École Centrale de Nantes

Algorithmique et Programmation

TP3
Fichiers et tris

FICHIERS À RENDRE :

```
-- rapport_TP3.pdf (contenant le lien vers le projet repl.it)
```

1 Introduction

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 connaitre les spécificités de chacun de ces algorithmes.
Nous vous proposons d'implémenter 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 tailles et couleurs différentes. Des fonctions d'affichage seront fournies pour la traduction en C++ avec la librairie ncurses. 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 (couleur et taille). 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 du fichier : `allumettes_donnees.txt` *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 nommé `Allumette`. Les opérations de tri s'effectueront en manipulant un vecteur d'allumettes, de type `VectAllumettes`. On donne la définition de ces types :

```
types
    Allumette : Enregistrement

            entier  : couleur
            entier  : taille
    Fin_enregistrement

    VectAllumettes : tableau de Allumette
```

Question 1. Écrivez 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 2. Écrivez l'algorithme d'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. **Implémentez cette fonction en C++ dans un fichier `utils.cpp`.**

Question 3. Écrivez l'algorithme d'une fonction qui écrit un vecteur d'allumettes dans un fichier, en suivant le format décrit ci-dessus. **Implémentez cette fonction en C++ dans votre fichier `utils.cpp`.**

Question 4. Illustrez le bon fonctionnement des fonctions écrites précédemment en écrivant en C++ la fonction `main`, 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 5. Pour afficher votre jeu d'allumettes, nous mettons à votre disposition trois fonctions dans le fichier `affichage.cpp` :

- `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`. Nous vous fournissons un fichier `Makefile` permettant la compilation avec cette librairie. Ce fichier est à déposer avec les fichiers sources, il sera utilisé par `repl.it` pour la compilation.

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 un algorithme adapté pour l'affichage des allumettes dans ce TP :

```
fonction tri_insertion

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

algorithme

variables locales
    entiers i, j // indices du vecteur d'allumettes
début
|   pour  i ← 1  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 fonction `inserer` insère l'élément d'indice i à l'indice j, avec $j \leq i$:

```
spécification :
    fonction :      allumettes ← inserer(allumettes, i,j)
    paramètres :    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 :     VectAllumettes allumettes // vecteur après l'insertion
```

Question 6. Écrivez l'algorithme de la fonction `inserer` et implémentez-la en C++ dans un fichier `tris.cpp`. Cette implantation devra n'utiliser qu'*un seul vecteur d'allumettes* qu'elle modifiera.

Question 7. Implémentez la fonction `tri_insertion` en C++ dans le fichier `tris.cpp`. Cette implantation devra n'utiliser qu'*un seul vecteur d'allumettes* qu'elle modifiera. **Complétez et utilisez votre fonction `main` pour mettre en évidence le bon fonctionnement de ce tri.**

Question 8. Modifiez la fonction `tri_insertion` de telle façon que, pour chaque valeur de i et j, le jeu d'allumettes soit affiché. **Testez à nouveau avec votre fonction `main`.**

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 10. Écrivez une fonction `permuter` qui permet d'échanger les emplacements de deux allumettes et implémentez-la en C++ dans le fichier `tris.cpp`.

Question 11. Implémentez l'algorithme d'une fonction de tri à bulles dans le fichier `tris.cpp`. Testez cette nouvelle fonction avec votre fonction `main`.

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) ? Implémentez ces modifications en C++ et illustrez le bon fonctionnement de l'ensemble sur quelques jeux d'essai significatifs.

Question 13 (Bonus). 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.

5 Comparaison

Il est maintenant intéressant de comparer les algorithmes de tri avec des jeux de données un peu plus réalistes.

Question 14. Écrivez un algorithme et implémentez la fonction `generationAleatoire` en C++ dans le fichier `utils.cpp` pour générer un tableau d'allumettes contenant un nombre d'allumettes passé en paramètre avec une répartition aléatoire de la taille et de la couleur.

Question 15. Comparez les temps d'exécution de vos algorithmes de tris avec des jeux de données de 1000, 10000, 100000 allumettes. Prenez soin de supprimer les fonctions d'affichage pour faire ces tests.

6 Tri rapide (Bonus)

Dans le tri rapide, on choisit un pivot (souvent le dernier élément du tableau), on range les éléments du tableau dans deux ensembles plus petit ou plus grand que le pivot, sans que ces éléments soient triés entre eux. Puis on replace le pivot à la place qui lui revient (puisqu'on connait le nombre d'élément plus petits). Et on recommence l'opération de tri avec les deux sous ensembles (plus petits et plus grands que le pivot).

Question 16. Écrivez un algorithme et implémentez une fonction de tri rapide dans le fichier `tris.cpp` et comparez-la avec les précédentes.

Question 17. Utilisez la méthode `sort` **de vector** pour faire le tri et comparez le résultats avec les implémentations précédentes.