Arbres binaires T.D.3.

Exercice 1

▶ On considère dans cet exercice des arbres binaires de recherche, dont les noeuds sont étiquetés par des entiers, et les feuilles ne portent pas d'information. On rappelle la propriété caractéristique d'un arbre binaire de recherche : l'étiquette de chaque noeud majore strictement toutes les étiquettes des noeuds du sous-arbre gauche et minore strictement toutes les étiquettes des noeuds du sous-arbre droit. La figure 1 présente un tel arbre, dans lequel on n'a pas fait figurer les feuilles par souci de simplification.

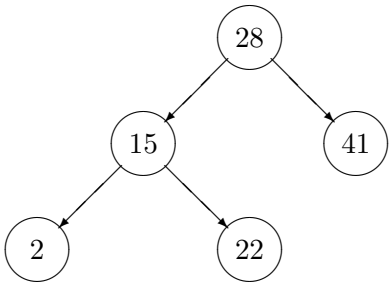


FIGURE 1 – un arbre binaire de recherche.

▶ Soient a un arbre binaire et s un entier. On note $\mathcal{T}_s(a)$ l'arbre binaire déduit de a par l'opération consistant à ajouter 1 à chaque étiquette de a au moins égale au seuil s. La figure 2 représente l'image par $\mathcal{T}_{20}$ de l'arbre binaire présenté à la figure 1.

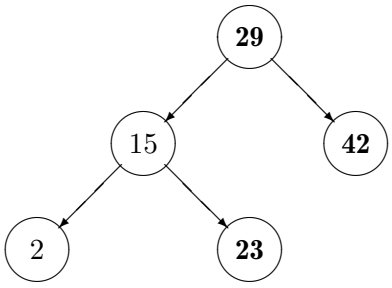


FIGURE 2 – l'image par $\mathcal{T}_{20}$ de l'arbre binaire de la figure 1.

Question 1 Montrez que si a est un arbre binaire de recherche, alors $\mathcal{T}_s(a)$ est lui aussi un arbre binaire de recherche.

▶ Pour décrire les arbres binaires de recherche étiquetés par des entiers, on définit le type OCaml suivant :

```
type abr = F | N of abr * int * abr;;
```

L'interprétation de ce type est la suivante : `F` est un arbre binaire réduit à une feuille ; `N(g,x,d)` est un arbre binaire dont la racine est étiquetée x, dont le sous-arbre gauche est g et dont le sous-arbre droit est d. Par exemple, l'arbre binaire de la figure 1 est décrit par :

```
N(N(N(F,2,F),15,N(F,22,F)),28,N(F,41,F))
```

Question 2 Rédigez en langage OCaml une fonction de signature :

```
compte : int -> abr -> int
```

spécifiée comme suit : `compte s a` compte le nombre de noeuds de l'arbre a, dont l'étiquette est au moins égale à s.

Question 3 Rédigez en langage OCaml une fonction de signature :

```
incr : int -> abr -> abr
```

spécifiée comme suit : `incr s a` construit l'arbre $\mathcal{T}_s(a)$.

Exercice 2

▶ Nous nous intéressons dans cet exercice à des arbres binaires, dont les feuilles sont étiquetées par des entiers. Voici un exemple :

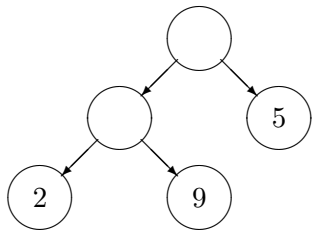


FIGURE 3 – un arbre binaire dont les feuilles sont étiquetées par des entiers

▶ La *profondeur* d'une feuille d'un arbre a est la longueur du chemin qui mène de la racine de a à cette feuille. Ainsi, dans l'arbre de la figure 3, l'une des feuilles est à la profondeur 1, et les deux autres à la profondeur 2.

▶ Pour décrire les arbres binaires de ce problème, nous définissons le type OCaml suivant :

```
type arbre = N of arbre * arbre | F of int ;;
```

Par exemple, `let a = N(N(F 2,F 9),F 5) ;;` décrit l'arbre de la figure 3.

Question 1 Soit a un arbre binaire ; notons $p(a)$ la profondeur minimale d'une feuille de a. Montrer que a contient au moins $2^{p(a)}$ feuilles.

Question 2 Rédiger en langage OCaml une fonction de signature :

```
etiquette_maximale : arbre -> int
```

spécifiée comme suit : `etiquette_maximale a` calcule la valeur maximale d'une étiquette d'une feuille de l'arbre a.

Question 3 Rédiger en langage OCaml une fonction de signature :

```
profondeur_minimale : arbre -> int
```

spécifiée comme suit : `profondeur_minimale a` calcule la profondeur minimale d'une feuille de l'arbre a.

Exercice 3

On souhaite classer les fromages français dans un arbre : les fromages les plus proches seront sur des branches « voisines » de l'arbre. Dans ce but est défini un type d'arbre étiqueté :

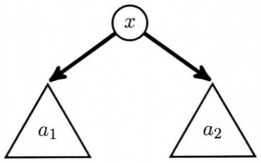
```
type arbre = Fromage of int | N of arbre * arbre * float;;
```

Pour chaque fromage, nous disposons d'un ensemble de mesures : apport énergétique, teneur en calcium, en protéines, ...

Nom	Apport énergétique (en kJ/1000g)	Calcium (en mg/100g)	...
Camembert	1150	333	...
Maroilles	348	335	...
Roquefort	374	601	...
⋮	⋮	⋮	⋮

Le tableau précédent sera représenté en OCaml par une matrice globale `fromages` de flottants. Par exemple, `fromages.(0).(1)` donne 333.0, la teneur en calcium (colonne numéro 1) du camembert (fromage numéro 0). Chaque fromage f est alors représenté par une ligne $[|f_0; \ldots; f_{q-1}|]$ de la matrice `fromages`. On pourra également supposer qu'on a défini une variable globale `q` de type entier contenant le nombre d'indicateurs dont on dispose pour chaque fromage.
Étant donné un tableau global `ponderation =` $[|a_0; \ldots; a_{q-1}|]$ de flottants strictement positifs, on définit la distance $d(f, g)$ entre deux fromages $f = [|f_0; \ldots; f_{q-1}|]$ et $g = [|g_0; \ldots; g_{q-1}|]$ par $d(f, g) = \sum_{k=0}^{q-1} a_k |f_k - g_k|$, puis on définit la distance $d(F, G)$ entre deux ensembles de fromages F et G par : $d(F, G) = \min_{f \in F, g \in G} d(f, g)$.
Enfin, on définit la distance entre deux arbres a_1 et a_2 comme la distance entre l'ensemble des fromages présents dans a_1 et l'ensemble des fromages présents dans a_2.

1. Écrire une fonction `distance : int -> int -> int -> float`.
   `distance i j` retournera la distance entre les formages de numéros `i` et `j`.
2. Définir en OCaml une matrice globale `dist`, telle que, pour tous fromages `i` et `j`, `dist.(i).(j)` est égal à la distance entre les fromages `i` et `j`.
3. Écrire une fonction `dist_arbr : arbre -> arbre -> float` qui, étant donné deux arbres de fromages, calcule la distance entre ces deux arbres.
4. La fusion de deux arbres a_1 et a_2 est un arbre de la forme ci-après où la racine est étiquetée par x, la distance entre a_1 et a_2.



   On appelle *forêt* une liste d'arbres. Un dendrogramme est un arbre construit de la manière suivante :
   (a) On crée une forêt contenant un arbre de la forme `Fromage(i)` par fromage.
   (b) Tant qu'il y a plusieurs arbres dans la forêt, on cherche deux arbres séparés par la plus petite distance possible, puis on les fusionne.
   (c) Lorsqu'il ne reste plus qu'un seul arbre, on renvoie celui-là.

   Écrire une fonction `dendrogramme : unit -> arbre` qui crée le dendrogramme de tous les fromages référencés dans la matrice globale `fromages`.