Tri par insertion [tr05] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

Java - Tri par insertion (Solution)

Mots-Clés Algorithmes de tris et rangs, Tri par insertion ■
Requis Axiomatique impérative (sauf Fichiers) ■
Difficulté • ○ ○ (45 min) ■

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

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 Java

```
final int TMAX = ...;
```

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

Téléchargez le fichier suivant et mettez-le dans votre dossier.

Java @[UtilsTR.java]

Copiez/collez ensuite les lignes suivantes : Les opérations seront accessibles en notation usuelle.

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]]?
```

Java @[saisirNombreElements] (dans UtilsTR.java)

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]
```

Java @[afficherTri] (dans UtilsTR.java)

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.

Java @[aleatoireTri] (dans UtilsTR.java)

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.

Java @[permuterTab] (dans UtilsTR.java)

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)...

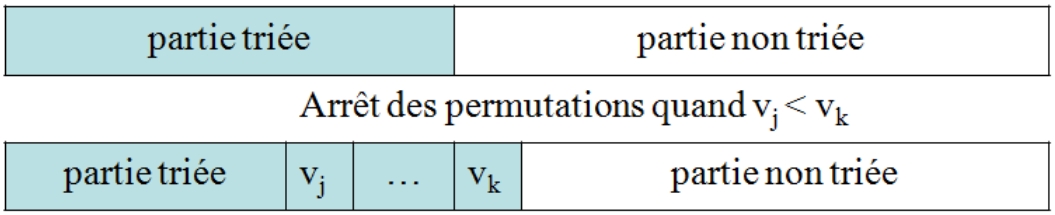
2 Tri par insertion / pginsertion

2.1 Tri par insertion basique

Principe du tri par insertion

Le **tri par insertion** d'un tableau `t[1..n]` de `n` éléments consiste, pour `j` variant de 1 à `n`, à insérer l'élément `t[j]` dans le sous-tableau trié `t[1..j-1]` de sorte que le tableau `t[1..j]` soit trié.

Etape j :



Exemple

(Avec affichage des tableaux successifs)

```
Nombre d'éléments dans [1..50]? 10
Tableau initial
[ 13 12  6  7 13 18 17  4 18  3  ]
Tri par insertion basique
[ 13|12  6  7 13 18 17  4 18  3 |]
[ 12 13| 6  7 13 18 17  4 18  3 |]
[  6 12 13| 7 13 18 17  4 18  3 |]
[  6  7 12 13|13 18 17  4 18  3 |]
[  6  7 12 13 13|18 17  4 18  3 |]
[  6  7 12 13 13 18|17  4 18  3 |]
[  6  7 12 13 13 17 18| 4 18  3 |]
[  4  6  7 12 13 13 17 18|18  3 |]
[  4  6  7 12 13 13 17 18 18| 3 |]
[  3  4  6  7 12 13 13 17 18 18||]
Tableau final
[  3  4  6  7 12 13 13 17 18 18  ]
```

Remarque

Le tri par insertion est similaire au tri du joueur de cartes qui insère les cartes une à une dans son jeu de manière à ce qu'à tout instant le jeu soit trié.

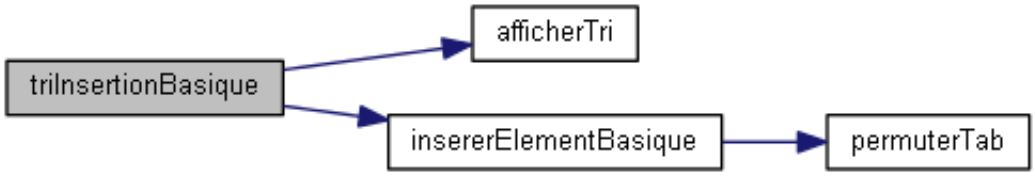
Écrivez une procédure `insererElementBasique(t,k)` qui insère l'élément `t[k]` dans le sous-tableau trié `t[1..k-1]`, `t` étant un `ITableau`. La procédure utilisera la procédure `permuterTab` pour réaliser l'échange de deux éléments de `t`.

Solution simple

Soit `j` l'indice de boucle.

Si `t[k]` est le plus petit élément de `t[1],...,t[k-1]`, la boucle ne doit pas provoquer un débordement vers la gauche du tableau avec la comparaison de `t[0]` au minimum courant. Il faut donc tester systématiquement si `j>1`.

Écrivez une procédure `triInsertionBasique(t,n)` qui effectue le tri par insertion de `n` éléments d'un `Tableau t`.



Validez vos procédures avec la solution.

Solution Java @[pginsertion.java]

```
/**
  Insere l'élément t[k] dans t[0..k-1] trie
  @param[in,out] t - un ITableau
  @param[in] k - indice d'insertion
*/

public static void insererElementBasique(int[] t, int k)
{
  boolean permutation = true;
  int j = k;
  while (j > 0 && permutation)
  {
    if (t[j] < t[j - 1])
    {
      UtilsTR.permuterTab(t,j,j - 1);
      --j;
    }
    else
    {
      permutation = false;
    }
  }
}
```

```java
/**
  Tri par insertion d'un ITableau
  @param[in,out] t - un ITableau
  @param[in] n - nombre d'éléments
*/

public static void triInsertionBasique(int[] t, int n)
{
  for (int j = 1; j < n; ++j)
  {
    UtilsTR.afficherTri(t,n,j,n);
    insererElementBasique(t,j);
  }
  UtilsTR.afficherTri(t,n,n,n);
}
```

Écrivez un programme qui saisit le nombre d'éléments, déclare un `ITableau2` et l'initialise de façon aléatoire en prenant 20 pour valeur maximale puis effectue son tri et l'affiche.

Testez.

2.2 Tri par insertion, version 2

Principe de la sentinelle

Afin d'éviter le débordement vers la gauche lors de l'insertion de l'élément `t[k]` dans le sous-tableau trié `t[1..k-1]`, une autre solution est de mettre une sentinelle en `t[1]`, c.-à-d. un élément dont on est certain qu'il va arrêter la boucle. Il suffit pour cela qu'il soit inférieur à tout élément rencontré, ce qui est le cas si l'on recherche au préalable l'élément minimum et qu'on le met à cette place.

Écrivez une fonction `indiceMinTab(t,n)` qui calcule et renvoie l'**indice** de l'élément contenant la plus petite valeur parmi les `n` premières valeurs d'un `ITableau t`. En cas d'ex-aequo, c'est l'indice le plus petit qui sera renvoyé.

Validez votre fonction avec la solution.

Solution Java @[UtilsTBOpers.java]

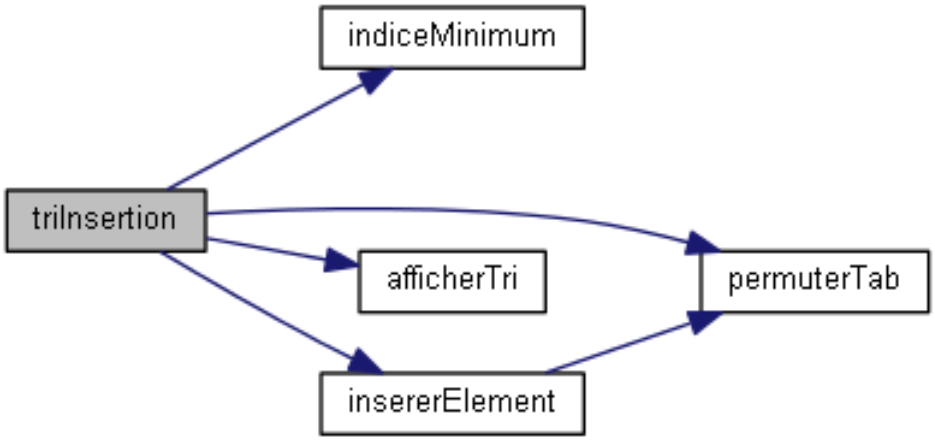
```java
/**
  Indice du minimum d'un ITableau
  @param[in] t - un ITableau
  @param[in] n - nombre de valeurs dans [1..TMAX[
  @return l'indice du minimum des n valeurs de t
*/

public static int indiceMinTab(final int[] t, int n)
{
  int imin = 0;
  int vmin = t[imin];
```

```
  for (int j = 1; j < n; ++j)
  {
    if (t[j] < vmin)
    {
      imin = j;
      vmin = t[imin];
    }
  }
  return imin;
}
```

Copiez/collez la procédure `insererELementBasique` en `insererElement(t,k)`, puis modifiez la procédure de sorte qu'elle insère l'élément `t[k]` dans le sous-tableau trié `t[1..k-1]`, l'élément minimal du tableau étant en `t[1]`. La procédure utilisera `permuterTab` pour réaliser l'échange de deux éléments de `t`.

Copiez/collez la procédure `triInsertionBasique` en `triInsertion(t,n)`, puis modifiez la procédure de sorte qu'elle recherche tout d'abord l'élément minimum des `n` éléments de `t` et le place en `t[1]` afin qu'il joue le rôle de sentinelle puis effectue la procédure d'insertion à partir de l'élément 2.



Validez vos procédures avec la solution.

Solution Java @[pginsertion.java]

```
/**
  Insère t[k] dans t[0..k-1] trie; élément minimum en t[0]
  @param[in,out] t - un ITableau
  @param[in] k - indice d'insertion
*/

public static void insererElement(int[] t, int k)
```

```
{
  int j = k;
  while (t[j] < t[j - 1])
  {
    UtilsTR.permuterTab(t,j,j - 1);
    --j;
  }
}
```

```
/**
  Tri par insertion d'un ITableau
  @param[in,out] t - un ITableau
  @param[in] n - nombre d'éléments
*/

public static void triInsertion(int[] t, int n)
{
  int j0 = indiceMinTab(t,n);
  UtilsTR.permuterTab(t,0,j0);
  for (int j = 1; j < n; ++j)
  {
    UtilsTR.afficherTri(t,n,j,n);
    insererElement(t,j);
  }
  UtilsTR.afficherTri(t,n,n,n);
}
```

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

Testez.

Validez votre programme avec la solution.

Solution Java @[pginsertion.java]

```
public static void main(String[] args)
{
  int nelems = UtilsTR.saisirNombreElements(TMAX);
  int[] tab = new int[TMAX];
  UtilsTR.aleatoireTri(tab,nelems,20);
  System.out.println("Tableau initial");
  UtilsTR.afficherTri(tab,nelems,0,nelems);
  System.out.println("Tri par insertion");
  triInsertionBasique(tab,nelems);
  triInsertion(tab,nelems);
  System.out.println("Tableau final");
  UtilsTR.afficherTri(tab,nelems,0,nelems);
}
```

2.3 Conclusion

Le nombre d'opérations du tri par insertion dépend fortement de l'ordre initial des éléments de la suite. Dans le cas où la suite d'éléments est déjà ordonnée, très peu d'opérations sont nécessaires. Le tri par insertion est donc un bon tri, si la suite a de bonnes chances d'être ordonnée (ou peu d'éléments ne sont pas dans le bon ordre).

3 Références générales

Comprend [Felea-PG1 :c3 :ex82], [Moliner-ML1 :c2 :ex14], [Tarlowski-PG1 :c3 :ex14] ■