Tri par sélection [tr04] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 21 mai 2018

Table des matières

C++ - Tri par sélection (Solution)

Mots-Clés Algorithmes de tris et rangs, Tri par sélection ■
Requis Axiomatique impérative (sauf Fichiers) ■
Difficulté • ○ ○ (30 min) ■

Objectif
Cet exercice réalise le tri par sélection 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 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++

```
const int TMAX = ...;
using ITableau = int[TMAX];
```

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

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

C++ @[UtilsTR.cpp]

Copiez/collez ensuite les lignes suivantes :

C++ **Au début** de votre programme :

```
#include "UtilsTR.cpp"
using namespace UtilsTR;
```

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.cpp)

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.cpp)

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.cpp)

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.cpp)

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].

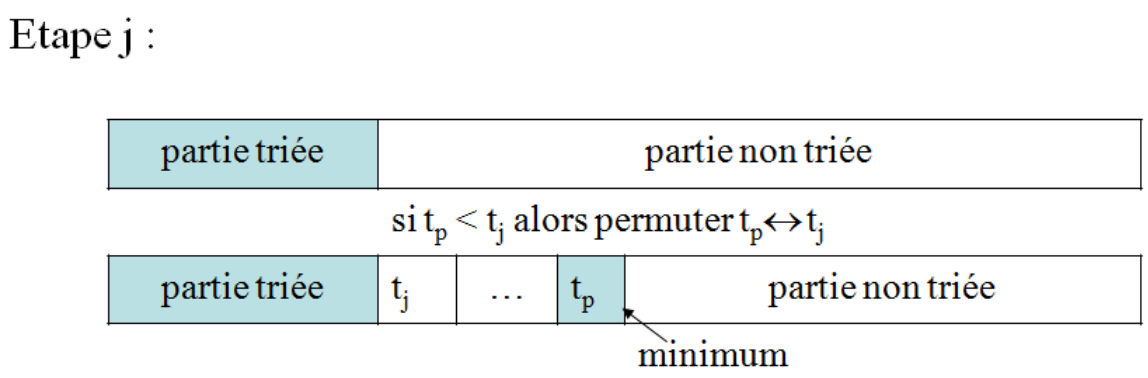
...(suite page suivante)...

2 Tri par sélection / pgselection

2.1 Tri par le minimum

Principe du tri par sélection

Le **tri par sélection** d'un tableau `t[1..n]` de `n` éléments consiste, pour `j` variant de 1 à `n-1`, à déterminer l'élément minimum du sous-tableau `t[j..n]` et à l'échanger avec l'élément `t[j]`.



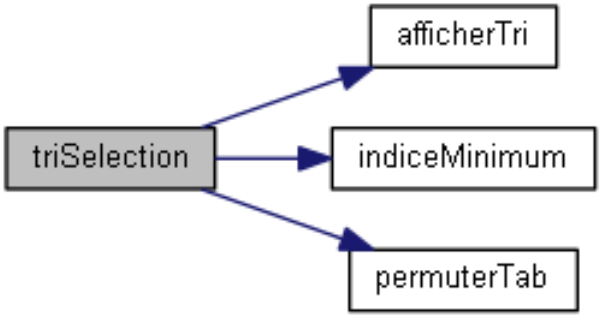
Exemple

(Avec affichage des tableaux successifs)

```
Nombre d'éléments dans [1..50]? 10
Tableau initial
[ 18 16  4  3 11  3  5  4 11 10  ]
Tri par selection
[|18 16  4  3 11  3  5  4 11 10 |]
[  3|16  4 18 11  3  5  4 11 10 |]
[  3  3| 4 18 11 16  5  4 11 10 |]
[  3  3  4|18 11 16  5  4 11 10 |]
[  3  3  4  4|11 16  5 18 11 10 |]
[  3  3  4  4  5|16 11 18 11 10 |]
[  3  3  4  4  5 10|11 18 11 16 |]
[  3  3  4  4  5 10 11|18 11 16 |]
[  3  3  4  4  5 10 11 11|18 16 |]
[  3  3  4  4  5 10 11 11 16|18 |]
Tableau final
[  3  3  4  4  5 10 11 11 16 18  ]
```

Écrivez une fonction `indiceMinSSTab(t,binf,bsup)` qui calcule et renvoie la position d'un minimum de `t[binf..bsup]` d'un `ITableau t`.

Écrivez une procédure `triSelection(t,n)` qui effectue le tri par sélection de `n` éléments d'un `ITableau t`. La procédure utilisera `permuterTab` pour réaliser l'échange de deux éléments de `t` et la fonction `indiceMinimum` pour la recherche de l'indice minimum.



Validez votre fonction et procédure avec la solution.

Solution C++ @[pgselection.cpp]

```
/**
  Position d'un minimum d'un sous-ITableau
  @param[in] t - un ITableau
  @param[in] binf - borne inférieure du sous-tableau
  @param[in] bsup - borne supérieure du sous-tableau
  @return Position d'un minimum de t[binf..bsup]
*/

int indiceMinSSTab(const ITableau& t, int binf, int bsup)
{
  int imin = binf;
  for (int j = binf + 1; j <= bsup; ++j)
  {
    if (t[j] < t[imin])
    {
      imin = j;
    }
  }
  return imin;
}
```

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

void triSelection(ITableau& t, int n)
{
  int bsup = n - 1;
  for (int j = 0; j < bsup; ++j)
  {
    UtilsTR::afficherTri(t,n,j,n);
    int k = indiceMinSSTab(t,j,bsup);
    if (k != j)
    {
```

```
      UtilsTR::permuterTab(t,j,k);
    }
  }
  UtilsTR::afficherTri(t,n,n,n);
}
```

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

Testez.

2.2 Tri par le maximum

Symétrie du tri par sélection

Par symétrie, nous pouvons déterminer l'élément maximum et l'échanger avec l'élément `t[j]`.

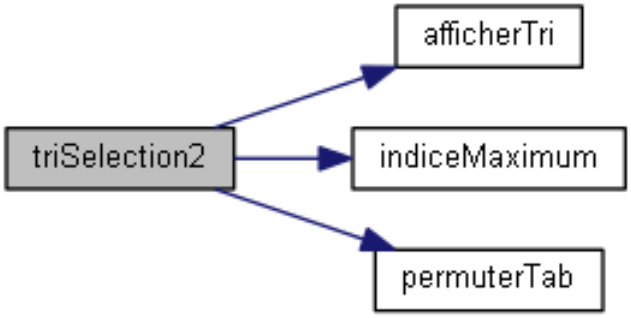
Exemple

(Avec affichage des tableaux successifs)

```
Nombre d'éléments dans [1..50]? 10
Tableau initial
[ 15  5  1 14  5  9 16  8 18 13  ]
Tri par selection (version 2)
[|15  5  1 14  5  9 16  8 18 13 |]
[|15  5  1 14  5  9 16  8 13 |18 ]
[|15  5  1 14  5  9 13  8 |16 18 ]
[| 8  5  1 14  5  9 13 |15 16 18 ]
[| 8  5  1 13  5  9 |14 15 16 18 ]
[| 8  5  1  9  5 |13 14 15 16 18 ]
[| 8  5  1  5 | 9 13 14 15 16 18 ]
[| 5  5  1 | 8  9 13 14 15 16 18 ]
[| 1  5 | 5  8  9 13 14 15 16 18 ]
[| 1 | 5  5  8  9 13 14 15 16 18 ]
Tableau final
[  1  5  5  8  9 13 14 15 16 18  ]
```

Écrivez une fonction `indiceMaxSSTab(t,binf,bsup)` qui calcule et renvoie la position d'un maximum de `t[binf..bsup]` d'un `ITableau t`.

Copiez/collez la procédure `triSelection` en `triSelection2(t,n)`, puis modifiez la procédure de sorte qu'elle utilise la fonction `indiceMaximum` en place de la fonction `indiceMinSSTab`.



Validez votre fonction et procédure avec la solution.

Solution C++ @[pgselection.cpp]

```
/**
  Position d'un maximum d'un sous-ITableau
  @param[in] t - un ITableau
  @param[in] binf - borne inférieure du sous-tableau
  @param[in] bsup - borne supérieure du sous-tableau
  @return Position d'un maximum de t[binf..bsup]
*/

int indiceMaxSSTab(const ITableau& t, int binf, int bsup)
{
  int imax = binf;
  for (int j = binf + 1; j <= bsup; ++j)
  {
    if (t[imax] < t[j])
    {
      imax = j;
    }
  }
  return imax;
}
```

```
/**
  Tri par séléction (par le maximum) d'un ITableau
  @param[in,out] t - un ITableau
  @param[in] n - nombre d'éléments
*/

void triSelection2(ITableau& t, int n)
{
  for (int j = n - 1; j > 0; --j)
  {
    UtilsTR::afficherTri(t,n,0,j);
    int k = indiceMaxSSTab(t,0,j);
    if (k != j)
    {
      UtilsTR::permuterTab(t,j,k);
    }
  }
  UtilsTR::afficherTri(t,n,0,0);
}
```

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

Testez.

Validez votre programme avec la solution.

Solution C++ @[pgselection.cpp]

```
int main()
{
  int nelems = UtilsTR::saisirNombreElements(TMAX);
  ITableau tab;
  UtilsTR::aleatoireTri(tab,nelems,20);
  cout<<"Tableau initial"<<endl;
  UtilsTR::afficherTri(tab,nelems,0,nelems);
  cout<<"Tri par selection"<<endl;
  triSelection(tab,nelems);
  triSelection2(tab,nelems);
  cout<<"Tableau final"<<endl;
  UtilsTR::afficherTri(tab,nelems,0,nelems);
}
```

3 Références générales

Comprend [Moliner-ML1 :c2 :ex15] ■