

## TD N° 12

## Tris par fusion

**1 Tri par fusion**

**a.** Faites la trace de l'algorithme du tri par fusion *simple* sur le fichier contenant les entiers suivants:

12 44 15 55 54 60 57 80 78 25 40

---

fa : 12 44 15 55 54 60 57 80 78 25 40

Éclatement n°1 : monotones de longueur 1

fb : 12 | 15 | 54 | 57 | 78 | 40

fc : 44 | 55 | 60 | 80 | 25

Fusion n°1 : monotones de longueur 1

fa : 12 44 | 15 55 | 54 60 | 57 80 | 25 78 | 40

nbMono = 6

Éclatement n°2 : monotones de longueur 2

fb : 12 44 | 54 60 | 25 78

fc : 15 55 | 57 80 | 40

Fusion n° 2 : monotones de longueur 2

fa : 12 15 44 55 | 54 57 60 80 | 25 40 78

nbMono = 3

Éclatement n°3 : monotones de longueur 4

fb : 12 15 44 55 | 25 40 78

fc : 54 57 60 80

Fusion n°3 : monotones de longueur 4

fa : 12 15 44 54 55 57 60 80 | 25 40 78

nbMono = 2

Éclatement n°4 : monotones de longueur 8

fb : 12 15 44 54 55 57 60 80

fc : 25 40 78

Fusion n°4 : monotones de longueur 8

fa : 12 15 25 40 44 54 55 57 60 78 80

nbMono = 1

On a fini : fa est trié...

.....

b. Et maintenant, faites la trace de l'algorithme du tri par fusion *naturelle* sur le même fichier.

Tout d'abord désigner les monotonies maximales :

12 44 | 15 55 | 54 60 | 57 80 | 78 | 25 40

Éclatement 1

12 44 54 60 78

15 55 57 80 | 25 40

Fusion

12 15 44 54 55 57 60 78 80 | 25 40

Éclatement 2

12 15 44 54 55 57 60 78 80

25 40

Fusion

12 15 25 40 44 54 55 57 60 78 80

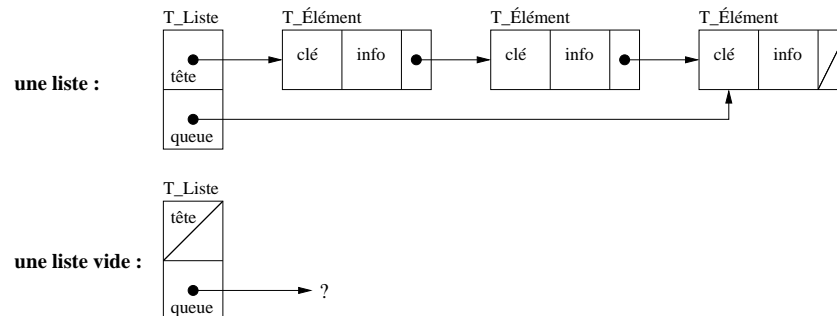
.....

## 2 Tri par fusion naturelle d'une liste

Dans cet exercice, nous allons appliquer le tri par fusion naturelle aux listes chaînées, qui sont un type de données séquentiel, comme les fichiers, mais interne cette fois. Nous mettrons en œuvre les deux solutions alternatives évoquées en cours qui s'appuient sur les contraintes suivantes :

1. faire attention, chaque fois que l'on ajoute une monotonie à la suite d'une autre, de réalement ajouter une nouvelle monotonie et non pas seulement de compléter la monotonie précédente ;
2. éclater en même temps que la fusion.

Les listes chaînées seront sans élément supplémentaire et disposeront, en plus du pointeur sur le premier élément de la liste, d'un pointeur sur le dernier élément de la liste selon les déclarations et le schéma suivants :



### types

T\_Clé = ? {muni d'une relation d'ordre}

T\_Contenu = ?

P\_Élément = **pointeur sur** T\_Élément

T\_Élément = **article**

clé : T\_Clé

info : T\_Contenu

suiv : P\_Élément

**finarticle**

T\_Liste = **article**

tête, queue : P\_Élément

**finarticle**

Nous utiliserons en outre les fonctions et procédures suivantes :

```
procédure initialiser (donnée-résultat L : T_Liste)
{Rôle : L+ est une liste vide (son champ tête prend la valeur nil)}
```

```
fonction estVide (L : T_Liste) : booléen
{Antécédent : L est initialisé}
{Rôle : retourne vrai si L est vide, faux sinon}
```

```
fonction tête (L : T_Liste) : T_clé
{Antécédent : L n'est pas vide}
{Rôle : retourne la clé de l'élément en tête de L}
```

```
fonction queue (L : T_Liste) : T_clé
{Antécédent : L n'est pas vide}
{Rôle : retourne la clé de l'élément en queue de L}
```

Dans un premier temps, nous allons écrire le tri en n'appliquant que la première contrainte.

**c.** Récapitulez les différentes procédures vues en cours qui interviennent dans le tri et précisez leur rôle dans le contexte de cet exercice.

---

Il s'agit ici

- de leur remettre en mémoire une nouvelle fois les grands traits de l'algorithme,
- de rappeler l'origine des deux contraintes supplémentaires (assurer une distribution égale des monotonies lors de l'éclatement et éviter de construire une liste (ou pire un fichier) à seule fin de l'éclater)
- et surtout de faire apparaître les procédures qui seront utilisées dans la solution et qui correspondent aux différents fragments de l'algorithme vu en cours.

Cela pourra par exemple être fait en s'appuyant sur une trace simplifiée du tri sur la liste 30 12 50 40 45 15.

Selon un schéma de conception descendante feront leur entrée :

- trier : qui alterne les phases d'éclatement et de fusion jusqu'à ce que la liste soit triée.
  - éclater : qui distribue les monotonies d'une liste vers deux nouvelles listes alternativement.
    - copierMonotonie : qui « déplace » une monotonie en tête d'une liste vers la queue d'une autre. Il faudra sans doute mettre en relief le fait qu'il n'y aura pas de duplication des éléments de la liste de départ mais uniquement des réarrangements de pointeurs.
      - copierÉlément : qui "déplace" un élément en tête d'une liste vers la queue d'une autre.
  - fusionner : qui assemble les monotonies successives de deux listes différentes dans une troisième liste.
    - fusionnerMonotonie : qui ajoute en queue d'une liste une monotonie construite à partir de deux autres monotonies présentes en tête de deux autres listes.

**d.** Écrivez la procédure trier qui prend en paramètre une liste et effectue le tri de cette liste selon l'algorithme de fusion naturelle et le découpage en procédures défini à la question précédente.

---

```
procédure trier (donnée-résultat L : T_Liste)
{Antécédent : L n'est pas vide}
{Rôle : L+ est la liste L- triée}
variables
```

```

L1, L2 : T_Liste
nb_mono : entier
début
  initialiser(L1); initialiser(L2)
  répéter
    éclater(L, L1, L2)
    fusionner(L1, L2, L, nb_mono)
  jusque nb_mono = 1
fin {de trier}

```

e. Écrivez les procédures nécessaires à la réalisation de l'opération d'éclatement d'une liste. Vous veillerez à respecter la première contrainte.

---

```

procédure éclater (données-résultats L, L1, L2 : T_Liste)
  {Antécédent : L n'est pas vide. L1 et L2 sont initialisés}
  {Rôle : distribuer les monotonies de L dans L1 et L2 alternativement,
   en prenant soin de réellement ajouter une monotonie.
   L1+ contient ainsi autant de monotonies, ou une de plus, que L2+.
   L+ est vide}
début
  copierMonotonie(L, L1)
  si non estVide(L) alors
    copierMonotonie(L, L2)
  finsi
  tantque non estVide(L) faire
    si tête(L) ≥ queue(L1) alors {compléter la monotonie précédente de L1}
      copierMonotonie(L, L1)
      si non estVide(L) alors {ajouter une nouvelle monotonie à L1}
        copierMonotonie(L, L1)
      finsi
    sinon
      copierMonotonie(L, L1)
    finsi
    si non estVide(L) alors
      si tête(L) ≥ queue(L2) alors {compléter la monotonie précédente de L2}
        copierMonotonie(L, L2)
        si non estVide(L) alors {ajouter une nouvelle monotonie à L2}
          copierMonotonie(L, L2)
        finsi
      sinon
        copierMonotonie(L, L2)
      finsi
    fintantque
  fin {de éclater}

```

Pour être sûr de toujours réellement ajouter une nouvelle monotonie, on pourrait être tenté de faire une boucle qui répète des `copierMonotonie` tant que celle qu'on ajoute complète la précédente. Pourtant si une première monotonie M1 en complète une autre, la monotonie M2 qui suit M1 dans L, si elle existe, ne peut pas compléter M1 par définition. Un deuxième appel à `copierMonotonie` suffit donc à garantir la contrainte que l'on s'est fixée.

```

procédure copierMonotonie (données-résultats src, dest : T_Liste)
  {Antécédent : src n'est pas vide. dest est initialisé}
  {Rôle : déplacer tous les éléments de la monotonie en tête de src-
   vers la queue de dest-}
début

```

```

répéter
  copierÉlément(src, dest)
  jusque estVide(src) ousi tête(src) < queue(dest)
fin {de copierMonotonie}

procédure copierÉlément (données-résultats src, dest : T_Liste)
{Antécédent : src n'est pas vide. dest est initialisé}
{Rôle : déplacer l'élément en tête de src vers la queue de dest}
début
  si estVide(dest) alors {cas de la liste vide}
    dest.tête ← src.tête
  sinon {cas général}
    (dest.queue)↑.suiv ← src.tête
  finsi
  dest.queue ← src.tête
  src.tête ← (src.tête)↑.suiv
  (dest.queue)↑.suiv ← nil
fin {de copierÉlément}

```

f. Terminez par l'écriture des procédures réalisant l'opération de fusion.

```

procédure fusionner (données-résultats L1, L2, L : T_Liste
  résultat nb : entier)
{Antécédent : L1 n'est pas vide. L2 et L sont initialisés}
{Rôle : fusionner et retirer chaque monotonie de L1 avec une de L2 et
  ajouter le résultat de chaque fusion en queue de L}
début
  nb ← 0
  tantque non estVide(L2) faire
    nb ←+ 1
    fusionnerMonotonie(L1, L2, L)
  fintantque
  si non estVide(L1) alors
    nb ←+ 1
    copier(L1, L)
  finsi
fin {de fusionner}

```

Nous pourrions utiliser `copierMonotonie` à la place de la nouvelle procédure `copier` mais pourquoi se priver des avantages des listes et ne pas « déplacer » la dernière monotonie en un seul bloc.

```

procédure copier (données-résultats src, dest : T_Liste)
{Antécédent : src et dest sont initialisés}
{Rôle : déplacer, en conservant l'ordre, les éléments de src en queue
  de dest}
début
  si estVide(dest) alors {cas de la liste vide}
    dest.tête ← src.tête
  sinon {cas général}
    (dest.queue)↑.suiv ← src.tête
  finsi
  src.tête ← nil {src est maintenant une liste vide}
  dest.queue ← src.queue
fin {de copier}

procédure fusionnerMonotonie (données-résultats L1, L2, L : T_Liste)

```

```

{Antécédent : L1 et L2 ne sont pas vides. L est initialisé}
{Rôle : fusionner et retirer une monotonie en tête de L1- et de
      L2- et ajouter le résultat en queue de L-}
variables
  fin_monol, fin_mono2 : booléen
début
  fin_monol ← faux; fin_mono2 ← faux
répéter
  si tête(L1) < tête(L2) alors
    copierÉlément(L1, L)
    fin_monol ← estVide(L1) ou si tête(L1) < queue(L)
    si fin_monol alors {fin de monotonie}
      copierMonotonie(L2, L)
    finsi
  sinon
    copierÉlément(L2, L)
    fin_mono2 ← estVide(L2) ou si tête(L2) < queue(L)
    si fin_mono2 alors {fin de monotonie}
      copierMonotonie(L1, L)
    finsi
  finsi
jusque fin_monol ou fin_mono2
fin {de fusionnerMonotonie}

```

.....

La réalisation de la seconde contrainte s'obtient en distribuant dans deux nouvelles listes les monotonies fusionnées (contrairement à l'algorithme précédent où elles étaient regroupées dans une même liste qui était ensuite éclatée).

**g.** Modifiez en conséquence les procédures trier et fusionner (que l'on peut maintenant appeler fusionnerÉclater).

---

```

procédure trier (donnée-résultat L : T_Liste)
variables
  L1, L2, L3 : T_liste
début
  initialiser(L1); initialiser(L2); initialiser(L3)
  éclater(L, L1, L2)
  fusionnerÉclater(L1, L2, L, L3)
  tantque non estVide(L3) faire {il reste plus d'une monotonie}
    L1 ← L {L1.tête = L.tête et L1.queue = L.queue}
    L.tête ← nil {L est vide}
    L2 ← L3
    L3.tête ← nil
    fusionnerÉclater(L1, L2, L, L3)
  fintantque
fin {de trier}

```

Pour savoir quand arrêter le tri, compter le nombre de monotonies, comme dans la version précédente, fonctionnerait très bien aussi.

```

procédure fusionnerÉclater (données-résultats L1, L2, LA, LB : T_Liste)
{Antécédent : L1 n'est pas vide. L2, LA et LB sont initialisés}
{Rôle : fusionner et retirer chaque monotonie de L1 avec une de L2 et
      ajouter le résultat de chaque fusion en queue de LA ou LB
      alternativement}
variable
  dernière : (listeA, listeB)

```

```
début
  fusionnerMonotonie(L1, L2, LA)
  dernière ← listeA
  si non estVide(L2) alors
    fusionnerMonotonie(L1, L2, LB)
    dernière ← listeB
  finsi
  tantque non estVide(L2) faire
    si tête(L1) ≥ queue(LA) et tête(L2) ≥ queue(LA) alors
      fusionnerMonotonie(L1, L2, LA) {nous complétons une monotonie}
      si non estVide(L2) alors
        fusionnerMonotonie(L1, L2, LA)
        dernière ← listeA
      finsi
    sinon
      fusionnerMonotonie(L1, L2, LA)
      dernière ← listeA
    finsi
  si non estVide(L2) alors
    si tête(L1) ≥ queue(LB) et tête(L2) ≥ queue(LB) alors
      fusionnerMonotonie(L1, L2, LB) {nous complétons une monotonie}
      si non estVide(L2) alors
        fusionnerMonotonie(L1, L2, LB)
        dernière ← listeB
      finsi
    sinon
      fusionnerMonotonie(L1, L2, LB)
      dernière ← listeB
    finsi
  finsi
fintantque
si non estVide(L1) alors
  si dernière = listeA alors
    copier(L1, LB)
  sinon
    copier(L1, LA)
  finsi
finsi
fin {de fusionnerÉclater}
```

.....