TP L2.2 : Algorithmique des arbres

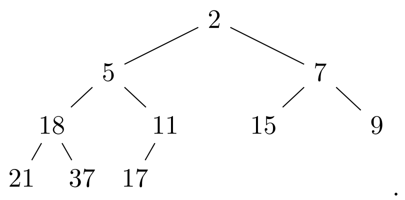
Tas min

Le but de ce TP est de compter les différents tas contenant les entiers de 1 à n.

Un tas de hauteur h est un arbre tel que:

- tous les niveaux sauf le dernier sont forcement remplis.
- toutes les feuilles sont au niveau h ou $h-1$, et toutes les feuilles du niveau h sont tassées sur la gauche;
- la valeur d'un nœud est plus grande que la valeur de son nœud parent.

Un exemple de tas est:



On peut représenter un tas par un tableau contenant ses valeurs, lues par niveaux successifs. L'exemple ci-dessus donne le tableau

2	5	7	18	11	15	9	21	37	17

Le parent du noeud d'indice $i > 0$ est le noeud d'indice $\lfloor \frac{i-1}{2} \rfloor$.

Exercice 0

Sortir un papier et un crayon.

Exercice 1

Écrire une fonction `int est_tas(int tab[], int taille)` qui renvoie 1 si le tableau passé en argument (de taille donnée dans le deuxième argument) est un tas, et 0 sinon.
Tester cette fonction sur différents tableaux, et afficher (ou dessiner avec dot) le tableau (ou l'arbre) quand c'est un tas.

Exercice 2

Le but de cet exercice est de compter les différents tas contenant exactement les valeurs de 1 à n. Par exemple pour $n = 4$, il y a 3 tas :

$$1, 2, 3, 4$$
$$1, 2, 4, 3$$
$$1, 3, 2, 4$$

On remarque qu'un tas est représenté par un tableau qui contient une permutation de $\{1, \ldots, n\}$; mais toutes les permutations ne forment pas des tas! L'ensemble des tableaux qui représentent un tas est inclus dans l'ensemble des tableaux qui représentent une permutation.

1. Écrire la fonction `enum_permutation`
   Une première étape est donc de construire les permutations de $\{1, \ldots, n\}$.
   Cette construction se fait par la fonction récursive `int enum_permutation(int t[], int premier, int n)` qui reçoit:

- `t`, un tableau de taille `n` contenant des entiers de 1 à n ainsi que des 0. 0 symbolise une case 'vide'.
- un entier *premier* à placer successivement dans chaque case 'vide'.
- un entier n, la valeur maximum.

Initialement toutes les cases sont 'vides', le tableau ne contient que des 0 et on cherche à placer 1. Le premier appel, pour construire les permutations de $\{1, \ldots, 5\}$ dans un tableau `permu` est donc `enum_permutation(permu,1,5)`. Cet appel cherchera à placer 2 par appel à `enum_permutation(permu,2,5)`et ainsi de suite jusqu'à avoir placé n
Algorithme:
Si $premier \leq n$, la fonction

- parcours le tableau pour placer l'entier *premier* dans une case vide,
- effectue un appel récursif pour placer les valeurs à partir de $premier + 1$ dans les cases vides restantes,
- au retour de l'appel récursif, libère la place (avec un 0) et poursuit la recherche en plaçant premier dans une autre case 'vide'.

Si $premier > n$ la permutation est complète. La fonction renvoie le nombre de permutations.

On pourra afficher les permutations trouvées (pour n petit).

2. Nombre de tas
En utilisant le code de la fonction précedente et la fonction `est_tas`, écrire une fonction `enum_tas` pour compter le nombre de tas.
On pourra afficher les tas trouvés (pour n petit).

3. Optimisations
Il y a beaucoup plus de permutations que de tas. Or on peut savoir avant construction complète qu'une permutation ne sera pas un tas.
Ainsi, en cours de construction, on peut remarquer que

2	1	0	0	0
1	0	0	2	0

   ne correspondent pas à un tas et stopper la construction.
Optimiser la fonction `est_tas` en une fonction `est_tas_partiel` et comparer les temps d'éxecution avec les deux versions pour determiner le nombre de tas possibles avec $n = 15$.

4. Quelle valeur maximun de n peut-on traiter en une minute?