IFT2015 :: hiver 2020

Cours de structures de données, Université de Montréal

JANVIER 13, 2020JANVIER 18, 2020 *by* CSUROSM

Tableaux: exercices

1. Exercices: tableau

1.1 Renverser un tableau

Donner un algorithme qui renverse l'ordre des éléments en place (donc O(1) espace de travail, aucun tableau auxiliaire).

Indice: utiliser deux indices avançant en sync à partir des extrémités vers le milieu $i \leftarrow 0, 1, \ldots$ et $j \leftarrow n-1, n-2, \ldots$ pour échanger les éléments.

1.2 Maximum local

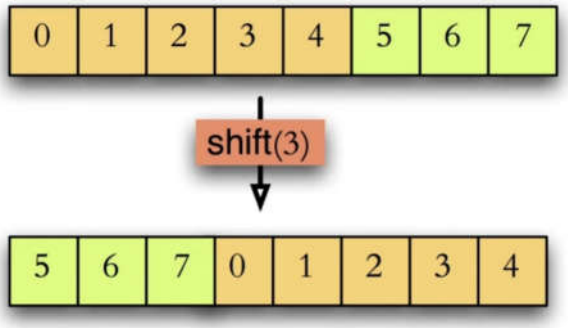
Dans un tableau $T[0..n-1]$, il y a un maximum local à $0 < k < n-1$ si $T[k-1], T[k+1] < T[k]$. Donner un algorithme qui trouve tous les maximums locaux.

1.3 Suppressions

Écrire une méthode delete$(T[0..n-1], x)$ qui supprime toute occurrence de x dans le tableau T en un seul parcours. La méthode retourne le nombre des éléments m qui restent; ces derniers se décalent vers la gauche pour occuper les cases $0..m-1$. (Le contenu des cases $m..n-1$ n'est pas important.)
$T = \{1,2,3,1,2,3,1,2,3\}$; delete$(T,3) \Rightarrow T = \{1,2,1,2,1,2,?,?,?\}$, retourne 6.

1.4 Décalage circulaire

Supposons qu'on veut faire des décalages circulaires dans un tableau $A[0..n-1]$. Un décalage par δ correspond à l'exécution *parallèle* des affectations $A[(i+\delta) \bmod n] \leftarrow A[i]$.



Quand $\delta = 1$, il est clair qu'on peut performer le décalage en place par une boucle simple:

```
Shift1(A[0..n-1])
{
    i ← 0; navette ← A[0];
    do
    {
        i ← (i+1) mod n;
        échanger navette ↔ A[i]
    } while (i≠0)
}
```

Donner un algorithme Shift(A, δ) qui exécute un décalage circulaire à δ positions **en place**. L'algorithme doit utiliser un espace de travail constant (excluant l'entrée $A[0..n-1]$) , et s'exécuter en un temps linéaire (en n), pour tout choix de δ. En particulier on n'a pas le droit d'utiliser un tableau auxiliaire de δ éléments.

Indice: Qu'est-ce qui se passe si on écrit $i \leftarrow (i+\delta) \bmod n$ dans la boucle au lieu de $i \leftarrow (i+1) \bmod n$?

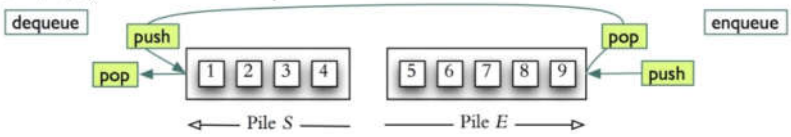
2. Exercices: queue et pile

2.1 Queue avec gestion de la taille

Montrer une implémentation de queue FIFO basée sur un tableau avec expansion/réduction dynamique.
Indice: Faites attention au copiage des éléments quand un nouveau tableau est alloué.

2.2 Deux c'est mieux

- Implanter une queue (file FIFO) avec les opérations $\mathsf{enqueue}(x)$ et $\mathsf{dequeue}()$, en utilisant deux piles.
  Indice: utilisez les deux piles pour stocker les éléments aux deux extremités.
  Dépilez/empilez ($E \to S$) seulement quand cela devient nécessaire.



- Ajouter l'opération $\mathsf{delete}(k)$ qui défile $k > 0$ fois ou arrête plus tôt quand la queue est vide. L'implantation ne devrait jamais mener à un débordement des piles sous-jacentes. L'opération ne retourne pas de valeur.
- Quel est le temps de calcul des opérations $\mathsf{enqueue}$ et $\mathsf{delete}(k)$ dans le pire cas? Donner une implantation qui garantit que le temps de calcul est au plus linéaire en m pour n'importe quelle séquence de m opérations — autrement dit, que le coût amorti des opérations est constant. Démontrer une borne spécifique.
  Indice: Montez une preuve crédit-débit.

Publicités