TP 1 : Révisions

1 Arbres binaires de recherche

On définit un arbre comme étant soit une feuille (vide), soit un nœud constitué d'une information (par exemple une chaîne de caractères) et de deux sous-arbres, c'est-à-dire le type CAML suivant :

```
type 'a arbre = Feuille
              | Noeud of 'a arbre * 'a * 'a arbre
```

où `'a` est le type de l'information. Dans ce TP, on pourra considérer des arbres d'entiers pour simplifier les choses (des `int arbre`).

1.1 Fonctions d'intérêt général

QUESTION 1 – Écrire des fonctions calculant la taille, le nombre de noeuds internes, le nombre de feuilles, la hauteur d'un arbre binaire.

QUESTION 2 – Un arbre binaire est dit *localement complet* si aucun des sous-arbres d'un noeud interne n'est vide. Il est dit *complet* si les feuilles ont toutes la même hauteur.

Écrire des fonctions testant la locale complétude et la complétude d'un arbre binaire, et évaluer leurs complexités.

1.2 Arbres binaires de recherche

Les arbres binaires de recherche constituent un moyen courant de stockage et de recherche (pour représenter des dictionnaires par exemple). Les algorithmes impliqués sont très classiques et se retrouvent dans de nombreuses situations (parcours d'un arbre, insertion dans un arbre...).

On suppose donc qu'il existe une relation d'ordre totale permettant de classer les données. La propriété d'un arbre binaire de recherche est la suivante :

Pour chaque nœud (g, i, d), les informations contenues dans le sous-arbre gauche g sont inférieures (ou égales) à i, les informations contenues dans le sous-arbre droit d supérieures (ou égales) à i.

On se propose d'écrire les principales fonctions de manipulation des arbres binaires de recherche, à savoir *la recherche*, *l'insertion* et *la suppression* dans un arbre binaire.

QUESTION 3 – Écrire une fonction `recherche : 'a arbre -> 'a -> bool` testant la présence d'un élément dans un arbre binaire de recherche.

Quelle est sa complexité en fonction de la taille N de l'arbre (nombre de nœuds de l'arbre) ? Distinguer les cas en moyenne et au pire. A quel type d'arbre correspond la complexité la plus médiocre ?

QUESTION 4 – Écrire une fonction `insertion : 'a arbre -> 'a -> 'a arbre` réalisant l'insertion d'un nouvel élément dans un arbre binaire de recherche. Quelle est la complexité de cette opération ?

En déduire une fonction `construit : 'a list -> 'a arbre` construisant un arbre binaire de recherche à partir d'une liste d'éléments.

QUESTION 5 – Écrire une fonction `parcours : 'a arbre -> 'a list` prenant un arbre binaire de recherche en argument et renvoyant la liste *triée* de ses éléments. Un tel parcours s'appelle *parcours préfixe* d'un arbre.

QUESTION 6 – Écrire une fonction `tri_ABR : 'a list -> 'a list` prenant une liste d'entiers en argument et renvoyant la liste *triée* de ses éléments. Évaluer sa complexité.

QUESTION 7 – Écrire une fonction `suppression : 'a arbre -> 'a -> 'a arbre` supprimant un élément dans un arbre binaire de recherche. Quelle est la complexité de cette opération ?

QUESTION 8 – (bonus) Lorsque l'arbre constitue une structure permanente (comme dans le cas d'un dictionnaire) et non pas une structure temporaire (construite pour classer des données par exemple), il est particulièrement intéressant de conserver l'équilibrage de l'arbre (le déséquilibre d'un noeud d'un arbre binaire désigne la différence des hauteurs entre les deux sous-arbres de ce noeud).

Écrire une fonction `equilibre : 'a arbre -> 'a arbre` équilibrant un arbre binaire de recherche. Il est sans doute nécessaire de découper cette tâche en plusieurs sous-tâches plus simples : mesure du déséquilibre, équilibrage d'un noeud...)

Quelle est la complexité de cette opération ?

2 Vecteurs et programmation impérative

QUESTION 9 – Écrire une fonction calculant le n-ième terme de la suite de Fibonacci en utilisant un vecteur où seront stockées les valeurs successives. Quelle est sa complexité temporelle ? spatiale ? La réécrire en utilisant des références.

QUESTION 10 – Écrire une fonction `appartient x a` d'appartenance d'un élément `x` à un vecteur `a` . Quelle est sa complexité ? Peut-on faire mieux si l'on suppose le tableau trié ? Programmer une version efficace dans ce cas.

QUESTION 11 – Écrire une fonction `concatene a1 a2` qui concatène deux tableaux unidimensionnels (il faudra créer un nouveau tableau). L'adapter ensuite aux matrices ayant le même nombre de lignes.

QUESTION 12 – Écrire une fonction `map_array f a` qui, sur le modèle de la fonction `map` pour les listes, renvoie un nouveau vecteur contenant les images des éléments de a par la fonction `f`.

3 Tri fusion

QUESTION 13 – Écrire une fonction qui sépare une liste en deux.

QUESTION 14 – Écrire une fonction qui réalise la fusion de deux listes triées en une seule liste triée.

QUESTION 15 – Écrire la fonction de tri fusion.