Feuille de Travaux Pratiques n° 2
Arbres binaires de recherche et Tas binaires

Le TP est à réaliser **en binôme** (un seul monôme est autorisé par groupe de TP, si celui-ci possède un nombre impair d'étudiants). Trois séances de TP encadrées sont consacrées à cette feuille de TP. Le TP sera évalué par un compte-rendu que vous devrez déposer sur madoc (voir les dates limites sur madoc en fonction de votre groupe de TP) au format PDF. Le nom du fichier devra respecter le format suivant : *TP2_nom1_nom2.pdf*

Un malus sera appliqué à chaque fois qu'une des consignes données ci-dessus n'aura pas été respectée. Notamment, en cas de retard, le malus sera le suivant : -1 point par heure de retard (toute heure entamée étant comptée comme complète).

Les objectifs de ce TP sont (1) de manipuler deux structures de données vues en cours et TD, à savoir les arbres binaires de recherche (ABR) et les tas binaires (TB) ; (2) de vérifier, par la pratique, les complexités données pour certaines routines de base ; (3) de comparer ces complexités entre les deux structures étudiées.

Le langage de programmation utilisé pour implémenter les algorithmes et pouvoir ainsi répondre aux différentes questions est laissé au choix.

Pour chaque exercice, les instructions sont séparées en deux parties : la partie programmation qui détaille les étapes nécessaires pour implémenter et tester les algorithmes (cette partie n'est pas à rendre) ; la partie compte-rendu qui précise les éléments à indiquer dans le rapport que vous devez déposer sur madoc.

Exercice 2.1 (Création d'ABR)

Dans cet exercice, nous allons comparer les temps d'exécution pour la création de deux types d'ABR : les *ABR complets* et les *ABR filiformes*.

Partie programmation :

1. Écrire la fonction d'insertion dans un ABR, telle qu'elle a été vue en cours.
2. Écrire une fonction `Creer-ABR-complet` qui prend en entrée un entier p, et qui crée un *ABR complet* d'entiers à $n = 2^{p+1} - 1$ nœuds dont les valeurs vont de 1 à n. Pour cela, il faut procéder en deux temps :
   (a) d'abord remplir un tableau T qui contiendra, dans un ordre bien choisi, les éléments de 1 à n. Cet ordre (donné plus bas) assure que l'insertion successive des éléments de T dans l'ABR donnera bien un ABR *complet*.
   (b) ensuite, procéder à la création de l'ABR en elle-même, qui consiste à insérer tous les élements de T, en lisant T de la gauche vers la droite.

   Exemple : si $p = 3$ (et donc $n = 15$), voici le tableau T attendu : $T = [8, 4, 12, 2, 6, 10, 14, 1, 3, 5, 7, 9, 11, 13, 15]$. Pour le construire, on peut utiliser l'algorithme suivant (on suppose ici que T est indicé à partir de 1) :

   $T[1] \leftarrow 2^p$
   $k = 2$
   Pour i de $p-1$ à 0 **faire**
     $T[k] \leftarrow 2^i$
     $k \leftarrow k + 1$
     Pour j de 1 à $2^{p-i} - 1$ faire
       $T[k] \leftarrow T[k-1] + 2^{i+1}$
       $k \leftarrow k + 1$
     FinPour
   FinPour

Et si l'on devait dessiner l'ABR complet obtenu lors de l'étape (b) à l'aide du tableau T quand $p = 3$, on obtiendrait l'ABR de la Figure 1.

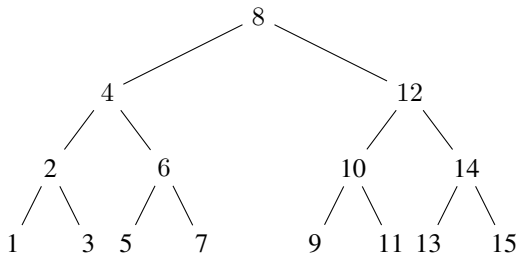

FIGURE 1 – ABR complet à $n = 15$ nœuds ($p = 3$).

3. Écrire une fonction `Creer-ABR-filiforme` qui prend en entrée un entier p , et qui crée un ***ABR filiforme*** d'entiers à $n = 2^{p+1} - 1$ nœuds, dont les valeurs vont de 1 à n. Ici aussi, il faudra d'abord (a) créer un tableau T contenant les entiers de 1 à n dans un ordre bien choisi, puis (b) procéder à l'insertion successive des élements de T dans l'ABR.
4. Réaliser un programme qui, pour un p donné, écrit dans un fichier les temps d'exécution de `Creer-ABR-complet` et `Creer-ABR-filiforme` (dans les deux cas : partie (b) seulement, donc sans la génération de T).
5. Tester ce programme pour toutes les valeurs de $p \geq 1$. On s'arrêtera quand, pour un p donné, le temps d'exécution de l'algorithme `Creer-ABR-complet` (toujours sans compter la génération de T) dépasse 3 minutes. On laissera cependant les tests se terminer pour cette valeur de p, qui sera donc la dernière testée.

Partie compte-rendu :

1. Donner le résultat du parcours suffixe de l'ABR complet à $n = 31$ nœuds. Donner le résultat du parcours suffixe de l'ABR complet à $n = 63$ nœuds.
2. Expliquer (en français) la méthode (l'algorithme) mise en place pour générer le tableau T qui permet de créer un ABR filiforme à $n = 2^{p+1} - 1$ nœuds.
3. Reporter sur un graphique (par exemple en utilisant un tableur) les temps d'exécution de `Creer-ABR-complet` et `Creer-ABR-filiforme` (en ordonnée) en fonction de n (en abscisse). Vous préciserez le nombre de valeurs de p (donc de valeurs n) que vous avez testées tout en respectant les contraintes ci-dessus.
4. Discuter les résultats obtenus en pratique par rapport aux résultats théoriques présentés en cours.

Exercice 2.2 (Suppression, insertion et recherche dans un ABR)

Dans cet exercice, nous allons nous intéresser aux temps d'exécution des routines de base, et plus particulièrement à celle de la recherche, sur différents types d'ABR.

Partie programmation :

1. Écrire les fonctions de recherche et de suppression dans un ABR, telles qu'elles ont été vues en cours (la fonction d'insertion ayant déjà été implementée dans l'exercice précédent).
2. Écrire une fonction `Manipuler-ABR-complet` qui prend en entrée un ABR *complet* A à $n = 2^{p+1} - 1$ nœuds, et qui renvoie un ABR A', obtenu de la manière suivante à partir de A : pour i de 2^p à 1, (a) supprimer i, puis (b) insérer i (directement après sa suppression).
3. Réaliser un programme qui, pour un p donné, écrit dans un fichier les temps d'exécution de la recherche de l'élément de valeur 1 dans A et A'.
4. Tester ce programme pour toutes les valeurs de $p \geq 1$ déjà testées à l'Exercice 2.1.

Partie compte-rendu :

1. Dessiner l'ABR A' obtenu pour $p = 3$ et pour $p = 4$.
2. De façon générale, quelle est la profondeur de l'ABR A' obtenu ? Justifier votre résultat.
3. Reporter sur un graphique (par exemple en utilisant un tableur) les temps d'exécution de la recherche de l'élément 1 dans A et A' (en ordonnée) en fonction de n (en abscisse).
4. Discuter les résultats obtenus en pratique par rapport aux résultats théoriques présentés en cours.

Exercice 2.3 (TB et ABR)

Dans cet exercice, nous allons nous intéresser à la comparaison des temps d'exécution (de création et de recherche) entre les TB et les ABR.

Partie programmation :

1. Écrire la procédure de création d'un max-tas binaire `Creer-TB` telle qu'elle a été vue en cours.
2. Écrire une fonction qui prend en entrée un entier $p \geq 1$ et qui renvoie un tableau T de $n = 2^{p+1} - 1$ entiers, tel que T contient les valeurs de 1 à n positionnées aléatoirement.
3. Réaliser un programme qui, pour un entier p donné, écrit dans un fichier les temps au mieux et au pire de `Creer-TB`, à partir d'un tableau aléatoire T contenant $n = 2^{p+1} - 1$ valeurs, tel qu'il a été construit à la question précédente. Pour chaque valeur de p testée, on générera un *grand échantillon* de tableaux d'entrée ; plus précisément, on arrêtera les tests quand le temps cumulé d'exécution de l'algorithme `Creer-TB` (sans compter la génération des tableaux T) dépasse 3 minutes.
4. Réaliser un programme qui, pour un entier $p \geq 1$ donné, écrit dans un fichier les temps au pire de la recherche de l'élément 1 dans les TB générés à la question précédente.
5. Tester les deux programmes pour toutes les valeurs de $p \geq 1$ déjà testées aux Exercices 2.1 et 2.2.

Partie compte-rendu :

1. Reporter sur un graphique (par exemple en utilisant un tableur) les temps d'exécution au mieux et au pire de `Creer-TB` (en ordonnée) en fonction de n (en abscisse). Discuter les résultats obtenus en pratique par rapport aux résultats théoriques présentés en cours.
2. Comparer les résultats obtenus pour la création des TB avec ceux obtenus pour la création des ABR à l'Exercice 2.1.
3. Reporter sur un graphique (par exemple en utilisant un tableur) les temps d'exécution au pire de la recherche de l'élément 1 dans les TB (en ordonnée) en fonction de n (en abscisse). Discuter les résultats obtenus en pratique par rapport aux résultats théoriques présentés en cours.
4. Comparer les résultats obtenus pour la recherche d'un élément dans un TB avec ceux obtenus pour cette même recherche dans les ABR (Exercice 2.2).