Tas : file de priorité minimum

Définition : Un *arbre partiellement ordonné* est un arbre binaire étiqueté tel que la valeur contenue dans tout nœud est inférieure ou égale aux valeurs contenues dans les sous-arbres de ce nœud.

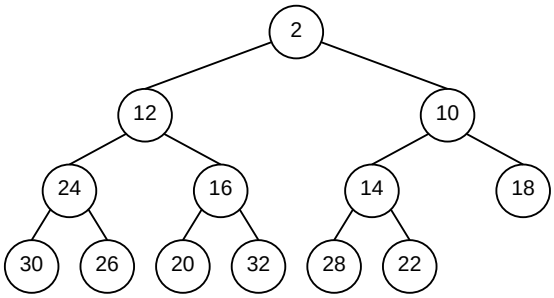


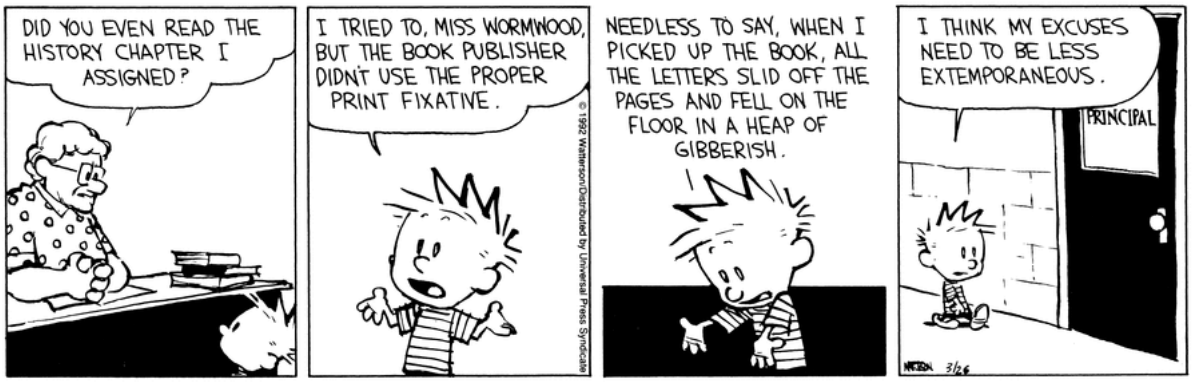
FIGURE 1 – Arbre parfait partiellement ordonné

Le *tas* permet de représenter un *arbre binaire* ***parfait*** *partiellement ordonné*. L'arbre étant parfait, on utilise un vecteur utilisant la numérotation hiérarchique pour le stocker. On y adjoint un entier représentant la taille de l'arbre représenté.

Ici, le vecteur contient des couples contenant chacun l'élément et sa valeur pour le tri.

L'arbre de la figure 1 contient les valeurs associées au éléments suivants (l'élément A a pour valeur 2...) :

A	B	C	D	E	F	G	H	I	J	K	L	M
2	10	12	14	16	18	20	22	24	26	28	30	32



Exercice 1 (Représentation par un tas)

1. Donner la représentation par tas (le vecteur contenant des couples (valeur, élément)) de l'arbre de la figure 1.

2. Soit V le vecteur représentant un arbre binaire parfait, n la taille de cet arbre :
   (a) Où est la racine ?
   (b) Comment retrouver les fils d'un nœud ?
   (c) Comment retrouver le père d'un nœud ?
   (d) Comment savoir si un nœud est une feuille ?
   (e) Comment savoir si un nœud est un point simple ?

Python

```
1    def newHeap():
2        return [None]
3
4    def isEmpty(H):
5        return len(H) == 1
```

Exercice 2 (Utilisation)

1. **Ajout :**
   (a) Comment ajouter un élément à un tas afin qu'il conserve toutes ses propriétés ?
   (b) Ajouter les éléments N de valeur 5 puis O de valeur 15 enfin P de valeur 1 à l'arbre de la figure 1 (donner l'arbre et le vecteur).
   (c) Écrire la fonction `heapPush(`H`,` x`)` qui ajoute x ($x = (val, elt)$) au tas H.
   (d) Quelle est la complexité de cette fonction (en fonction de n, la taille du tas) ?
2. **Suppression :**
   (a) Comment supprimer d'un tas l'élément de valeur minimum ?
   (b) Supprimer le plus petit élément de l'arbre obtenu à la question précédente (donner l'arbre et le vecteur).
   (c) Écrire la fonction `heapPop(`H`)` qui retourne x de valeur minimum ($x = (val, elt)$) du tas H après l'avoir supprimé.
   (d) Quelle est la complexité de cette fonction ?

Exercice 3 (Modification ?)

1. **Minimisation :**
   (a) Dans le tas obtenu à l'exercice 2, la valeur de M change, elle passe à 4. Donner le nouveau tas obtenu après modification de cette valeur.
   (b) Écrire la procédure `heapUpdate(`H`,` x`,` pos`)` qui réorganise le tas H après minimisation de la valeur de x ($x = (val, elt)$) en position pos (dans le vecteur).
2. **Optimisation :**
   (a) Quelle serait la complexité d'une fonction permettant de trouver la position d'un élément quelconque ?
   (b) Que faudrait-il ajouter à la représentation pour pouvoir modifier le tas en cas de minimisation d'un élément quelconque du tas en gardant une complexité optimale ?

Exercice 4 (Tri par tas)

Utiliser les fonctions précédentes pour écrire une fonction qui trie une liste en ordre croissant.

Exercice 5 (Heapify)

Soit un arbre binaire parfait déjà représenté dans une liste (avec la numérotation hiérarchique). Écrire la fonction `heapify` qui transforme cet arbre en tas.

```
1  >>> T = [None,
       (20,'A'),(5,'B'),(10,'C'),(12,'D'),(15,'E'),(8,'F'),(2,'G'),(6,'H'),(2,'I'),(9,'J')]
2  >>> heapify(T)
3  [None,
       (2,'G'),(2,'I'),(8,'F'),(5,'B'),(9,'J'),(20,'A'),(10,'C'),(6,'H'),(12,'D'),(15,'E')]
```

À noter que la solution n'est pas unique.