Sujet 1 (16/10/2024)

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

Question de cours

Donner le principe de parcours en profondeur, et les 3 variantes de celui-ci. Illustrer un parcours en profondeur infixe sur un arbre d'au moins 6 nœuds.

Exercice 1 : Arbre Binaire de Recherche

On définit un arbre binaire de recherche comme un arbre binaire tel que pour chaque nœud $N(x, \text{sous arbre gauche}, \text{sous arbre droit})$:

- x est plus petit que tous les éléments du sous arbre gauche.
- x est plus grand que tous les éléments du sous arbre droit.

1. Donner le type d'un arbre binaire `'a arbre_binaire`.

2. Écrire la fonction `est_abr : 'a arbre_binaire -> bool` qui renvoie vrai si l'arbre passé en argument est un arbre binaire de recherche.

3. Écrire la fonction `est_dans_arbre : 'a -> 'a arbre_binaire -> bool` qui renvoie vrai si un élément est présent dans un arbre binaire.

4. Écrire la fonction `est_dans_abr : 'a -> 'a arbre_binaire -> bool` qui renvoie vrai si un élément est présent dans un arbre binaire de recherche.

5. Est-ce qu'il est plus efficace de chercher dans un arbre binaire ou dans un arbre binaire de recherche ? Donner un exemple.

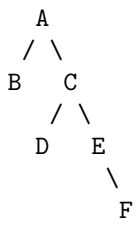
6. Donner une borne du nombre d'appels récursif de `est_dans_abr`.

Exercice 2 : 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. Écrire une fonction `diametre : 'a arbre_binaire -> int` qui calcule le diamètre d'un arbre binaire.

4. Rappeler le type pour un arbre général et écrire une fonction `diametre : 'a arbre -> int` qui calcule le diamètre d'un arbre général.

Sujet 2 (16/10/2024)

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

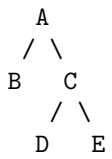
Question de cours

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 : Tri topologique

On souhaite établir l'ordre topologique des nœuds d'un arbre. Le tri topologique consiste à ordonner les nœuds d'un arbre de manière à ce que, pour chaque nœud u et v tel que v est le fils de u, le nœud v apparaisse avant le nœud u dans l'ordre. Ce type de tri est très utile dans des situations comme la planification des tâches ou la gestion des dépendances dans des systèmes complexes.

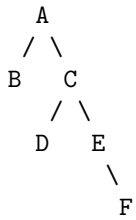
1. Donner le tri topologique de l'arbre suivant :



2. Écrire la fonction `tri_topologique : 'a arbre_binaire -> 'a list` qui effectue le tri topologique sur un arbre binaire.

Exercice 2 : Plus court chemin entre feuilles

On cherche à établir dans un arbre binaire le plus court chemin entre deux feuilles.



Par exemple dans cet arbre, les nœuds D et F sont à 3 de distance par le chemin `[D;C;E;F]`

On définit l'ancêtre commun le plus proche x (ACPP) de deux nœuds u et v comme le nœud de profondeur maximale tel que u et v sont dans les sous-arbres de x.

1. Montrer que pour l'ACPP de deux feuilles u et v, u et v n'appartiennent pas au même sous-arbre.

2. Montrer que le chemin entre deux feuilles passe nécessairement par leur ACPP.

3. Écrire une fonction `pcc : 'a arbre_binaire -> 'a -> 'a -> int` qui calcule la distance entre deux feuilles, en supposant que les feuilles soient bien dans l'arbre.

4. Écrire une fonction `pcc_avec_valeurs : 'a arbre_binaire -> 'a -> 'a -> 'a list` qui retourne les valeurs des nœuds sur le chemin le plus court entre deux feuilles.

Sujet 3 (16/10/2024)

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

Exercice de cours

1. Rappeler la définition d'un arbre binaire strict.

2. Montrer que le nombre de feuilles d'un arbre binaire strict est égal à son nombre de nœuds interne + 1.

Exercice 1 : Les zippers

Un **zipper** permet de parcourir un arbre tout en se souvenant du chemin déjà parcouru, offrant ainsi la possibilité de s'arrêter à un certain point du parcours et de reprendre plus tard, sans perdre la structure de l'arbre.

Un zipper pour un arbre peut être défini de la manière suivante :

```
type 'a chemin =
  | Top
  | Gauche of 'a chemin * 'a * 'a arbre_binaire  (* En descendant à gauche *)
  | Droite of 'a arbre_binaire * 'a * 'a chemin  (* En descendant à droite *)

type 'a zipper = 'a arbre_binaire * 'a chemin
```

avec :

- `Top` représente la racine.
- `Gauche (valeur, droite, chemin)` garde en mémoire que vous êtes descendu à gauche, avec la valeur du nœud parent, le sous-arbre droit (non visité), et le chemin pour remonter.
- `Droite (valeur, gauche, chemin)` garde en mémoire que vous êtes descendu à droite, avec le sous-arbre gauche (non visité), et le chemin pour remonter.

1. Écrire une fonction `start : 'a arbre_binaire -> 'a zipper` qui transforme un arbre en un zipper.

2. Écrire une fonction `down_left : 'a zipper -> 'a zipper option` qui permet de descendre dans l'enfant gauche du nœud.

3. Écrire une fonction `down_right : 'a zipper -> 'a zipper option` qui permet de descendre dans l'enfant droit du nœud.

4. Écrire une fonction `up : 'a zipper -> 'a zipper option` qui permet de remonter vers le nœud parent du nœud actuel.

Exercice 2 : Arbre presque complet

1. Écrire une fonction `entiers : int -> int arbre_binaire` qui à partir d'un entier $n \geq 1$ renvoie un arbre binaire contenant n nœuds et qui contient les entiers de 1 à n.

2. Quelle est la hauteur des arbres générés par votre fonction `tree` ? Si ce n'est pas déjà le cas, proposer une fonction `entiers_pc : int -> int arbre_binaire` tel que l'arbre généré soit presque complet.