

TD2 Listes

VAN DE MERGHEL Robin

2023

Table des matières

Exercice 2.1	1
Question 1	1
Méthode 1 : Tableau dans un intervalle	2
Méthode 2 : Tableau où l'on cherche les cases libres	3
Question 2	4
Question 3	6
Exercice 2.3	7
Question 1	7

Exercice 2.1

Question 1

Proposer une implémentation des listes chaînées avec un tableau (le suivant dans la liste n'est pas forcément le suivant dans le tableau). Pensez à gérer la liste des cases libres.

0	1	2	3	4	5	6	7	8	9
				5	2	8	9		

1. Suivant dans la liste c'est le suivant dans le tableau
2. Suivant dans la liste n'est pas forcément le suivant dans le tableau

0	1	2	3	4	5	6	7	8	9
9		5			8				5
-1		9			2				8

Il faut gérer dans la liste l'ensemble des cases du tableau qui ne contiennent pas un élément de la liste libre : $\{ 1, 3, 6, 7, 8 \}$

Ex : 1. `insérerDebut(1, 20)`

- On commence par chercher un index libre dans la liste des cases libres
- 2. Si un élément est supprimé de la liste, son index dans le tableau doit être inséré dans l'ensemble libres

Fonctions linéaire :

- `créerListe()`
- `debutListe(l)`
- `estVide(l)`
- `ajouterDebut(l, e)`

- ajouterFin(l, e)
- supprimerDebut(l)
- supprimerFin(l)

3. Faire de même avec les cellules (que l'on demande au système), la liste n'est pas stockée dans un tableau.

On implémente les deux méthodes :

Méthode 1 : Tableau dans un intervalle

```
function creerListe() {
    n = 42
    Liste l
    tab = new int[n]
    l.tab = tab
    l.taille = 0
    l.debut = 0
    l.fin = 0
    return l
}

function estVide(l) {
    return l.taille == 0
}

function debutListe(l) {
    if (estVide(l)) {
        return -1
    }
    return l.debut
}

function ajouterDebut(l, e) {
    if (l.taille == n) {
        return -1
    }
    l.tab[l.debut] = e
    l.debut = (l.debut + 1) % n
    l.taille = l.taille + 1
    return 0
}

function ajouterFin(l, e) {
    if (l.taille == n) {
        return -1
    }
    l.tab[l.fin] = e
    l.fin = (l.fin - 1) % n
    l.taille = l.taille + 1
    return 0
}

function supprimerDebut(l) {
    if (estVide(l)) {
        return -1
    }
    l.debut = (l.debut - 1) % n
}
```

```

    l.taille = l.taille - 1
    return 0
}

function supprimerFin(l) {
    if (estVide(l)) {
        return -1
    }
    l.fin = (l.fin + 1) % n
    l.taille = l.taille - 1
    return 0
}

```

Méthode 2 : Tableau où l'on cherche les cases libres

```

function creerListe() {
    n = 42
    Liste l = new Liste()
    tab = new int[n]
    l.tab = tab
    File f = new File()
    for (i = 0; i < n; i++) {
        ajouterFin(f, i)
    }
    l.libres = f
    l.taille = 0
    l.debut = 0
    l.fin = 0
    return l
}

function estVide(l) {
    return l.taille == 0
}

function debutListe(l) {
    if (estVide(l)) {
        return -1
    }
    return l.debut
}

function ajouterDebut(l, e) {
    if (estVide(l.libres)) {
        return -1
    }
    i = supprimerDebut(l.libres)
    l.tab[i] = e
    l.debut = i
    l.taille = l.taille + 1
    return 0
}

function ajouterFin(l, e) {
    if (estVide(l.libres)) {
        return -1
    }
}

```

```

    i = supprimerDebut(l.libres)
    l.tab[i] = e
    l.fin = i
    l.taille = l.taille + 1
    return 0
}

function supprimerDebut(l) {
    if (estVide(l)) {
        return -1
    }
    ajouterFin(l.libres, l.debut)
    l.debut = l.tab[l.debut]
    l.taille = (l.taille - 1) % n
    return 0
}

function supprimerFin(l) {
    if (estVide(l)) {
        return -1
    }
    ajouterFin(l.libres, l.fin)
    l.fin = l.tab[l.fin]
    l.taille = (l.taille - 1) % n
    return 0
}

```

Question 2

Faire de même avec les pointeurs.

On redéfinit la structure d'une cellule, on a un pointeur sur son suivant :

```

enregistrement Cellule {
    T val
    Cellule suiv
}

```

Le système a deux fonctions :

- `allocation()` : alloue une cellule (`malloc` en C, `new` en Java)
- `liberation(c)` : libère une cellule (`free` en C, automatique en Java)

On créera une liste avec un pointeur sur sa première cellule :

```

enregistrement Liste {
    Cellule debutListe
}

```

Pour le code concret, on doit créer une cellule constante qui représentera la fin de la liste (comme le `null` en Java, ou le `None` en Python) :

`Cellule cVide = ...` *# Dépend du langage*

En C on peut faire :
Cellule cVide = NULL

En Java on peut faire :
Cellule cVide = null

```
# En Python on peut faire :  
# cVide = None
```

Maintenant, on peut implémenter les fonctions :

```
Liste creerListe() {  
    Liste l = new Liste()  
    l.debutListe = cVide  
    return l  
}  
  
bool estVideListe(Liste l) {  
    return l.debutListe == cVide  
}  
  
Cellule debutListe(Liste l) {  
    if (estVideListe(l)) {  
        return cVide  
    }  
    return l.debutListe  
}  
  
void ajouterDebut(Liste l, T e) {  
    Cellule c = allocation()  
  
    # En C on vérifie si l'allocation a réussi  
    # if (c == NULL) {  
    #     printf("Erreur d'allocation\n")  
    #     exit(1)  
    # }  
  
    c.val = e  
    c.suiv = l.debutListe  
    l.debutListe = c  
}  
  
void ajouterFin(Liste l, T e) {  
    Cellule c = allocation()  
    c.val = e  
    c.suiv = cVide  
  
    if (estVideListe(l)) {  
        l.debutListe = c  
    } else {  
        Cellule c2 = l.debutListe  
        while (c2.suiv != cVide) {  
            c2 = c2.suiv  
        }  
        c2.suiv = c  
    }  
}  
  
Liste supprimerDebut(Liste l) {  
    if (estVideListe(l)) {  
        return l  
    }
```

```

    }
    Cellule c = l.debutListe
    l.debutListe = c.suiv
    liberation(c)
    return l
}

Liste supprimerFin(Liste l) {
    if (estVideListe(l)) {
        return l
    }
    if (l.debutListe.suiv == cVide) {
        liberation(l.debutListe)
        l.debutListe = cVide
        return l
    }
    Cellule c = l.debutListe
    while (c.suiv.suiv != cVide) {
        c = c.suiv
    }
    liberation(c.suiv)
    c.suiv = cVide
    return l
}

```

*# On pourrait faire une fonction auxiliaire pour aller au dernier élément
 # Et l'appeler dans ajouterFin et supprimerFin, car le code est le même*

Question 3

Comment modifier vos implémentations pour implémenter une liste chaînée bi-directionnelle ? Une liste circulaire ?

Pour une liste chaînée bi-directionnelle, on peut modifier l'enregistrement `Cellule` pour avoir un pointeur sur la cellule précédente :

```

enregistrement Cellule {
    T val
    Cellule suiv
    Cellule prec
}

```

Un exemple de représentation :



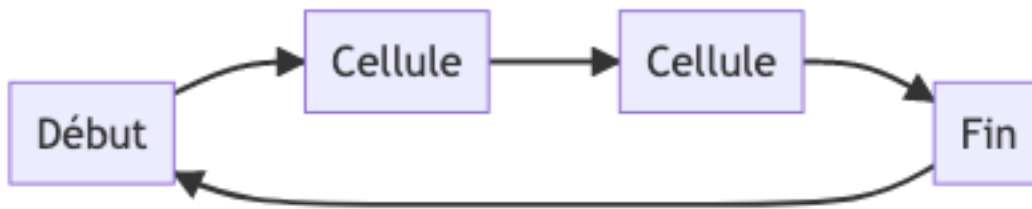
Pour une liste circulaire, peut modifier l'enregistrement `Liste` pour avoir un pointeur sur la dernière cellule :

```

enregistrement Liste {
    Cellule debutListe
    Cellule finListe
}

```

Un exemple de représentation :



Exercice 2.3

Question 1

Écrire une procédure de concaténation de deux listes en temps $O(1)$

On rappelle notre implémentation : on a une liste chaînée avec pointeur sur le début et la fin de la liste.

```
enregistrement Liste {  
    Cellule debutListe  
    Cellule finListe  
}
```

On peut donc concaténer deux listes en faisant pointer la fin de la première liste sur le début de la deuxième liste :

```
Liste concatener(Liste l1, Liste l2) {  
    if (estVideListe(l1)) {  
        return l2  
    }  
    if (estVideListe(l2)) {  
        return l1  
    }  
    l1.finListe.suiv = l2.debutListe  
    l1.finListe = l2.finListe  
    return l1  
}
```

Le temps est bien en $O(1)$ car on ne fait que modifier les pointeurs.