corrige

November 6, 2024

1 Corrige 06/11/2024

1.1 Sujet 1

1.1.1 Exercice 1

[2]:
```
type 'a arbre = F | N of 'a * 'a arbre * 'a arbre
```

[2]:
```
type 'a arbre = F | N of 'a * 'a arbre * 'a arbre
```

[4]:
```
let est_parfait t =
    let rec aux tp = match tp with
    | F -> true, 0
    | N(_,l,r) -> let pl,hl = aux l in
                  let pr,hr = aux r in
                  (pl && pr && hl=hr), (hl+1)
    in let r,_ = aux t in r
```

[4]:
```
val est_parfait : 'a arbre -> bool = <fun>
```

[5]:
```
let est_symetrique t =
    let rec mirroir t = match t with
    | F -> t
    | N(x,l,r) -> N(x, (mirroir r), (mirroir l))
    in
    let rec eq t1 t2 = match t1,t2 with
    |F,F -> true
    |N(x1,l1,r1), N(x2,l2,r2) -> x1=x2 && eq l1 l2 && eq r1 r2
    |_,_ -> false
    in
    eq t (mirroir t)
```

[5]:
```
val est_symetrique : 'a arbre -> bool = <fun>
```

1.1.2 Exercice 2

[55]:
```
let rec inserer_abr t x = match t with
|F -> N(x,F,F)
|N(v,l,r) -> if x<v then N(v,inserer_abr l x,r) else N(v,l,inserer_abr r x)
```

[55]:
```
val inserer_abr : 'a arbre -> 'a -> 'a arbre = <fun>
```

[57]:
```
let rec hauteur t = match t with
|F -> 0
|N(_,l,r) -> 1 + (max (hauteur l) (hauteur r))
```

[57]:
```
val hauteur : 'a arbre -> int = <fun>
```

[59]:
```
let rotation_droite t = match t with
|N(a,N(b,c,d),e) -> N(b,c,N(a,d,e))
|_ -> failwith "rotation impossible"
```

[59]:
```
val rotation_droite : 'a arbre -> 'a arbre = <fun>
```

[60]:
```
let rotation_gauche t = match t with
|N(b,c,N(a,d,e)) -> N(a,N(b,c,d),e)
|_ -> failwith "rotation impossible"
```

[60]:
```
val rotation_gauche : 'a arbre -> 'a arbre = <fun>
```

[ ]:
```

```

1.2 Sujet 2

1.2.1 Exercice 1

[10]:
```
let abr1 = N(6,N(3,N(1,F,F),N(5,F,F)),N(9,F,F))
```

[10]:
```
val abr1 : int arbre = N (6, N (3, N (1, F, F), N (5, F, F)), N (9, F, F))
```

[8]:
```
let kpg t k =
    let rec aux t acc nacc = match t with
    | F -> acc, nacc
    | N(x,l,r) -> let accl, naccl = aux r acc nacc in
                  if naccl >= k then accl, naccl
                  else
                      aux l (x::accl) (naccl+1)
    in let r,_ = aux t [] 0 in r
```

[8]:
```
val kpg : 'a arbre -> int -> 'a list = <fun>
```

[16]:
```
kpg abr1 5
```

[16]:
```
- : int list = [1; 3; 5; 6; 9]
```

Une autre solution est de faire un parcours infixe et de prendre les k derniers

1.3 Sujet 3

1.3.1 Exercice 2

[21]:
```
let rec dfs_pref t = match t with
|F->[]
|N(x,l,r) -> x::((dfs_pref l) @ (dfs_pref r))
```

[21]:
```
val dfs_pref : 'a arbre -> 'a list = <fun>
```

[22]:
```
let rec dfs_inf t = match t with
|F->[]
|N(x,l,r) -> ((dfs_inf l) @ x::(dfs_inf r))
```

[22]:
```
val dfs_inf : 'a arbre -> 'a list = <fun>
```

[23]:
```
let a1 = N(1,N(2,N(4,F,F),N(5,F,F)),N(3,F,F))
```

[23]:
```
val a1 : int arbre = N (1, N (2, N (4, F, F), N (5, F, F)), N (3, F, F))
```

[24]:
```
dfs_pref a1
```

[24]:
```
- : int list = [1; 2; 4; 5; 3]
```

[25]:
```
dfs_inf a1
```

[25]:
```
- : int list = [4; 2; 5; 1; 3]
```

[26]:
```
let cut l x =
    let rec aux lp acc = match lp with
    |[] -> failwith "no"
    |h::t -> if h=x then (List.rev acc), t else aux t (h::acc)
    in aux l []
```

[26]:
```
val cut : 'a list -> 'a -> 'a list * 'a list = <fun>
```

[27]:
```
let rec first l x = match l with
|[] -> []
|h::t -> if x=0 then [] else h::(first t (x-1))
```

[27]:
```
val first : 'a list -> int -> 'a list = <fun>
```

[50]:
```
let rec last l x = match l with
|[] -> []
|h::t -> if x=1 then t else last t (x-1)
```

[50]:
```
val last : 'a list -> int -> 'a list = <fun>
```

[53]:
```
let rec reconstruire pref inf = match pref with
|[] -> F
|h::t -> let linf, rinf = cut inf h in
         let lpref, rpref = (first t (List.length linf)), (last t (List.length␣
 ↪linf)) in
         N(h,(reconstruire lpref linf),(reconstruire rpref rinf))
```

[53]:
```
val reconstruire : 'a list -> 'a list -> 'a arbre = <fun>
```

[54]:
```
reconstruire (dfs_pref a1) (dfs_inf a1)
```

[54]:
```
- : int arbre = N (1, N (2, N (4, F, F), N (5, F, F)), N (3, F, F))
```