Sujet 1 (13/11/2024)

contact : jean-baptiste.doderlein@ens-rennes.fr

Question de cours

1. Rappeler la définition d'un arbre binaire de recherche. Définir un type OCaml `'a abr`.

2. Écrire une fonction `recherche : 'a abr -> 'a -> bool`. En quoi cette fonction diffère d'une fonction de recherche dans un arbre binaire ? Est-elle plus rapide dans tous les cas ?

Exercice 1 : Vérification de propriétés d'un arbre binaire

1. Montrer qu'un arbre parfait (tous les niveaux sont complets) de hauteur h possède $2^h - 1$ nœuds.

2. Écrire une fonction `est_parfait : 'a arbre -> bool` qui vérifie si l'arbre est parfait.

3. Écrire une fonction `est_symetrique : 'a abr -> bool` qui vérifie si l'arbre est symétrique par rapport à son axe vertical (l'arbre est son propre miroir).

4. Montrez que le parcours en profondeur du miroir d'un arbre est identique au parcours infixe de l'arbre original, mais dans l'ordre inverse.

Exercice 2 : AVL

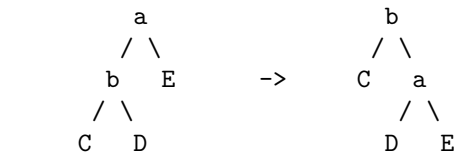
1. Écrire une fonction `inserer_abr : 'a abr -> 'a -> 'a abr` qui insère un élément dans un ABR.

Pour permettre un accès plus rapide aux éléments d'un ABR, on cherche à *équilibrer l'arbre*. Ici, on utilise le critère suivant : Un ABR est un arbre AVL si la hauteur du sous arbre gauche et celle du sous arbre droit diffèrent au plus de un pour chaque nœud de l'arbre.

2. Donner un exemple d'un ABR AVL.

3. Écrire la fonction `hauteur : 'a abr -> int`.

La rotation d'un arbre binaire de recherche consiste à faire remonter un nœud dans l'arbre et à en faire redescendre un autre, tout en conservant la propriété d'ordre des éléments de l'ABR. Par exemple une rotation à droite donne :



4. Écrire les fonctions `rotation_droite : 'a abr -> 'a abr` et `rotation_gauche : 'a abr -> 'a abr`.

On souhaite réaliser une insertion dans un arbre AVL. Pour cela, on va utiliser la fonction `inserer_abr` puis rééquilibrer avec des rotations.

5. Est-il toujours possible de rééquilibrer un AVL après une insertion avec une rotation ? deux rotations ? Donner un exemple dans lequel deux rotations sont nécessaires.

6. Écrire `equilibre : 'a abr -> 'a abr` qui équilibre un ABR après une insertion.

7. Exprimer la taille minimale d'un arbre AVL de hauteur h. On considèrera que $F_n < \frac{\phi^n}{4}$ où F_n est le n-ieme nombre de la suite de Fibonacci.

8. En déduire une borne sur la hauteur d'un arbre AVL.

Sujet 2 (13/11/2024)

contact : jean-baptiste.doderlein@ens-rennes.fr

Question de cours

Montrer que pour un arbre binaire A, on a :

$$h(A) + 1 \leq n(A) \leq 2^{h(A)+1} - 1$$

Exercice 1 : k plus grands éléments d'un ABR

1. Montrer que le parcours en profondeur infixe d'un ABR renvoie une liste triée.

2. Écrire une fonction `kpg : 'a arbre -> int -> 'a list` qui renvoie les k plus grands éléments d'un ABR.

Exercice 2 : Arbres 2-3

Un arbre 2-3 est un type d'arbre de recherche équilibré qui possède des caractéristiques spécifiques permettant de garder l'arbre équilibré :

- Tous les chemins de la racine aux feuilles d'un arbre 2-3 ont la même longueur. Cela signifie que toutes les feuilles se trouvent au même niveau (ou profondeur) dans l'arbre
- Les nœuds peuvent être de deux types :
  - Nœuds 2 : Ils contiennent une seule clé et ont deux sous-arbres (fils gauche et droit).
  - Nœuds 3 : Ils contiennent deux clés et ont trois sous-arbres (fils gauche, central et droit).
- Dans chaque nœud 2, la clé sépare les sous-arbres : toutes les clés dans le sous-arbre gauche sont inférieures à la clé du nœud, et toutes les clés dans le sous-arbre droit sont supérieures.
- Dans chaque nœud 3, les deux clés séparent les sous-arbres en trois parties :
  - Toutes les clés dans le sous-arbre gauche sont inférieures à la première clé.
  - Toutes les clés dans le sous-arbre central sont comprises entre les deux clés.
  - Toutes les clés dans le sous-arbre droit sont supérieures à la deuxième clé.

1. À partir d'un arbre 2-3 vide, ajouter successivement : `10, 20, 30, 40, 50, 60`.

2. Montrez qu'un arbre 2-3 de hauteur h possède entre $2h$ et $3h$ feuilles.

3. Utilisez cette borne pour en déduire une limite pour le nombre total de nœuds dans l'arbre.

4. Proposer un type OCaml `'a a23` pour les arbres 2-3.

5. Écrire `ajouter : 'a a23 -> 'a -> 'a a23` qui ajoute un élément dans un arbre 2-3.

Exercice 3 : Arbre infini

1. Montrer que dans tout arbre ayant une infinité de nœuds et dans lequel chaque nœud a un nombre fini de fils, il existe un chemin de longueur infinie partant de la racine et descendant le long des arêtes de l'arbre.

Sujet 3 (13/11/2024)

contact : jean-baptiste.doderlein@ens-rennes.fr

Question de cours

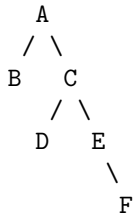
Donner le type OCaml d'un arbre binaire. Écrire la fonction `dfs : 'a arbre -> 'a list` qui effectue un parcours en profondeur postfixe d'un arbre.

Exercice 1 : Diamètre d'un arbre

Le diamètre d'un arbre est défini comme la plus grande distance entre deux nœuds de l'arbre. Autrement dit, c'est la longueur du chemin le plus long entre deux nœuds quelconques de l'arbre.

1. Rappeler le type OCaml d'un arbre binaire.

2. Donner le diamètre de l'arbre suivant :



3. Soit A un arbre, montrer que diamètre$(A) < 2\times$ hauteur(A)

4. Donner une expression du diamètre d'un arbre, en fonction des diamètres et hauteurs des deux sous arbres.

4. Écrire une fonction `diametre : 'a arbre_binaire -> int` qui calcule le diamètre d'un arbre binaire.

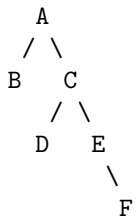
Exercice 2 : Sérialisation d'arbre binaire

On cherche à retrouver la forme d'un arbre binaire à partir de ses parcours en profondeur préfixe et infixe.

1. Donner un type `'a arbre` d'arbre binaire en OCaml

2. Écrire `dfs_prefixe : 'a arbre -> 'a list`, le parcours en profondeur préfixe d'un arbre.

3. Donner les parcours en profondeur préfixe et infixe de



On considère les fonctions suivantes déjà implémentées : - `cut : 'a list -> 'a -> 'a list * 'a list`, qui pour une liste `l` et un élément `x` dans `l` renvoie la liste des éléments avant `x` dans `l` et la liste des éléments après `x` dans `l`. Par exemple `cut [1;2;3;4] 3` renvoie `[1;2], [4]`. `-first : 'a list -> int -> 'a list * 'a list` qui à partir d'une liste `l` et d'un entier `n`, renvoie une liste contenant les `n` premiers éléments et une liste contenant les éléments restants.

4. Écrire `reconstruire : 'a list -> 'a list -> 'a arbre` qui à partir du parcours en profondeurs préfixe et infixes reconstruit l'arbre.

5. A-t-on besoin du parcours infixe pour reconstruire un ABR ?