Exercice 6. Suppression d'un noeud (méthode 2)

Question 1 –

```
let rec fusion g d = match g with
  | Vide -> d
  | N(e, g2, d2) -> N(e, g2, fusion d2 d);;
```

Question 2 –

```
let rec supprimer_bis e a = match a with
  | Vide -> failwith "L'étiquette n'apparait pas"
  | N(e2, g, d) when e < e2 -> N(e2, supprimer_bis e g, d)
  | N(e2, g, d) when e > e2 -> N(e2, g, supprimer_bis e d)
  | N(_, g, d) -> fusion g d;;
```

Question 3 – Pour fusionner `G` et `D`, il suffit de repérer dans `D` le noeud d'étiquette minimale (ce noeud se trouve en bas à gauche de `D`) et de remplacer son fils gauche (qui vaut `Vide`) par `G`.

Exercice 7. Gestion d'intervalles

Question 1.a –

```
let rec add (ens: uid) (a,b): uid = match ens with
  | [] -> [a,b]
  | (a',_) :: q as ens when b < a' -> (a,b) :: ens
  | (a',b') :: q when b' < a -> (a',b') :: add q (a,b)
  | (a',b') :: q -> add q (min a' a, max b' b);;
```

Question 1.b – La fonction s'exécute en $\mathcal{O}(n)$ où n est la taille de la liste.

Question 2.a –

```
let rec uid_of_ui (li : ui): uid = match li with
  | [] -> []
  | interv :: q -> add (uid_of_ui q) interv;;
```

Question 2.b – La fonction s'exécute en $\mathcal{O}(n^2)$ où n est la taille de la liste.

Question 3 –

```
let rec coupe_gauche (ens: uid2) a = match ens with
  | Vide -> Vide, a
  | N((a'',_), g, d) when a < a'' -> coupe_gauche g a
  | N((a'',b''), g, d) when a <= b'' -> g, a''
  | N(interv, g, d) ->
     let d', a' = coupe_gauche d a in
     N(interv, g, d'), a';;
```

Question 4 – On commence par écrire une fonction `coupe_droite` qui prend en entrée E et b, et qui renvoie le couple (E', b') où :

- Si $b \in E$, alors $b' = b''$ et $E' = E \cap]b''; +\infty[$ où $[a'', b'']$ est l'intervalle contenant b.
- Si $b \notin E$, alors $b' = b$ et $E' = E \cap]b; +\infty[$.

```
let rec coupe_droite (ens: uid2) b = match ens with
  | Vide -> Vide, b
  | N((_,b''), g, d) when b > b'' -> coupe_droite d b
  | N((a'',b''), g, d) when b >= a'' -> d, b''
  | N(interv, g, d) ->
     let g', b' = coupe_droite g b in
     N(interv, g', d), b';;
```

```
let rec add2 (ens: uid2) (a,b) =
  let g', a' = coupe_gauche ens a in
  let d', b' = coupe_droite ens b in
  N((a',b'), g', d');;
```

Question 5.a –

```
let rec uid2_of_ui (li : ui): uid2 = match li with
  | [] -> Vide
  | interv :: q -> add2 (uid2_of_ui q) interv;;
```

```
let parcours_infixe ens0 =
  let rec aux acc ens = match ens with
    | Vide -> acc
    | N(e, g, d) -> aux (e::aux acc d) g
  in aux [] ens0;;
```

```
let uid_of_ui_bis (ens: ui): uid = parcours_infixe (uid2_of_ui ens);;
```

Question 5.b – Les fonctions `coupe_droite` et `coupe_gauche` s'exécutent en temps $\mathcal{O}(h)$ où h est la hauteur de l'arbre en entrée, soit en $\mathcal{O}(n)$ où n est le nombre de noeuds dans l'arbre. La fonction `uid2_of_ui` s'exécute en temps $\mathcal{O}(n^2)$ où n est le nombre d'éléments dans la liste donnée en entrée. La fonction `parcours_infixe` s'exécute en temps $\Theta(n)$ où n est le nombre de noeuds dans l'arbre. Finalement, la fonction `uid_of_ui_bis` s'exécute en temps $\mathcal{O}(n^2)$ où n est le nombre d'éléments dans la liste initiale.