EXOOD4 Tutorials
Video games development

Les listes chaînées

ALGORITHMIE

Structures de données

Table des matières

La liste est la forme la plus usuelle des structures de données. Une liste est une suite finie, éventuellement vide, d'éléments de même type. L'ordre des éléments dans une liste est sans importance. Par contre, la place de chaque élément est importante.

Si $[e_1, e_2, \ldots, e_n]$ représentent les éléments d'une liste, cette liste est notée :

$$L = [e_1, e_2, \ldots, e_n]$$

l'élement e_2 succède à l'élément e_1 et précède e_3 . La notion de place implique souvent des algorithmes séquentiels sur les éléments d'une liste.

1 Spécification fonctionnelle d'une Liste

1.1 Opérations

Soient **LISTE** le type liste et **ELEMENT** le type des éléments de la liste.

Type d'opérations	Profil et définition de l'opération
construction	**liste_vide** : → **LISTE** constante désignant une liste vide
consultation	**longueur** : **LISTE** → **ENTIER** fournit la longueur d'une liste **L**
	ieme : **LISTE** x **ENTIER** → **ELEMENT** fournit l'élément de rang **k** d'une liste **L**
modification	**insérer** : **LISTE** x **ENTIER** x **ELEMENT** → **LISTE** modifie l'élément de rang **k** d'une liste **L** par un élément **e**
	supprimer : **LISTE** x **ENTIER** → **LISTE** suppression de l'élément de rang **k** d'une liste **L**

→ : retour de la fonction

x : séparateur de paramètre (LISTE x ENTIER signifie que la fonction admet un paramètre de type LISTE et un autre de type ENTIER)

1.2 Propriétés

Les opérations **ieme** et **supprimer** ne sont pas définies pour les listes vides. On parle alors d'opérations partielles. Pour les appliquer, il est nécessaire de définir des préconditions.

opération	précondition
ieme (L,i)	L ? liste_vide
supprimer(L,i)	L ? liste_vide

En supposant les préconditions vérifiées, on peut écrire les propriétés suivantes :

- **longueur** (**liste_vide**) = 0
- **longueur** (**insérer** (L, k, e)) = **longueur** (L) + 1
- **longueur** (**supprimer** (L, k, e)) = **longueur** (L) – 1
- 1 = i < k → **ieme** (**insérer** (L, k, e), i) = ieme (L, i)
  si on insère un élément **e** dans une liste **L** au rang **k**, **k** plus grand que **i**, la place de l'élément de rang **i** n'est pas modifié
- **ieme** (**insérer** (L, k, e), k) = e
- k < i = **longueur** (L) + 1 → **ieme** (**insérer** (L, k, e), i) = **ieme** (L, i-1)
  si on insère un élément **e** dans une liste **L** au rang **k**, **k** plus petit que **i**, la place de l'élément de rang **i** est celle qui se trouvait au rang **i-1**
- 1 = i < k → **ieme** (**supprimer** (L, k), i) = **ieme** (L, i)
  si l'on supprime un élément de la liste **L** au rang **k**, **k** plus grand que **i**, la place de l'élément **i** n'est pas modifié
- 1 = k < i = **longueur** (L) – 1 → **ieme** (**supprimer** (L, k), i) = **ieme** (L, i+1)
  si l'on supprime dans la liste **L**, l'élément de rang **k**, **k** plus petit que **i**, la place de l'élément **i** est celle qui se trouvait au rang **i+1**

2 Représentation physique

2.1 Représentation statique

Les éléments de la liste sont mémorisés dans un tableau. On a besoin également d'une variable pour mémoriser le nombre significatif d'éléments du tableau (à cause des opérations d'insertion et de suppression). On regroupe ces deux informations dans un enregistrement :

VALEURS	e_1	e_2		e_i		e_n			
NB_VALEURS	n								

On écrit alors :

Enregistrement liste composé de 2 champs :
VALEURS : tableau de lg_max ELEMENTS
NB_VALEURS : variable indiquant le nombre d'éléments dans le tableau VALEURS

Remarque :

- La fonction d'accès à un élément de la liste, est la fonction d'accès des tableaux. Si L est de type LISTE, l'accès à un élément s'écrit : L.VALEURS[i]

- La fonction longueur est directement codée, dans cette représentation, par la variable NB_VALEURS

Les seules opérations complexes à programmer sont l'insertion et la suppression car il est nécessaire de décaler les éléments du tableau

Exemple : Supprimer

```
// supprimer l'élément de rang k d'une liste L
SI la liste L n'est pas vide ALORS
        SI le rang k de suppression est valide ALORS
                décaler les éléments e(k+1), …, e(n) vers la gauche;
        SINON
                afficher le message d'erreur "rang non valide";
        FIN SI;
SINON
        afficher le message d'erreur "liste vide";
FIN SI;
```

Dans un pseudo langage de programmation, on écrira :

```
// supprimer l'élément de rang K d'une liste L
PROCEDURE supprimer (MISE A JOUR L, ENTREE K)
GLOSSAIRE
        L : enregistrement composé de 2 champs :
                un tableau VALEURS et une variable NB_VALEURS;
        K : rang de l'élément à supprimer;
        I : variable de parcours du tableau L.VALEURS pour les décalages;
DEBUT
        SI L.NB_VALEURS != 0 ALORS
                SI K >=1 ET K <= L.NB_VALEURS ALORS
                        // décaler les éléments e(k+1), …, e(n) vers la gauche
                        I ← K;
                        TANT QUE I < L.NB_VALEURS FAIRE
                                L.VALEURS[I] ← L.VALEURS[I+1];
                                I ← I + 1;
                        FIN TANT QUE;
                        // décrémenter le nombre d'éléments de la liste L
                        L.NB_VALEURS ← L.NB_VALEURS - 1;
                SINON
                        ERREUR("rang non valide");
                FIN SI;
        SINON
                ERREUR("liste vide");
        FIN SI;
FIN
```

2.2 Représentation dynamique

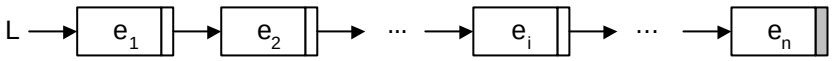
En représentation dynamique, on représente chaque élément de la liste par une cellule en mémoire :

Une cellule = un enregistrement composé de 2 champs :

- 1 champ VALEUR

- 1 champ SUIVANT : adresse (pointeur) de la cellule suivante

La liste en elle-même sera représentée par un pointeur sur la première cellule.



On déclarera :

Pointeur Liste sur enregistrement CELLULE;
Enregistrement CELLULE composé de 2 champs
VALEUR (de type ELEMENT) et
SUIVANT (de type pointeur sur enregistrement CELLULE);

Nous avons ici un cas de structure de données **récursive** (SUIVANT étant un pointeur sur enregistrement de même type)

Remarque

- Pour accéder à l'élément e_i il faudra balayer séquentiellement les i-1 éléments qui précédent (s'ils existent)

- Le passage d'une cellule à la cellule suivante, si L désigne l'adresse de la cellule courante, s'écrira :

L ← L^.SUIVANT

Algorithme de l'opération supprimer

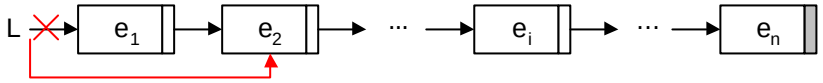
```
// supprimer l'élément de rang k d'une liste L
SI la liste L n'est pas vide ALORS
        SI le rang k est égal à 1 ALORS
                supprimer la première cellule;
        SINON
                rechercher séquentiellement la cellule k;
                SI cette cellule k existe ALORS
                        supprimer la cellule;
                SINON
                        afficher le message d'erreur "rang incorrect";
                FIN SI;
        FIN SI;
SINON
        afficher le message d'erreur "liste vide";
FIN SI;
```

Pour traduire cette algorithme en pseudo langage de programmation, il est nécessaire de déclarer 2 pointeurs pour parcourir la liste :

PTR_PRECEDENT
PTR_COURANT

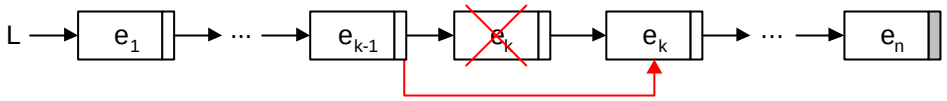
Deux cas de figure se présentent :

- Suppression en tête de liste :



L ← L^.SUIVANT;

- Suppression dans la liste :



PTR_PRECEDENT^.SUIVANT ← PTR_COURANT^.SUIVANT;

D'où l'écriture de la fonction supprimer :

```
// supprimer l'élément de rang K d'une liste L
PROCEDURE supprimer (MISE A JOUR L, ENTREE K)
GLOSSAIRE
        L : pointeur sur un enregistrement CELLULE composé de 2 champs
                VALEUR et SUIVANT;
        K : rang de l'élément à supprimer;
        I : compteur de cellules pour la recherche séquentielle de la cellule K;
                PTR_COURANT : pointeur sur la cellule courante;
                PTR_PRECEDENT : pointeur sur la cellule précédente vérifiant
                        PTR_COURANT = PTR_PRECEDENT^.SUIVANT;
DEBUT
        SI L != NULL ALORS
                SI K = 1 ALORS
                        // suppression en tête de liste
                        L ← L^.SUIVANT;
                SINON
                        // suppression dans la liste
                        // rechercher l'élément de rang K de la liste
                        I ← 1;
                        PTR_COURANT ← L;
                        TANT QUE (I <= K) ET (PTR_COURANT != NULL) FAIRE
                                I ← I + 1;
                                PTR_PRECEDENT ← PTR_COURANT;
                        PTR_COURANT ← PTR_COURANT^.SUIVANT;
                        FIN TANT QUE;
                        SI PTR_COURANT != NULL ALORS
                                // la cellule de rang K existe, la supprimer
                                PTR_PRECEDENT^.SUIVANT ← PTR_COURANT^.SUIVANT;
                        SINON
                                ERREUR("Rang invalide");
                        FIN SI;
                FIN SI;
        SINON
                ERREUR("Liste vide");
        FIN SI;
FIN
```