Programmation avancée

Répétition 3: Pile, file, liste, vecteur et séquence

Jean-Michel Begon

23 octobre 2015

Exercice 1

1. Soient deux nœuds `n` et `m` de listes simplement liées. Que provoquent les opérations suivantes sur la liste auquel `n` appartient (mais pas `m`) ?
   (a) `n.next = n.next.next`
   (b) `m.next = n.next; n.next = m`
   (c) `n.next = m; m.next = n.next`
2. Soit `L` une liste simplement liée.
   (a) Ecrire une fonction `AvantAvantDernier(L)` permettant de récupérer l'avant-avant dernier noeud de `L`.
   (b) Ecrire une fonction `Reverse(L)` permettant d'inverser (en place) la liste `L`.
   (c) Ecrire une fonction `Delete(L, n)` supprimant le noeud `n` de la liste `L`.

Exercice 2

1. Soit une pile `S`. Illustrer l'état de `S` au fil des instructions.

```
push(S, 5)
push(S, 3)
pop(S)
push(S, 2)
push(S, 8)
pop(S)
pop(S)
push(S, 9)
push(S, 1)
pop(S)
push(S, 7)
push(S, 6)
pop(S)
pop(S)
push(S, 4)
pop(S)
pop(S)
```

2. Soit une file `Q`. Illustrer l'état de `Q` au fil des instructions.

```
enqueue(Q, 5)
enqueue(Q, 3)
dequeue(Q)
enqueue(Q, 2)
enqueue(Q, 8)
dequeue(Q)
dequeue(Q)
enqueue(Q, 9)
enqueue(Q, 1)
dequeue(Q)
enqueue(Q, 7)
enqueue(Q, 6)
dequeue(Q)
dequeue(Q)
enqueue(Q, 4)
dequeue(Q)
dequeue(Q)
```

Exercice 3

Concevez une structure de données contenant 2 piles et implémentée avec un seul tableau. Implémentez les opérations `push` et `pop`.

Exercice 4

Réécrire les fonctions `Enqueue(Q, x)` et `Dequeue(Q)` du cours de sorte à gérer les erreurs. Considérer l'implémentation par tableau et l'implémentation par liste liée.

Exercice 5

1. Implémenter une file à l'aide de deux piles. Quelle est la complexité des opérations `Enqueue` et `Dequeue` ?
2. Implémenter une pile à l'aide deux files. Quelle est la complexité des opérations `push` et `pop` ?

Bonus

Bonus 1

Dans de nombreux domaines de l'informatique appliquée (bioinformatique, optimisation, etc.), on souhaite modéliser un effet de mémoire de capacité bornée où les données les plus récentes remplacent les plus anciennes. Par exemple, pour une mémoire de capacité 7, on aurait :

```
M = create-memroy(7)
store(M, 1)
store(M, 2)
store(M, 3)
store(M, 4)
store(M, 5)
store(M, 5)
store(M, 7)
store(M, 8)
print-memory(M)
>>> 2, 3, 4, 5, 5, 7, 8 //L'ordre est préservé
```

1. Proposer une structure de données pour ce cas d'utilisation qui permettrait un accès rapide aux éléments par leur index courant.
2. On aimerait rajouter la caractéristique de doubler spontanément la capacité de la mémoire s'il y a $2n$ ajouts d'éléments consécutifs sans lecture de la mémoire. Adapter ou proposer une structure de données pour ce cas d'utilisation.

Bonus 2

On souhaiterait construire une file à priorité simplifiée. Elle dispose des trois opérations suivantes

— `append(Q, x)` : ajoute l'élément x à la fin de la file Q.
— `preprend(Q, x)` : ajoute l'élément x au début de la file Q.
— `dequeue(Q)` : retourne le premier élément de la file Q.

Proposer une structure de données pour répondre à ce cas d'utilisation.