Devoir maison

Informatique théorique

Le but de ce projet est de comparer deux implémentations de file avec priorités bornées.

Queues : les éléments arrivent avec un « ticket » contrairement aux files classiques, c’est leur priorité (compris entre 0 . On fait donc sortir de la queue selon les tickets et pas l’ordre d’arrivée (le premier sorti est le premier entré avec la plus haute priorité, c’est-à-dire la plus petite valeur). Cependant, s’il y a deux éléments avec la même priorité (ticket) alors là on prend en compte l’ordre d’arrivée (premier arrivé=premier sorti). Plus la valeur de la priorité est faible, plus la priorité est élevée.

Fichier simpleQueue.py : fonctionnement pour file sans priorité

Faire toutes les fonctions pour les deux méthodes

* **BoundedOneQueue** : file avec priorité borné avec une seule liste simplement chaînée, munie de 2 sentinelles

|  |  |
| --- | --- |
| V | P |
| V | P |
| … | … |
| … | … |

* **BoundedListQueue** : file avec priorité bornée avec liste de taille fixe, le ième élément de la liste correspond à la file de priorité. Les informations stockées sont les valeurs.

**Liste Indice (donne priorité)**

|  |
| --- |
| V V V |
| V V |
| V |
| … |

0

1

2

…

Il faut les comparer en termes de complexité : Un algorithme de forte complexité a un comportement moins efficace qu'un algorithme de faible complexité, il est donc généralement plus lent.

Ici nous allons utiliser des files avec priorité bornée. Pour caractériser une file on peut identifier les fonctions suivantes :

|  |  |
| --- | --- |
| Lexique fr | Réalisation |
| Créer  Priorité max  Longueur  Ôter  Ajouter  Premier  Vide  Liste présente  Élément priorité  Sommaire priorité | \_\_init\_\_  max\_priority  \_\_len\_\_  pop  push  first  empty  to\_list  howmany  summary |

**Types :**

Ici désigne l’ensemble des valeurs, est l’ensemble des priorités, est l’ensemble

Queue [] désignera les queues permettant de stocker des éléments de type .

**Fonctions :**

Cette partie précise les opérations disponibles sur le TdA et leurs signatures, c’est à-dire le nombre et le type de paramètres attendus ainsi que le type du résultat.

**BoundedOneQueue**

* \_\_init\_\_ : BoundedOneQueue []
* max\_priority : BoundedOneQueue []
* \_\_len\_\_ : BoundedOneQueue [
* pop : BoundedOneQueue []None



* push : None
* first : BoundedOneQueue []



* empty : BoundedOneQueue [] Bool
* to\_list : BoundedOneQueue []list[]
* howmany :
* summary : BoundedOneQueue [ list[]

**BoundedListQueue**

* \_\_init\_\_ : BoundedListQueue []
* max\_priority : BoundedListQueue []
* \_\_len\_\_ : BoundedListQueue [
* pop : BoundedListQueue [] None



* push : None
* first : BoundedListQueue []



* empty : BoundedListQueue [] Bool
* to\_list : BoundedListQueue []list[]
* howmany :
* summary : BoundedListQueue [ list[]

**Axiomes :**

Les axiomes sont des formules logiques toujours vraies, ils permettent de spécifier le comportement des fonctions.

* \_\_init\_\_ :

ensure empty(\_\_init\_\_( ??))==True *elle doit être vide*

ensure \_\_len\_\_(\_\_init\_\_( ??))==0 *le nombre d’élément de la file juste créée =0*

* max\_priority :
* \_\_len\_\_ :

Quand on enlève un élément, la longueur diminue de 1

* pop :

require empty(???)==False

ensure « Element enlevé » a la plus basse priorité

* push :

ensure \_\_len\_\_(push(???))== \_\_len\_\_+1 la taille de la liste augmente de 1

* first :

require empty(???)==False

* empty :

ensure empty(\_\_init\_\_( ??))==True *Une file créée est toujours vide*

ensure empty(push(???))==False

* to\_list :

si la liste est vide ne renvoie rien (False ?)

* howmany :

si la liste est vide alors howmany=0

* summary :

**Préconditions :**

Du fait de l’existence de fonctions partielles dans le TdA, il faut exprimer à quelle(s) condition(s) la fonction fournira un résultat. On utilisera le mot clef « nécessite » ou « require », pour décrire quand les arguments appartiennent au domaine de la fonction.

* v,p Queue], pop(v,p) nécessite not empty(v,p)
* v,p Queue], first(v,p) nécessite not empty(v,p)