Algorithmes de tri

Jean Simard

Mercredi 5 novembre 2008

1 Préalable

Nous allons mettre en place une base de programme sur laquelle nous allons tester nos fonctions de tri. Donc chaque exercice sera une amélioration apportée à ce qui a déjà été fait. C'est pour cette raison que vous ne devriez avoir qu'un seul fichier source pour l'ensemble des exercices de ce TP.

Nous allons travailler sur des listes doublement chaînées. Chaque maillon de cette liste sera constitué d'un nombre (puis des pointeurs vers le maillon précédent et suivant).

Exercice 1 *Écrire la structure qui remplira la tâche de maillon d'une liste doublement chaînée et contenant un entier non-signé.*

Correction de l'exercice 1 –

Code 1: Tri.c

```
5  typedef struct sNombre
6  {
7    unsigned int uiNombre;
8    struct sNombre * pSuiv;
9    struct sNombre * pPrec;
10 } Nombre;
```

Dans notre programme, nous allons devoir initialiser la liste doublement chaînée avec des valeurs. Ces valeurs doivent être variées pour que le tri ait un intérêt.

Exercice 2 *Écrire une fonction qui créé et initialise les différents maillons de chaîne doublement chaînée. Cette fonction sera appelée depuis la fonction* `main` *avec un paramètre indiquant le nombre de maillons à créer. Le nombre de maillons à créer sera une valeur que l'utilisateur donnera (au choix, par argument ou par l'intermédiaire de la fonction* `scanf`*). Les maillons seront initialisés avec un entier aléatoire entre zéro et le nombre de maillon.*

Correction de l'exercice 2 –

Code 2: Tri.c

```
12 Nombre * InitialiserListe( unsigned int uiNbElements )
13 {
14   Nombre * pListe = NULL;
15   Nombre * pTemp = NULL;
16   unsigned int i = 0;
17   srand( uiNbElements );
18   for( i = 0; i < uiNbElements; i++ )
19   {
20     pTemp = (Nombre *) malloc( sizeof( Nombre ) );
21     pTemp->uiNombre = (unsigned int)( (double)rand()\
22       * (double)uiNbElements / ( (double)RAND_MAX + 1.0 ) );
23     pTemp->pSuiv = pListe;
24     if( pListe != NULL )
25     {
26       pTemp->pSuiv->pPrec = pTemp;
27     }
28     pTemp->pPrec = NULL;
29     pListe = pTemp;
30   }
31   return pListe;
32 }
```

Exercice 3 *Écrire une fonction qui libère l'espace mémoire de cette liste doublement chaînée. Cette fonction sera appelée juste avant de quitter la fonction* ***main****.*

Correction de l'exercice 3 –

Code 3: Tri.c

```
34 void LibererListe( Nombre * pListe )
35 {
36   Nombre * pTemp = pListe;
37   while( pTemp != NULL )
38   {
39     pListe = pTemp->pSuiv;
40     free( pTemp );
41   }
42   return;
43 }
```

Exercice 4 *Écrire une fonction qui permettre l'affichage des valeurs entières non-signées de la liste doublement chaînées. Cette fonction sera utilisée pour afficher la liste aléatoirement créée puis la liste triées.*

Correction de l'exercice 4 –

Code 4: Tri.c

```
45 void AfficherListe( Nombre * pListe )
46 {
47   while( pListe != NULL )
48   {
49     printf( "%u ", pListe->uiNombre );
50     pListe = pListe->pSuiv;
51   }
52   printf( "\n" );
53   return;
54 }
```

2 Tri à bulles

Pour rappel, l'algorithme de tri à bulles consiste à parcourir la liste en comparant deux à deux les valeurs puis à les inverser si elles ne sont pas dans le bon ordre. On effectue cette opération jusqu'à atteindre la fin de la liste. On parcours de nouveau la liste tant que la liste n'est pas triée.

Exercice 5 *Écrire une fonction qui effectue un tri à bulle de la liste doublement chaînée.*

Correction de l'exercice 5 –

Code 5: Tri.c

```
56 Nombre * InverserElement( Nombre * pN1, Nombre * pN2 )
57 {
58   Nombre * pTemp = NULL;
59   if( pN1->pSuiv != NULL )
60   {
61     pTemp = pN1->pSuiv;
62     pN1->pSuiv = pN2->pSuiv;
63     if( pN1->pSuiv != NULL )
64     {
65       pN1->pSuiv->pPrec = pN1;
66     }
67     pN2->pSuiv = pTemp;
68     pN2->pSuiv->pPrec = pN2;
69   }
70   else
71   {
72     pN1->pSuiv = pN2->pSuiv;
73     if( pN1->pSuiv != NULL )
```

```
74      {
75        pN1->pSuiv->pPrec = pN1;
76      }
77      pN2->pSuiv = NULL;
78    }
79    if( pN1->pPrec != NULL )
80    {
81      pTemp = pN1->pPrec;
82      pN1->pPrec = pN2->pPrec;
83      if( pN1->pPrec != NULL )
84      {
85        pN1->pPrec->pSuiv = pN1;
86      }
87      pN2->pPrec = pTemp;
88      pN2->pPrec->pSuiv = pN2;
89    }
90    else
91    {
92      pN1->pPrec = pN2->pPrec;
93      if( pN1->pPrec != NULL )
94      {
95        pN1->pPrec->pSuiv = pN1;
96      }
97      pN2->pPrec = NULL;
98    }
99    return pN2;
100 }
101
102 Nombre * TriBulle( Nombre * pListe )
103 {
104   Nombre * pTemp = pListe;
105   unsigned int Tri = 0;
106   while( Tri != 1 )
107   {
108     Tri = 1;
109     pTemp = pListe;
110     while( pTemp->pSuiv != NULL )
111     {
112       if( pTemp->uiNombre > pTemp->pSuiv->uiNombre )
113       {
114         pTemp = InverserElement( pTemp, pTemp->pSuiv );
115         if( pTemp->pPrec == NULL )
116         {
117           pListe = pTemp;
118         }
119         Tri = 0;
120       }
121       pTemp = pTemp->pSuiv;
122     }
123   }
```

```
124   return pListe;
125 }
```

3 Tri par sélection

Le tri par sélection consiste à chercher le maillon dont la valeur est la plus petite puis à le placer au début de la liste. Ensuite, on effectue la même opération sur la liste sur le reste de la liste qui n'est pas encore trié.

Exercice 6 *Écrire une fonction qui effectue un tri par sélection de la liste doublement chaînée.*

Correction de l'exercice 6 –

CODE 6: Tri.c

```
127 Nombre * TriSelection( Nombre * pListe )
128 {
129   Nombre * pMobile = pListe;
130   Nombre * pMin = NULL;
131   Nombre * pTemp = pMobile;
132   while( pMobile != NULL )
133   {
134     pTemp = pMobile;
135     pMin = pTemp;
136     while( pTemp != NULL )
137     {
138       if( pTemp->uiNombre < pMin->uiNombre )
139       {
140         pMin = pTemp;
141       }
142       pTemp = pTemp->pSuiv;
143     }
144     pMobile = InverserElement( pMobile, pMin );
145     if( pMobile->pPrec == NULL )
146     {
147       pListe = pMobile;
148     }
149     pMobile = pMobile->pSuiv;
150   }
151   return pListe;
152 }
```

4 Tri par insertion

Le tri par insertion va le plus souvent être plus efficace que les deux algorithmes précédents. Il va parcourir la liste est lorsqu'il trouve un élément qui n'est pas à la bonne place, il va le ramener dans la liste en l'insérant à sa place.

Exercice 7 *Écrire une fonction qui effectue un tri par insertion de la liste doublement chaînée.*

Correction de l'exercice 7 –

CODE 7: Tri.c

```
154 Nombre * Insertion( Nombre * pLieu, Nombre * pInsertion )
155 {
156   if( pInsertion->pPrec != NULL )
157   {
158     pInsertion->pPrec->pSuiv = pInsertion->pSuiv;
159   }
160   if( pInsertion->pSuiv != NULL )
161   {
162     pInsertion->pSuiv->pPrec = pInsertion->pPrec;
163   }
164   pInsertion->pSuiv = pLieu;
165   pInsertion->pPrec = pLieu->pPrec;
166   pLieu->pPrec = pInsertion;
167   if( pInsertion->pPrec != NULL )
168   {
169     pInsertion->pPrec->pSuiv = pInsertion;
170   }
171   return pInsertion;
172 }
173
174 Nombre * TriInsertion( Nombre * pListe )
175 {
176   Nombre * pMobile = pListe;
177   Nombre * pInsertion = NULL;
178   while( ( pMobile != NULL ) && ( pMobile->pSuiv != NULL ) )
179   {
180     if( pMobile->uiNombre > pMobile->pSuiv->uiNombre )
181     {
182       pInsertion = pMobile->pSuiv;
183       while( pInsertion->pPrec != NULL )
184       {
185         if( pInsertion->pPrec->uiNombre < pMobile->pSuiv->uiNombre )
186         {
187           break;
188         }
189         pInsertion = pInsertion->pPrec;
190       }
```

```
191         pInsertion = Insertion( pInsertion, pMobile->pSuiv );
192         if( pInsertion->pPrec == NULL )
193         {
194           pListe = pInsertion;
195         }
196       }
197       else
198       {
199         pMobile = pMobile->pSuiv;
200       }
201     }
202     return pListe;
203 }
```