Corrigé **D.M.1**

(1) $C_1 = \{\{0_A, 0_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}\}$ est, par exemple, un couplage de cardinal 3 dans G_0

(2) Il n'existe pas de couplage de cardinal 4 dans G_0 car dans un tel couplage chaque sommet doit appartenir dans une arête exactement donc un tel couplage devrait contenir $\{1_A, 3_B\}$ et $\{3_A, 3_B\}$ ce qui est absurde car ces deux arêtes sont incidentes.

(3) L'idée la plus naïve est de parcourir le tableau C en testant si chacune de ses arêtes appartient bien au graphe et si les sommets en B sont bien deux à deux distincts (càd si les entiers positifs du tableau C sont 2 à 2 distincts)

```
let verifie g c = let n = Array.length cand b = ref true and i = ref 0 in
 while (!i < n) && !b do
  if c.(!i)= -1 then incr i
  else if not(g.(!i).(c.(!i))) then b := false
  else let j = ref 0 in
   while (!j < !i) && (c.(!j)<>c.(!i)) do incr j done;
   b:=(!j = !i);
   incr i
 done;
 !b;;
```

Dans le pire des cas, on fait n passages dans la 1ère boucle while et pour chaque $i \in [\![0, n-1]\!]$, on fait i passages dans la 2e boucle while donc au total, une complexité en $O(n^2)$ (car $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$)

2ème version: On peut éviter la 2e boucle while en utilisant un tableau appelé "Dispo" qui indique les sommets "B" non encore utilisés. Dans ce cas la complexité est en $O(n)$.

```
let verifie2 g c =
 let n = Array.length c and b = ref true and i = ref 0 in
 let dispo = Array.make  n true in
 while (!i < n) && !b do
  if c.(!i)= -1 then incr i
  else if not(g.(!i).(c.(!i))) or not(dispo.(c.(!i))) then b := false
  else begin dispo.c.( !i))<- false ; incr i end
 done;
 !b;;
```

Question (4)

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

(II) (5) On a deux arêtes avec une somme de degrés minimale ($=4$) qui sont $\{1_A, 3_B\}$ et $\{3_A, 3_B\}$: on choisit par exemple $\{1_A, 3_B\}$. A ce stade $C = \{\{1_A, 3_B\}\}$

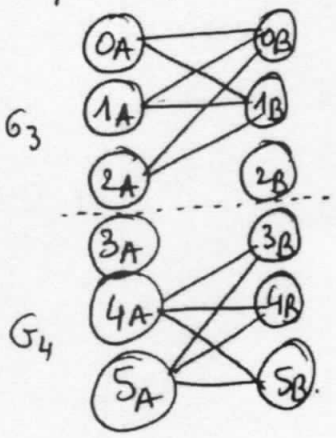
Dans le nouveau graphe on a 6 arêtes avec une somme de degrés minimale ($=5$) on choisit par ex $\{0_A, 0_B\}$. A ce stade $C = \{\{1_A, 3_B\}, \{0_A, 0_B\}\}$

Dans le nouveau graphe il reste 2 arêtes toutes deux avec une somme de degrés $= 3$. On choisit par ex $\{2_A, 1_B\}$. On supprime alors toutes les arêtes restantes et l'algorithme s'arrête et a conduit à $C = \{\{1_A, 3_B\}, \{0_A, 0_B\}, \{2_A, 1_B\}\}$

qui est bien un couplage maximal.

(6) Une seule arête $\{3_A, 2_B\}$ a une somme de degrés égale à 4, toutes les autres ayant une somme de degrés égale à 5 ou 6. On a donc $a_1 = \{3_A, 2_B\}$.

Après suppression des arêtes incidents à a_1 ainsi que de a_1 on a le graphe



le graphe obtenu est la "réunion" de 2 graphes disjoints G_3 et G_4

En appliquant la suite de l'algorithme algo-approché on ne pourra retenir au maximum que 4 arêtes : 2 parmi celles partant de $\{0_A, 4_A, 2_A\}$ (car elles n'aboutissent qu'à 2 sommets B) et 2 parmi celles partant de $\{4_A, 5_A\}$.

Algo-approché conduit donc à un couplage de cardinal au plus 5 alors qu'il existe un couplage de cardinal 6 à savoir $\{\{0_A, 0_B\}, \{1_A, 1_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}, \{4_A, 4_B\}, \{5_A, 5_B\}\}$

(7) On parcourt la matrice décrivant le graphe (coût quadratique)
À chaque arête rencontrée on calcule la somme des degrés (nombre de "true" dans sa ligne + nombre de "true" dans sa colonne).
Si cette somme est < au min précédent on modifie a et on achnalise somme min.
Le coût est en $O(n^3)$

```
let arete_min g a = let n = Array.length g and trouve = ref false in
  let min = ref (2*n + 1) in
    for i = 0 to n-1 do
      for j = 0 to n-1 do.
        if g.(i).(j) then let s = ref 0 in
          for k = 0 to n-1 do if g(i).(k) then s := !s + 1 done;
          for k = 0 to n-1 do if g.(k).(j) then s := !s + 1 done;
          if (!s < !min) then
            begin min := !s ; a.(0) ← i ; a.(1) ← j ; trouve := true end

    done;
  done;
! trouve ;;
```

(8) On met des "false" dans tous les coeff de G ayant un indice commun avec a.
Le coût sera linéaire.

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

(9)

```
let dupliquer g = let n = Array.length g in
  let g1 = Array.make_matrix n n true in
    for i = 0 to n-1 do g1.(i) ← Array.copy g.(i) done;
    g1 ;;

let algo_appode g = let n = Array.length g in
  let g1 = dupliquer g and a = [|-1 ; -1|] and c = Array.make n (-1) in
    while arete_min g1 a do
      supprimer g1 a ;
      c.(a.(0)) ← a.(1)
    done;
    c ;;
```

le coût est en $O(n^4)$

III

(10)
```
let une_arete g a =
  let n = Array.length g and i = ref 0 and j = ref 0 and trouve = ref false in
  while not(!trouve) && !i < n do
    if g.(!i).(!j) then begin trouve := true ; a.(0) <- i ; a.(1) <- j end
    else if !j < n-1 then incr j else begin incr i ; j := 0 end
  done ;
  !trouve ;;
```

(11)
```
let rec meilleur_couplage g =
  let n = Array.length g and a = [|-1;-1|] in
  let c = Array.make n (-1) in
  match une_arete g a with
  | false -> c
  | _ -> let g1 = dupliquer g and g2 = dupliquer g in
         g1.(a.(0)).(a.(1)) <- false ;
         supprime g2 a ;
         let c1 = meilleur_couplage g1 and c2 = meilleur_couplage g2 in
         if (cardinal c1 > (cardinal c2) + 1) then c1
         else begin c2.(a.(0)) <- a.(1) ; c2 end
;;
```