

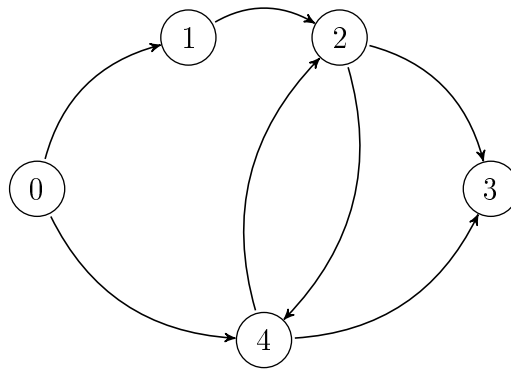
TD 4

Représentation d'un graphe

Correction

1 Représentation par matrice d'adjacence

La matrice d'adjacence est une matrice booléenne (carrée et d'ordre n) telle que $m.(i).(j)$ vaut true si et seulement si on a un arc du sommet i au sommet j .



Nous utiliserons alors le type suivant :

```
type graphe_1 = bool array array ;;
```

1. Définir l'exemple avec ce type.

Correction

Listing 1 – L'exemple 1

```
type graphe_1 = bool array array ;;

let (exemple_1:graphe_1) = let g = Array.make_matrix 5 5 false in
  g.(0).(1) <- true ;
  g.(1).(2) <- true ;
  g.(2).(3) <- true ;
  g.(0).(4) <- true ;
  g.(4).(3) <- true ;
  g.(2).(4) <- true ;
  g.(4).(2) <- true ;
  g;;
```

2. Écrire deux fonctions `taille` et `nb_arcs` comptant respectivement le nombre de sommets et le nombre d'arcs d'un graphe.

```
taille : graphe_1 -> int
```

```
nb_arc : graphe -> int
```

CorrectionListing 2 – Les fonctions `taille` et `nb_arcs`

```
let taille (graphe:graphe_1) = Array.length graphe.(0) ;;

let nb_arc (graphe:graphe_1) =
  let nb = ref 0 in
  for i = 0 to taille(graphe) -1 do
    for j = 0 to taille(graphe) -1 do
      if graphe.(i).(j) then incr nb ;
    done;
  done;
  !nb;;
```

3. Écrire les fonctions `ajout_arc` et `supprime_arc` :

- La fonction `ajout_arc` prend en argument un graphe et les sommets i et j , et ajoute l'arc (i, j) .

`ajout_arc : graphe_1 -> int -> int -> unit`

CorrectionListing 3 – La fonction `ajout_arc`

```
let ajout_arc (graphe:graphe_1) i j = graphe.(i).(j) <- true;;
```

- La fonction `supprime_arc` prend en argument un graphe et les sommets i et j , et supprime l'arc (i, j) .

`supprime_arc : graphe_1 -> int -> int -> unit`

CorrectionListing 4 – La fonction `supprime_arc`

```
let supprime_arc (graphe:graphe_1) i j = graphe.(i).(j) <- false;;
```

4. Écrire la fonction `supprime_sommet` qui rend inaccessible le sommet i du graphe.

`supprime_sommet : graphe_1 -> int -> unit`

CorrectionListing 5 – La fonction `supprime_sommet`

```
let supprime_sommet (graphe:graphe_1) i =
  let n = taille(graphe) in
  for j = 0 to n-1 do
```

```

    supprime_arc graphe i j ;
    supprime_arc graphe j i ;
done;;

```

5. Écrire la fonction `desorienter` qui transforme un graphe orienté en graphe non orienté.

`desorienter : graphe_1 -> unit`

Correction

Listing 6 – La fonction `desorienter`

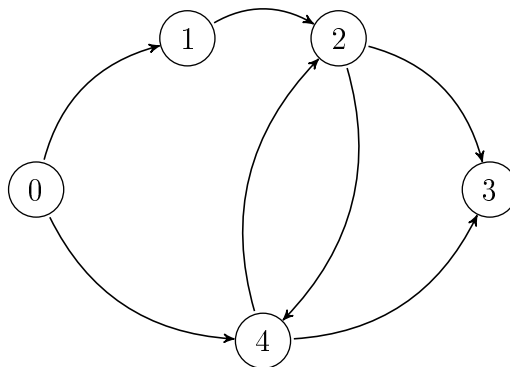
```

let desorienter graphe =
  let n = taille(graphe) in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if graphe.(i).(j) then graphe.(j).(i) <- true
    done;
  done;
;;

```

2 Représentation par listes d'adjacence

Les listes d'adjacence sont représentées par un tableau v de listes d'entiers tel que $v.(s)$ contient la liste des voisins du sommet s .



Nous utiliserons alors le type suivant :

```
type graphe_2 = int list array ;;
```

1. Définir l'exemple avec ce type.

Correction

Listing 7 – L'exemple 2

```

type graphe_2 = int list array ;;

let (exemple_2 : graphe_2) = [| [4; 1]; [2]; [4; 3]; []; [3; 2] |];;

```

2. Écrire deux fonctions `taille` et `nb_arcs` comptant respectivement le nombre de sommets et le nombre d'arcs d'un graphe.

`taille : graphe_2 -> int`

`nb_arc : graphe_2 -> int`

Correction

Listing 8 – Les fonctions `taille` et `nb_arcs`

```
let taille (graphe:graphe_2) = Array.length graphe ;;

let nb_arc (graphe:graphe_2) =
  let nb = ref 0 in
  for i = 0 to taille(graphe) -1 do
    nb := !nb + List.length( graphe.(i))
  done;
  !nb;;
```

3. Écrire les fonctions `ajout_arc` et `supprime_arc` :

- La fonction `ajout_arc` prend en argument un graphe et les sommets i et j , et ajoute l'arc (i, j) .

`ajout_arc : graphe_2 -> int -> int -> unit`

Correction

Listing 9 – La fonction `ajout_arc`

```
let rec appartient liste element =
  match liste with
  | [] -> false
  | a::reste -> (a = element) || (appartient reste element) ;;

let ajout_arc (graphe:graphe_2) i j =
  let liste = graphe.(i) in
  if not (appartient liste j) then graphe.(i)<- j::liste;;
```

- La fonction `supprime_arc` prend en argument un graphe et les sommets i et j , et supprime l'arc (i, j) .

`supprime_arc : graphe_2 -> int -> int -> unit`

Correction

On écrit la fonction `supprime` qui retire un élément d'une liste. On utilise alors cette fonction dans la fonction `supprime_arc`.

Listing 10 – La fonction `supprime_arc`

```
let rec supprime liste i = match liste with
  | [] -> []
```

```

| a::reste when a = i -> reste
| a::reste -> a::(supprime reste i);;

let supprime_arc (graphe:graphe_2) i j =
  let liste = graphe.(i) in
  graphe.(i) <- supprime liste j;;

```

4. Écrire la fonction `supprime_sommet` qui rend inaccessible le sommet i du graphe.

`supprime_sommet : graphe_2 -> int -> unit`

Correction

Listing 11 – La fonction `supprime_sommet`

```

let supprime_sommet (graphe:graphe_2) i =
  let n = taille(graphe) in
  for j = 0 to n-1 do
    supprime_arc graphe i j ;
    supprime_arc graphe j i ;
  done;;

```

5. Écrire la fonction `desorienter` qui transforme un graphe orienté en graphe non orienté.

`desorienter : graphe_2 -> unit`

Correction

Listing 12 – La fonction `desorienter`

```

let desorienter graphe =
  let n = taille(graphe) in
  let rec ajout_arc_liste l i = match l with
    | [] -> ()
    | a::reste ->
      begin ajout_arc graphe a i ;
        ajout_arc_liste reste i
      end in
  for i = 0 to n-1 do
    ajout_arc_liste graphe.(i) i
  done
;;

```

6. Écrire les fonctions `mat_of_liste` et `liste_of_mat` :

- La fonction `mat_of_liste` renvoie la matrice d'adjacence du graphe dont on connaît les listes d'adjacence.

`mat_of_liste : int list array -> bool array array`

Correction

Listing 13 – La fonction `mat_of_liste`

```
let mat_of_liste graphe =  
  let n = Array.length graphe in  
  let tab = Array.make_matrix n n false in  
  let rec lecture_liste l i = match l with  
    | [] -> ()  
    | a::reste ->begin  
      tab.(i).(a) <- true;  
      lecture_liste reste i;  
    end  
  in  
  for i = 0 to n-1 do  
    lecture_liste graphe.(i) i ;  
  done;  
  tab;;
```

- La fonction `liste_of_mat` renvoie les listes d'adjacence du graphe dont on connaît la matrice d'adjacence.

`liste_of_mat : bool array array -> int list array`

Correction

Listing 14 – La fonction `liste_of_mat`

```
let liste_of_mat graphe =  
  let n = Array.length graphe.(0) in  
  let t = Array.make n [] in  
  for i = 0 to n-1 do  
    for j = 0 to n-1 do  
      if graphe.(i).(j) then  
        t.(i) <- j::(t.(i));  
      done;  
    done;  
  done;  
  t;;
```