Récursivité

1 Avec des tris

Vous connaissez le principe des tris, aujourd'hui nous allons voir des algorithmes récursifs permettant de trier un tableau d'entiers. Nous allons utiliser le même environnement logiciel qu'en algorithmique, qui nous permet de visualiser le déroulement de notre tri dans une fenêtre graphique. Vous pouvez y trouver en particulier les méthodes `afficherTableau` et `echanger` de la classe `util` qui mettent à jour l'affichage graphique.

1.1 Tri fusion en place (exemple de base)

Ce premier tri vous est donné en tant qu'exemple pour vous permettre de comprendre le principe d'un algorithme récursif de tri. Dans les tris fusion, l'idée est de trier récursivement chaque moitié de tableau puis de les fusionner. Pour la fusion de ce premier tri fusion, nous utiliserons la méthode `fusionEnPlace` présente dans la classe `UtilFusion`. Cette méthode est déjà écrite et n'est pas destinée à être étudiée aujourd'hui. Si vous êtes curieux et souhaitez absolument comprendre son fonctionnement, nous vous conseillons de le faire chez vous et de consulter l'article indiqué en commentaire (attention il s'agit d'un algorithme assez difficile à comprendre). La fusion fournie peut être appelée de la manière suivante :
`UtilFusion.fusionEnPlace(util, T, debut1, taille1, debut2, taille2);`
où `T[debut1..debut1+taille1-1]` et `T[debut2..debut2+taille2-1]` sont les deux parties à fusionner avec la contrainte `debut1+taille1 = debut2`.

1.2 Tri rapide

Pour le tri rapide, on choisit un élément, appelé pivot, qui va nous servir à contruire une partition de l'ensemble des autres éléments : une partie contenant les éléments inférieurs au pivot, et une autre contenant les éléments supérieurs au pivot. Supposons que nous choisissons systématiquement le dernier élément comme pivot, nous pouvons alors construire notre partition de la manière suivante :

```
// Tableau T à trier indicé de p (premier élément) à d (dernier élément)
j = p
pivot = d
pour i allant de p à d-1
faire
    // Ici on va s'arranger pour que :
    // - les éléments d'indice [p..j-1] soient strictement inférieurs à T[pivot]
    // - les éléments d'indice [j..i] soient supérieurs ou égaux à T[pivot]
    si T[i] < T[pivot]
    alors
        échanger T[i] avec T[j]
        j = j+1;
échanger T[j] avec T[pivot]
```

Cet algorithme permet de partitionner correctement mais risque de partitionner de manière deséquilibrée si les éléments de T ne suivent pas une distribution uniforme. Pour éviter ce problème, il suffit de choisir comme pivot un élément de T choisi aléatoirement par un tirage uniforme et de l'échanger avec le dernier élément (pour retomber sur le cas traité par l'algorithme).

Pour compléter cet algorithme de tri, il suffit de trier récursivement chaque partie (de chaque coté du pivot) selon le même principe.

1.3 Tri fusion en place (fusion récursive)

Même principe général que pour tous les tris fusion (voir section 1.1). En revanche, ici, on vous demande de travailler sur une fusion recursive dont le principe est le suivant : soient A et B les deux "tranches" d'indices sur lesquels il faut opérer la fusion, on coupe ces 2 tranches en 2 soit A=A1A2 et B=B1B2 (si les tranches A et B ne sont constituées que d'un seul élément, on est dans le cas de base, il n'y a pas d'appel récursif). On fusionne alors successivement les couples de tranches :

— A1 et B1
— A2 et B2
— A2 et B1

Attention : pour que cet algorithme de fusion fonctionne il faut que la taille du tableau soit une puissance de 2. On vous donne une classe nommée `Etendeur` qui s'occupe d'étendre un tableau en y ajoutant autant d'éléments de valeur `Integer.MAX_VALUE` que nécessaire pour que sa taille devienne une puissance de 2. Elle s'utilise de la manière suivante :

```
Etendeur e = new Etendeur(TableauOriginal);
...
monAlgoDeTri(..., e.tableauEtendu(),...);
...
// finalise copie les valeurs triées dans le tableau original
e.finalise();
```

1.4 Tri bitonique

Le tri bitonique se fonde sur le fait qu'une séquence bitonique peut être fusionnée en une séquence triée par un réseau de tri appelé "papillon". Une séquence bitonique est une séquence de valeurs d'abord croissante, puis décroissante ou l'inverse (d'abord décroissante puis croissante).

Pour la fusion d'une séquence bitonique de taille n, le réseau "papillon" effectue des comparaisons entre toutes les paires d'éléments d'indice i et i+(n/2) pour i variant entre le début de la séquence et sa moitié. Pour un tri par ordre croissant, et une paire d'élément à comparer, le maximum des deux ira à l'indice le plus haut et le minimum à l'indice le plus bas. Pour un tri par ordre décroissant, c'est l'inverse. Pour terminer la fusion bitonique, il faut l'appliquer récursivement aux deux moitiés de la séquence ainsi obtenue. **Attention :** pour que cet algorithme de fusion fonctionne il faut que la taille du tableau soit une puissance de 2.

Enfin, pour que notre séquence soit bitonique, il suffit de trier la première moitié par ordre croissant, et la seconde par ordre décroissant (avec un tri bitonique par exemple).

1.5 Tri 2-3

Une séquence X-triée est une séquence telle que pour tout z entre 0 et X-1, la séquence des éléments d'indice de la forme z+kX est triée. Si une séquence est à la fois 2-triée et 3-triée alors les éléments sont tous soit à leur place soit juste à coté de leur place. Le tri 2-3 consiste à 2-trier et 3-trier la séquence puis à la parcourir séquentiellement pour remettre à leur place les éléments mal placés.

2 Bonus : les tours de Hanoï

Pour ceux qui iraient un peu trop vite ou en auraient vraiment marre des tris, un problème classique. Les tours de Hanoï sont constituées de trois piquets A, B et C. Sur le piquet A sont empilés N disques de taille décroissante. Le but est de placer tous les disques sur le piquet C. Pour cela, le seul mouvement autorisé est le déplacement d'un disque à la fois vers un autre piquet qui ne contient pas déjà un disque plus petit.

Attention : pour cette question aucun affichage ne vous est fourni. Vous devez le réaliser vous même à l'aide de la machine à tracer. Pour cela, en plus des méthodes que vous avez déjà vu lors des TPs de dessin, vous pouvez utiliser les méthodes supplémentaires :

— `effacerTout() : comme son nom l'indique`
— `attendre(d)` : où `d` est une durée entière exprimée en millisecondes