Une classe liste FIFO

Projet individuel à rendre pour le 5 novembre 2018

Dans les listes chaînées que nous avons rencontré jusqu'à présent, nous pouvons avoir accès au premier élément de la liste en temps constant (O(1)). En revanche avoir accès au dernier élément (celui entré en premier dans la liste) prend un temps proportionnel à la taille de la liste (O(n)).

Nous souhaiterions mettre en place une liste spécifique permettant un accès FIFO (first-in, first-out) c'est-à-dire une liste dans laquelle on peut retirer le dernier élément en temps raisonnable. Une telle liste est souvent appelée « queue ». Nous souhaiterions en outre conserver l'accès au premier élément en temps constant.

Une façon de faire cela consiste à utiliser en interne deux listes nommées `in` et `out`, entrée, et sortie. La liste d'entrée reçoit les éléments que l'on ajoute au fil du temps. La seconde reste vide tant que l'on n'a pas besoin de retirer d'élément à la fin. Dès que cette occasion se présente, on inverse la liste d'entrée et on l'assigne à la sortie, la liste d'entrée devient alors vide.

L'opération d'inversion ne se produit en retirant les éléments à la fin que si la liste de sortie est vide. C'est pourquoi on dit que la structure fonctionne en temps constant « amorti ». Voici un exemple :

```
A = Queue[Int](Nil, Nil)     // in = Nil, out = Nil
B = A.enqueue(1)             // = Queue(1->Nil, Nil)
C = B.enqueue(2)             // = Queue(2->1->Nil, Nil)
D = C.enqueue(3)             // = Queue(3->2->1->Nil, Nil)
(x, E) = D.dequeue()         // = (1, Queue(Nil, 2->3->Nil) ici 'in' est inversée et placée
en `out`
(y, F) = E.dequeue()         // = (2, Queue(Nil, 3->Nil)
(z, G) = F.dequeue()         // = (3, Queue(Nil, Nil))
G.isEmpty                    // = True
```

Voici le prototype d'une telle structure de donnée :

```
/** Une classe liste FIFO. */
case class Queue[T](in:List[T] = Nil, out:List[T] = Nil) {
    /** Ajoute un élément `x` en tête. */
    def enqueue(x:T):Queue[T] = ???
    /** Retire le dernier élément. */
    def dequeue():(T,Queue[T]) = ???
    /** Accès au premier élément, s'il existe. */
    def headOption():Option[T] = ???
    /** Vrai si la liste est vide. */
    def isEmpty:Boolean = in.isEmpty && out.isEmpty
}
```

Travail à réaliser

- Implantez les méthodes de la classe `Queue`.
- Implantez une méthode `length` qui retourne la taille de le queue.
- Implantez une méthode `rearOption` qui retourne le dernier élément de la queue sans la modifier.
- Implantez une méthode `toList` qui convertit la `Queue` en liste chaînée.
- Implantez une méthode `map` sur `Queue`.
- Implantez une méthode `foldLeft` sur `Queue`.
- La méthode `dequeue` peut ne pas fonctionner telle que spécifié. Corrigez la.

Rendez le projet sur Eureka sous forme d'archive nommée avec vos noms et prénoms, et contenant : les sources, un ensemble de tests le plus exhaustif possible, et un rapport d'une page recto-verso maximum décrivant l'implantation réalisée, les difficultés, les méthodes utilisée, et tout détail d'utilisation que vous jugerez nécessaire.