Tri bulle, tri caillou, tri shaker [tr03] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 21 mai 2018

Table des matières

C - Tri bulle et associés (Solution)

Mots-Clés Algorithmes de tris et rangs, Tri bulle, Tri caillou, Tri shaker ■
Requis Axiomatique impérative (sauf Fichiers) ■
Difficulté ●●○ (1 h 15) ■

Objectif
Cet exercice réalise le tri bulle, tri caillou, tri shaker d'un tableau d'entiers. Dans le même ordre d'idées, l'exercice @[Tri par sélection] réalise le tri par sélection et l'exercice @[Tri par insertion] celui par insertion.

1 Téléchargements – Utilitaires

Cet exercice utilise les opérations suivantes, toutes définies dans un bon nombre d'exercices de cet espace thématique :

- Fonction `saisirNombreElements`
- Procédure `afficherTri`
- Procédure `aleatoireTri`
- Procédure `permuterTab`

Elles ont été regroupées dans une bibliothèque.

Définitions C

```c
enum { TMAX = ... };
typedef int ITableau[TMAX];
```

Fixez la constante `TMAX=50` (nombre maximum d'éléments).

Téléchargez le fichier suivant et mettez-le dans votre dossier.
C @[UtilsTR.c]

Copiez/collez ensuite les lignes suivantes :
C **Au début** de votre programme :

```c
#include "UtilsTR.c"
```

Soit la fonction `saisirNombreElements(nmax)` qui renvoie le nombre d'éléments, saisi par l'utilisateur, entier compris dans `[1..nmax]`. Elle affiche l'invite :

```
Nombre d'éléments dans [1..[nmax]]?
```

C @[saisirNombreElements] (dans UtilsTR.c)

Soit la procédure `afficherTri(t,n,g,h)` qui affiche, à la queue-leu-leu séparés par un espace le tout entre crochet, les `n` premières valeurs d'un `ITableau t`, les indices `g` et `h` indiquant le sous-intervalle du tri et représentés par une barre verticale. La barre de gauche est avant `g` et celle de droite est après `h`. Exemple :

```
afficherTri(t,10,4,9) ==> [1 2 3 |4 5 6 7 8 9 |10]
```

C @[afficherTri] (dans UtilsTR.c)

Soit la procédure `aleatoireTri(t,n,vmax)`H qui initialise les `n` premiers éléments d'un `ITableau t` en utilisant `vmax` comme valeur maximale pour la fonction de génération d'un entier pseudo-aléatoire.
C @[aleatoireTri] (dans UtilsTR.c)

Soit la procédure `permuterTab(t,j,k)` qui permute les éléments d'indice `j` et `k` d'un `ITableau t`. Les indices sont supposés valides.

C @[permuterTab] (dans UtilsTR.c)

Remarque
Si la fonction et les procédures n'ont pas été réalisées, il vous est conseillé de la(les) rédiger dans l'exercice @[Utilitaires Tris et Rangs].

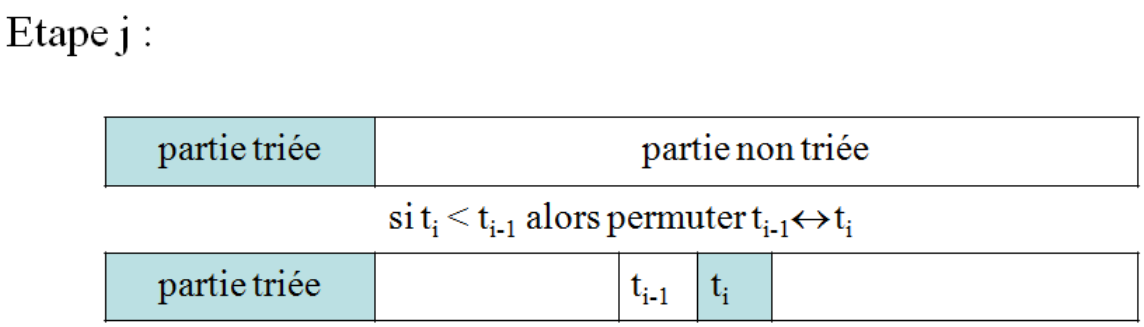
...(suite page suivante)...

2 Tri bulle et associés / pgbulles

2.1 Tri bulle basique

Principe du tri bulle

Le **tri bulle** d'un tableau `t[1..n]` de `n` éléments consiste, pour `j` variant de 1 à `n-1`, à faire « remonter » le plus petit élément du sous-tableau `t[j..n]` en position `j` par une suite d'échanges de deux éléments consécutifs.



Exemple

(Avec affichage des tableaux successifs)

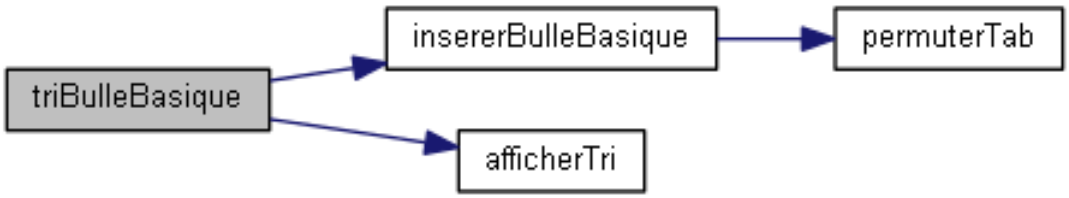
```
Nombre d'éléments dans [1..50]? 10
Tableau initial
[  1 19  8  7  4  8 14 15  6  5  ]
Tri bulle basique
[| 1  4 19  8  7  5  8 14 15  6 |]
[  1| 4  5 19  8  7  6  8 14 15 |]
[  1  4| 5  6 19  8  7  8 14 15 |]
[  1  4  5| 6  7 19  8  8 14 15 |]
[  1  4  5  6| 7  8 19  8 14 15 |]
[  1  4  5  6  7| 8  8 19 14 15 |]
[  1  4  5  6  7  8| 8 14 19 15 |]
[  1  4  5  6  7  8  8|14 15 19 |]
[  1  4  5  6  7  8  8 14|15 19 |]
Tableau final
[  1  4  5  6  7  8  8 14 15 19  ]
```

Remarque

Le nom du tri bulle provient de ce que les éléments les plus petits remontent au fur et à mesure des échanges vers la gauche du tableau, de la même façon que les bulles d'un élément plus léger remontent vers la surface d'un élément plus lourd.

Écrivez une procédure `insererBulleBasique(t,g,h)` qui fait remonter le plus petit élément du sous-tableau `t[g..h]| en \lstinlinet[g]@` par permutation de deux éléments consécutifs, `t` étant un `Tableau`. La procédure utilisera la procédure `permuterTab` pour réaliser l'échange de deux éléments de `t`.

Écrivez une procédure `triBulleBasique(t,n)` qui effectue le tri bulle de `n` éléments d'un `ITableau t`.



Validez vos procédures avec la solution.

Solution C @[pgbulles.c]

```c
void insererBulleBasique(Tableau t,int g,int h)
{
  int ix;
  for (ix=h-1 ; ix>=g ; --ix)
  {
    if (t[ix+1]<t[ix])
    {
      permuterTab(t,ix+1,ix);
    }
  }
}
```

```c
void triBulleBasique(Tableau t,int n)
{
  int j;
  for (j=0 ; j<n ; ++j)
  {
    insererBulleBasique(t,j,n);
    afficherTri(t,n,j,n);
  }
}
```

Écrivez un programme qui saisit le nombre d'éléments puis déclare un `ITableau` et l'initialise de façon aléatoire en prenant 20 pour valeur maximale, en effectue son tri bulle et l'affiche.

Testez.

2.2 Tri bulle, version 2

Amélioration du tri bulle

Nous pouvons améliorer la procédure du tri bulle en faisant la remarque suivante : si aucun échange n'est effectué lors du parcours d'un sous-tableau `t[j..n]`, c'est que le tableau est trié. Nous pouvons donc écrire la procédure d'insertion sous la forme d'une fonction qui renvoie `Vrai` si un échange a été effectué, `Faux` sinon.

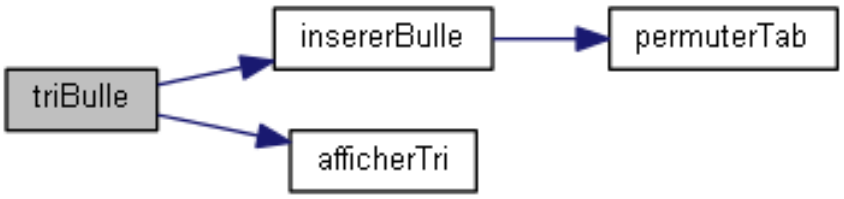
Exemple

(Avec affichage des tableaux successifs)

```
Nombre d'éléments dans [1..50]? 10
Tableau initial
[ 12 11 15 17  7  7 13  1 18 18  ]
Tri bulle
[| 1 12 11 15 17  7  7 13 18 18 |]
[  1| 7 12 11 15 17  7 13 18 18 |]
[  1  7| 7 12 11 15 17 13 18 18 |]
[  1  7  7|11 12 13 15 17 18 18 |]
[  1  7  7 11|12 13 15 17 18 18 |]
Tableau final
[  1  7  7 11 12 13 15 17 18 18  ]
```

Copiez/collez la procédure `insererBulleBasique` en la **fonction** modifiée `insererBulle(t,g,h)`. Modifiez-la de sorte qu'elle remonte l'élément le plus petit du sous-tableau `t[g..h]` en `t[g]` par une suite d'échanges et renvoie `Vrai` si un échange a été réalisé, `Faux` sinon.

Copiez/collez la procédure `triBulleBasique` en `triBulle(t,n)`, puis modifiez la procédure pour qu'elle utilise la fonction `insererBulle` en place de la procédure `insererBulleBasique` et ce tant que la fonction renvoie `Vrai`.



Validez votre fonction et votre procédure avec la solution.

Solution C @[pgbulles.c]

```
bool insererBulle(Tableau t,int g,int h)
{
  bool permutation = false;
  int ix;
  for (ix=h-1 ; ix>=g ; --ix)
  {
    if (t[ix+1]<t[ix])
    {
      permuterTab(t,ix+1,ix);
      permutation = true;
    }
  }
  return permutation;
}
```

```
void triBulle(Tableau t,int n)
{
  bool permutation=true;
  int j=0;
  while (j<n && permutation)
  {
    permutation = insererBulle(t,j,n);
    afficherTri(t,n,j,n);
    j += 1;
  }
}
```

Modifiez votre programme afin qu'il effectue le tri bulle (version 2).

Testez.

2.3 Tri caillou

Symétrie du tri bulle

Par symétrie, le tri caillou provient de ce que les éléments les plus lourds descendent au fur et à mesure des échanges vers la droite du tableau, de la même façon que les cailloux d'un élément plus lourd descendent vers le fond d'un élément plus léger.

Exemple

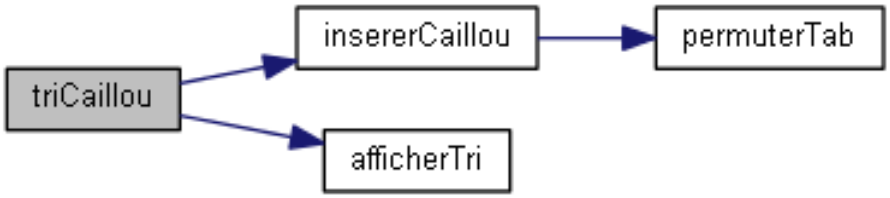
(Avec affichage des tableaux successifs)

```
Nombre d'éléments dans [1..50]? 10
Tableau initial
[  2  4 11  3 12 18  3  9 18 11  ]
Tri caillou
[| 2  4  3 11 12  3  9 18 11 18 |]
[| 2  3  4 11  3  9 12 11 18 |18 ]
[| 2  3  4  3  9 11 11 12 |18 18 ]
[| 2  3  3  4  9 11 11 |12 18 18 ]
```

```
[| 2  3  3  4  9 11 |11 12 18 18 ]
Tableau final
[  2  3  3  4  9 11 11 12 18 18  ]
```

Copiez/collez la fonction `insererBulle` en `nsererCaillou(t,g,h)`, puis modifiez la fonction de sorte à faire descendre l'élément le plus lourd du sous-tableau `t[g..h]` en `t[h]` par une suite d'échanges et renvoie `Vrai` si un échange a été réalisé, `Faux` sinon.

De même, copiez/collez la procédure `triBulle` en `triCaillou(t,n)`, puis modifiez la procédure de sorte qu'elle utilise la fonction `insererCaillou` en place de la procédure `insererBulle` et ce tant que la fonction renvoie `Vrai`.



Validez votre fonction et votre procédure avec la solution.

Solution C @[pgbulles.c]

```
}

bool insererCaillou(Tableau t,int g,int h)
{
  bool permutation=false;
  int ix;
  for (ix=g+1 ; ix<=h ; ++ix)
  {
    if (t[ix-1]>t[ix])
    {
      permuterTab(t,ix-1,ix);
      permutation = true;
    }
  }
  return permutation;
}
```

```
void triCaillou(Tableau t,int n)
{
  bool permutation=true;
  int j=n-1;
  while (j>0 && permutation)
  {
    permutation = insererCaillou(t,0,j);
```

```
    afficherTri(t,n,0,j);
    --j;
  }
}
```

Modifiez votre programme afin qu'il effectue le tri caillou.

Testez.

2.4 Tri Shaker

Stratégie du tri shaker

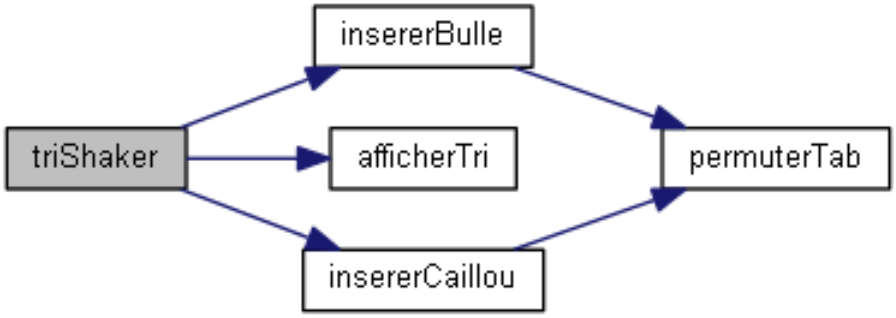
Le nom du **tri shaker** provient de ce que le tri alterne l'insertion bulle et l'insertion caillou, c.-à-d. qu'une fois sur deux, soit l'indice de gauche `g` est incrémenté, soit l'indice de droite `h` est décrémenté.

Exemple

(Avec affichage des tableaux successifs)

```
Nombre d'éléments dans [1..50]? 10
Tableau initial
[ 11 12 13  1 16 19  0 15  8  3  ]
Tri shaker
[| 0 11 12 13  1 16 19  3 15  8 |]
[  0|11 12  1 13 16  3 15  8 19 |]
[  0| 1 11 12  3 13 16  8 15 |19 ]
[  0  1|11  3 12 13  8 15 16 |19 ]
[  0  1| 3 11  8 12 13 15 |16 19 ]
[  0  1  3| 8 11 12 13 15 |16 19 ]
[  0  1  3| 8 11 12 13 |15 16 19 ]
Tableau final
[  0  1  3  8 11 12 13 15 16 19  ]
```

Écrivez une procédure `triShaker(t,n)` qui effectue le tri shaker de `n` éléments d'un `ITableau t`. La procédure appellera la fonction `insererBulle` ou la fonction `insererCaillou` selon l'entier `sens` valant $+1$ ou -1.



Validez votre procédure avec la solution.

Solution C @[pgbulles.c]

```
void triShaker(Tableau t,int n)
{
  bool permutation=true;
  int g=0;
  int h=n-1;
  int sens=1;
  while (g<h && permutation)
  {
    if (sens==1)
    {
      permutation = insererBulle(t,g,h);
      afficherTri(t,n,g,h);
      ++g;
    }
    else
    {
      permutation = insererCaillou(t,g,h);
      afficherTri(t,n,g,h);
      --h;
    }
    sens = -sens;
  }
}
```

Modifiez votre programme afin qu'il effectue le tri shaker.

Testez.

Validez votre programme avec la solution.

Solution C @[pgbulles.c]

```
int main(void)
{
  Tableau tab;
  int nelems = saisieNombreElements();
  aleatoireTri(tab,nelems,20);
  printf("Tableau initial\n");
  afficherTri(tab,nelems,0,nelems);
  printf("Tri bulle\n");
  //triBulleBasique(tab,nelems);
  //triBulle(tab,nelems);
  //triCaillou(tab,nelems);
  triShaker(tab,nelems);
  printf("Tableau final\n");
  afficherTri(tab,nelems,0,nelems);
}
```

3 Références générales

Comprend [Chappelier :c7 :ex37], [Moliner-ML1 :c2 :ex19], [Tarlowski-PG1 :c3 :ex15] ■