Corrigé : couplage dans un graphe biparti équilibré (mines 2012)

Partie I. Généralités

Question 1. Il existe dans G_0 un couplage de cardinal 3, par exemple $\{0_A, 1_B\}$, $\{1_A, 3_B\}$ et $\{2_A, 0_B\}$.
En revanche, il n'existe pas de couplage de cardinal 4 ; en effet, si un tel couplage existait les sommets 1_A et 3_A seraient tous deux couplés à 3_B et les deux arêtes correspondantes seraient incidentes.

Question 2.

```
let verifie g c =
  let n = vect_length c in
  let dejavu = make_vect n false in
  let rec aux = function
    | i when i = n                              -> true
    | i when c.(i) = -1                         -> aux (i+1)
    | i when g.(i).(c.(i)) && not dejavu.(c.(i)) -> dejavu.(c.(i)) <- true ; aux (i+1)
    | _                                         -> false
  in aux 0 ;;
```

Pour chaque nouveau couplage rencontré il faut vérifier si l'arête correspondante existe dans le graphe et si elle n'est pas incidente à une arête déjà rencontrée. Pour cela on utilise un tableau **dejavu** qui marque les sommets de B dès lors qu'ils sont incidents à un sommet de A.
L'accès aux éléments du tableau **dejavu** est de complexité constante donc le coût de cette fonction est en $O(n)$.

Question 3. Il suffit de dénombrer le nombre de valeurs du tableau c qui ne sont pas égales à -1 :

```
let cardinal c =
  let n = vect_length c in
  let rec aux acc = function
    | i when i = n       -> acc
    | i when c.(i) <> -1 -> aux (acc+1) (i+1)
    | i                  -> aux acc (i+1)
  in aux 0 0 ;;
```

La complexité de cette fonction est bien évidemment un $\Theta(n)$.

Partie II. Un algorithme pour déterminer un couplage maximal

Question 4. Dans le graphe initial toutes les arêtes ont une somme des degrés des extrémités égale à 4, 5 ou 6. Seules deux ont une somme égale à 4 : les arêtes $\{1_A, 3_B\}$ et $\{3_A, 3_B\}$. On choisit par exemple l'arête $\{1_A, 3_B\}$.
Une fois les arêtes incidentes éliminées, il ne reste que des arêtes de somme 5 ; on choisit l'arête $\{0_A, 0_B\}$ qu'on retire ainsi que les arêtes incidentes.
Il ne reste alors que deux arêtes : $\{2_A, 1_B\}$ et $\{2_A, 2_B\}$; on choisit la première et les deux arêtes restantes sont alors éliminées : l'algorithme se termine en renvoyant le couplage $\{\{1_A, 3_B\}, \{0_A, 0_B\}, \{2_A, 1_B\}\}$.

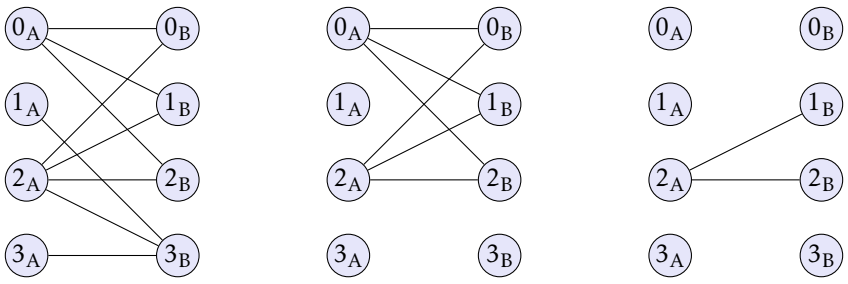
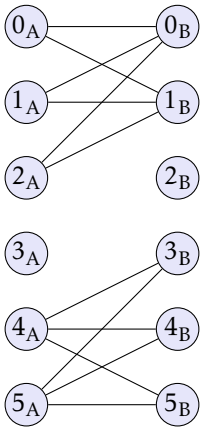


Figure 1 – Les trois étapes d'élimination de l'algorithme.

Question 5. Dans G_1 toutes les arêtes ont une somme des degrés des extrémités égale à 4, 5 ou 6. Une seule a une somme égale à 4 : l'arête $a_1 = \{3_A, 2_B\}$. Une fois celle-ci éliminée ainsi que les arêtes incidentes il reste le graphe :



On constate que le graphe obtenu n'est plus connexe ; chacune de ses deux composantes peut posséder en son sein au plus deux couplages (il est d'ailleurs facile de constater que c'est effectivement le cas) donc l'algorithme retournera un couplage de cardinal inférieur ou égal à 5. Or il existe un couplage de cardinal 6 : $\{\{0_A, 0_B\}, \{1_A, 1_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}, \{4_A, 4_B\}, \{5_A, 5_B\}\}$; l'algorithme ne garantit donc pas d'obtenir un couplage de cardinal maximal.

Question 6. On commence par rédiger une fonction qui calcule les degrés des sommets de A et des sommets de B :

```
let calcule_degre g =
  let n = vect_length g in
  let degA = make_vect n 0 in
  let degB = make_vect n 0 in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if g.(i).(j) then (degA.(i) <- degA.(i) + 1 ; degB.(j) <- degB.(j) + 1)
    done
  done ;
  degA, degB ;;
```

Cette première fonction est bien évidemment de complexité quadratique.
Il reste ensuite à partir à la recherche de l'arête minimale :

```
let arete_min g a =
  let n = vect_length g in
  let degA, degB = calcule_degre g in
  let s = ref (2 * n + 1) in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if g.(i).(j) && degA.(i) + degB.(j) < !s then
            (s := degA.(i) + degB.(j) ; a.(0) <- i ; a.(1) <- j)
    done
  done ;
  !s <= 2 * n ;;
```

On notera que la somme des degrés des extrémités d'une arête ne peut excéder $2n$, ce qui explique la valeur initiale de la référence `s`.
Cette fonction est de complexité quadratique $O(n^2)$.

Question 7. La fonction demandée fait passer à `false` toutes les valeurs de la ligne d'indice i et de la colonne d'indice j :

```
let supprimer g a =
  let n = vect_length g in
  let i = a.(0) and j = a.(1) in
  for k = 0 to n-1 do
    g.(i).(k) <- false ; g.(k).(j) <- false
  done ;;
```

Cette fonction est de coût linéaire.

Question 8.

Remarque. Sachant que la fonction **copy_vect** réalise la copie d'un vecteur on peut dupliquer une matrice en utilisant la fonction suivante :

```
let copy_matrix m = map_vect copy_vect m ;;
```

On définit la fonction principale de l'algorithme *algo_approche* :

```
let algo_approche g =
  let gg = copy_matrix g in
  let n = vect_length g in
  let c = make_vect n (-1) in
  let a = [| 0; 0 |] in
  while arete_min gg a do
    c.(a.(0)) <- a.(1) ;
    supprimer gg a
  done ;
  c ;;
```

Sachant que le coût de la fonction **arete_min** est en $O(n^2)$ et qu'un couplage maximal comporte au maximun n couplages, le coût total de cet algorithme est en $O(n^3)$ (le coût de la création de **gg** est un $\Theta(n^2)$ et celui de **c** un $\Theta(n)$).

Partie III. Recherche exhaustive d'un couplage de cardinal maximum

Question 9. On utilise le mécanisme de rattrapage d'une exception pour cesser la recherche une fois trouvée une arête.

```
exception Trouve ;;

let une_arete g a =
  let n = vect_length g in
  try
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if g.(i).(j) then (a.(0) <- i ; a.(j) <- j ; raise Trouve)
    done
  done ;
  false
  with Trouve -> true ;;
```

Question 10. Lorsque le graphe G contient au moins une arête a, on réalise deux copies G_1 et G_2 de G. Dans la première on supprime l'arête a et dans la seconde l'arête a ainsi que toutes les arêtes incidentes.
Tout couplage C_1 de G_1 est un couplage de G ne contenant pas l'arête a ; tout couplage C_2 de G_2 à qui on ajoute a est un couplage de G contenant l'arête a. Ceci conduit à l'algorithme suivant :

```
let rec meilleur_couplage g =
  let n = vect_length g in
  let a = [| 0; 0 |] in
  if une_arete g a then
    begin
      let g1 = copy_matrix g in
      g1.(a.(0)).(a.(1)) <- false ;
      let g2 = copy_matrix g in
      supprimer g2 a ;
      let c1 = meilleur_couplage g1 and c2 = meilleur_couplage g2 in
      c2.(a.(0)) <- a.(1) ;
      if cardinal c1 < cardinal c2 then c2 else c1
    end
  else make_vect n (-1) ;;
```

Partie IV. L'algorithme hongrois

Question 11. Le seul sommet non couplé de A est le sommet 2_A ; il doit être au départ de toute chaîne alternée relativement à C_1. Le seul sommet non couplé de B est le sommet 4_B ; il doit être à l'arrivée de toute chaîne alternée augmentante relativement à C_1.
Un tel exemple de chaîne alternée augmentante est par exemple $2_A, 2_B, 3_A, 3_B, 4_A, 4_B$.

Question 12. Considérons une chaîne alternée augmentante $(x_0, x_1, \dots, x_{2p+1})$ relativement à un couplage C dans G et considérons l'ensemble d'arêtes C′ constitué :

- des arêtes de C qui ne sont pas utilisées dans la chaîne ;
- des arêtes $\{x_{2k}, x_{2k+1}\}$ pour $k \in [\![0, p]\!]$.

Autrement dit on remplace les p arêtes $\{x_{2k+1}, x_{2k+2}\}$ de C par les $p+1$ arêtes $\{x_{2k}, x_{2k+1}\}$. On a donc $\operatorname{card} C' = \operatorname{card} C + 1$.
Les arêtes restantes dans C ne font pas intervenir les sommets $x_1, \dots, x_{2p}$; elles ne font pas non plus intervenir les sommets x_0 et x_{2p+1} puisque ces sommets ne sont pas couplés dans C. Ainsi, C′ constitue un nouveau couplage de cardinal $n+1$.
Dans l'exemple précédent cela revient à considérer le nouveau couplage $\{\{0_A, 0_B\}, \{1_A, 1_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}, \{4_A, 4_B\}, \{5_A, 5_B\}\}$ qui est de cardinal 6.

Ceci montre donc que si un couplage est de cardinal maximum il ne peut exister de chaîne alternée augmentante. La réciproque de ceci est admise par l'énoncé.

Question 13. La marque permet de reconstituer la chaîne en partant de la fin ; dans le cas du sommet S_B cela donne : $3_B \leftarrow 3_A \leftarrow 2_B \leftarrow 2_A \leftarrow 1_B \leftarrow 0_A \leftarrow 0_B \leftarrow 1_A$; la chaîne alternée arrivant au sommet 3_A est donc : $1_A, 0_B, 0_A, 1_B, 2_A, 2_B, 3_A, 3_B$ et c'est bien une chaîne augmentante.

Question 14. On parcourt la chaîne à rebours en ajoutant l'arête au couplage lorsqu'on passe de B vers A.

```
let rec actualiser c r mA mB = function
  | -1 -> ()
  | i  -> r.(i) <- mB.(i) ; c.(mB.(i)) <- i ;
          actualiser c r mA mB mA.(mB.(i)) ;;
```

Question 15. Deux chaînes sont possibles, toutes deux partent de 1_A : $1_A, 3_B, 3_A, 4_B, 4_A$ et $1_A, 4_B, 4_A, 3_B, 3_A$. Aucune des deux n'est augmentante.

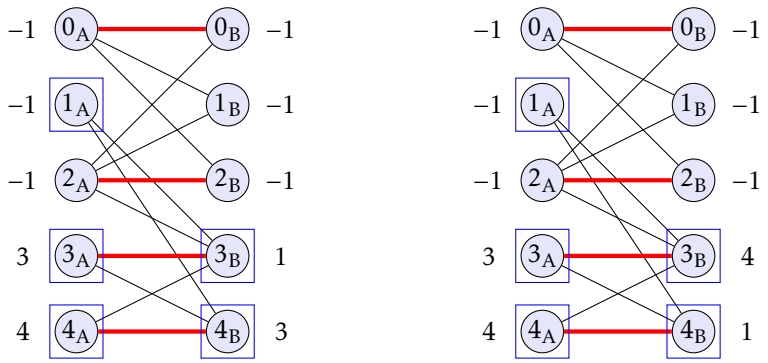


Figure 2 – Les deux marquages possibles.

Question 16. Deux fonctions sont dites *mutuellement récursives* lorsque chacune d'elles intervient dans la définition de l'autre ; elles se définissent à l'aide de la syntaxe `let rec f = ... and g = ... ;;`.

```
let rec chercheA g c r mA mB x =
  let rec aux = function
    | y when y = vect_length g -> -1
    | y when g.(x).(y) && c.(x) <> y && mB.(y) = -1 -> mB.(y) <- x ;
      let s = chercheB g c r mA mB y in if s = -1 then aux (y+1) else s
    | y                                              -> aux (y+1)
  in aux 0
and chercheB g c r mA mB y = match r.(y) with
  | -1                  -> y
  | x  when mA.(x) = -1 -> mA.(x) <- y ; chercheA g c r mA mB x
  | _                   -> -1 ;;
```

La fonction **chercheA** parcourt les différents voisins $y \in B$ de x. Si on rencontre un sommet y non couplé avec x et non marqué ce dernier est marqué par x et la recherche se poursuit à partir de y. Si cette recherche est fructueuse et renvoie un sommet s non couplé de B, ce dernier est renvoyé : une chaîne alternée augmentante a été trouvée. Dans le cas contraire la recherche se poursuit parmi les voisins de x.
La fonction **chercheB** est plus simple : si y n'est pas couplé la recherche se termine en renvoyant y ; si y est couplé à un sommet non marqué x la recherche se poursuit à partir de x ; dans les autres cas la recherche échoue.

Question 17. On parcourt les sommets de A à la recherche de sommets non couplés. Lorsqu'on en rencontre on cherche à l'aide de la fonction **chercheA** à créer une chaîne alternée augmentante.

```
let rec chaine_alternee g c r mA mB =
  let rec aux = function
    | x when x = vect_length g -> -1
    | x when c.(x) = -1        -> let y = chercheA g c r mA mB x in
                                  if y = -1 then aux (x+1) else y
    | x                        -> aux (x+1)
  in aux 0 ;;
```

Question 18.

```
let algorithme_hongrois g =
  let n = vect_length g in
  let c = make_vect n (-1) and r = make_vect n (-1) in
  let rec aux () =
    let mA = make_vect n (-1) and mB = make_vect n (-1) in
    match chaine_alternee g c r mA mB with
      | -1 -> c
      | y  -> actualiser c r mA mB y ; aux ()
  in aux () ;;
```