prise en charge**.

# Quel type abstrait de données ?

Pour enregistrer les données, c’est-à-dire les patients qui arrivent aux urgences, et faciliter leur prise en charge, il vous faut choisir entre deux ***types abstraits de données*** *(TAD)*pour l’application : La **Pile** ou la **File**

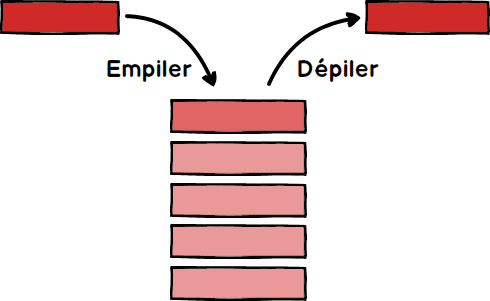
Voici les procédures et spécifications d’une Pile et d’une File (*Interfaces* entre l’application et la structure de données):

**Pile**

* **Constructeur** pile() : Création d’une pile

*Postcondition : La Pile est une pile vide*

* **Empiler(***elm***)** : ajoute un élément sur le dessus de la pile.



**Premier élément**

**Dernier élément**

*Postcondition : L’élément est ajouté en sommet de la pile*

* **Dépiler()** : retire le premier élément de la pile. Cette fonction renvoie en résultat la valeur de cet élément.

*Précondition : la pile n’est pas vide  
Postcondition : le sommet de la pile est dépilé et retourné*

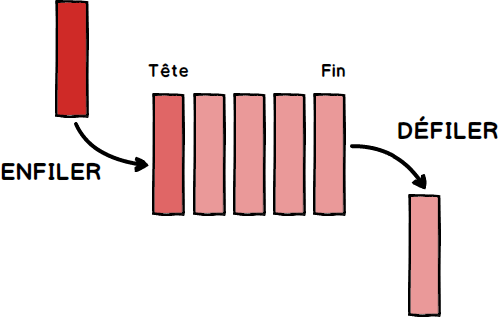
* **Vider()** : Vide la pile de tous ses éléments.

*Postcondition : la pile ne contient plus aucun élément*

* **estVide()** : renvoie True si la pile est vide, False sinon

# File

* **Constructeur** file() : Création d’une file



**Premier de file**

**Queue de file**

*Postcondition : La file est une file vide*

* **Enfiler(***elm***)** : ajoute un élément à la file.

*Postcondition : L’élément est ajouté en queue de file*

* **Défiler()** :

*Précondition : la file n’est pas vide  
Postcondition : le premier de la file est supprimé et retourné*

* **Vider()** : Vide la file de tous ses éléments.

*Postcondition : la file ne contient plus aucun élément*

* **estVide()** : renvoie True si la file est vide, False sinon
* **longueur()**: renvoie le nombre d’éléments en attente dans la file

1. Déterminer lequel de ces deux types abstraits de données bien connus est le plus adapté à l’application que l’on souhaite développer ? Justifiez

Le type abstrait de donné le plus adapté pour enlever le patient le plus ancien et non le plus récent est d’utiliser une File qui défile le plus ancien élément.

# Quelle structure de données ?

Un type abstrait de données permet de définir d’un point de vue théorique les seules actions de base qui seront possibles sur les données, ici les noms de patients.

Maintenant pour implémenter physiquement ce **T**ype **A**bstrait de **D**onnées dans un langage de programmation, il faut choisir une **structure de données**. Ce qui dicte ce choix est avant tout **l’efficacité du programme autrement sa complexité temporelle**.

Plusieurs choix d’implémentation du TAD s’offre à vous :

* Avec un tableau dynamique (list python)
* Avec une liste simplement chainée
* Avec une liste doublement chainée

1. Laquelle de ces structures vous parait la plus appropriée en termes d’efficacité (complexité temporelle) ? Justifiez.

***(S’aider des documents en annexe pour répondre)***

La structure la plus appropriée car la moins complexe au niveau temporel et sans aucun doute la liste doublement chaîné. Celles-ci permettent d’ajouter un élément en début de liste -> complexité en O(1),

d’ajouter un élément en fin de liste -> complexité en O(1), d’obtenir l’élément en fin de liste-> O(1) et d’obtenir l’élément en début de liste -> O(1). Tout ceci qui est « excellent » et donc le plus avantageux peut importe la longueur de la file.

# File implémentée par un tableau dynamique (list python)

## Implémentation

Malgré ce qui a pu être trouvé à la question précédente, pour des raisons de facilité de compréhension et programmation, nous allons implémenter dans un 1er temps la **FILE** avec une **list** **Python** (tableau dynamique)

Ouvrez le fichier incomplet Class\_File\_ListPython.py à l’aide d’un IDE de votre choix (Spyder, Pyzo, ...).

Prenez connaissance de la classe File, de ses méthodes de classes, et faites le lien avec le Type Abstrait de Données [*décrit plus haut*](#_File).

1. Implémenter les méthodes de la classe File : **longueur()**, **enfiler(elm)**, et **defiler()**

Vous veillerez à **enfiler en fin de list python**, et **défiler en début de list (rang 0) Fais**

1. Tester vos 3 méthodes implémentées à l’aide du fichier tester\_file.py, pour cela décommentez au fur et à mesure les lignes. **Fais**
2. Implémenter la dernière méthode de la classe File : **DixProchains()**

Cette méthode doit renvoyer dans une liste les 10 prochains éléments qui seront défilés sans les supprimer de la File.

Décommenter les dernières lignes du fichier tester\_file.py pour tester votre méthode. **Fais**

s

## Utilisation dans l’application hospitalière

Votre classe File implémentées par une list Python est désormais fonctionnelle, nous allons maintenant la tester dans l’application hospitalière d’accueil des patients aux urgences. Le chiffre correspondant à l’urgence ne sert à rien pour le moment...

1. Exécutez le fichier Application\_Hopital.py.

Amusez-vous à admettre des patients, les prendre en charge, vider la liste d’attente, etc, ...

Retrouvez (en devinant ou visionnant le code) quelles méthodes sont utilisées pour les boutons :

* Mettre en attente : def metEnAttente()
* Prendre en charge patient : def prendEnCharge()
* Vider la liste : def viderListeAttente()

1. Testez l’efficacité de l’implémentation de la File avec une list Python en relevant les temps des tests suivants dans la console de l’IDE (ils dépendent du matériel) :

* 100 personnes (enfilées/défilées) : 0,36 ms
* 20000 personnes (enfilées/défilées) : 115,64 ms

# File implémentée par une liste doublement chainée

## Découverte de l’implémentation

Comme nous l’avons vu précédemment, l’implémentation de la File avec un tableau dynamique n’est pas la solution la plus judicieuse, mais plus facile à coder.

Rappel sur les listes doublement chainées :

* Voir cours
* [Article wikipedia](https://fr.wikipedia.org/wiki/Liste_cha%C3%AEn%C3%A9e#:~:text=Liste%20doublement%20cha%C3%AEn%C3%A9e,-Liste%20doublement%20cha%C3%AEn%C3%A9e&text=La%20diff%C3%A9rence%20avec%20une%20liste,ou%20de%20pr%C3%A9d%C3%A9cesseur%20en%20pr%C3%A9d%C3%A9cesseur.)

Ouvrez le fichier Class\_File\_LDChainee.py à l’aide d’un IDE de votre choix (Spyder, Pyzo, ...).

Prenez connaissance de la classe FileLDC, de ses méthodes de classes, et faites le lien avec le Type Abstrait de Données [*décrit plus haut*](#_File).

1. Rédiger les docstrings des méthodes : **enfiler(***infoAInserer***)**, **defiler() Fais**
2. Pour les plus rapides, implémenter la méthode **vider() Fais**

## Utilisation dans l’application hospitalière

Dans le fichier Application\_Hopital.py commenter les lignes 14 et 15 concernant l’import de la class File implémentée par une list Python.

Décommenter les lignes 19 et 20 afin d’importer la classe FileLDC depuis le fichier Class\_File\_LDChainee.py

Amusez-vous à admettre des patients, les prendre en charge, vider la liste d’attente, etc, ...

1. Testez l’efficacité de l’implémentation de la structure File avec une liste doublement chainée, relever les temps des tests suivants (dans la console de l’IDE) :

* 100 personnes (enfilées/défilées) : 0,62 ms
* 20000 personnes (enfilées/défilées) : 62,34 ms

Comparer à l’implémentation précédente avec une list Python, justifier les résultats, et conclure.

On observe qu’avec une simple liste python, il met 0,36ms pour traiter 100 patients et 115ms pour 20 000, or avec une liste doublement chainée 100 personnes met 0,62ms et 20 000 seulement 62ms. On peut en conclure que pour beaucoup de patients, le plus rapide est la liste doublement chainée en O(1) alors que la liste python est en O(n).

# File de priorité implémentée par une list python

1. Quel paramètre important lié au contexte de l’application que l’on souhaite développer, n’a pas été pris en compte jusqu’à présent ?

Etant donné le contexte de travail, cette application dois prendre en compte la priorité des patients comme il ce passe en temps réel à l’hopital.

Afin de prendre en compte l’état du patient admis aux urgences, le Type Abstrait de données "**file de priorité"** devient plus approprié.

Le principe est d’enfiler l’élément (patient) dans la file (d’attente) en fonction de sa priorité, tout en respectant en cas d’égalité de priorité, l’ordre d’arrivée dans la file.

Nous allons implémenter la file de priorité à l’aide d’une list Python, et les éléments stockés seront maintenant des tuples (priorité, nom) :

* Priorité de prise en charge du patient : entier (int) de 1 à 5.

*1 -> immédiate 2 -> très urgente 3 -> urgente 4 -> standard 5 -> non urgente*

* Nom du patient : chaine de caractères (string)

# File de priorité

Procédures et spécifications d’une File de priorité (*Interfaces* entre l’application et la structure de données):

* **Constructeur** FilePrioritaire() : Création d’une file de priorité

*Postcondition : La file de priorité est une file vide*

* **Enfiler(***elm***)** : ajoute un élément à la file de priorité.

*Postcondition : L’élément est ajouté dans la file en fonction de sa priorité, et en respectant l’ordre d’arrivée en cas d’égalité de priorité*

* **Défiler()** :

*Précondition : la file n’est pas vide  
Postcondition : le premier de la file est supprimé et retourné*

* **Vider()** : Vide la file de priorité de tous ses éléments.

*Postcondition : la file de priorité ne contient plus aucun élément*

* **estVide()** : renvoie True si la file de priorité est vide, False sinon
* **longueur()**: renvoie le nombre d’éléments en attente dans la file de priorité

Dans le fichier Application\_Hopital.py, commenter les lignes 19/20, et décommenter les lignes 24/25 afin d’utiliser la class **FilePrioritaire** implémentée par une list Python.

1. Ouvrez le fichier incomplet Class\_File\_Prioritaire\_ListPython.py et analyser la classe **FilePrioritaire** avec ses méthodes de classes.

En observant les méthodes **defiler()**, **et DixProchains()** compléter le texte suivant :

Les patients prioritaires (priorité 1) sont insérés(enfilés) plutôt en fin de list Python, alors que les patients dont la priorité est 5 (non urgente) sont insérés plutôt en début de list Python.

Le patient pris en charge (défilé) est celui en fin de list Python.

Les 10 patients renvoyés par la méthode **DixProchains()** pour l’affichage sont Les 10 patients les plus prioritaires(priorités 1 en général) donc les 10 prochains éléments à sortir (qui seront défilés) de la File sans les supprimer de la File

1. A partir des observations de la question précédente, implémenter la méthode **enfiler(***element***)**. Testez votre code directement dans l’application Application\_Hopital.py

Amusez-vous avec l’application en admettant des patients avec des priorités différentes.

Testez l’efficacité de l’implémentation en relevant les temps des tests suivants (dans la console de l’IDE) :

* 100 personnes (enfilées/défilées) : 1,34 ms
* 10000 personnes (enfilées/défilées) : 45,03 ms
* 20000 personnes (enfilées/défilées) : 188,31 ms

# File de priorité implémentée par une liste doublement chainée

1. Commenter/Décommenter les lignes au début du fichier Application\_Hopital.py afin d’utiliser la classe FilePrioritaireLDC.

Amusez-vous avec l’application en admettant des patients avec des priorités différentes.

Testez l’efficacité de l’implémentation en relevant les temps des tests suivants (dans la console de l’IDE) :

* 100 personnes (enfilées/défilées) : 2,13 ms
* 10000 personnes (enfilées/défilées) : 54,03 ms
* 20000 personnes (enfilées/défilées) : 13444,37 ms

Conclusion et justification sur l’utilisation d’une liste doublement chainée :

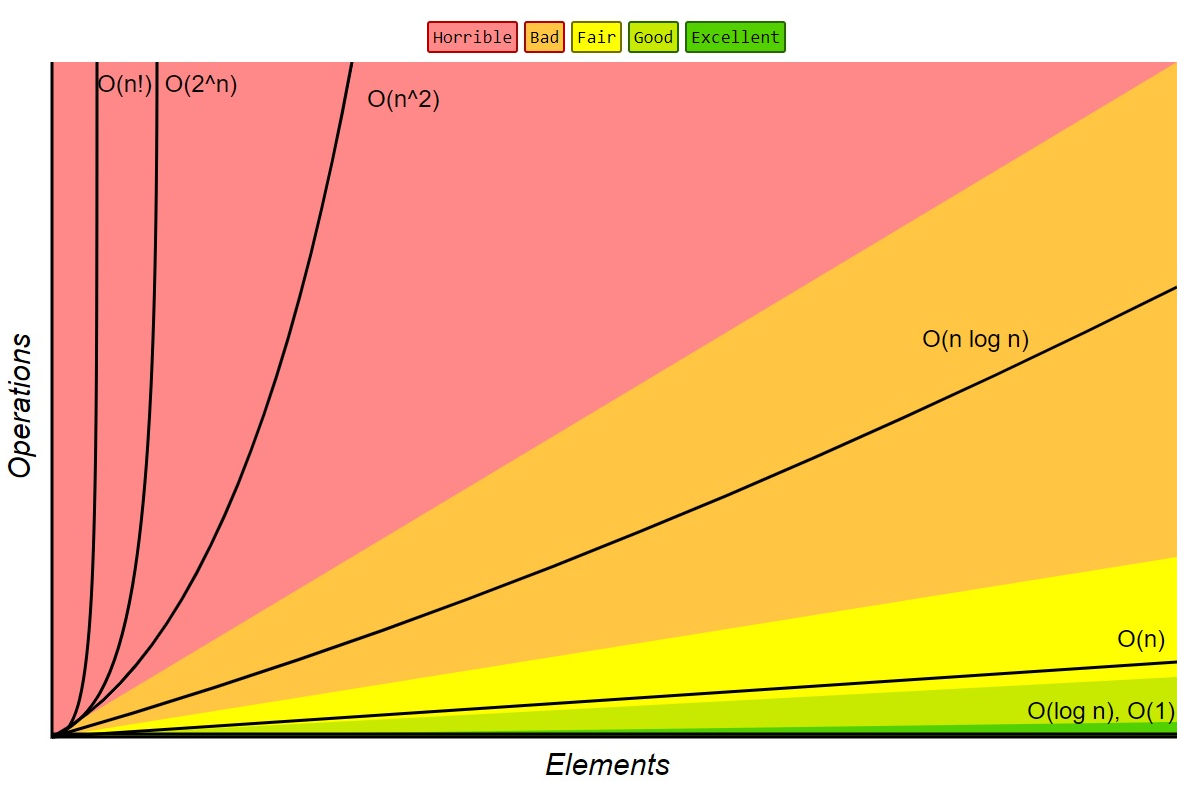
Lorsque la file d’attente est longue avec des priorité la liste doublement chainé mais très longtemps (en O(n)) car une vérification un à un par éléments ce fais.

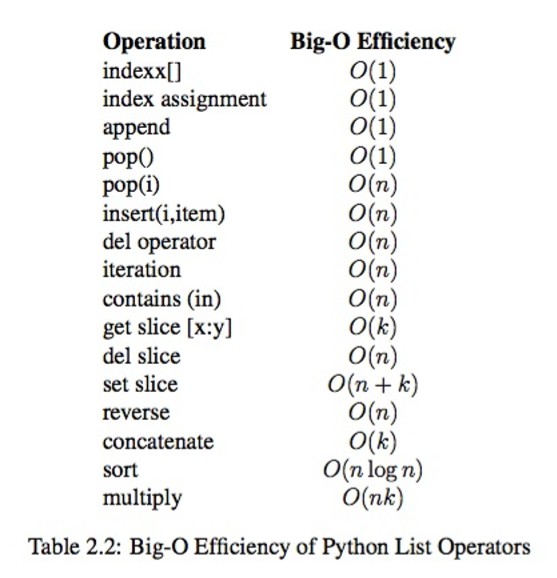
# Pour aller plus loin (en projet ?)

Implémentation d’une liste simplement chainée circulaire de 10 éléments (10 informations sur l’actualité). Un bandeau (de taille une information) dans l’application hospitalière permet d’afficher successivement en boucle les 10 informations.

Eventuellement, pour les meilleurs, renouveler les informations une par une, en supprimant la plus ancienne.

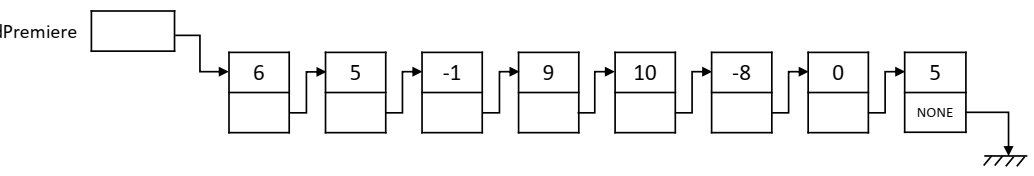
ANNEXES

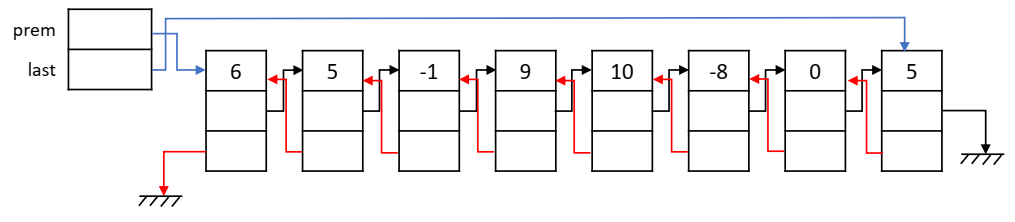
Aide pour le travail 2 : Complexité

**List Python :**

* Ajout d’un élément en début de list python -> insert (0,elem)
* Ajout d’un élément en fin de list python -> append(elem)
* Obtenir l’élément en fin de list python -> pop()
* Obtenir l’élément en début de list python -> pop(0)

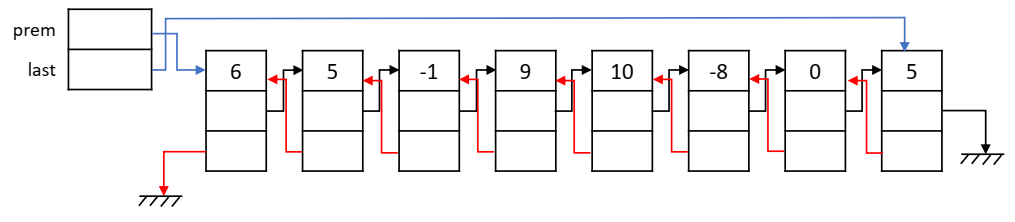
**Liste simplement chaînée :**





* Ajout d’un élément en début de liste chaînée -> complexité en O(1)
* Ajout d’un élément en fin de liste chaînée -> complexité en O(n)
* Obtenir l’élément en fin de liste chaînée -> O(n)
* Obtenir l’élément en début de liste chaînée -> O(1)

**Liste doublement chaînée :**



* Ajout d’un élément en début de liste doublement chaînée -> complexité en O(1)
* Ajout d’un élément en fin de liste doublement chaînée -> complexité en O(1)
* Obtenir l’élément en fin de liste doublement chaînée -> O(1)
* Obtenir l’élément en début de liste doublement chaînée -> O(1)