Algorithmique et Programmation

TP4
Tris

1 Introduction

L'étude des mécanismes de tris a fait l'objet de nombreuses recherches en informatique. L'informatique de gestion est, par exemple, très demandeuse de ce genre de méthodes puisque les données y sont presque toujours préalablement triées avant d'être traitées.

Un algorithme de tri est un algorithme permettant d'ordonner un ensemble d'éléments selon un ordre déterminé. Il existe une grande variété d'algorithmes de tri. Leurs performances varient en fonction de l'ensemble considéré : un algorithme peut être très rapide pour ordonner une liste totalement désordonnée alors qu'il se révélera particulièrement médiocre avec une liste quasiment déjà triée. Il est donc important, pour le concepteur d'un programme informatique, de connaître les spécificités de chacun de ces algorithmes.

Nous vous proposons de partir à la découverte de quelques-uns des tris les plus connus. Le principe théorique de ces algorithmes sera illustré par un exemple servant de fil conducteur au TP : le tri d'allumettes de taille différentes et de nombre maximum ne dépassant pas une certaine valeur fixée (`MAXALLUMETTES` égale à 40). Les allumettes à trier peuvent avoir différentes couleurs afin de pouvoir rendre le tri plus visuel (des fonctions d'affichage seront fournies pour la traduction en C). Dans une première approche, il est demandé de ne trier les allumettes qu'en fonction de leur taille.

2 Entrées - Sorties

Le but de cette partie est de construire deux fonctions qui liront et écriront les fichiers relatifs au stockage des informations sur les allumettes (taille et couleur). Les données concernant chaque allumette sont lues à partir d'un fichier d'entrée et stockées dans un enregistrement qui peut alors être utilisé dans le corps du programme. On désire aussi être capable de sauvegarder la configuration du jeu d'allumettes (par exemple, pour garder une trace d'une configuration triée) : il est donc utile de disposer d'une fonction qui écrive dans un fichier l'ensemble des informations relatives à un état du jeu d'allumettes.

Format des fichiers. *Le format de fichier que nous avons choisi pour stocker les informations relatives à un jeu d'allumettes est le suivant : le fichier commence toujours par une ligne contenant un entier indiquant le nombre d'allumettes à trier. Suivent ensuite les informations propres à chaque allumette (une allumette par ligne) : d'abord un entier (compris entre 0 et 6) correspondant à sa couleur, puis, séparé avec un espace, un autre entier (compris entre 1 et 20) représentant sa taille. L'ordre dans lequel ces données sont lues correspond à l'ordre dans lequel les allumettes sont disposées.*

Format des données. Les informations sur les allumettes seront stockées dans un type enregistrement dédié baptisé t_Allumette. Les opérations de tri s'effectueront en manipulant un vecteur d'allumettes, de type t_VectAllumettes. On donne la définition de ces types :

```
types
    t_Allumette : Enregistrement

            entier  : couleur
            entier  : taille
    Fin_enregistrement

    t_VectAllumettes : vecteur de t_Allumette
```

Question 0. Ecrivez en C ces deux types de données dans un fichier entête `tris_def.h`. Nommez bien ainsi le fichier et respectez l'orthographe des noms des types, car ils sont utilisés tels quels dans les fonctions qui vous sont ensuite fournies dans le TP.

Question 1. Écrivez une fonction qui lit dans un fichier les données relatives à un jeu d'allumettes et les stocke au fur et à mesure dans un vecteur d'allumettes. **Implantez cette fonction en C dans un fichier `lectecr.c`.**

Question 2. Écrivez une fonction qui écrit un vecteur d'allumettes dans un fichier, en suivant le format décrit en bas de la page 1. **Implantez cette fonction en C dans votre fichier `lectecr.c`.**

Question 3. Illustrez le bon fonctionnement des fonctions écrites précédemment en écrivant en C une fonction `main`, dans un fichier `main.c`, qui lit un vecteur d'allumettes dans un fichier donné et le réécrit tel quel dans un autre fichier. Vous pourrez utiliser pour vos tests le fichier `allumettes_donnees.txt` disponible sur le serveur pédagogique.

Question 4. Pour afficher votre jeu d'allumettes, nous mettons à votre disposition trois fonctions dans le fichier `affichage.o` :

- `init` à appeler *une fois pour toutes* avant le premier affichage ;
- `finish` à appeler *une fois* à la fin du programme ;
- `affiche` à appeler dès que vous voulez afficher le vecteur d'allumettes à l'écran.

Les déclarations (prototypes) de ces fonctions sont données dans le fichier `affichage.h`.

L'affichage nécessite par ailleurs des fonctions définies dans la bibliothèque `ncurses`. Il faut donc ajouter cette bibliothèque lors de **l'édition de liens**. Celle-ci se fait alors par :

```
gcc -o main.out main.o lectecr.o affichage.o -lcurses
```

Complétez votre fonction `main` afin que le jeu d'allumettes lu soit affiché.

3 Tri par insertion

C'est le tri généralement utilisé par un joueur pour trier sa main dans un jeu de cartes. Il consiste à considérer chaque élément successivement et à l'insérer au bon endroit dans l'ensemble des éléments déjà triés. En voici, l'algorithme :

```
fonction tri_insertion

problème : Trier les allumettes par taille.

spécification :
    fonction :     allumettes ← tri_insertion(nbAllumettes, allumettes)
    paramètres :   entier nbAllumettes // le nombre d'allumettes à trier
                   t_VectAllumettes allumettes // vecteur des allumettes à trier
    résultats :    t_VectAllumettes allumettes // vecteur des allumettes triées
```

algorithme

```
variables locales
    entiers i, j // indices du vecteur d'allumettes

début
|   pour  i ← 0  à  nbAllumettes-1  faire
|   |   // initialisation
|   |   j ← 0
|   |   tant que  (j<i) et (allumettes(j).taille ≤ allumettes(i).taille)  faire
|   |   |   j ← j+1
|   |   fin tant que
|   |   // insertion de l'allumette au bon endroit
|   |   allumettes ← inserer(allumettes,i,j)
|   fin pour
fin
```

La spécification de la fonction `inserer` est la suivante :

```
spécification :
    fonction :     allumettes ← inserer(allumettes, i,j)
    paramètres :   t_VectAllumettes allumettes // vecteur des allumettes à trier
                   entier i // l'indice de l'allumette à insérer
                   entier j // l'indice auquel insérer l'allumette
    résultats :    t_VectAllumettes allumettes // vecteur des allumettes après l'insertion de
                   l'élément d'indice i à l'indice j, j ≤ i
```

Question 5. Écrivez l'algorithme de la fonction `inserer` et implantez-la en C dans un fichier `inserer.c`. Cette implantation devra n'utiliser qu'*un seul vecteur d'allumettes* qu'elle modifiera.

Question 6. Implantez la fonction `tri_insertion` en C dans un fichier `tri_insertion.c`. Cette implantation devra n'utiliser qu'*un seul vecteur d'allumettes* qu'elle modifiera. **Complétez et utilisez votre fonction C `main` pour mettre en évidence le bon fonctionnement de ce tri.**

Question 7. Modifiez la fonction `tri_insertion` de telle façon que, pour chaque valeur de i et j, les allumettes d'indices i et j et elles seules soient de couleur `BLANC`. `BLANC` est une constante entière définie dans `affichage.h` : vous pouvez donc colorier en blanc une allumette `a` par l'instruction `a.couleur ← BLANC`. **Testez à nouveau avec votre fonction `main`.**

Question 8. L'algorithme proposé trie le vecteur d'entrée en le modifiant. Ainsi son utilisation mémoire est minimale. Cependant, à chaque insertion, un décalage de tous les éléments plus grands

que l'élément inséré vers la fin du vecteur [1] est nécessaire.

En s'autorisant l'utilisation d'un vecteur supplémentaire (éventuellement plus grand) et en supposant que les éléments à trier sont tous différents quant au critère de tri, **décrire le principe d'un algorithme similaire mais se passant du coûteux décalage.**

Cet algorithme est-il toujours préférable ?

4 Tri à bulle

Soit n la taille du vecteur à trier. Le principe du tri à bulle est de comparer deux à deux les éléments successifs A_k et A_{k+1} du vecteur. S'ils ne sont pas dans le bon ordre, ils sont permutés. Puis on s'intéresse aux éléments A_{k+1} et A_{k+2} et ainsi de suite. Lorsque les éléments A_{n-1} et A_n ont été comparés, l'élément le plus grand (ou le plus petit selon l'ordre voulu) se trouve à l'emplacement n. Il faut ensuite itérer ce traitement sur les n-1 valeurs non triées, puis les n-2 valeurs et ainsi de suite jusqu'à trier tout le vecteur.

Exemple. Soit à trier la suite d'entiers $8, 4, 9, 1$. L'application de l'algorithme donne (à lire colonne par colonne) :

$\boxed{8, 4}, 9, 1$	$\boxed{4, 8}, 1, 9$	$\boxed{4, 1}, 8, 9$
$4, 8, 9, 1$	$4, 8, 1, 9$	$1, 4, 8, 9$
$4, \boxed{8, 9}, 1$	$4, \boxed{1, 8}, 9$	
$4, 8, 9, 1$	$4, 1, 8, 9$	
$4, 8, \boxed{9, 1}$		
$4, 8, 1, 9$		

Question 9. Écrivez une fonction `permuter` qui permet d'échanger les emplacements de deux allumettes et implantez-la en C dans un fichier `bulles.c`.

Question 10. Donnez l'algorithme d'une fonction de tri à bulles et implantez-la dans le fichier `bulles.c`. Testez cette nouvelle fonction avec votre fonction `main`. Puis, comme à la question 7, **mettez en évidence** les allumettes qui sont comparées à chaque instant en les coloriant en blanc.

Question 11. Quel est le « pire cas », c'est-à-dire la configuration des éléments à trier pour laquelle l'algorithme fait le plus de permutations ? Donnez un majorant du nombre d'opérations (affectations et comparaisons) dans ce cas. Afin d'évaluer la performance (en temps) des algorithmes, on s'intéresse en général à ce nombre d'opérations au pire cas quand la taille des données à traiter tend vers l'infini. On l'appelle *complexité asymptotique au pire cas*. On peut également regarder la complexité asymptotique au meilleur cas et en moyenne pour compléter l'analyse.

Question 12. Que faut-il modifier pour trier les allumettes non seulement par taille, mais aussi par couleur (en supposant donc préalablement défini un ordre sur les couleurs) ? Implantez ces modifications en C et illustrez le bon fonctionnement de l'ensemble sur quelques jeux d'essai significatifs.

1. Selon l'ordre choisi pour le tri