TP n°8

Recherche et tris dans les tableaux

1 La recherche dans un tableau

Nous souhaitons écrire une application permettant de comparer les différents algorithmes de recherche vus en CMTD. Pour cela, nous allons créer un tableau contenant un ensemble d'entiers aléatoires, puis nous calculerons les temps nécessaires pour rechercher une série de valeurs aléatoires pour chaque algorithme de recherche. Afin de pouvoir comparer les trois algorithmes de recherche, nous effectuerons les recherches dans le même tableau et nous rechercherons la même série de valeurs.

Les quatre premières questions doivent être réalisées rapidement. Il s'agit de questions qui ont déjà été abordées dans des TP précédents.

1. Écrivez une fonction `aleatoire` qui prend en paramètres les bornes de l'intervalle $[a, b]$ et qui génère un entier aléatoire dans cet intervalle.
2. Écrivez une fonction `genererTableau` qui prend en paramètre un nombre de cases n, et les bornes de l'intervalle $[a, b]$. La fonction génère un tableau d'entiers de n cases, contenant des entiers aléatoires dans l'intervalle spécifié.
3. Écrivez la procédure `afficher` qui prend en paramètre un tableau d'entiers et qui affiche ses valeurs sur une seule ligne, séparées par un espace. À la fin de la fonction, on effectue un retour à la ligne.
4. Écrivez un `main` qui demande à l'utilisateur de saisir une taille et les bornes de l'intervalle $[a, b]$. Il génère ensuite un tableau contenant des valeurs aléatoires dans $[a, b]$ et l'affiche à l'écran.
5. Écrivez une fonction `rechercheSequentielle` qui prend en paramètres un tableau d'entiers et une valeur à rechercher. Cette fonction recherche la valeur spécifiée en paramètre et retourne l'indice de la case contenant cette valeur. Si la valeur n'est pas présente, la fonction retourne -1.
6. Modifiez le `main` pour qu'il demande à l'utilisateur un nombre de recherches à effectuer. Puis, un tableau de cette taille est créé et initialisé avec des valeurs aléatoires. Le `main` recherche ensuite chaque valeur de ce tableau dans le tableau précédent. Pour chaque valeur, il affiche la valeur recherchée et la position trouvée (ou -1 si la valeur n'existe pas).
7. Écrivez une fonction `rechercheSequentielleTriee` qui suppose que le tableau passé en paramètre est trié.
8. Modifiez votre `main` pour trier le tableau de valeurs afin de pouvoir comparer les deux algorithmes de recherche sur les mêmes bases. Vous utiliserez l'instruction `Arrays.sort(t)`, où t est le nom du tableau à trier (ajoutez l'instruction `import java.util.Arrays` au début de votre fichier). Après avoir affiché les recherches avec la fonction `rechercheSequentielle`, affichez les mêmes recherches avec la fonction `rechercheSequentielleTriee`.

Nous souhaitons maintenant comparer le temps moyen nécessaire pour chaque algorithme. Pour cela, nous souhaitons utiliser la fonction suivante (à ajouter dans votre classe) :

```
public static long getTemps() {
    Date d = new Date();
    return d.getTime();
}
```

Cette fonction retourne le temps écoulé depuis un temps de référence, en millisecondes (c'est ce qui est appelé un *timestamp*). Ainsi, pour calculer le temps nécessaire pour une série de recherches, il suffit de mémoriser le temps courant avant et après les recherches et de faire la différence entre les deux.

9. Réalisez ce calcul pour chaque algorithme de recherche et affichez le bilan à la fin de l'exécution.

Pour obtenir des temps différents, il est conseillé d'utiliser un tableau suffisamment grand (1000000 de cases par exemple), un intervalle de valeurs aléatoires assez grand ([0, 100000] par exemple) puis de générer un nombre de recherches suffisant (par exemple 10000).

Attention ! Il faut supprimer tous les affichages intermédiaires car l'affichage est plus coûteux que les recherches en elles-mêmes. Plutôt que de supprimer complètement les lignes, mettez-les en commentaires.

10. Quels sont les temps affichés pour chaque algorithme ?
11. Écrivez une fonction `rechercheDichotomique` qui implémente la recherche par dichotomie.
12. Modifiez votre `main` pour pouvoir comparer les trois algorithmes de recherche.

2 Comparaison entre les tris

1. Dans une classe `Tri`, recopiez les fonctions/procédures suivantes : `getTemps`, `aleatoire` et `genererTableau`.
2. Écrivez la fonction `clone` qui prend en paramètre un tableau et qui retourne un nouveau tableau ayant le même contenu.
3. Écrivez la procédure `triSelection` qui prend en paramètre un tableau d'entiers et qui le trie suivant l'algorithme du tri par sélection.

Dans un TP précédent, il était demandé d'écrire une fonction. Il faudra donc adapter votre code.

4. Écrivez un `main` qui demande à l'utilisateur la taille du tableau à trier et l'intervalle des valeurs aléatoires générées. Il génère ensuite un tableau en fonction de ces données. Avant d'appeler la procédure `triSelection`, faites une copie de ce tableau : c'est cette copie qui sera triée. Comme pour les algorithmes de recherche, déterminez le temps de calcul nécessaire.
5. Écrivez la procédure `triSelectionEchange` qui permet de trier un tableau passé en paramètre suivant l'algorithme du tri par sélection/échange.

6. Quels temps obtenez-vous pour un tableau de 1000 valeurs (peu importe l'intervalle des valeurs générées) pour les deux algorithmes ?
7. Faites de même pour 2000, 3000 et 10000 valeurs.

Les questions suivantes sont facultatives.

8. Écrivez la procédure `triInsertion` qui implémente l'algorithme du tri par insertion correspondant à l'exercice vu en TD.
9. Modifiez votre main pour comparer les différences entre les trois algorithmes.