Exercice 1. Implémentation de l'algorithme de Berge

Question 1 – Voir les question intermédiaire ci-dessous.

Question 2 –

```
let make_graphe (g_bip: graphe_bip): graphe =
  let g = {
      mat_adj = Array.make_matrix g_bip.n0 g_bip.n1 0;
      s_adj = Array.make g_bip.n0 true;
      t_adj = Array.make g_bip.n1 true;
    } in
  List.iter (fun (s1,s2) -> g.mat_adj.(s1).(s2) <- 1) g_bip.aretes;
  g;;
```

Question 3 –

```
let get_couplage (g: graphe): (int*int) list =
  let n0 = Array.length g.s_adj in
  let n1 = Array.length g.t_adj in
  let coupl = ref [] in
  for u0 = 0 to n0 - 1 do
    for u1 = 0 to n1 - 1 do
      if g.mat_adj.(u0).(u1) = -1 then
        coupl := (u0,u1) :: !coupl
    done;
  done;
  !coupl;;
```

Question 4 –

```
(* c = chaine M-augmentante dont le premier sommet appartient à S0 *)
(* On remarque que les nouveaux sommets saturés sont les extrémités de
   la chaine *)
let maj_graphe (g: graphe) (c: int list): unit =
  let rec aux = function
    | u0 :: u1 :: [] ->
       g.mat_adj.(u0).(u1) <- -1;
       g.t_adj.(u1) <- false
    | u0 :: u1 :: v0 :: q ->
       g.mat_adj.(u0).(u1) <- -1;
       g.mat_adj.(v0).(u1) <- 1;
       aux (v0 :: q)
    | _ -> failwith "maj_graphe: liste vide ou de taille impaire en entrée"
  in
  aux c;
  g.s_adj.(List.hd c) <- false;;
```

Question 5.a –

```
let rec list_init (n: int) (pred: int -> bool): int list = match n with
  | -1 -> []
  | n when pred n -> n :: list_init (n-1) pred
  | n -> list_init (n-1) pred;;
```

Question 5.b – On effectue un parcours en profondeur du graphe. Pour cela, on utiliser une variable « `vis: bool array array` » telle que :

- `vis.(0).(i)` indique si le sommet numéro `i` de S_0 a déjà été visité.
- `vis.(1).(j)` indique si le sommet numéro `j` de S_1 a déjà été visité.

La fonction `voisins` renvoie la liste des sommets non visités voisins de `u`.
Le paramètre $k \in \{0, 1\}$ en entrée indique si `u` appartient à S_0 ou S_1.
La fonction `parc_prof` renvoie un chemin de `s` vers `t` dans `g` sous la forme d'une liste chaînée de sommets (ou la liste vide si un tel chemin n'a pas été trouvé).

```
let voisins (g: graphe) (u: int) (vis: bool array array) (k: int) =
  if k = 0 then begin
    let n1 = Array.length vis.(1) in
    let pred v = g.mat_adj.(u).(v) = 1 in
    list_init (n1-1) pred
    end
  else begin
      let n0 = Array.length vis.(0) in
      let pred v = g.mat_adj.(v).(u) = -1 in
      list_init (n0-1) pred
    end;;
let get_CMA (g: graphe) =
  let n0 = Array.length g.s_adj in
  let n1 = Array.length g.t_adj in
  let vis = [|Array.make n0 false; Array.make n1 false|] in

  let rec parc_prof (li: int list) (k: int): int list = match li with
    | [] -> []
    | u :: q when k = 1 && g.t_adj.(u) -> [u]
    | u :: q when vis.(k).(u) -> parc_prof q k
    | u :: q ->
       vis.(k).(u) <- true;
       let c = parc_prof (voisins g u vis k) (1-k) in
       if c = [] then parc_prof q k else u :: c
  in

  let pred u0 = g.s_adj.(u0) in
  parc_prof (list_init (n0-1) pred) 0;;
```

Question 6 –

```
let berge (g_bip: graphe_bip) =
  let g = make_graphe g_bip in
  let c = ref (get_CMA g) in
  while !c <> [] do
    maj_graphe g !c;
    c := get_CMA g;
  done;
  get_couplage g;;
```