

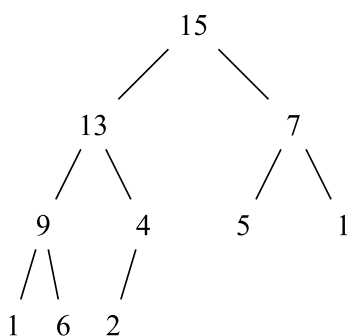
TP n° 9

Tas binaire et tri dans un collection


L'objectif de cette séance de travaux pratiques est de comprendre l'intérêt des structures de données permettant un tri des éléments.


Exercice 1 – Tas binaire


Dans cet exercice, nous proposons une implémentation d'un tas binaire. Pour rappel un tas est représenté comme un arbre contigu, c'est à dire un tableau de valeurs. En vertu de la propriété de tas, l'élément à l'indice 0 du tableau est la valeur la plus grande du tas.




0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	13	7	9	4	5	1	1	6	2					

 Dans une nouvelle classe **Tas**, on déclarera un tas binaire comme un tableau **valeurs** d'éléments comparables, ainsi qu'un entier **indiceMax** permettant de stocker l'indice du "dernier" élément de l'arbre (i.e. la feuille à droite de plus grande profondeur). Écrire le constructeur du **Tas**.

 Écrire l'algorithme de la fonction **insérer** qui permet l'insertion d'une valeur dans le tas. Une nouvelle valeur est toujours ajoutée comme "dernier" élément de l'arbre, puis remontée si nécessaire par échanges avec ses pères successifs.

 Écrire l'algorithme des fonctions **supprimer** et **tamiser**. Lors de la suppression d'un élément i de l'arbre, il est remplacé (i.e. écrasé) par le "dernier" élément de l'arbre j , puis un processus de tamisage est appliqué si nécessaire pour faire descendre j jusqu'à sa place naturelle.

 Mettre en place une procédure de tests unitaires pour valider le fonctionnement du tas. On pourra implémenter par exemple le test suivant :

Données : *ch*, *chRes* : Chaîne ; *c* : Caractère ; *t* : Tas<Caractère>

début


```

  ch ← "hello world"
  pour chaque c ∈ ch faire
    t.insérer(c) ;
  tant que t.indiceMax > 0 faire
    c ← t.valeurs[t.indiceMax]
    chRes ← chRes + c
    t.supprimer(0)
  retourner chRes = "wroolllhed"


```

Exercice 2 – TAD File à Priorité

Dans cet exercice, nous proposons une implémentation d'une file à priorité à partir d'un tas binaire.


 télécharger l'interface **File** sur Moodle. Créer une nouvelle classe **FilePriorité** qui implémente l'interface **File** en dérivant la classe **Tas**.


 Écrire les algorithmes des opérateurs du TAD File à partir des méthodes du tas afin d'obtenir l'implémentation d'une file à priorité.


 Tester cette implémentation.

Exercice 3 – Tri Java

Dans cet exercice, nous utilisons la classe `java.util.Collections` pour trier des éléments stockés dans une liste Java. Cette classe contient un ensemble de méthodes statiques permettant la manipulation de collections d'éléments (recherche du min, du max, tri, etc).

 Dans un nouveau paquetage de votre projet, créer une classe **Tâche** qui implémente l'interface **Comparable** et qui permet de représenter une tâche par une importance (entier ∈ [1,100]) et un libellé (chaîne de caractères). La comparaison de 2 tâches doit s'effectuer en priorité sur l'importance (tri décroissant), puis en cas d'importances égales, sur le libellé (tri alphabétique).

 Dans le programme principal, créer une liste de tâches, en y insérant dans cet ordre les tâches suivantes : (100, "Study some maths"), (80, "Order takeaway"), (150, "Learn Java"), (50, "Chill/watch movie") et (80, "Get some sleep").

 Utiliser la classe **Collections** pour trier (*sort*) votre liste de tâches et obtenir l'ordre exact donné ci-dessous :


```


(150) Learn Java
(100) Study some maths
(80) Get some sleep
(80) Order takeaway
(50) Chill/watch movie

```

Exercice 4 – Planification des vols

Dans cet exercice, un programme informatique assure rôle d'un opérateur d'une tour de contrôle en charge de réguler les atterrissages de vols se présentant à intervalles réguliers (toutes les secondes). Chaque vol est caractérisé par un identifiant et une quantité de carburant restante. À intervalles réguliers, le programme donne une autorisation d'atterrir à un vol en attente.

 Télécharger le paquetage de classes `tour_de_contrôle`. Dans cette version du programme, les vols sont mis en attente dans une file. Exécuter le programme et constatez la survenue rapide d'un crash.

 En utilisant la classe `PriorityQueue`, apporter les modifications nécessaires au programme de gestion de la tour de contrôle afin d'endiguer cette série de catastrophes aériennes.