Feuille de Travaux Pratiques n^o 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).

L'objectif de ce TP est de manipuler les structures vues en cours à savoir les arbres binaires de recherche (ABR) et tas binaires (TB), de vérifier, par la pratique, les complexités données pour les routines de base et de comparer ces complexités entre les 2 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 vue en cours.
2. Écrire une fonction `creeABR-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.
   Remarque : la figure 1 représente l'ABR complet avec $p = 3$ (c'est-à-dire à $n = 2^4 - 1 = 15$ nœuds). Pour créer un tel arbre, on pourra procéder par "niveau" (i.e. nœuds de même profondeur) :
   — le premier niveau (ou profondeur $= 0$) contient la racine de l'arbre (donc $1 = 2^0$ seul nœud) de valeur 2^p (sur l'exemple, la racine vaut 8) ;
   — le second niveau (profondeur$= 1$) contient $2 = 2^1$ nœuds : en partant de 2^{p-1} (sur l'exemple, la valeur 4) et en ajoutant 2^p pour chaque nouveau nœud inséré (sur l'exemple, $4 + 2^3 = 12$) ;
   — on continue ainsi jusqu'au dernier niveau (profondeur $= p$) pour lequel on insérera 2^p nœuds : le premier nœud inséré pour ce niveau est $2^0 = 1$ et la valeur de chaque nouveau nœud est obtenue en ajoutant $2 = 2^1$ à la valeur du nœud précédemment inséré (sur l'exemple, les valeurs 1, 3, 5, 7, 9, 11 et 13, respectivement).
3. Écrire une fonction `creeABR-filiforme` d'entiers 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.
4. Réaliser un programme qui, pour un p donné, écrit dans un fichier les temps d'exécution de `creeABR-complet` et `creeABR-filiforme`.
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 `creeABR-complet` dépasse 3 minutes. On laissera cependant les tests se terminer pour cette valeur de p, qui sera donc la dernière testée.

Partie compte-rendu :

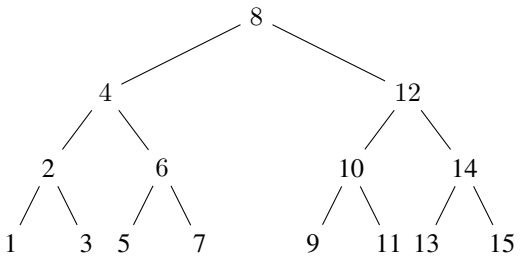

FIGURE 1 – ABR complet à $n = 15$ nœuds ($p = 3$).

1. Donner le résultat du parcours suffixe des ABR complets à $n = 31$ et à $n = 63$ nœuds.
2. Expliquer (en français) la méthode (l'algorithme) mise en place pour 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 `creeABR-complet` et `creeABR-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 suppression et de recherche dans un ABR vues en cours (la fonction d'insertion a déjà été implementée dans l'exercice précédent).
2. Écrire une fonction `manipulerABR-complet` qui prend en entrée un ABR *complet* A à $n = 2^{p+1} - 1$ nœuds et qui renvoie un ABR A' tel que A' est obtenu de la manière suivante à partir de A : pour i de 2^p à 1, on supprime i et on l'ajoute à nouveau 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 ?
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 tas binaires et les arbres binaires de recherche.

Partie programmation :

1. Écrire la procédure de création d'un tas binaire `creeTB` décrite dans le cours.
2. Écrire une fonction qui prend en argument un entier p 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 p donné, écrit dans un fichier les temps au mieux et au pire de `creeTB`. Pour chaque valeur de p testée, on générera un *grand échantillon* de tableaux d'entrée. On arrête les tests quand le temps cumulé d'exécution de l'algorithme `creeTB` sans compter la génération des tableaux dépasse 3 minutes.

4. Réaliser un programme qui, pour un p donné, écrit dans un fichier les temps au pire de la recherche de l'élément 1 dans les tas binaires 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 `creeTB` (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 tas binaires avec ceux obtenus pour la création des arbres binaires de recherche 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 tas binaires (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 tas binaire avec ceux obtenus pour cette même recherche dans les arbres binaires de recherche (Exercice 2.2).