Tas

Feuille n°6

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 intégralement remplis ;
— toutes les feuilles sont donc au niveau h ou $h-1$;
— 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

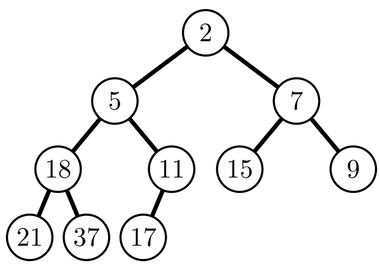

FIGURE 1 – Tas

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 nœud d'indice $i \geqslant 1$ est le nœud d'indice $\lfloor (i-1)/2 \rfloor$.

0. **Sortir un papier et un crayon** — Si vous ne réussissez pas cet exercice, votre enseignant ne viendra pas vous aider en cas de problème lors d'une des questions qui suivent.
1. **Vérification** — Écrire une fonction `int est_tas(int tab[], int taille)` qui renvoie `1` si le tableau `tab` (de longueur `taille`) est un tas, et renvoie `0` sinon. Tester cette fonction sur différents tableaux, et afficher (ou dessiner avec `dot`) le tableau (ou l'arbre) quand c'est un tas.

2. **Énumération** — 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 que toutes les permutations ne forment pas des tas ! L'ensemble des tableaux qui représentent un tas est donc inclus dans l'ensemble des tableaux qui représentent une permutation.

a. Écrire une fonction récursive `int enum_permutation(int tab[], int premier, int n)` dont le fonctionnement est le suivant. On suppose que `tab` est un tableau de longueur `n`, et contenant exactement une occurrence de chacun des entiers $1, 2, \ldots,$ `premier` $- 1$, et dont les autres cases sont occupées par l'entier 0. La fonction doit renvoyer l'entier k égal au nombre de manières de compléter ce tableau, en rajoutant les entiers `premier`$, \ldots, n$, et correspondant à une permutation de $\{1, \ldots, n\}$. Les cases contenant l'entier 0 sont assimilées à des cases vides.

Pour ce faire, la fonction devra procéder comme suit. Si `premier` $> n$, le tableau `tab` représente une permutation complète ; dans ce cas, la fonction doit renvoyer `1`. Si `premier` $\leqslant n$, la fonction doit :

— parcourir le tableau `tab`, pour placer l'entier `premier` dans la première case vide (c'est-à-dire contenant l'entier `0`) rencontrée ;
— effectuer un appel récursif à `enum_permutation(tab, premier+1, n)` pour placer les entiers `premier` $+ 1, \ldots, n$ dans le tableau et compter combien de permutations on a obtenu ;
— au retour de l'appel récursif, libérer la case (en y mettant un `0`) et aller placer l'entier `premier` dans la prochaine case vide rencontrée ;
— renvoyer le nombre total de permutation obtenues en essayant les différentes positions de l'entier `premier`.

Quand n est petit, on pourra afficher toutes les permutations obtenues.

b. Écrire, en adaptant la fonction précédente, une fonction récursive `int enum_tas_naif(int tab[], int premier, int n)` qui renvoie le nombre total de tas obtenus en complétant le tableau (partiellement rempli) `tab`.

Quand n est petit, on pourra afficher tous les tas obtenus.

c. Il y a beaucoup plus de permutations que de tas. Or, dans certains cas, on peut identifier qu'un tableau partiellement rempli ne pourra jamais aboutir à la construction d'un tas. Ainsi, en cours de construction, on peut remarquer que les tableaux

2	1	0	0	0

1	0	0	2	0

0	0	0	1	0

ne correspondent pas à un tas, et stopper la construction.

Écrire une fonction `int est_tas_partiel(int tab[], int n)` qui vérifie qu'un tableau partiellement rempli peut être complété en (au moins) un tas. Écrire ensuite une fonction `int enum_tas_verif(int tab[], int premier, int n)` qui permette d'optimiser le fonctionnement de `enum_tas_naif` en s'aidant de la fonction `est_tas_partiel`.

d. Pourquoi, pour répondre à la question précédente, aurait-il en fait été pratique de représenter les cases vides du tableau en y mettant l'entier `n+1` plutôt que l'entier `0` ?

e. Dans la fonction précédente, on passe beaucoup de temps à effectuer des vérifications. Expliquer pourquoi, dans la fonction précédente, et au lieu de placer l'entier `premier` dans chaque case vide puis de vérifier que les tableaux obtenus pouvaient être complétés en un tas, on aurait pu se contenter de ne placer l'entier `premier` que dans les cases d'indice i telles que `tab`$[i] = 0$ et telles que soit $i = 0$, soit `tab`$[\lfloor (i-1)/2 \rfloor] \neq 0$.

Écrire ensuite une fonction `int enum_tas(int tab[], int premier, int n)` pour tenir compte de cette remarque.

f. Quelle valeur maximum de n peut-on traiter en une minute ?