LINFO1122 - Méthodes de conception de programme : DEVOIR N°2 2023

Auteurs :
Jérôme Lechat
Arthur Louette
Arthur Fraipont

NOMA :
5035-18-00
5108-21-00
7738-20-00

1 Introduction

L'objectif de ce deuxième devoir dans le cadre du cours de Méthode de Conception de Programmes est simple : définir et spécifier une file de priorité (ou Priority Queue en anglais). C'est une structure de données qui représente une collection d'éléments associées chacun à une priorité. Elle est munie de trois opérations : tester si elle est vide, ajouter un élément à la file, et retirer l'élément de priorité maximale.
Nous devons donc donner une implémentation complète de cette structure de données. Le retrait de l'élément de priorité maximale doit se faire en O(1).

2 Invariant de Représentation

L'invariant de représentation est une propriété qui doit toujours être vraie pour une instance de la classe, avant et après chaque opération. Voici l'invariant de représentation ok() sur PQueue :

1. La taille de la file (size) est égale à la longueur du tableau data.

2. Les éléments dans le tableau data sont triés par ordre décroissant de priorité.

3. L'élément de priorité maximale est toujours en tête du tableau data (premier élément).

3 Fonction d'Abstraction

Seconde question de ce devoir, il nous est demandé de donner la fonction d'abstraction abs() sur PQueue vue comme un multiensemble d'entiers. Une Fonction d'Abstraction est une fonction offrant une vue simplifiée où chaque élément est considéré comme un entier, tout en conservant la notion de priorité. Voici la fonction d'abstraction en langage Java :

```
class PQueue {
    var data: array<int>;
    var size: int;
    ...

    // Invariant de representation (tel que defini precedemment)

    // Fonction d'abstraction
    function abs() -> multiset<int>:
        return multiset<int>(data[0:size])
}
```

4 Différentes Spécifications

Voici les différentes spécifications attendues pour ce devoir. On attend les pré et post-conditions de ces 3 fonctions suivantes : isEmpty(), insert() et removeMax().

4.1 isEmpty()

4.1.1 Preconditions

Aucune pré-conditions n'est requise.

$$@Pre : true$$

4.1.2 Postconditions

b retourne true si et seulement si pq est vide.

$$@Post : b \iff isEmpty(pq)$$

4.2 insert()

4.2.1 Preconditions

pq.data est un tableau trié par ordre décroissant (plus petit au plus grand).

$$@Pre : \forall i \in [1, pq.size - 1] :: pq.data[i] \leq pq.data[i + 1]$$

4.2.2 Preconditions

pq.size est bien la taille actuelle de la file de priorité.

$$@Pre : pq.size' = pq.size$$

4.2.3 Postconditions

pq.data est toujours trié par ordre croissant et contient bien l'élément e.

$$@Post : (\exists i[1, pq.size] :: e = pq.data[i]) \wedge (\forall j \in [1, pq.size - 1] :: pq.data[i] \leq pq.data[i + 1])$$

4.2.4 Postconditions

pq.size est incrémenté de 1.

$$@Post : pq.size' = pq.size + 1$$

4.3 removeMax()

4.3.1 Preconditions

La file de priorité (pq) est non vide.

$$@Pre : \neg isEmpty(pq)$$

La file de priorité (pq) est triée par ordre croissant.

$$@Pre : \forall i, j \ (0 \leq i < j < pq.length \Rightarrow pq[i] \leq pq[j])$$

4.3.2 Postconditions

l'élément résultant result[0] est présent dans le tableau trié pq.data.

$$@Post : (\exists i \in [1, pq.size] : pq.data[i] = result[0]) \wedge (\forall j \in [1, pq.size - 1] : pq.data[j] \leq pq.data[j+1])$$

4.4 join()

4.4.1 Préconditions

Les deux files de priorité (pq et p2) sont triées par ordre croissant.

$$@Pre : sorted(pq) \wedge sorted(p2)$$

4.4.2 Postconditions

La file de priorité résultante (pq) est la concaténation de pq et p2.

$$@Post : pq = old(pq) + old(p2)$$

La file de priorité résultante (pq) est triée par ordre croissant.

$$@Post : \forall i, j \ (0 \leq i < j < pq.length \Rightarrow pq[i] \leq pq[j])$$

5 Implémentation de removeMax()

```
public static removeMax(){
    if (isEmpty()) {
            throw new IllegalStateException("La file de priorite
               est vide.");
    }

    pq.size--;

    int toRet = pq.data[pq.size];
    pq.data[pq.size] = null;

    return toRet;
}
```

6 Conclusion

Ceci conclu notre second devoir dans le cadre du cours de Méthode de conception de programmes. Nous avons correctement réparti le travail entre les trois membres du groupe : Arthur Louette s'est occupé de la structure générale du document ainsi que l'invariant de représentation et la fonction d'abstraction, tandis que Jérôme Lechat a travaillé sur les deux première spécifications et l'implémentation de removeMax(). Arthur Fraipont quant à lui a fait les spécifications restantes et l'implémentation de removeMax() en Java.