📂 Dossier Algorithmique : Insertion dans une liste chaînée triée

# 1. Problème

On souhaite insérer un élément dans une liste simplement chaînée de sorte que la liste reste toujours triée dans l’ordre croissant. L’objectif est de maintenir l’ordre tout en minimisant le parcours de la liste.

# 2. Principe

Le principe de l’insertion dans une liste chaînée triée repose sur :  
- Créer un nouveau nœud contenant la valeur à insérer.  
- Si la liste est vide ou que la valeur est plus petite que la tête, insérer en début.  
- Sinon, parcourir la liste jusqu’à trouver la bonne position.  
- Insérer le nœud en reliant correctement les pointeurs.

# 3. Dictionnaire de données

|  |  |  |
| --- | --- | --- |
| Nom | Type | Rôle |
| head | Pointeur (Node\*) | Pointe vers le premier élément de la liste |
| data | Entier (int) | Valeur contenue dans un nœud |
| next | Pointeur (Node\*) | Lien vers le nœud suivant |
| newNode | Node\* | Nouveau nœud à insérer |
| current | Node\* | Pointeur utilisé pour parcourir la liste |

# 4. Algorithme (pseudo-code)

ALGORITHME InsererListeTriee  
ENTREE : une liste chaînée L, un entier x  
SORTIE : la liste L avec x inséré en respectant l’ordre trié  
  
1. Créer un nouveau nœud N contenant la valeur x  
2. SI (L est vide) OU (x < valeur du premier nœud)  
 alors insérer N en tête de liste  
 retourner N comme nouvelle tête  
3. Sinon  
 - Initialiser un pointeur courant ← tête  
 - Tant que courant.next ≠ NULL ET courant.next.data < x  
 avancer courant ← courant.next  
 - Insérer N après courant  
4. Retourner la tête de la liste  
FIN ALGORITHME

# 5. Complexité

La complexité dépend de la position d’insertion :  
- Cas meilleur (insertion en tête) : O(1)  
- Cas moyen (au milieu) : O(n)  
- Cas pire (fin de la liste) : O(n)  
=> Complexité globale : O(n) dans le cas général.