UNIVERSITÉ PARIS DESCARTES

DUT Informatique - 1ère année
SDA - Structures de Données et Algorithmes

TRAVAUX DIRIGES SUR MACHINE N°7
Implémentation d'algorithmes de tri

Thèmes : Utilisation de la fonction `qsort` et programmation d'un tri par dénombrement

Les algorithmes de tri sont très nombreux. Les meilleurs ont une complexité temporelle moyenne de l'ordre de `n`.log `n` (où `n` désigne la taille des données). Toutefois, il est possible de définir des algorithmes plus efficaces en exploitant des propriétés des données à trier et/ou en consommant de la mémoire. C'est le cas de l'algorithme de tri par dénombrement. Pour pouvoir valider votre implémentation et pouvoir estimer ses performances, nous commençons par étudier l'utilisation de la fonction de tri de la bibliothèque standard du C. Nous nous limiterons à trier par ordre croissant des tableaux d'entiers.

Exercice 1. Étude du programme de test et de comparaison de deux algorithmes de tri

1.1. Créez un nouveau projet de type « `Projet C++` » sem07-tp-Cpp1. A partir des sources de TP, importez du répertoire `Exo1` le fichier `testCountingSortVsQuickSort.cpp`.

1.2. Le programme principal réalise le traitement suivant :

1. Saisie des paramètres : i) la taille du tableau à trier, ii) la borne strictement supérieure des valeurs générées aléatoirement
2. Création dynamique de deux tableaux
3. Initialisation d'un des deux avec des valeurs aléatoires
4. Affichage (contrôlé) des valeurs du tableau initial (à trier)
5. Copie de celui-ci dans le deuxième tableau
6. Tri (chronométré) du 1er tableau par un appel à la fonction `quickSort`
7. Tri (chronométré) du 2ème tableau par un appel à la fonction `countingSort`
8. Vérification de la validité des tris et affichage (contrôlé) des résultats (temps d'exécution des tris et des tableaux triés)

Les points 3, 4, 5 et 8 sont réalisés par le biais de fonctions déclarées dans `util.h`. Leurs noms sont suffisamment explicites pour comprendre leur rôle.

Les fonctions `quickSort` et `countingSort` sont déclarées dans `sort.h`. Votre travail (dans les exercices suivants) va consister à les programmer (dans `sort.cpp`).

Exercice 2. Utilisation de la fonction `qsort`

2.1. Créez un nouveau projet de type « `Projet C++` » sem07-tp-Cpp2. Importez les fichiers du répertoire `Exo2` : `sort.h`, `sort.cpp`, `util.h`, `util.cpp` et `testQuickSort.cpp`.

2.2. La bibliothèque standard du C comporte une fonction de tri nommée `qsort`. Cette fonction implémente l'algorithme de tri rapide. Nous rappelons que la complexité moyenne en temps de cet algorithme est Θ(n.log `n`) et Θ (log `n`) en espace. La fonction `qsort` permet de trier des tableaux contenant des éléments de n'importe quel type. Pour se faire, il faut lui fournir en paramètre (un pointeur vers) la fonction permettant de comparer deux éléments. Le prototype de cette fonction (déclaré dans `stdlib.h`) est le suivant :

```
void qsort ( void* tab,
             size_t nb_elem,
             size_t taille_elem,
             int (*compare)(const void*, const void*));
```

La fonction `qsort` trie une table contenant `nb_elem` éléments de taille `taille_elem`. Le paramètre `tab` pointe sur le début de la table. Le contenu de la table est trié en ordre croissant, en utilisant la fonction de comparaison pointée par `compare`, laquelle est appelée avec deux arguments pointant sur les objets à comparer. La fonction de comparaison doit renvoyer un entier inférieur, égal, ou supérieur à zéro si le premier argument est respectivement considéré comme inférieur, égal ou supérieur au second. Si la comparaison des deux arguments renvoie une égalité (valeur de retour nulle), l'ordre des deux éléments est indéfini (l'algorithme peut les permuter ou non).

La fonction de comparaison doit respecter le prototype indiqué ci-dessus. Les deux paramètres sont des pointeurs quelconques (vers n'importe quel type de données). Bien entendu, dans votre application, les pointeurs désigneront des entiers. Pour pouvoir les employer en tant que pointeur d'entier, vous devez réaliser une conversion explicite. Si `p` est de type `void*`, sa conversion en un pointeur d'entier est notée `(int*)p`.

Complétez le code des fonctions `compare` et `quicksort` dans le fichier `sort.cpp`. Exécutez le programme de test, faites varier les paramètres (taille et valeur borne) pour apprécier les temps d'exécution.

Exercice 3. Implémentation d'un tri par dénombrement

3.1. Créez un nouveau projet de type « `Projet C++` » sem07-tp-Cpp3. Importez les fichiers du répertoire `Exo3` : `sort.h`, `sort.cpp`, `util.h`, `util.cpp` et `testCountingVsQuickSort.cpp`.

3.2. Le tri par dénombrement (*counting sort*) est très efficace lorsque la plage des valeurs contenues dans le tableau est limitée. Le principe du tri consiste à déterminer pour chaque valeur distincte à quel indice du tableau trié les éléments ayant cette valeur commenceront. Une fois ces indices

définis, il devient simple de construire le tableau trié. Le calcul de ces indices est réalisé de la manière suivante :

1. Calculer la plus petite (`min`) et la plus grande (`max`) valeurs présentes dans le tableau `t` devant être trié.
2. Allouer dynamiquement un tableau `tcpt` permettant de stocker un compteur par valeur possible (sa taille est donc `max - min + 1`) et initialiser chacun de ces compteurs à zéro.
3. Faire en sorte que chaque compteur `tcpt[i]` indique combien d'éléments du tableau `t` ont une valeur correspondant à ce compteur diminuée de `1` (attention, le compteur `tcpt[i]` correspond à l'élément de valeur `min + i`).
4. Faire en sorte que chaque compteur `tcpt[i]` indique combien d'éléments de `t` ont une valeur strictement inférieure à la valeur correspondant à ce compteur (donc une valeur strictement inférieure à `min + i`).

Le tableau contient à présent les indices en question.

Soit `t = [9, 4, 3, 7, 3, 2]`,

1. `min = 2` ; `max = 9`.
2. `tcpt = [0, 0, 0, 0, 0, 0, 0, 0]`
3. `tcpt = [0, 1, 2, 1, 0, 0, 1, 0]`
4. `tcpt = [0, 1, 3, 4, 4, 4, 5, 5]`

Le tableau `tcpt` nous indique que les éléments ayant la valeur `2` (le compteur correspondant est `tcpt[0]`) doivent être rangé à partir de l'indice `0` et que ceux portant la valeur `7` (`tcpt[5]`) à partir de l'indice `4`.

Dans le cas général, le tableau `t` ne peut pas être trié sur place (directement dans `t`). En effet, lorsque des données complémentaires sont associées aux données triées, le fait de savoir que les personnes ayant 2 ans doivent être placées à partir de l'indice `0` n'est pas suffisant pour reconstruire `t` dans le bon ordre. Les informations complémentaires stockées dans `t` ne peuvent être écrasées sans perte.

En conséquence, il est nécessaire :

5. d'allouer un nouveau tableau nommé `tmp` ayant la même taille que `t`,
6. de l'initialiser en appliquant l'algorithme suivant :

pour `i` **variant de** `0` **à** `size - 1` **faire**

- placer `t[i]` dans `tmp` à l'indice indiqué par le compteur correspondant.
- Incrémenter ce compteur de `1` de façon à ce que le prochain élément ayant la même valeur soit placer au bon endroit.

fin pour

Le tableau `tmp` peut alors être recopié dans `t`. Programmer cet algorithme en suivant les différentes étapes ci-dessus. Exécutez le programme de test, faites varier les paramètres pour apprécier les temps d'exécution.