Devoir maison n°1

À rendre le lundi 15 septembre

⋆ ⋆ ⋆

Arbres et tas binomiaux

1 Préliminaires

Un **arbre** $\mathcal{A} = (r, \mathcal{L})$ est défini par la donnée d'un entier r, appelé **nœud**, et d'une liste d'arbres $\mathcal{L}$, éventuellement vide. Si la liste $\mathcal{L}$ est vide, on dit que r est une **feuille**. Sinon, la liste $\mathcal{L}$ contient ℓ éléments $\mathcal{T}_i = (r_i, \mathcal{L}_i)$, $1 \leqslant i \leqslant \ell$, on dit que r est un **nœud interne**, que les nœuds r_i sont les **enfants** de r, et que r est le **parent** des r_i.

- Tous les nœuds de $\mathcal{A}$ sont supposés distincts. On note $\mathcal{N}(\mathcal{A})$ l'ensemble de tous les nœuds de $\mathcal{A}$ et $|\mathcal{A}|$ leur nombre.
- Dans un arbre, les **voisins** d'un nœud sont son parent (s'il existe) et ses enfants.
- Entre deux nœuds s et t de $\mathcal{A}$ il existe un unique **chemin**, composé de nœuds x_0, x_1, …, x_k deux à deux distincts, tels que $x_0 = s$, $x_k = t$, et x_i et x_{i+1} sont voisins pour $0 \leqslant i \leqslant k-1$.

  On note $chemin(s, t)$ ce chemin et on dit que k est sa **longueur**.
- Le nœud r est appelé la **racine** de $\mathcal{A}$.
- La **profondeur** d'un nœud s est la longueur de $chemin(s, r)$; en particulier, la racine r a pour profondeur 0. La **hauteur** de $\mathcal{A}$, notée $h(\mathcal{A})$ est le maximum de la profondeur de ses nœuds.

Les arbres seront représentés en OCaml par le type suivant :

```
type arbre = N of int * arbre list
```

Question 1 Écrire une fonction `taille (a : arbre) : int` qui calcule la taille d'un arbre.

2 Arbres binomiaux

Dans cette partie, on étudie des arbres particuliers, appelés « binomiaux ».

Soit k un entier positif ou nul. Un arbre binomial d'ordre k, noté $\mathcal{B}_k$, est défini comme suit :

- si $k = 0$, $\mathcal{B}_0$ est un arbre réduit à sa racine ;
- si $k > 0$, $\mathcal{B}_k$ est la forme $(r_k, (\mathcal{B}_{k-1}, \mathcal{B}_{k-2}, \ldots, \mathcal{B}_1, \mathcal{B}_0))$.

Question 2 Dessiner $\mathcal{B}_4$, avec une numérotation des nœuds telle que le parcours préfixe se fait par ordre croissant et contient tous les entiers de $[\![0, |\mathcal{B}_4| - 1]\!]$.

Question 3 Montrer que $|\mathcal{B}_k| = 2^k$. Combien de feuilles contient $\mathcal{B}_k$?

Question 4 Pour $k > 0$, montrer qu'on peut aussi définir récursivement $\mathcal{B}_k$ à l'aide de deux copies de $\mathcal{B}_{k-1}$.

Question 5 Écrire une fonction `decale (n : int) (a : arbre) : arbre` qui prend en arguments un entier n et un arbre $\mathcal{A}$ et renvoie une copie de $\mathcal{A}$ dans laquelle chaque nœud de numéro i est remplacé par un nœud de numéro $i + n$.

Question 6 Écrire une fonction `bk (k : int) : arbre` qui prend en arguments un entier $k \geqslant 0$ et renvoie $\mathcal{B}_k$ avec une numérotation des nœuds telle que le parcours en profondeur préfixe se fait par ordre croissant et contient tous les entiers de $[\![0, |\mathcal{B}_k| - 1]\!]$.

On rappelle que `1 lsl n` *renvoie la valeur* 2^n.

Question 7 Quelle est la hauteur de $\mathcal{B}_k$? Quelle est la longueur maximale d'un chemin entre deux nœuds ?

Question 8 Combien de nœuds ont une profondeur donnée ℓ ?

3 Tas binomiaux

Un **tas binomial** est un ensemble fini d'arbres étiquetés binomiaux d'ordres deux à deux distincts qui sont de plus des arbres **tournois**, c'est-à-dire tels que la valeur d'un nœud non racine est toujours inférieure ou égale à celle de son parent. Les arbres sont rangés par ordres croissants.

La figure 1 est un exemple de tas binomial à 13 éléments.

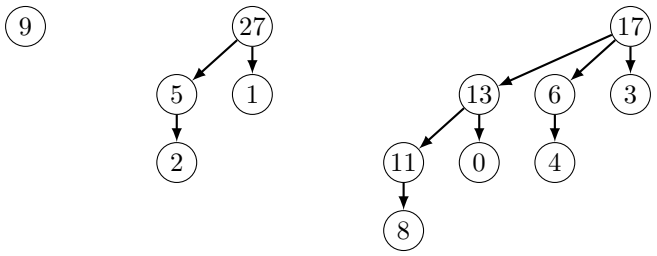

FIGURE 1 – Un tas binomial à 13 éléments

La **taille** d'un tas binomial $\mathcal{T}$, notée $|\mathcal{T}|$, est son nombre total de nœuds. La **longueur** d'un tas binomial $\mathcal{T}$, notée $\ell(\mathcal{T})$, est le nombre d'arbres binomiaux qui le composent.

Question 9 On considère un tas binomial $\mathcal{T}$ de taille n. Déterminer $\ell(\mathcal{T})$, et les ordres des arbres binomiaux qui composent $\mathcal{T}$, en fonction de l'écriture en binaire de n.

On considère les types suivants :

```
type binom = int * arbre

type tas = binom list
```

Tel qu'un arbre binomial est donné par un couple `(k, N(r, lst))`, où `N(r, lst)` forme un arbre binomial d'ordre k (on donne l'information de l'ordre directement pour éviter de le calculer). Le tas donné en exemple peut alors s'écrire :

```
let t = [(0, N(9, []));
         (2, N(27, [N(5, [N(2, [])]);
                    N(1, [])]));
         (3, N(17, [N(13, [N(11, [N(8, [])]);
                           N(0, [])]);
                    N(6, [N(4, [])]);
                    N(3, [])]))]
```

Question 10 Écrire une fonction `fusion_arbre (a : binom) (b : binom) : binom` qui prend en argument deux arbres binomiaux **tournois** de même ordre k et renvoie un arbre binomial **tournoi** d'ordre $k + 1$

contenant les étiquettes des deux arbres. Cette fonction devra avoir une complexité constante et renvoyer une erreur si les deux arbres ne sont pas de même ordre.

Question 11 En déduire une fonction `fusion (t1 : tas) (t2 : tas) : tas` qui fusionne efficacement deux tas binomiaux. Cette fonction devra avoir une complexité linéaire en la longueur des deux tas.

Indication : On pourra écrire une fonction `fusion` *dont le premier argument est soit un tas binomial, soit une liste d'arbres binomiaux tournois à au moins deux éléments* `a1 :: b1 :: q1` *où* `b1 :: q1` *est un tas binomial et où le rang de* `a1` *est égal au rang de* `b1`*.*

Question 12 En déduire une fonction `insertion (t : tas) (x : int) : tas` qui insère un élément dans un tas binomial. Cette fonction devra avoir une complexité logarithmique en la taille du tas dans le pire cas, et on demande de le justifier.

Question 13 Quitte à modifier la fonction précédente, montrer qu'effectuer 2^m insertions successives à partir d'un tas vide a une complexité totale en $\mathcal{O}(2^m)$. Quelle est la complexité de `insertion` dans le pire cas en fonction de la taille n du tas ?

Pour extraire l'élément maximal d'un tas binomial, on commence par extraire l'arbre binomial qui le contient.

Question 14 Écrire une fonction `extraire_arbre (t : tas) : binom * tas` qui prend en argument un tas et renvoie un couple formé de l'arbre binomial qui contient l'élément maximal du tas et d'un nouveau tas formé de l'ancien auquel on a retiré cet arbre binomial. On affichera un message d'erreur si le tas est vide.

Question 15 En déduire une fonction `extraire_max (t : tas) : int * tas` qui prend en argument un tas et renvoie un couple formé de l'élément maximal du tas et d'un nouveau tas formé de l'ancien auquel on a retiré cet élément.

Question 16 Déterminer la complexité temporelle de `extraire_max` en fonction de la taille n du tas. Commenter.

⋆ ⋆ ⋆