Tris de tableaux

L'objectif de ce chapitre est de découvrir des algorithmes permettant de trier un tableau. Bien qu'il soit inutile de vous préciser l'importance de tels algorithmes, je me permets simplement de vous rappeler que sans tri, pas de recherche Google, pas moyen de trouver le plus court chemin entre votre appartement et l'O'Regans, et plus important que tout, impossible de trouver le handspinner [1] le moins cher du marché sur Amazon.

Nous allons découvrir dans ce chapitre trois tris différents. Nous verrons que chaque tri a ses propres caractéristiques et sera plus ou moins efficace suivant les données à trier. Pour aborder ces trois tris, la même technique sera abordée à chaque fois. Dans un premier temps, vous devrez regarder une vidéo vous montrant un exemple. Vous pourrez visionner cette vidéo autant de fois que vous le souhaitez. À partir de là, vous devrez rédiger en français le principe de l'algorithme avant de rédiger l'algorithme complet. Vient alors le temps de la programmation et des tests. Enfin, vous comparerez les différents algorithmes au travers d'une étude de complexité.

L'objectif est donc de répéter trois fois ce TD : les mêmes exercices sont valables pour chacun des algorithmes. N'hésitez pas à solliciter votre enseignant dès qu'une notion n'est pas claire.

Pour toute la suite de ce TD, les tris seront réalisés de façon croissante.
N'utilisez pas Internet pour trouver un algorithme tout prêt de chaque tri.

1 Travail à réaliser pour chaque tri

1.1 Compréhension du tri

La première partie de ce travail consiste à étudier un exemple de tri. Pour cela, il vous est demandé d'étudier une vidéo. Cette dernière débute par une (très) courte introduction, puis montre un exemple de tri sur le tableau 4, 2, 5, 3, 1. À vous de comprendre le fonctionnement du tri à l'aide de cet exemple. N'hésitez pas à visionner la vidéo autant de fois que nécessaire.

Voici les liens vers les vidéos [2] :

- tri à bulles : `https://youtu.be/OAv1kwSGFLw`
- tri sélection : `https://youtu.be/Q6b-1xk-1qA`
- tri insertion : `https://youtu.be/t-KynfOR6mw`

1. Trop 2017 ...
2. Vidéos qui ne sont plus du tout d'actualité, mais c'est ce qui fait leur charme !

Exercice 1. Vérification de la compréhension..
À partir de cet exemple, vous devriez être capable de comprendre le fonctionnement général de l'algorithme.

1) Complétez un tableau identique à celui présenté en annexe. La première ligne présente les cartes dans leur ordre initial. Écrivez l'état du tableau sur une nouvelle ligne dès qu'une permutation est effectuée dans la vidéo. Pour le tri insertion, nous considérons une insertion comme une seule étape.
2) Combien de parcours sont nécessaires pour trier l'ensemble du tableau ?
3) Dans chaque parcours, combien d'éléments compare-t-on au maximum ?
4) Résumez en quelques mots (pas plus de 3 lignes) le principe du tri.

1.2 Rédaction de l'algorithme

Avant de coder un tri, il faut bien évidemment rédiger l'algorithme. Les données d'entrées et de sortie sont identiques pour chacun d'eux, à savoir un tableau `tab` de `N` entiers en entrée et le même tableau trié en sortie.

Exercice 2. Algorithme.
À l'aide de vos réponses à l'exercice précédent, répondez aux questions suivantes.

1) La boucle principale de l'algorithme est-elle une boucle `POUR` ou `TANT QUE` ?
2) Même question pour la boucle réalisant un parcours.
3) Selon votre ressenti, rédigez d'abord un pseudo-algorithme puis l'algorithme complet du tri, ou rédigez directement l'algorithme final.
4) Vérifiez que votre algorithme fonctionne en le déroulant sur l'exemple 2, 4, 3, 6, 1, 5 en utilisant le format du tableau en annexe. Entourez au fur et à mesure les éléments qui, selon votre algorithme, ne seront plus permutés.

1.3 Programmation

Une fois que les algorithmes sont rédigés et testés, il est temps passer à la programmation. Conservez bien les fonctions que vous allez réaliser dans ce TD, elles vous resserviront au cours de l'année.

Exercice 3.
À l'aide de vos algorithmes rédigés lors de l'exercice précédent, vous devez rédiger une fonction C prenant en paramètres un pointeur vers un tableau d'entiers et un entier donnant le nombre d'éléments du tableau. Codez une fonction par tri permettant de réaliser un tri croissant du tableau.

Vous pouvez également coder une fonction permettant d'afficher l'ensemble des éléments d'un tableau. Tout comme les fonctions de tri, elle doit prendre en paramètres le pointeur vers le tableau et la taille de ce dernier.

Exercice 4.
Afin de vérifier que vos tris fonctionnent correctement, lancez-les sur les valeurs suivantes. Tous vos programmes doivent se terminer sans erreur en ayant correctement trié le tableau de façon croissante.

1) Éléments différents : 2, 4, 3, 6, 1, 5.
2) Répétitions : 3, 4, 1, 2, 1, 4.
3) Élément unique : 1, 1, 1, 1, 1, 1.
4) Tableau déjà trié : 1, 2, 3, 4, 5, 6.

2 Complexité

2.1 Optimisation

Vous avez pu observer que certains tris réalisent moins d'opérations que d'autres, en particulier le tri insertion. Il existe cependant des optimisations simples que l'on peut ajouter, notamment au tri à bulles. En effet, le principe de ce dernier est de parcourir `N` fois le tableau de `N` éléments. Cependant, on peut remarquer qu'à chaque parcours, le plus grand élément non trié remonte en fin de tableau.

Exercice 5.

1) Optimisez votre tri à bulles afin de réduire le nombre d'éléments comparés à chaque parcours.
2) Optimisez votre tri à bulles afin d'arrêter l'algorithme si le tableau est trié avant la fin des `N` parcours.
3) Optimisez vos autres tris afin d'arrêter l'algorithme si le tableau est trié.

2.2 Comparaison de performances

Afin de vérifier la théorie donnant le tri insertion comme le plus efficace de ces trois tris, nous allons créer un protocole de test. Évidemment, sur des tableaux de petites tailles, tous les tris se déroulent en quelques millisecondes. Il va falloir trier des grands tableaux et compter le nombre d'opérations réalisées dans chacun d'entre eux pour voir une différence.

Exercice 6. Basecode d'évaluation.

1) Ajoutez à vos codes deux variables globales nommées `comparaisons` et `echanges`.
2) Pour chaque tri, incrémentez les deux variables précédentes pour toute comparaison ou échange effectué.
3) Créez une fonction `initTabUniforme()` prenant en paramètres un pointeur `tab` sur un tableau d'entiers ainsi que sa taille `N`. Cette fonction doit initialiser le tableau en le remplissant des valeurs de 1 à `N` puis en le mélangeant à l'aide du *Knuth shuffle*[3].
4) Créez une fonction `initTabPresqueTrie()` prenant en paramètres un pointeur `tab` sur un tableau d'entiers déjà alloué ainsi que sa taille `N` et un entier `P`. Cette fonction doit initialiser le tableau en le remplissant des valeurs de 1 à `N` puis en permutant `P` fois deux éléments pris au hasard dans le tableau.
5) Créez une fonction `initTabRepetition()` prenant en paramètres un pointeur `tab` sur un tableau d'entiers déjà alloué, sa taille `N` et un entier `R`. Cette fonction doit initialiser le tableau en le remplissant d'entiers de 1 à `R` de façon aléatoire.

Exercice 7. Mesure de performance.

Il est maintenant temps de mesurer les performances de chaque tri. Utilisez le basecode créé dans l'exercice précédent afin de répondre aux questions suivantes.

1) Lancez 10 fois chacun de vos tris sur des tableaux comprenant les valeurs de 1 à 10000 mélangés uniformément. Quel algorithme est le plus rapide ?
2) Même question avec des tableaux presque triés (seuls 100 échanges ont été réalisés).
3) Même question sur des tableaux de 10000 éléments entre 1 et 100.

3. Voir la description sur Internet

3 À retenir

Vous avez découvert et implémenté trois tris. Vous devez être capable de rédiger chacun de ces algorithmes, les dérouler sur un exemple quelconque et les implémenter. Vous devez également connaitre leur caractéristiques principales et savoir quel algorithme à utiliser en fonction des données d'entrée. Résumons maintenant leurs caractéristiques.

Exercice 8. Synthèse.

Pour chacun des tris vus dans ce TD, répondez aux questions suivantes afin de résumer leurs propriétés principales. On cherche à trier un tableau de `N` éléments de façon croissante.

1) De façon générale, combien de parcours sont effectués au maximum pour trier complètement le tableau ?
2) Résumez le fonctionnement de chaque algorithme en une phrase.
3) Donnez un exemple d'entrées dans lequel chaque tri est particulièrement efficace.

Afin de finir de vous convaincre, allez voir la course aux tris sur cette page : `https://www.toptal.com/developers/sorting-algorithms`

Annexe

Vous trouverez ci-dessous le tableau à compléter pour chaque tri. Celui-ci donne sur la première ligne l'état initial du tableau, puis sur les lignes 2 et 3 les deux premiers échanges effectués par le tri à bulles.

tri à bulle

4	2	5	3	1
2	4	5	3	1
2	4	3	5	1
2	**3**	**4**	**5**	**1**
2	**3**	**4**	**1**	**5**
2	**3**	**1**	**4**	**5**
2	**1**	**3**	**4**	**5**
1	**2**	**3**	**4**	**5**

Annexe

Vous trouverez ci-dessous le tableau à compléter pour chaque tri. Celui-ci donne sur la première ligne l'état initial du tableau, puis sur les lignes 2 et 3 les deux premiers échanges effectués par le tri à bulles.

insertion

4	2	5	3	1
2	4	5	3	1
2	3	4	5	1
1	2	3	4	5

sélectif

4	2	5	3	1
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5