Université de Paris

L2 informatique - Année 2020–2021

TD d'Éléments d'Algorithmique n° 8

Une file est une structure de données *abstraite* sur laquelle sont définies trois opérations :

- `empty(F)` qui teste si la file `F` est vide ;
- `put(x,F)` qui ajoute un élément `x` en queue de la file `F` ;
- `get(F)` qui enlève l'élément en tête de la file `F` et le renvoie.

Exercice 1. *Sedgewick.*
Dans la suite suivante, une lettre indique un *put* et un astérisque indique un *get*. Donnez la suite de *get* exécutés lorsqu'on applique à une file (initialement vide) la suite d'opérations indiquée :

```
E A S * Y * Q U E * * * S T * * * I O * N * * *
```

Exercice 2. *Listes avec pointeur de queue.*

1. Implémentez les files à l'aide d'une liste chaînée où la tête de la file est en tête de la liste. Quel problème constatez-vous ?

Pour résoudre ce problème, on se propose d'utiliser des *listes avec pointeur de queue*. Une telle liste est représentée par une structure `ListEnd` dans laquelle la structure `Cellule` est utilisée pour représenter les éléments ou les nœuds de la liste :

```
1 class Cellule {
2   Object val
3   Cellule next
4 }
```

```
1 class ListEnd {
2   Cellule first
3   Cellule last
4 }
```

Le champ `first` contient une référence à la première cellule, comme pour une liste ordinaire. Le champ `last` contient une référence à la dernière cellule. (Si la liste est vide, les deux champs valent `null`.)

2. Implémentez les files à l'aide de listes avec pointeur de queue.

On se propose maintenant d'implémenter les files avec deux listes ordinaires :

```
1 class List2 {
2   Cellule first
3   Cellule second
4 }
```

On enfile les éléments dans la liste `first`, et on les défile depuis la liste `second`. Lorsque `second` est vide, on inverse la liste `first` et on la stocke dans `second`.

3. Implémentez les files à l'aide de deux listes. Combien faut-il de temps pour enfiler un élément ? Pour défiler un élément ? Pour enfiler n éléments et les défiler tous ?
4. Quels sont les avantages et les défauts de chacune de ces implémentations ?

Exercice 3. *File bornée.*
On suppose que la file a une taille maximale de N. Proposez une implémentation de la file avec un tableau. Quelle est la complexité en nombre d'opérations d'un `get()` et d'un `put()` dans votre implémentation. Proposez, si ce n'est pas le cas, une implémentation de la file telle que le nombre d'opération pour le `get()` et le `put()` soit constant. **Exercice 4.** *listes circulaires.*

Une liste est circulaire si elle contient $n \geq 1$ cellules `c1`, `c2` ..., `cn` telles que `c1.next=c2`, `c2.next=c3`,...,`cn.next=c1`.

1. Dessinez chacune des listes créées par les suites d'instructions suivantes, et indiquez si elle est circulaire.

```
a := new Cellule(1, null)
b := new Cellule(2, a)
c := new Cellule(3, a)
a.next := c
b.val := 4
L := new List(b)
```

```
a := new Cellule(1, null)
b := new Cellule(2, a)
c := new Cellule(3, null)
a.next := c
b.val := 4
L := new List(b)
```

```
a := new Cellule(1, null)
a.next := a
L := new List(a)
```

2. Écrivez une fonction qui calcule la longueur d'une liste (ordinaire). Comment se comporte votre fonction si on lui passe une liste circulaire ?
3. Écrivez une fonction `est-circulaire` qui prend une liste en paramètre et retourne vrai si et seulement si cette liste est circulaire. Vous pourrez vous servir d'une liste chaînée pour conserver les cellules déjà vues. Pouvez-vous éviter de consommer de l'espace en plus ?
4. Même question en espace constant et temps linéaire. (Vous pourrez par exemple utiliser deux références à des cellules, dont une qui avance deux fois plus vite que l'autre.)