université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

PFA

L3 Info, L3 Mag Info

TP 12

Présentation

Ce TP est la suite de la préparation au TP noté. Il contient :

— Un exercice sur les arbres binaires de recherche

— Un exercice sur le backtracking

— Un exercice sur les structures de données mutables

1 Arbres binaires de recherche

On suppose que l'on est dans un foncteur `Make` permettant de construire un ensemble en prenant en argument un module définissant un type de valeurs et une fonction de comparaison.

```
module type COMPARABLE =
sig
  type t
  val compare : t -> t -> int
end
module Make (E : COMPARABLE) =
struct
  (* Arbres équilibrés de type AVL *)
  type t = Nil  (* arbre vide *)
         | Node of (int, t, E.t, t) (* hauteur, gauche, element, droite *)

  let empty = Nil
  (** [merge t1 t2] renvoie l'union de [t1] et [t2] sous
      forme d'un arbre équilibré *)
  let merge t1 t2 = ...
 end
```

Le code contient aussi le corrigé des fonctions du TP 9.5. Rajouter les fonctions suivantes :

1. `min_major : t -> E.t -> E.t option` telle que `min_major e t` renvoie le plus petit élément strictement supérieur à `e` dans `t`, sous la forme d'une option (`None` si aucun élément n'est trouvé et `Some v` si un tel élément `v` existe).
2. `check_avl : t -> int -> unit` la fonction vérifie que l'arbre est bien un AVL. Plus précisément, `check_avl t n` lève une exception (avec `failwith`) si `t` contient un nœud interne ayant deux sous-arbre dont les auteurs diffèrent de plus de `n`. Attention, vous ne devez pas utiliser la hauteur stockée dans les nœuds mais vérifier que l'arbre est correct en recalculant les hauteurs.

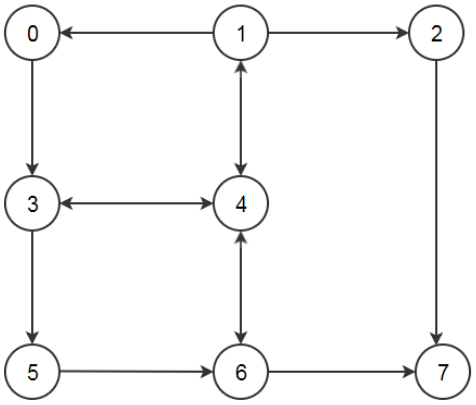
2 Retrour arrière

On suppose qu'un graphe est représenté par le type OCaml suivant :

```
type graph = (int, int list) Hashtbl.t
```

Les sommets du graphe sont des entiers entre 0 et $n-1$ où n est la taille de la table de hachage (le nombre de clés, donné par Hashtbl.length), associés à la liste des sommets voisin. Par exemple

```
let graph = Hashtbl.create 16
let () =
Hashtbl.add graph 0 [3];
Hashtbl.add graph 1 [0;2;4];
Hashtbl.add graph 2 [7];
Hashtbl.add graph 3 [4;5];
Hashtbl.add graph 4 [1;3;6];
Hashtbl.add graph 5 [6];
Hashtbl.add graph 6 [4;7];
Hashtbl.add graph 7 []
```



1. Donner une fonction `circuit : (int list -> unit) -> graph -> unit` prenant en argument une fonction `f` et appelant cette dernière sur tous les circuits Hamiltoniens, c'est à dire tous les chemins qui passent par l'ensemble des sommets du graphe une seule fois. Pour le graphe précédent, `[0;3;5;6;4;1;2;7]` est une solution (la seule).

3 Structures de données mutables

3.1 Nœud de Landin

On considère le code OCaml suivant :

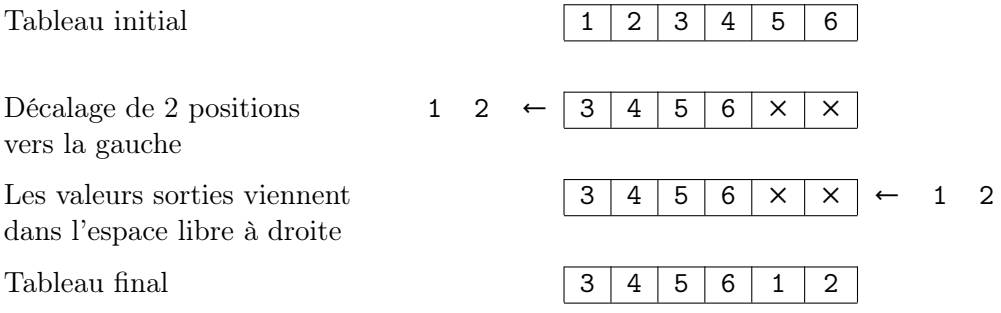
```
let fact =
   let f = ref (...) in
   ...
```

Le compléter les zone `...` pour que `fact` contienne la fonction qui calcule factorielle de façon récursive naïve (`fact` n `=` n `*` `fact` $(n-1)$ et `fact 0 = 1`) sans utiliser de boucle ou le mot clé **`let rec`**.

Indication on pourra stocker dans la référence `f` une fonction bidon levant une exception, puis mettre dans `f` une fonction anonyme qui utilise la référence pour l'apple récursif. Cette technique permettant de faire des fonctions récursive dans des langages qui ne possède pas cette construction est appelé un nœud de Landin [1].

3.2 Rotation de k positions, en place

On présente un algorithme capable d'effectuer une rotation *en place* de k éléments vers la gauche dans un tableau. On donne l'exemple avec un tableau de taille 6 contenant initialement les entiers de 1 à 6 avec une rotation de 2 positions vers la gauche :



Le problème ici est qu'il faut stocker les deux éléments qui sortent dans un tableau auxiliaire. En effet, Si on commence par décaler le `3` de deux positions à gauche, alors on écrase le `1`. Si on commence par prendre le `1` pour le mettre dans l'avant-dernière case, on écrase le `5`.

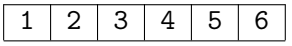
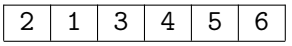
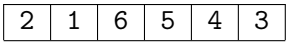
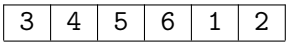
Il existe cependant une technique permettant d'effectuer cet opértion *en place*, sans utiliser d'espace auxiliaire, en se basant sur l'observation suivante. Décaler les valeurs de k positions vers la gauche est équivalent à :

— Renverser les valeurs entre les indices 0 et $k-1$

— Renverser les valeurs entre les indices k et $n-1$ (pour un tableau de taille n)

1. P. J. Landin, *The mechanical evaluation of expressions. Journal of Computer Science, 1964*

— Renverser tout le tableau

Tableau initial	
Renversement des indices 0 à 1	
Renversement des indices 2 à 5	
Renversement de tout le tableau, tableau final	

1. Implémenter une fonction

   `reverse_in_place : 'a array -> int -> int -> unit`

   telle que `reverse_in_place tab i j` inverse l'ordre des éléments entre les indices `i` et `j` du tableau (les indices sont supposés valides et `i <= j`).

   Remarque pour vous entrainez, vous pouvez écrire deux version de cette fonction, l'une récursive (faisant croitre `i` et diminuer `j`), l'autre avec une boucle **`for`** et indice `k` allant de 0 à $\frac{\texttt{j - i}}{2}$.
2. Implémenter la fonction `rotate_left : 'a array -> int -> unit` qui effectue la rotation demandée.

Remarque : cet algorithme se généralise facilement à des valeurs de k négatives et de valeurs absolues supérieure à n (la taille du tableau).

— Si $-n < k < 0$, alors cela peut s'interpréter comme effectuer une rotation vers la droite. Pour cela il suffit d'exécuter une rotation vers la gauche de $n - k$ (en effet dans notre exemple, faire une rotation de 2 cases à gauches ou $6 - 2 = 4$ cases vers la droite est la même chose)

— Si $|k| \geq n$, alors il suffit de considérer $k \mod n$. En effet, dans notre exemple, faire une rotation de 32 cases vers la gauche, c'est faire d'abord 5 rotations complètes qui ne changent rien (les valeurs reviennent à leurs positions initiales) puis 2 rotations restantes.