STRUCTURES DE DONNÉES ET ALGORITHMES · CHAPITRE 4

Suivre des liens plutôt que des positions

Construisez une liste simplement chaînée avec des emplacements explicites. Distinguez accès, parcours et modification locale des liens.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Suivre une tête et des liens pour retrouver l'ordre logique.
- La tête et les liens définissent l'ordre logique, indépendamment des emplacements de stockage.
- Insérer après un nœud connu peut être constant ; retrouver ce nœud peut rester linéaire.

Prérequis. Avoir étudié les chapitres DSA précédents ; savoir accéder à une collection et suivre des indices.

Partir d'un besoin

Peut-on insérer une valeur sans décaler toutes les suivantes ? Oui, en reliant des éléments séparés. En échange, atteindre la troisième valeur demande de suivre les liens depuis le début.

Définir la structure et les conventions

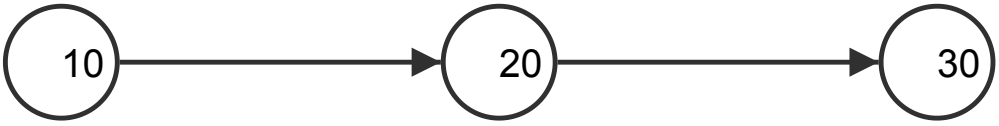
Un **nœud** contient une valeur et un lien vers le nœud suivant. La **tête** désigne le premier nœud. Une **sentinelle** représente l'absence de suivant. La liste simplement chaînée ne possède qu'un lien vers l'avant ; elle n'est pas un tableau de valeurs contiguës dans son modèle usuel.

Nous simulons les nœuds avec deux tableaux parallèles : values contient les valeurs ; links contient les indices des successeurs. Le même indice désigne donc une valeur et son lien. Python utilise -1 pour l'absence de nœud, MATLAB utilise 0 : ces sentinelles sont testées avant

tout accès. Elles ne doivent jamais être utilisées comme un indice.

Les emplacements contiennent [10, 30, 20], mais les liens font lire 10, puis 20, puis 30. Ordre physique et ordre logique sont distincts. Le dernier lien mène à la sentinelle. Une liste valide pour ce parcours ne comporte pas de cycle.

Suivre les états



Liste chaînée : la tête mène à 10, puis 20, puis 30. Chaque lien pointe vers le nœud suivant.

Nœuds et liens en Python

Indice	Valeur	Indice suivant
0	10	2
1	30	-1
2	20	1

Comparer les réalisations

Python construit la sortie par append. MATLAB utilise zeros pour réserver un vecteur de trois zéros, puis remplit ses cases ; ce nombre correspond ici au nombre connu de nœuds accessibles. Le parcours suit les indices 0, 2, 1 en Python et 1, 3, 2 en MATLAB.

Pseudo-code

```
// example.pseudo
// follow_links
// Slot indices start at 0; -1 means no node.
current <- head
WHILE current != -1 DO
    PRINT values[current]
    current <- links[current]
END WHILE
```

Python

```python
# follow_links.py
values = [10, 30, 20]
links = [2, -1, 1]
current = 0
ordered = []
while current != -1:
    ordered.append(values[current])
    current = links[current]
print(ordered)
```

MATLAB

```
% follow_links.m
values = [10, 30, 20];
links = [3, 0, 2];
current = 1;
ordered = zeros(1, numel(values));
count = 0;
while current ~= 0
    count = count + 1;
    ordered(count) = values(current);
    current = links(current);
end
disp(ordered);
```

Justifier et compter

Parcourir n nœuds est Θ(n). Insérer après un nœud déjà connu peut ne modifier que deux liens, donc Θ(1), une fois le nouveau nœud alloué. Rechercher d'abord ce nœud reste Θ(n) au pire. La simulation par tableaux n'élimine pas ses propres coûts d'allocation. Le parcours seul utilise Θ(1) mémoire de travail ; conserver la sortie ajoute Θ(n).

Prévoir puis expérimenter

Modifiez le lien du nœud 20 pour viser la sentinelle : la sortie devient [10, 20]. Ne faites pas pointer le dernier nœud vers la tête sans ajouter une détection de cycle : le parcours ne s'arrêterait plus.

Exercice corrigé

▼ **Quels liens changent pour insérer 15 après 10 ?**

Créez un nœud 15 dont le suivant est l'ancien successeur de 10, puis faites pointer 10 vers 15. Conserver l'ancien successeur évite de perdre le reste de la liste.

À retenir

- La tête et les liens définissent l'ordre logique, indépendamment des emplacements de stockage.
- Insérer après un nœud connu peut être constant ; retrouver ce nœud peut rester linéaire.
- Testez la sentinelle avant tout accès et excluez les cycles si le parcours ne les détecte pas.

Version web : https://www.clearcode.fr/courses/programming/dsa/linear/linked-lists