1 Exercice du cours

Liste chaînée

Définir à la main une chaîne contenant `[0;1;2;3;4]` où chaque élément sera
- un couple `(u, p)` où `u` est la valeur stockée et `p` un pointeur vers l'élément suivant
- sauf si il s'agit du dernier élément auquel cas ce sera `u`.

Quel est le type de l'objet ainsi construit ?

2 Le crible d'Ératosthène

Soit n un entier fixé. Le crible d'Ératosthène[1] est un algorithme qui permet de déterminer de manière efficace les entiers premiers strictement inférieurs à n. On construit pour cela un tableau t de booléens de taille n dont toutes les cases sont initialisées à `true`, sauf les cases 0 et 1 qui contiennent initialement `false`. Pour i variant de 2 à la partie entière de $\sqrt{n}$, on itère alors le procédé suivant : si `t.(i)` contient `true` alors on met `false` dans toutes les cases du tableau dont l'indice est un multiple strict de i. À la fin de l'algorithme, `t.(i)` contient `true` si et seulement si l'entier i est premier.

Question 1 Écrivez une fonction `eratosthene` qui implémente cet algorithme.

```
val eratosthene : int -> bool array
```

Note : on peut utiliser le test `i * i <= n` plutôt que de calculer explicitement $\lfloor\sqrt{n}\rfloor$.

Question 2 Écrivez une fonction `decompose_somme` telle que `decompose_somme t s` retourne un booléen indiquant si l'entier `s` se décompose comme la somme de deux entiers premiers. (`t` est un tableau de booléens de taille supérieure à `s` tel que `t.(x)` indique si `x` est premier.)

```
val decompose_somme : bool array -> int -> bool
```

3 Polynômes

Nous étudions une implémentation des polynômes à une variable et des opérations élémentaires sur les polynômes. Un polynôme :

$$P = a_0 + a_1X + a_2X^2 + \cdots + a_nX^n$$

sera représenté en machine par le tableau suivant en Ocaml :

```
p = [|a0 ; a1 ; a2 ; ... ; an |]
```

On se contentera de manipuler des polynômes à coefficients entiers donc $a_0, \cdots, a_n$ seront de type `int` et `p` de type `int array`.

1. Ératosthène fut un astronome, géographe, philosophe et mathématicien grec du IIIe siècle av. J.-C. (Cyrène, v. -276 – Alexandrie, Égypte, v. -194).

1) Addition, soustraction, multiplication

Si

$$P = \sum_{i=0}^{d_P} a_i X^i \quad \text{et} \quad Q = \sum_{i=0}^{d_Q} b_i X^i$$

alors

$$P \pm Q = \sum_{i=0}^{\max(d_P, d_Q)} (a_i \pm b_i) X^i$$

en convenant que les a_i, b_i manquants valent zéro. Pour implémenter ceci en Ocaml une méthode consiste à définir un tableau résultat, `r`, suffisamment long et initialisé à zéro, à y recopier `p` puis à lui ajouter ou retrancher `q`.
Voici le code d'une fonction `(++)` additionnant deux polynômes. Une fonction dont le nom est constitué de symbôles ($+ - */=$ etc) et dont le nom est entouré de parenthèse sera utilisable en notation *infixe* : ici on va pouvoir faire `p ++ q` au lieu de `somme_polynome p q` (notation *préfixe*).

```
(* addition *)
let (++) p q =
  let lp = Array.length p
  and lq = Array.length q
  in
  let r = Array.make (max lp lq) 0 in
  for i=0 to lp-1 do (* on recopie *)
    r.(i) <- p.(i)
  done;
  for i=0 to lq-1 do (* on ajoute *)
    r.(i) <- r.(i) + q.(i)
  done;
  r;;
```

Recopier ce code, définir de même la soustraction et la multiplication des polynômes qui seront notées `(--)` et `( ** )` (attention aux espaces entre les parenthèses et les étoiles, sinon c'est considéré comme un commentaire vide).
Tester vos fonctions avec les polynômes `p` et `q` définis ci-dessous ou avec d'autres.

```
let p = [| 1; 2; 3 |]      (* 1 + 2x + 3x^2 *)
and q = [| 4; 5; 6; 7|]    (* 4 + 5x + 6x^2 + 7x^3 *)
;;

p ++ q;;
```

2) Les polynômes de Tchebychev

Il s'agit d'une suite de polynômes définis par les relations :

$$\begin{aligned} T_0 &= 1, \\ T_1 &= X, \\ T_n &= 2XT_{n-1} - T_{n-2} \qquad \text{si } n > 2. \end{aligned}$$

Programmer itérativement le calcul du $n$ème polynôme de Tchebychev à l'aide des opérations `++`, `--` et `**` définies précédemment.
Vous pouvez au choix utiliser une matrice `t` avec `t.(i)` $= T_i$ les `t.(i)` étant calculés de proche en proche ou, mais c'est plus compliqué à implémenter, n'utiliser que deux polynômes variables (donc en fait des références sur des polynômes) contenant les deux derniers polynômes calculés et faire *évoluer* ces variables pour calculer les T_i de proche en proche. Quelque soit votre choix, votre fonction `tchebychev : int -> int array` ne devra retourner que le polynôme dont l'indice est fourni en argument et sa complexité devra être linéaire en i.
Vérifier que $T_7 = 64X^7 - 112X^5 + 56X^3 - 7X$.

3) Composition

Si $P = \sum_{i=0}^{d_P} a_i X^i$ et Q est un polynôme quelconque alors $P \circ Q = \sum_{i=0}^{d_P} a_i Q^i$. En particulier si a est un entier, $P(a) = P \circ a$ où le deuxième a désigne le polynôme constant aX^0. La composition des polynômes est une opération de même nature que l'évaluation en un point et les algorithmes d'évaluation peuvent être utilisés pour calculer une composée. En particulier, on peut utiliser l'algorithme de Hörner qui va consister à calculer $P \circ Q$ sous la forme $a_0 + Q\,(a_1 + Q\,(a_2 + \cdots + Qa_n)\cdots)$.
Programmer cette opération de composition en Caml, elle sera notée `compose`. L'utiliser pour calculer :

```
compose [|0; 0; 0; 0; 0; 0; 1|] [|1; 1|]
```

et expliquer quel est le résultat obtenu.

4) Division euclidienne

On considère l'algorithme suivant :

Entrée $A = \sum_i^n a_i X^{\in}\mathbb{R}[X]i$ et $B = \sum_{i=0}^m b_i X^i \in \mathbb{R}[X]$ avec $b_m \neq 0$

Sortie Le couple $(Q, R) \in \mathbb{R}[X]$ tel que $A = BQ + R$ et $\deg R < m$

1. Si $n < m$, renvoyer $Q = 0$ et $R = A$

2. $R \leftarrow A$, $u \leftarrow b_m^{-1}$

3. pour $i = n - m, n - m - 1, \cdots, 0$, faire
- si $\deg R = m + i$ alors
  - $q_i \leftarrow \mathrm{cd}(R)u$,
  - $R \leftarrow R - q_i X^i B$
- sinon
  - $q_i \leftarrow 0$

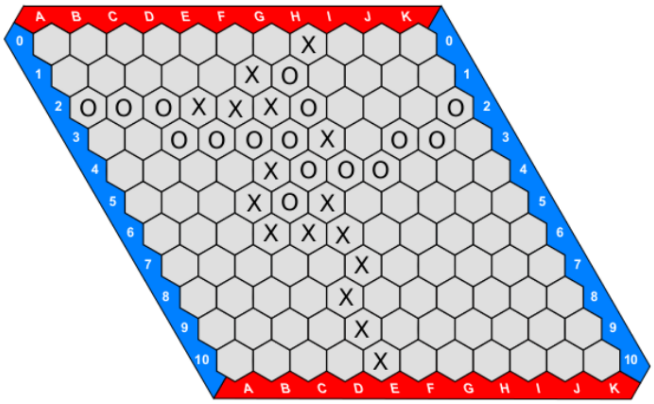
4. Renvoyer $Q = \sum_{i=0}^{n-m} q_i X^i$ et R

où $\mathrm{cd}(R)$ désigne le coefficient dominant d'un polynôme.

1) Que fait cet algorithme ?
2) Quelle est sa complexité en fonction de n et m ?
3) Est-ce que l'algorithme fonctionne avec des polynômes à coefficients entiers ?
4) Implémenter l'algorithme.

4 Jeu de Hex

Le jeu de Hex co-inventé par le mathématicien John Nash est un jeu de stratégie sur plateau à deux joueurs. Il se joue sur un plateau en forme de losange divisé en 14×14 (ou 11×11) cases hexagonales.
Les deux joueurs `O` et `X` jouent en posant une pièce à tour de rôle ; le but pour chacun des joueurs est de connecter un bord du plateau à son bord opposé par une chaîne connexe de pièces lui appartenant.



Sur cette image le joueur `O` a gagné. Une des particularités de ce jeu est que si l'on ne peut plus jouer, alors il y a toujours un et un seul joueur victorieux.
On codera le tableau hexagonal sous forme de matrice en faisant correspondre les lignes et en associant à une diagonale la colonne correspondante de la matrice (sur le dessin `A`, `B` et `H` sont envoyées respectivement sur les colonnes 0, 1 et 7 de la matrice).

1) Quelles sont les cases voisines de la case (i, j) sur un plateau de taille n ? Écrire une fonction `voisins : int -> int -> int -> (int * int) list` qui prend en paramètre i, j et n. On fera attention aux cas limites.
2) Écrire une fonction qui étant donné un plateau de jeu sous forme d'une matrice $n \times n$ de `char` (`'O'`, `'X'` ou `' '` pour les cases vides) détermine si un joueur a gagné et si oui lequel.
3) S'il vous reste du temps, démontrer rigoureusement qu'il ne peut pas y avoir de match nul et qu'il existe toujours un gagnant.