Devoir maison 2020 – File de priorité bornée

1 Impératifs

Travail à rendre par mail à `marc-michel.corsini@u-bordeaux.fr` en utilisant une adresse officielle de la forme
`prenom.nom@etu.u-bordeaux.fr`, vous indiquerez dans le corps du mail les noms et prénoms des membres du groupe et vous prendrez soin de mettre en copie vos camarades **CC**, afin que tous reçoivent l'accusé de réception du mail. Date limite du rendu le **Lundi 07 décembre à 08h00 (matin)**, tout retard sera assorti d'une **forte** pénalité. Pour ce DM vous travaillerez par groupe de 2 **exceptionnellement** 3[1], évidemment la note sera adaptée en fonction.
Le rendu est constitué d'un document au format **PDF** pour la partie théorique (TdA, axiomes, complexité) et un code **python 3.7+** orienté objet qui outre l'implémentation devra fournir les tests correspondant aux axiomes que vous avez définis.

2 Présentation du sujet et des contraintes

Le projet de cette année porte sur la comparaison de deux implémentations de file avec **priorités bornées**. Les deux implémentations satisfont **aux mêmes axiomes**, mais diffèrent par les complexités des opérations. La comparaison a pour but de déterminer l'implémentation la plus efficace (en terme de complexité) pour *chaque* méthode.
La structure sous-jacente sera basée sur une liste simplement chaînée munie de deux sentinelles « head » et « tail », qui sont inaccessibles aux utilisateurs. Le fichier **simpleQueue.py** vous donne le fonctionnement pour une file sans priorité.

2.1 simpleQueue

Les opérations sont la création `__init__`, la longueur `__len__`, l'ajout `push`, la suppression `pop`, la valeur du premier élément à traiter `first`, savoir si la structure est vide `empty`, la liste python des éléments présents `to_list`. Le code est fourni avec les axiomes en commentaires, et un code permettant de vérifier les axiomes.

méthode QNode	signature	instructions	complexité
__init__	$\mathbf{V} \to \mathbf{E}$	2	$O(1)$
item	$\mathbf{E} \to \mathbf{V}$	1	$O(1)$
get_next	$\mathbf{E} \to \mathbf{E} \cup \{\texttt{None}\}$	1	$O(1)$
set_next	$\mathbf{E} \cup \{\texttt{None}\} \times \mathbf{E} \cup \{\texttt{None}\} \to \texttt{None}$	$2 + \max(1,0)$	$O(1)$
méthode SQueue	signature	instructions	complexité
__init__	$\emptyset \to \mathbf{Queue}[\mathbf{E}]$	3	$O(1)$
__len__	$\mathbf{Queue}[\mathbf{E}] \to \mathbb{N}$	1	$O(1)$
push	$\mathbf{Queue}[\mathbf{E}] \times \mathbf{E} \to \texttt{None}$	$2 + 1 + \max(2,2)$	$O(1)$
pop	$\mathbf{Queue}[\mathbf{E}] \not\to \texttt{None}$	3	$O(1)$
first	$\mathbf{Queue}[\mathbf{E}] \not\to \mathbf{V}$	1	$O(1)$
empty	$\mathbf{Queue}[\mathbf{E}] \to \mathbf{Bool}$	1	$O(1)$
to_list	$\mathbf{Queue}[\mathbf{E}] \to \mathbf{list}[\mathbf{V}]$	$2 + 3n + 1$	$O(n)$

2.2 File avec priorité bornée

Une donnée est une paire contituée d'une valeur et d'une priorité entière. Plus la valeur de la priorité est faible, plus élevée est la priorité.

[1] Dans ce cas veuillez me contacter par mail **au plus tard le lundi 02 nov. 2020 12h00**

Une file avec priorité bornée gère des éléments dont les priorités sont comprises entre $0 \leq p < n$. Si la priorité est hors de l'intervalle, l'élément n'est pas pris en compte.
Si tous les éléments ont la même priorité, le fonctionnement est identique à celui d'une file (premier entré, premier sorti), lorsque les éléments ont des priorités différentes, le premier sorti est le premier entré avec la plus haute priorité – plus petite valeur de priorité.

2.2.1 Une seule liste

La classe `BoundedOneQueue`, implémente une file avec priorité bornée en n'utilisant qu'une seule liste simplement chaînée, munie de 2 sentinelles.

- Le constructeur prend en entrée un entier n, et va gérer des éléments de la forme (v, p) une paire valeur, priorité, tels que la priorité sera $0 \leq p < n$.
- `max_priority` est une **property** qui renvoie la valeur utlisée lors de la création de la file.
- La longueur `__len__`, renvoie le nombre d'éléments dans la file
- `pop` enlève le premier élément de plus basse priorité, ne renvoie rien
- `push` reçoit une valeur v et une priorité p. Si p ne vérifie pas $0 \leq p < max\_priority$, pas d'insertion. Sinon, on insère dans la liste de telle sorte que tout ce qui est avant est de priorité inférieure ou égale à p, tout ce qui est après est de priorité strictement supérieure à p. `push` ne renvoie rien
- `first` renvoie la première paire (v, p) de la liste
- `empty` renvoie **True** si la file est vide, **False** sinon
- `to_list` renvoie une liste python des paires (v, p) présentes dans la file
- `howmany` prend en entrée un entier naturel, et renvoie le nombre d'éléments dans la file ayant cette priorité
- `summary` renvoie une liste python de taille `max_priority` et contenant la distribution des priorités ordonnées de manière croissante.

2.2.2 Plusieurs listes chaînées

La classe `BoundedListQueue` implémente une file avec priorité bornée en utilisant une liste python de taille fixe, le ième élément de la liste correspond à la file de priorité i. Les informations stockées sont uniquement les valeurs, la priorité étant connue par l'index de la file. Les méthodes sont **exactement** les mêmes que celles de la classe `BoundedOneQueue`. Les files sont implémentées à l'aide de liste simplement chaînée munies de 2 sentinelles.

2.3 Contraintes

- Il n'y a aucune variable publique dans les 2 classes.
- Il doit être impossible, pour l'utilisateur de rajouter des attributs ou méthodes aux 2 classes.
- Les axiomes et les signatures sont identiques pour les deux classes.
- **V** est l'ensemble des valeurs, **P** est l'ensemble des priorités, **E** est l'ensemble $\mathbf{V} \times \mathbf{P}$, **Bool** désigne les booléens, $\mathbb{N}$ les entiers naturels, $\mathbb{Z}$ les entiers relatifs. `int` correspond aux entiers relatifs. `list` désigne les listes en python.