Devoir surveillé n°1

Durée : 2 heures, calculatrices et documents interdits

I. Autour des arbres binaires.

On considère des arbres binaires étiquetés par des entiers. Pour cela, on utilise le type suivant :

```
type arbre =
  | Vide
  | Noeud of int * arbre * arbre
;;
```

1) Écrire une fonction `mini : arbre -> int` qui prend en argument un arbre et qui renvoie le minimum de ses étiquettes.

2) On rappelle que la profondeur d'un nœud est sa distance par rapport à la racine (la profondeur de la racine est 0, celle des fils éventuels de la racine est 1, etc.).
Écrire une fonction `noeuds_profondeur : arbre -> int -> int list` qui prend en argument un arbre et un entier p et qui renvoie la liste des nœuds de l'arbre dont la profondeur est p.
On pourra utiliser l'opérateur `@` de concaténation sur les listes.

II. Numérotations de $\mathbb{N}^2$.

On étudie ici la fonction `cantor` définie par

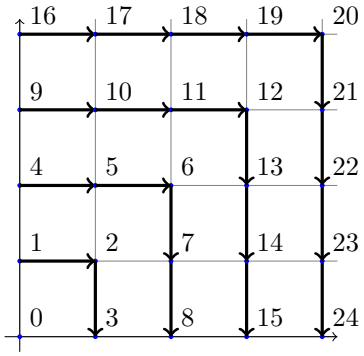
$$\forall (x,y) \in \mathbb{N}^2,\ \texttt{cantor}(x,y) = x + \frac{(x+y)(x+y+1)}{2}$$

1) Cette fonction, appelée fonction de couplage de Cantor, réalise une numérotation dite *en triangle* des éléments de $\mathbb{N}^2$. Justifier par une figure cette terminologie ; on pourra commencer par calculer `cantor`$(0,1)$, `cantor`$(1,0)$ et `cantor`$(1,1)$.

2) Écrire en OCaml la fonction `cantor : int -> int -> int`.

3) Écrire une fonction **récursive** `cantor_rec : int -> int -> int` réalisant **de proche en proche** cette même numérotation ; le cas terminal correspondra au cas $(0,0)$.

4) Dans cette question, on numérote les éléments de $\mathbb{N}^2$ en carré comme illustré sur la figure ci-dessous ; écrire une fonction récursive `carre : int -> int -> int` réalisant **de proche en proche** cette numérotation ; le cas terminal correspondra au cas $(0,0)$.



III. Multiplication égyptienne.

Soit a un entier naturel non nul ; a admet une unique écriture binaire $a = \sum_{i=0}^{n} 2^i a_i$ où $a_i \in \{0;1\}$ et $a_n = 1$. Le nombre a_i est appelé $i^{\text{ème}}$ bit de a. On représente l'entier positif non nul a par la liste de ses bits : $[a_0; a_1; \ldots; a_n]$, la liste vide représentant 0.

1) Écrire les fonctions suivantes :

a) `estPair : int list -> bool`, qui prend en argument un entier positif a et qui renvoie `true` si a est pair, `false` sinon.

b) `egalUn : int list -> bool`, qui prend en argument un entier positif a et qui renvoie `true` si a vaut 1, `false` sinon.

c) `double : int list -> int list`, qui prend en argument un entier positif a et qui renvoie le double de a.

d) `moitie : int list -> int list`, qui prend en argument un entier positif a et qui renvoie $\left\lfloor \frac{a}{2} \right\rfloor$.

2) **a)** Écrire une fonction `successeur : int list -> int list`, qui prend en argument un entier positif a et qui renvoie $a + 1$.

b) A l'aide de cette seule fonction, écrire une fonction (**récursive**) `add : int list -> int list -> int list` qui additionne deux entiers positifs.

3) Pour calculer le produit de deux entiers positifs, on se base sur le principe suivant :

$$a \cdot b = \begin{cases} a & \text{si } b = 1 \\ (2a) \cdot (b/2) & \text{si } b \text{ est pair} \\ (2a) \cdot [(b-1)/2] + a & \text{si } b \text{ est impair} \end{cases}$$

a) En utilisant uniquement les fonctions précédemment définies, écrire une fonction récursive `mult : int list -> int list -> int list` qui calcule le produit de deux entiers positifs.

b) Justifier que la fonction `mult` termine. On pourra raisonner par récurrence sur la longueur d'une des deux listes.

c) Au total, combien d'appels à la fonction `mult` sont-ils effectués lors du calcul de ab ?

IV. Suite de Syracuse.

Soit $n \in \mathbb{N}^*$. On considère la *suite de Syracuse* $(u_k)_{k\in\mathbb{N}}$ définie par

$$u_0 = n \text{ et } \forall k \in \mathbb{N}, u_{k+1} = \begin{cases} u_k/2 \text{ si } u_k \text{ est pair} \\ 3u_k + 1 \text{ sinon} \end{cases}$$

1) Écrire une fonction `suivant : int -> int` qui, lorsqu'elle prend en argument l'entier correspondant à u_k, renvoie u_{k+1}.

La *conjecture de Syracuse*, un problème encore ouvert, affirme que quelle que soit la valeur de n, il existe un rang k_0 tel que $u_{k_0} = 1$; à partir du rang k_0, la suite prendrait alors périodiquement les valeurs 1, 4, 2, répétant un cycle de longueur 3.
Cette conjecture a été vérifiée pour tous les entiers $n < 1,25 \times 2^{62}$.

2) On appelle *vol* de n, si elle existe, la liste $[u_0; u_1; \ldots; u_{k_0}]$ pour laquelle $u_0 = n$ et k_0 est le plus petit entier k tel que $u_k = 1$.
Écrire une fonction `vol : int -> int list` qui prend en argument un entier n et qui renvoie le vol de n.

Par exemple, `vol 6` renverra `[6; 3; 10; 5; 16; 8; 4; 2; 1]`

3) On appelle *altitude maximale* de n la valeur maximale du vol de n.
Écrire une fonction `altitude_max : int -> int` qui prend en argument un entier n et qui renvoie son altitude maximale.

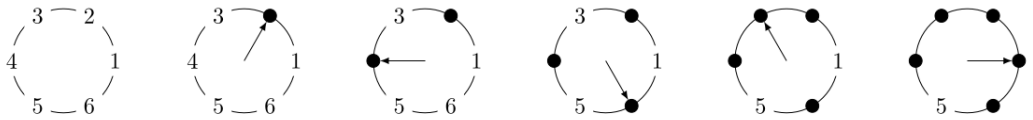
Par exemple, `altitude_max 6` renverra `16`

4) On appelle *durée de vol en altitude* de n le nombre maximum de termes consécutifs du vol de n ayant une valeur strictement supérieure à celle du point de départ $u_0 = n$.
Écrire une fonction `duree_vol_alt : int -> int` qui prend en argument un entier n et qui renvoie sa durée de vol en altitude.

Par exemple, le vol de 9 étant $[9; 28; 14; 7; 22; 11; 34; 17; 52; 26; 13; 40; 20; 10; 5; 16; 8; 4; 2]$, l'appel `duree_vol_alt 9` renverra 10 (obtenue pour la sous-liste $[22; 11; 34; 17; 52; 26; 13; 40; 20; 10]$).

V. Problème de Josèphe.

Une assemblée de n personnes, numérotées de 1 à n, forme un cercle. On décide de parcourir le cercle dans l'ordre croissant des numéros et d'éliminer une personne sur deux (qui sort alors du cercle), et ainsi de suite jusqu'à ce qu'il ne reste plus qu'une personne dans le cercle. On note J la fonction qui, à chaque entier $n \geqslant 1$, associe le numéro de la dernière personne restant après élimination de tous les autres candidats.



Par exemple, pour $n = 6$, l'ordre d'élimination des candidats est $(2, 4, 6, 3, 1)$ et le dernier candidat restant est le numéro 5, donc $J(6) = 5$.

1) On suppose dans cette question que n est pair et on pose $p = \frac{n}{2}$. Après un tour du cercle, p candidats ont été éliminés. On renumérote les p candidats restants de 1 à p. Exprimer l'ancien numéro x_i de chaque candidat restant en fonction de son nouveau numéro i et en déduire une relation liant $J(2p)$ à $J(p)$.

2) Trouver de même une relation liant $J(2p + 1)$ à $J(p)$.

3) Écrire une fonction récursive `josephe`, de type `int -> int`, calculant $J(n)$.

— **FIN** —