

UE ALGO5 — TD2 — Séance 2 : Tri par segmentation d'une liste chaînée

Objectifs

À la fin de cette séance, vous devriez être capable de :

- structurer un algorithme en utilisant des sous fonctions ayant une signature et des pré- et post-conditions ;
- manipuler des listes chaînées ;
- écrire des algorithmes récursifs comme le tri par segmentation.

Exercice 1. Tri par segmentation d'une liste chaînée

On rappelle le principe du tri par segmentation :

- 1 TriSegmentation(E):
 - si** la taille **de** E est > 1 **alors**
 - 3 Soit p une valeur «pivot» dans E
 - Partitionner E en E_1, E_2 t.q.:
 - 5 — E_1 contient les valeurs **de** E inférieures ou égales à p
 - E_2 contient les valeurs **de** E supérieures à p
 - 7 TriSegmentation(E_1)
 - TriSegmentation(E_2)
 - 9 Reconstituer E en assemblant E_1, p et E_2

En cours on a vu comment réaliser ce tri dans le cas où E est représenté par un tableau; on s'intéresse ici à traduire cet algorithme dans le cas d'une liste chaînée.

Réalisation

- Q1. Quelle serait une spécification acceptable pour TriSegmentation? (signature, pré- et post-conditions). Attention à tout ce dont on va avoir besoin pour réaliser le tri.

Corrigé —

On peut déjà réfléchir à la signature de TriSegmentation. Comme d'usage dans beaucoup d'algorithmes de tri et de manipulation de liste chaînée, on modifiera par effet de bord la liste reçue en argument. Cela implique qu'il faudra «remballer» les deux sous-listes à trier avant de les communiquer aux appels récursifs.

Quant à la reconstruction de la liste, elle est ici plus subtile qu'avec les tableaux : il faut connaître le dernier maillon de E_1 pour le raccrocher avec la suite. En plus de modifier la liste reçue, on renverra donc l'adresse de son dernier maillon.

- 1 TriSegmentation(donnée L : la liste à trier
résultat D : l'adresse **de** son dernier maillon après le tri)
- 3 { Trie la liste chaînée L par effet de bord.
Pré-condition : aucune
- 5 **Post-condition** :
 - L a conservé les mêmes éléments globalement
 - 7 — ils sont rangés en ordre croissant
 - D est le dernier maillon de la liste triée (non spécifié si L est vide)

-
- Q2. Le partitionnement sera réalisé par une procédure intermédiaire nommée Partition. Donner la spécification de cette procédure (signature, pré- et post-conditions).
-

Corrigé —

L'idée est ici de réfléchir à la spécification sans réaliser tout de suite la procédure.

— Valeur du pivot ? Est-elle déterminée à l'intérieur de Partition ou connue à l'appel de procédure ? Ici il sera plus simple de le déterminer avant (on aura besoin du maillon correspondant pour reconstituer E dans la procédure de tri)

— Il faut choisir (arbitrairement) où mettre les valeurs égales à p.

— Comme on a déjà pris le pivot dans le premier maillon, on peut passer en réalité à Partition un morceau de liste chaînée (privée du pivot) dont on donne l'adresse du premier maillon.

— On renvoie un couple de listes.

Par exemple :

```
Partition(données p : un élément, M : l'adresse d'un maillon ;
2   résultat L1, L2 : deux listes)
{ Pré-condition : aucune.
4 Post-condition :
  — L1 ∪ L2 contient les mêmes valeurs que celles suivant M
6  — toutes les valeurs de L1 sont inférieures ou égales à p
  — toutes les valeurs de L2 sont supérieures à p
8 }
```

Q 3. Donner une réalisation de TriSegmentation en utilisant Partition.

Corrigé —

```
TriSegmentation(L):
2   p : un élément
   L1, L2 : deux listes
4   D1, D2 : deux adresses
debut
6   si L.tete == NULL :
   renvoyer NULL
8
   p = L.tete.valeur
10  L1, L2 = Partition(p, L.tete.suivant)
   D1 = TriSegmentation(L1)
12  D2 = TriSegmentation(L2)
14
   // On raccroche L2 triée à la suite du pivot
   L.tete.suivant = L2.tete // Ce sera NULL si L2 est vide
16  si L2.tete == NULL :
   D2 = L.tete // le vrai dernier maillon de la liste triée
18  fin
20
   // On raccroche le pivot à la suite de L1 triée
   si L1.tete != NULL :
22     D1.suivant = L.tete
     L.tete = L1.tete
24   // sinon rien à faire, c'est le pivot le premier maillon de L triée
   fin
26   renvoyer D2
fin
```

Q 4. Donner une réalisation de Partition :

— *Indication* : il est possible de réaliser Partition en une seule boucle qui parcourt la liste.

— Commencer par exprimer sous forme de schéma le principe général de cette boucle.

Corrigé —

```

1 Partition(p, M):
   L1, L2 : deux listes
3   TMP : une adresse
debut
5   L1.tete = NULL
   L2.tete = NULL
7   tant que M != NULL
   TMP = M.suivant
9   si M.valeur <= x alors
   M.suivant = L1.tete
11  L1.tete = M
   sinon
13  M.suivant = L2.tete
   L2.tete = M
15  fin si
   M = TMP
17 fin tant que
   renvoyer L1, L2
19 fin

```

Analyse

Q 5. Quel est le coût dans le pire cas, dans le meilleur cas de Partition? de TriSegmentation? Donner des exemples (de taille 10) de pire et meilleurs cas.

Corrigé —

Partition : coût de $O(n)$ dans tous les cas (n =taille de la liste; n passages dans la boucle puisque M avance d'un maillon à chaque itération).

TriSegmentation :

—Meilleur cas : à chaque appel récursif, partition en deux listes de taille identique.

Coût $C(n) = 2.C(n/2) + O(n) = O(n \log n)$

Exemple (à construire à partir des appels récursifs) :

```

          Tri(0,9)
        [4,1,0,3,2,8,7,6,5,9]
         /       \
    Tri(0,3)   Tri(5,9)
    [2,1,0,3] [7,6,5,9,8]
     /  \     /  \
Tri(0,1) Tri(3,3) Tri(5,6) Tri(8,9)
 [0,1]   [3]   [5,6]   [8,9]

```

—Pire cas : à chaque appel récursif, partition en deux listes de tailles $(n-1)$ et 0.

Coût $C(n) = C(n-1) + O(n) = O(n^2)$

Exemple : tous les éléments de la liste ont la même valeur.

Q 6. Donner le coût de Partition(T) dans le cas où tous les éléments de T ont la même valeur. Proposer une correction de Partition susceptible d'améliorer ce coût.

Corrigé —

On peut remarquer que la question se pose de manière plus générale pour les tableaux comportant un nombre important de fois la même valeur (ce qui arrive dans de nombreuses applications).

Identifier le problème : la partition effectuée met toutes les valeurs "égales" au pivot d'un seul côté. Il faudrait au contraire minimiser la taille de la plus grande partition.

Plusieurs pistes possibles :

- les valeurs égales au pivot peuvent très bien être placées dans l'une ou l'autre partition : alterner entre les deux (à l'aide d'un booléen par exemple) ?
- plus rigoureux : faire un vrai «drapeau hollandais» (voir Cours 5), en regroupant toutes les valeurs égales au milieu de la liste.

```
1 Partition(p, M):
  L1, L2, Lp : trois listes
3   TMP : une adresse
debut
5   L1.tete = NULL
   L2.tete = NULL
7   Lp.tete = NULL
   tant que M != NULL
9     TMP = M.suivant
     si M.valeur == x alors
11      M.suivant = Lp.tete
      Lp.tete = M
13     sinon si M.valeur < x alors
      M.suivant = L1.tete
15      L1.tete = M
     sinon
17      M.suivant = L2.tete
      L2.tete = M
19     fin si
     M = TMP
21 fin tant que
   renvoyer L1, Lp, L2
23 fin
fin
```

On peut ensuite remarquer que la liste Lp contenant les valeurs égales au pivot est triée : on peut changer la spécification de Partition pour donner comme résultat les trois partitions, mais seules L1 et L2 auront besoin d'un appel récursif. Il faut également modifier l'algorithme de tri pour raccrocher les trois listes ensemble correctement.
