Algorithmes de tri Compte Rendu du MiniProjet

Data · April 2011

CITATIONS
0

READS
1,081

1 author:

Mohamed Housni
Sorbonne Université
1 PUBLICATION **0** CITATIONS

SEE PROFILE

Compte Rendu du MiniProjet
Algorithmes de tri en langage C

Réalisé par
- Mohamed HOUSNI
  1ère GEGM I-1

Encadré par
- Pr Mohamed BENAHMED

Contenu

(capture d'écran)

```
Programme  : Algorithmes De Tri / © Version 01 Le 12/10/2010
Réalisé par: Mohamed HOUSNI     / 1ère GEGM I-1
Encadré par: Pr Mohammed BENAHMED

Donner le nombre d'entiers que vous voulez trier: 10

Donner l'entier 1: 9
Donner l'entier 2: 1
Donner l'entier 3: 6
Donner l'entier 4: 2
Donner l'entier 5: 4
Donner l'entier 6: 5
Donner l'entier 7: 8
Donner l'entier 8: 3
Donner l'entier 9: 0
Donner l'entier 10: 7

1. Le tri à bulle
2. Le tri par séléction
3. Le tri par permutation
4. Le tri par insertion
5. Le tri par fusion
6. Le tri rapide

Veuillez saisir le numéro de de l'algorithme de tri à appliquer: 7

(!) Ce numéro ne figure pas dans la liste !

Veuillez saisir le numéro de de l'algorithme de tri à appliquer: 6

TRIÉS! :   0  1  2  3  4  5  6  7  8  9

Appuyez sur une touche pour continuer...
```

Cachier des charges

1. Demander à l'utilisateur de remplir un tableau Tab avec des entiers.
2. Proposer à l'utilisateur de choisir entre les algorithmes de tri suivant:
    1. Le tri à bulle
    2. Le tri par sélection
    3. Le tri par permutation
    4. Le tri par insertion
    5. Le tri par fusion
    6. Le tri rapide
3. Renvoyer le tableau dans lequel les entiers sont classés par ordre croissant.

Présentation des algorithmes de tri utilisés

Algorithme	**Code C**
Le tri à bulle L'algorithme parcourt la liste, et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet de la liste, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que la liste est triée : l'algorithme peut s'arrêter.	*// tri à bulle* #define TRUE 1 #define FALSE 0 **void** tri_a_bulle(**int** *t, **int** n) { **int** j = 0; **int** tmp = 0; **int** en_desordre = 1; **while**(en_desordre) { en_desordre = FALSE; **for**(j = 0 ; j < n- 1 ; j++){ **if**(t[j] > t[j+1]){ tmp = t[j+1]; t[j+1] = t[j]; t[j] = tmp; en_desordre = TRUE; } } } }
Le tri par séléction Le tri par sélection (ou tri par extraction) est un des algorithmes de tri les plus triviaux. Il consiste en la recherche soit du plus grand élément (ou le plus petit) que l'on va replacer à sa position finale c'est-à-dire en dernière position (ou en première), puis on recherche le second plus grand élément	*// tri par sélection* **void** tri_selection(**int** *t, **int** n) { **int** i, min, j , tmp; **for**(i = 0 ; i < n - 1 ; i++) { min = i; **for**(j = i+1 ; j < n ; j++) **if**(t[j] < t[min]) min = j; **if**(min != i) {

(ou le second plus petit) que l'on va replacer également à sa position finale c'est-à-dire en avant-dernière position (ou en seconde), etc., jusqu'à ce que le tableau soit entièrement trié.

```
            tmp = t[i];
            t[i] = t[min];
            t[min] = tmp;
        }
    }
}
```

Le tri par permutation

Le tri par permutation est le tri du jeu de cartes, il consiste à parcourir le tableau jusqu'à ce que l'on trouve un élément plus petit que le précedent donc mal placé.
On prend cet élément et on le range à sa place dans le tableau puis on continue la lecture.
On s'arrête à la fin du tableau.

```
// tri par permutation
void tri_permutation(int *t, int n)
{
      int i,s=0,k;
      int nb[n];
      int res [n];
      for(i=0;i<n;i++)
            {
                  for(k=0;k<n;k++){
                        if(t[i]>t[k]) s++;
                           nb[i]=s;
                  }
            res[s]=t[i];
                  s=0;
            }
            for( i=0;i<n;i++)
                  t[i]=res[i];
}
```

Tri par insertion

Le tri par insertion consiste à parcourir la liste : on prend les éléments dans l'ordre. Ensuite, on les compare avec les éléments précédents jusqu'à trouver la place de l'élément qu'on considère. Il ne reste plus qu'à décaler les éléments du tableau pour insérer l'élément considéré à sa place dans la partie déjà triée.

```
// tri par insertion
void tri_insertion(int *t, int n)
{
   int i,p,j;
   int x;

   for (i = 1; i < n; i++)
   {
      x = t[i];
      p = i-1;
      while (t[p] > x && p-- > 0) {}
      p++;
      for (j = i-1; j >= p; j--) {
         t[j+1] = t[j];
      }
      t[p] = x;
   }
}
```

Tri par fusion

Le tri fusion est construit

```
// tri par fusion
void fusion(int *t,int deb1,int fin1,int fin2)
```

suivant la stratégie "diviser pour régner", en anglais "divide and conquer". Le principe de base de la stratégie "diviser pour régner" est que pour résoudre un gros problème, il est souvent plus facile de le diviser en petits problèmes élémentaires. Une fois chaque petit problème résolu, il n'y a plus qu'à combiner les différentes solutions pour résoudre le problème global.
La méthode "diviser pour régner" est tout à fait applicable au problème de tri : plutôt que de trier le tableau complet, il est préférable de trier deux sous tableaux de taille égale, puis de fusionner les résultats.
L'algorithme proposé ici est récursif. En effet, les deux sous tableaux seront eux même triés à l'aide de l'algorithme de tri fusion. Un tableau ne comportant qu'un seul élément sera considéré comme trié : c'est la condition sine qua non sans laquelle l'algorithme n'aurais pas de conditions d'arrêt. Etapes de l'algorithme :
- Division de l'ensemble de valeurs en deux parties
- Tri de chacun des deux ensembles
- Fusion des deux ensembles

```
{
   int *table1;
   int deb2=fin1+1;
   int compt1=deb1;
   int compt2=deb2;
   int i;
   table1=(int *)malloc((fin1-deb1+1)*sizeof(int));
   for(i=deb1;i<=fin1;i++)
        table1[i-deb1]=t[i];
   for(i=deb1;i<=fin2;i++){
         if (compt1==deb2) break;
         else if (compt2==(fin2+1)) {
            t[i]=table1[compt1-deb1];
            compt1++;
            }
         else if (table1[compt1-deb1]<t[compt2]){
            t[i]=table1[compt1-deb1];
            compt1++;
            }
         else{
            t[i]=t[compt2];
            compt2++;
            }
         }
   free(table1);
}
void tri_fusion_bis(int *t,int deb,int fin)
{
if (deb!=fin){
      int milieu=(fin+deb)/2;
      tri_fusion_bis(t,deb,milieu);
      tri_fusion_bis(t,milieu+1,fin);
      fusion(t,deb,milieu,fin);
      }
}

void tri_fusion(int *t,int n)
{
   if (n>0) tri_fusion_bis(t,0,n-1);
}
```

Le tri rapide
La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui lui sont inférieurs soient à sa gauche et que tous ceux qui lui sont supérieurs soient à sa

```
// tri rapide
void echanger(int *t, int i, int j)
    {
    int tmp;
    tmp=t[i];
    t[i]=t[j];
    t[j]=tmp;
    }
```

droite. Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

```
int partition(int *t, int deb, int fin)
{
   int compt=deb;
   int pivot=t[deb];
   int i;

   for(i=deb+1;i<=fin;i++){
      if(t[i]<pivot){
         compt++;
         echanger(t,compt,i);
         }
      }
   echanger(t,compt,deb);
   return(compt);
   }

void tri_rapide_bis(int *t, int debut,int fin)
   {
   if(debut<fin)
      {
      int pivot=partition(t,debut,fin);
      tri_rapide_bis(t,debut,pivot-1);
      tri_rapide_bis(t,pivot+1,fin);
      }
   }

void tri_rapide(int *t,int n)
    {
    tri_rapide_bis(t,0,n-1);
    }
```

Code source du programme en C

```
#include <stdio.h>
#include <stdlib.h>
/* Définition des fonctions de tri */
// tri à bulle
#define TRUE 1
#define FALSE 0
void tri_a_bulle(int *t, int n)
{
	int j   = 0;
	int tmp = 0;
	int en_desordre = 1;
	while(en_desordre)
	{
		en_desordre = FALSE;
```

```
            for(j = 0 ; j < n- 1 ; j++)
            {
                  if(t[j] > t[j+1])
                  {
                        tmp = t[j+1];
                        t[j+1] = t[j];
                        t[j] = tmp;
                        en_desordre = TRUE;
                  }
            }
      }
}

// tri par sélection
void tri_selection(int *t, int n)
{
    int i, min, j , tmp;
    for(i = 0 ; i < n - 1 ; i++)
    {
       min = i;
       for(j = i+1 ; j < n ; j++)
           if(t[j] < t[min])
              min = j;
       if(min != i)
       {
          tmp = t[i];
          t[i] = t[min];
          t[min] = tmp;
       }
    }
}

// tri par permutation
void tri_permutation(int *t, int n)
{
      int i,s=0,k;
      int nb[n];
      int res [n];
      for(i=0;i<n;i++)
            {
                  for(k=0;k<n;k++){
                        if(t[i]>t[k]) s++;
                           nb[i]=s;
               }
            res[s]=t[i];
                  s=0;
            }
            for( i=0;i<n;i++)
                  t[i]=res[i];
}
```

```
// tri par insertion
void tri_insertion(int *t, int n)
{
   int i,p,j;
   int x;

   for (i = 1; i < n; i++)
   {
      x = t[i];
      p = i-1;
      while (t[p] > x && p-- > 0) {}
      p++;
      for (j = i-1; j >= p; j--) {
         t[j+1] = t[j];
      }
      t[p] = x;
   }
}

// tri par fusion
void fusion(int *t,int deb1,int fin1,int fin2)
{
   int *table1;
   int deb2=fin1+1;
   int compt1=deb1;
   int compt2=deb2;
   int i;
   table1=(int *)malloc((fin1-deb1+1)*sizeof(int));
   for(i=deb1;i<=fin1;i++)
        table1[i-deb1]=t[i];
   for(i=deb1;i<=fin2;i++){
         if (compt1==deb2) break;
         else if (compt2==(fin2+1)) {
            t[i]=table1[compt1-deb1];
            compt1++;
            }
         else if (table1[compt1-deb1]<t[compt2]){
            t[i]=table1[compt1-deb1];
            compt1++;
            }
         else{
            t[i]=t[compt2];
            compt2++;
            }
         }
   free(table1);
}
void tri_fusion_bis(int *t,int deb,int fin)
{
```

```
if (deb!=fin){
      int milieu=(fin+deb)/2;
      tri_fusion_bis(t,deb,milieu);
      tri_fusion_bis(t,milieu+1,fin);
      fusion(t,deb,milieu,fin);
      }
}
void tri_fusion(int *t,int n)
{
   if (n>0) tri_fusion_bis(t,0,n-1);
}

// tri rapide
void echanger(int *t, int i, int j)
    {
    int tmp;
    tmp=t[i];
    t[i]=t[j];
    t[j]=tmp;
    }

int partition(int *t, int deb, int fin)
{
   int compt=deb;
   int pivot=t[deb];
   int i;

   for(i=deb+1;i<=fin;i++){
      if(t[i]<pivot){
         compt++;
         echanger(t,compt,i);
         }
      }
   echanger(t,compt,deb);
   return(compt);
   }

void tri_rapide_bis(int *t, int debut,int fin)
   {
   if(debut<fin)
      {
      int pivot=partition(t,debut,fin);
      tri_rapide_bis(t,debut,pivot-1);
      tri_rapide_bis(t,pivot+1,fin);
      }
   }

void tri_rapide(int *t,int n)
    {
    tri_rapide_bis(t,0,n-1);
```

```
    }
/* Fin de la définition des fonctions de tri */

int main(void)
{
   int nb_entiers; // nombre d'entiers à entrer
   int *tab; // tableau des entiers
   int i; // compteur
   int num;

   // présentation
   printf("Programme  : Algorithmes De Tri / %c Version 01 Le 12/10/2010\n",184);
   printf("R%calis%c par: Mohamed HOUSNI     / 1%cre GEGM I-1\n",130,130,138);
   printf("Encadr%c par: Pr Mohammed BENAHMED\n\n",130);

   // lire nb_entiers
   printf("Donner le nombre d'entiers que vous voulez trier: ");
   scanf("%d",&nb_entiers);

   // allouer la mémoire pour tab[nb_entiers]
   tab=(int *)malloc(nb_entiers*sizeof(int));

   // remplir tab[nb_entiers]
   printf("\n");
   for(i=0;i<nb_entiers;i++){
     printf("Donner l'entier %d: ",i+1);
     scanf("%d",&tab[i]);
     }

   // liste des algorithmes de tri
   printf("\n1. Le tri %c bulle\n",133);
   printf("2. Le tri par s%cl%cction\n",130,130);
   printf("3. Le tri par permutation\n");
   printf("4. Le tri par insertion\n");
   printf("5. Le tri par fusion\n");
   printf("6. Le tri rapide\n");

   // choisir l'algorithme à appliquer
   do{
     printf("\nVeuillez saisir le num%cro de de l'algorithme de tri %c appliquer: ",130,133);
     scanf("%d",&num);
     if((num>6)||(num<1)) printf("\n(!) Ce num%cro ne figure pas dans la liste !\n",130);}
   while((num>6)||(num<1));

   // appliquer l'algorithme choisi
   if(num==1) tri_a_bulle(tab,nb_entiers);
   if(num==2) tri_selection(tab,nb_entiers);
   if(num==3) tri_permutation(tab, nb_entiers);
   if(num==4) tri_insertion(tab,nb_entiers);
   if(num==5) tri_fusion(tab,nb_entiers);
```

```
    if(num==6) tri_rapide(tab,nb_entiers);

    // résultat
    printf("\nTRI%cS! : ",144);
    for(i=0;i<nb_entiers;i++) printf("%3d",tab[i]);

    printf("\n\n");
    system("PAUSE");
    return 0;
}
```

Vous pouvez faire un copier coller de ce code pour l'essayer chez vous! ☺

Exemple d'utilisation

```
Programme  : Algorithmes De Tri / © Version 01 Le 12/10/2010
Réalisé par: Mohamed HOUSNI     / 1ère GEGM I-1
Encadré par: Pr Mohammed BENAHMED

Donner le nombre d'entiers que vous voulez trier: 10

Donner l'entier 1: 9
Donner l'entier 2: 1
Donner l'entier 3: 6
Donner l'entier 4: 2
Donner l'entier 5: 4
Donner l'entier 6: 5
Donner l'entier 7: 8
Donner l'entier 8: 3
Donner l'entier 9: 0
Donner l'entier 10: 7

1. Le tri à bulle
2. Le tri par séléction
3. Le tri par permutation
4. Le tri par insertion
5. Le tri par fusion
6. Le tri rapide

Veuillez saisir le numéro de de l'algorithme de tri à appliquer: 7

(!) Ce numéro ne figure pas dans la liste !

Veuillez saisir le numéro de de l'algorithme de tri à appliquer: 6

TRIÉS! :   0  1  2  3  4  5  6  7  8  9

Appuyez sur une touche pour continuer...
```

Vous pouvez faire un copier coller du code source de la page précédente pour l'essayer chez vous!

☺ ☺ ☺