Programmation avancée

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

Jean-Michel Begon

novembre 2014

Exercice 1

(a) Soient deux noeuds `n` et `m` de listes simplement liées. Que provoquent les opérations suivantes sur la liste auquel `n` appartient (mais pas `m`) ?

1. `n.next = n.next.next`
2. `m.next = n.next; n.next = m`
3. `n.next = m; m.next = n.next`

(b) Soit `S` une liste simplement liée. Ecrire une fonction `AvantAvantDernier(S)` permettant de récupérer l'avant-avant dernier noeud de `S`.

(c) Soit `S` une liste simplement liée. Ecrire une fonction `Reverse(S)` permettant d'inverser (en place) la liste `S`.

(d) Soit `S` une liste simplement liée. Ecrire une fonction `Delete(S, n)` supprimant le noeud `n` de la liste `S`.

Exercice 2

(a) Soit une pile `S`. Décrire la sortie des opérations suivantes. Quel est l'état de `S` en fin d'exécution ?

```
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)
```

(b) Soit une file `Q`. Décrire la sortie des opérations suivantes. Quel est l'état de `Q` en fin d'exécution ?

```
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

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

Exercice 6

Modifier l'implémentation de `RemoveAtRank(V, r)` de sorte à réduire par 2 l'espace mémoire utilisé par le vecteur `V` si le nombre d'éléments qu'il contient devient inférieur à `V.c / 4` (où `V.c` est la capacité courante de `V`).

Exercice 7

Implémenter une structure efficace qui réponde au TDA d'une séquence :
— `Insert-Before(S, p, x)` : insère x avant p dans la séquence.
— `Insert-After(S, p, x)` : insère x après p dans la séquence.
— `Remove(S, p)` : retire l'élément à la position p.
— `Replace(S, p, x)` : remplace par l'objet x l'objet situé 'à la position p.
— `First(S)`, `Last(S)` : renvoie la première, resp. dernière position dans la séquence.

— `Prev(S, p)`, `Next(S, p)` : renvoie la position précédant (resp. suivant) p dans la séquence.
— `Elem-At-Rank(S,r)` : retourne l'élément au rang r dans la séquence.
— `Replace-At-Rank(S,r, x)` : remplace l'élément situé au rang r par x et retourne cet objet.
— `Insert-At-Rank(S,r, x)` : insère l'élément x au rang r, en augmentant le rang des objets suivants.
— `Remove-At-Rank(S,r)` : extrait l'élément situé au rang r et le retire de la séquence, en diminuant le rang des objets suivants.
— `Size(S)` : renvoie la taille de la séquence.
— `At-Rank(S,r)` : retourne la position de l'élément possédant le rang r.
— `Rank-Of(S, p)` : retourne le rang de l'élément situé à la position p.

Note : Vous pouvez utiliser d'autres structures comme éléments de base.

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
```

(a) Proposer une structure de données pour ce cas d'utilisation qui permettrait un accès rapide aux éléments par leur index courant.

(b) 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.