# TP7 - Graphes (2) (éléments de réponses)

## Généralités

**Question 1.** Les arêtes $\{0_A, 0_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}$ forment un couplage de cardinal 3.
On a également $\{0_A, 1_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}$ ou encore $\{0_A, 0_B\}, \{2_A, 2_B\}, \{1_A, 3_B\}$. Et bien d'autres encore.

**Question 2.** Supposons qu'il existe un couplage de cardinal 4. Tous les sommets de $A$ (et tous ceux de $B$) doivent y participer, pour des raisons de cardinal. Mais $1_A$ et $3_A$ ne peuvent être couplés qu'avec $3_B$, ce qui donne deux arêtes incidentes. C'est contradictoire ; il n'existe donc pas de tel couplage dans ce graphe.

**Question 3.** Compte tenu des hypothèses sur $c$, il reste à vérifier que les arêtes mentionnées existent réellement et qu'il n'y en a pas deux incidentes. Ce dernier point est clair du côté de $A$ mais on doit vérifier l'injectivité du côté de $B$.

```
let verifie g c =
  let n = Array.length g in       (* c'est aussi la taille de c par hypothèse *)
  let i = ref 0 in
  let ok = ref true in            (* jusqu'ici c n'a pas de problème *)
  while !i < n && !ok do
    let j = c.(!i) in
    if j <> -1 then
      if not g.(!i).(j) then
        ok := false               (* arête factice *)
      else begin
        let k = ref 0 in          (* sommet déjà utilisé ? *)
        while (!k < !i) && (c.(!k) <> j) do incr k done;
        ok := (!k = !i)
      end;
    incr i
  done;
  !ok
```

La complexité au pire est en $O(n^2)$ car chaque vérification de non-incidence a un coût linéaire.

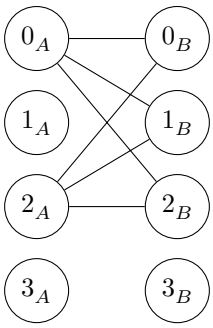
**Question 4.**

```
let cardinal c =
  let nb = ref 0 in
  for i = 0 to Array.length c - 1 do
    if c.(i) <> -1 then incr nb
  done;
  !nb
```

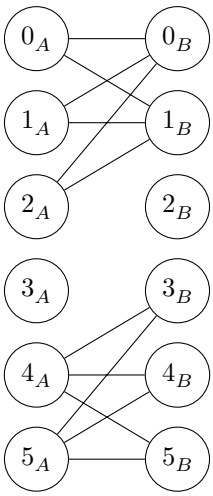
La complexité au pire est linéaire.

## Couplage maximal

**Question 5.** On observe que le minimum pour les sommes de degrés vaut 4, réalisé pour les arêtes $\{1_A, 3_B\}$ et $\{3_A, 3_B\}$. On enlève par exemple $\{1_A, 3_B\}$. Il reste alors le graphe ci-dessous dont on retire n'importe quelle arête (la somme vaut toujours 5), par exemple $\{0_A, 0_B\}$, pour obtenir un graphe sans arête.




**Question 6.** L'arête $\{3_A, 2_B\}$ est la seule de somme 4 et il n'y en a pas de somme moindre. C'est donc l'arête $a_1$. On obtient




qui comporte deux composantes connexes (en excluant les sommets sans arête). Dans chacune, il n'y a pas de couplage de cardinal strictement supérieur à 2 (pas assez de sommets pour cela), donc l'algorithme fournira un couplage de cardinal au plus $1+2+2=5$.
Mais il y a clairement un couplage de cardinal 6 en prenant les $\{i_A, i_B\}$. Finalement le couplage fourni par l'algorithme étudié ne peut pas être de cardinal maximal.

**Question 7.** Il n'est pas raisonnable d'écrire une fonction auxiliaire pour calculer le degré d'un sommet car le calcul dépend de si ce sommet est dans $A$ ou dans $B$.

```
let arete_min g a =
  let n = Array.length g in
  let non_vide = ref false in
  let min_sdegres = ref 0 in  (* cette valeur ne sera jamais lue *)
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if g.(i).(j) then
        let sdegres = ref 0 in
        for k = 0 to n-1 do
          if g.(i).(k) then incr sdegres;
          if g.(k).(j) then incr sdegres
        done;
        if (not !non_vide) || (!sdegres < !min_sdegres) then (
          non_vide := true;
          a.(0) <- i; a.(1) <- j;
          min_sdegres := !sdegres;
        )
    done
  done;
  !non_vide
```

La complexité est clairement cubique.

**Question 8.**

```
let supprimer g (i,j) =
  let n = Array.length g in
  for k = 0 to n - 1 do
    g.(i).(k) <- false;
    g.(k).(j) <- false
  done
```

Complexité linéaire.

**Question 9.** On peut proposer une solution *à la main.*

```
let copier_matrice m =
  let n = Array.length m in
  let p = Array.length m.(0) in
  let mc = Array.make_matrix n p true in
  for i = 0 to n-1 do
    for j = 0 to p-1 do
      mc.(i).(j) <- m.(i).(j)
```

```
    done;
  done;
  mc
```

On peut également utiliser des fonctions de la bibliothèque OCaml comme la fonction `Array.copy` qui copie un tableau unidimensionnel.

```
let copier_matrice m =
  Array.init (Array.length m) (fun i -> Array.copy m.(i))
```

Complexité $O(n^2)$.

**Question 10.**

```
let algo_approche g =
  let g = copier_matrice g in
  let n = Array.length g in
  let c = Array.make n (-1) in
  let a = Array.make 2 0 in
  while arete_min g a do
    c.(a.(0)) <- a.(1);
    supprimer g (a.(0), a.(1))
  done;
  c
```

On a vu que `arete_min` est cubique et que les autres opérations sont au plus quadratiques. Dans le pire cas (tous les sommets sont de degré 1), la suppression n'enlève qu'une seule arête à chaque fois, et on a donc $n$ tours de boucle. L'algorithme est donc en $O(n^4)$.

## Couplage maximum

**Question 11.**

```
let une_arete g =
  let n = Array.length g in
  let i = ref 0 and trouve = ref None in
  while !trouve = None && !i < n do
    let j =  ref 0 in
    while !trouve = None && !j < n do
      if g.(!i).(!j) then trouve := Some((!i,!j));
      incr j
    done;
    incr i
  done;
  !trouve
```

**Question 12.**

```
let rec meilleur_couplage g =
  let n = Array.length g in
  match une_arete g with
    | None -> Array.make n (-1)
    | Some((i,j)) ->
        let gavec = copier_matrice g in
        supprimer gavec (i,j);  (* on fait comme si on avait couplé i avec j *)
        let cavec = meilleur_couplage gavec in
        cavec.(i) <- j;  (* on couple i avec j *)
        let gsans = copier_matrice g in
        gsans.(i).(j) <- false;  (* on supprime (i,j) seulement *)
        let csans = meilleur_couplage gsans in
        if cardinal cavec > cardinal csans then cavec else csans
```