Chemins dans un graphe

Question 1

$A \oplus B = B \oplus A$
$(A \oplus B) \oplus C = A \oplus (B \oplus C)$
$A \oplus 0_n = 0_n \oplus A = A$

$A \oplus A = A$
$(A \oplus B) \otimes C = (A \otimes C) \oplus (B \otimes C)$
$A \otimes (B \oplus C) = (A \otimes B) \oplus (A \otimes C)$
$A \otimes I_n = I_n \otimes A = A$

Puis, faire une petite récurrence.

Question 2

Pour la somme, cela revient à effectuer un "ou" logique terme à terme : une double boucle permet de conclure :

```
let somme a b =
  let n = Array.length a in
  let c = Array.make_matrix n n true in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      c.(i).(j) <- a.(i).(j) || b.(i).(j)
    done
  done ;
  c ;;
```

Pour le produit, cela revient à effectuer un produit matriciel où l'addition est remplacée par le "ou" logique, et la multiplication par le "et" logique.
On utilise une fonction auxiliaire qui étant donné deux entier i et j et un entier k calcule le coefficient (i, j) du produit ; l'indice k débute à 0 et permet de s'arrêter au plus tard après n étapes : on s'arrête dès que l'on obtient vrai (évaluation paresseuse du "ou" ||).

```
let produit a b =
  let n = Array.length a in
  let d = Array.make_matrix n n true in
  let rec aux i j = function (* fait le test récursivement pour tout k entre 0 et n-1*)
    | k when k = n -> false
    | k            -> (a.(i).(k) && b.(k).(j)) || aux i j (k+1)
  in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      d.(i).(j) <- aux i j 0
    done
  done ;
  d ;;
```

Question 3

On code l'identité id, puis on élève la matrice $(A \oplus I)$ à la puissance n : on choisit l'exponentiation rapide ici, effectuée par la fonction aux qui prend un entier k en entrée et renvoie la matrice $(A \oplus I)^k$.

```
let accessible a =
  let n = Array.length a in
  let id = Array.make_matrix n n false in
  for i = 0 to n-1 do
    id.(i).(i) <- true
  done ;
  let b = somme a id in
  let rec aux = function
    | 1 -> b
    | k when k mod 2 = 0 -> let d = aux (k/2) in produit d d
    | k -> let d = aux (k/2) in produit b (produit d d)
  in aux n ;;
```

Question 4

```
let rec affiche_chemin = function (* à partir d'une liste de sommets, affiche le chemin*)
  | []   -> ()
  | [t]  -> print_int t ; print_newline ()
  | t::q -> print_int t ; print_string " -> " ; affiche_chemin q ;;
```

La fonction auxiliaire prend en entrée une liste de chemins (donc une liste de listes d'entiers), une liste des sommets exclus (qui évitera de passer plusieurs fois par le même sommet, afin d'éviter les cycles), un sommet i qui est le sommet courant, ainsi qu'un sommet k représentant le sommet suivant à visiter. Elle renvoie la liste des chemins allant de i à j. Il restera à afficher tous ces chemins.

Détaillons le parcours effectué :
Si à partir du sommet i, le sommet k est un voisin de i non encore visité, on peut le choisir en construisant une nouvelle liste de chemins allant de k à j sans passer par i : il restera à ajouter i au début de chaque chemin afin d'avoir la liste des chemins allant de i) j en passant par k. On peut aussi choisir de ne pas passer par k et donc de passer directement à (k+1).

```
let chemins a i j =
  let n = Array.length a in
  let rec aux chemins exclus i = function
    | _ when i = j                          -> [[j]]
    | k when k = n                          -> chemins
    | k when a.(i).(k) && not (List.mem k exclus) ->
                let lst = List.map (function l -> i::l) (aux [] (i::exclus) k 0)
                in aux (lst @ chemins) exclus i (k+1)
    | k                                     -> aux chemins exclus i (k+1)
  in List.iter affiche_chemin (aux [] [] i 0) ;;
```