DOSSIER ALGORITHMIQUE

Insertion dans une liste doublement chaînée triée

1. Problème

On se propose d'implémenter l'insertion d'un élément dans une liste doublement chaînée triée.

Une liste doublement chaînée triée est une structure de données composée de nœuds reliés entre eux par deux pointeurs :
- suiv (vers l'élément suivant)
- prec (vers l'élément précédent)

La particularité de cette liste est que ses éléments sont toujours maintenus dans un ordre croissant. Chaque insertion doit donc placer le nouvel élément à la bonne position pour préserver le tri.

2. Principe algorithmique

Pour insérer un élément dans une liste doublement chaînée triée :
1. Créer un nouveau nœud contenant la valeur à insérer.
2. Si la liste est vide, le nouvel élément devient la tête.
3. Si la valeur est plus petite que la valeur du premier nœud, insérer en tête.
4. Sinon, parcourir la liste jusqu'à trouver la première valeur supérieure ou égale à celle à insérer.
5. Insérer le nouveau nœud avant cette valeur.
6. Si aucune valeur n'est plus grande, insérer en fin de liste.

3. Dictionnaire des données

Nom	Type	Rôle
Noeud	Structure	Représente un élément de la liste
val	entier	Valeur contenue dans un nœud
suiv	pointeur	Pointeur vers le nœud suivant
prec	pointeur	Pointeur vers le nœud

		précédent
P	pointeur	Pointe sur le premier élément de la liste (tête)
nouveau	pointeur	Nœud à insérer
courant	pointeur	Sert au parcours de la liste

4. Algorithme

pseudo

```
type Noeud = ^Element
Element = enregistrement
   val : entier
   suiv : Noeud
   prec : Noeud
fin enregistrement

procedure inserer_trie(P : Noeud, val : entier)
var nouveau, courant : Noeud

début
   // Création du nouveau nœud
   nouveau ← allouer Noeud
   nouveau^.val ← val
   nouveau^.suiv ← nul
   nouveau^.prec ← nul

   // Cas 1 : liste vide
   si P = nul alors
      P ← nouveau
      retourner

   // Cas 2 : insertion en tête
   si val <= P^.val alors
      nouveau^.suiv ← P
      P^.prec ← nouveau
      P ← nouveau
      retourner

   // Cas 3 : parcours pour trouver la position
   courant ← P
   tant que courant^.suiv ≠ nul et courant^.suiv^.val < val faire
```

```
    courant ← courant^.suiv
  fin tant que

  // Insertion après courant
  nouveau^.suiv ← courant^.suiv
  nouveau^.prec ← courant

  si courant^.suiv ≠ nul alors
    courant^.suiv^.prec ← nouveau
  fin si

  courant^.suiv ← nouveau
fin
```

Exemple d'exécution

- Liste initiale : [5] ⇆ [10] ⇆ [20] ⇆ [30]
- Opération : insérer 15
- Résultat : [5] ⇆ [10] ⇆ [15] ⇆ [20] ⇆ [30]

5. Complexité

Complexité temporelle :

- Cas meilleur (insertion en tête) : O(1)
- Cas moyen/pire (parcours de la liste) : O(n)

Complexité spatiale :

- O(1) → seulement un nouveau nœud est alloué