Graphes bipartis et couplage

OPTION INFORMATIQUE - TP n° 3.4 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ reconnaître et caractériser un graphe biparti
- ☞ utiliser le parcours en largeur d'un graphe pour caractériser un graphe biparti
- ☞ établir un chemin alternant
- ☞ trouver un couplage de cardinal maximum

A Graphes bipartis ou bi-colorables

■ **Définition 1 — Graphe biparti.** un graphe $G = (V, E)$ est biparti si l'ensemble V de ses sommets peut être divisé en deux sous-ensembles disjoints U et W tels que chaque arête de E possède une extrémité dans U et l'autre dans W.

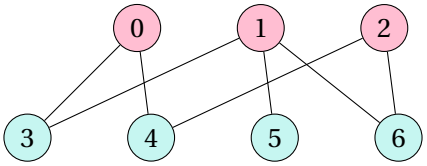


FIGURE 1 – Graphe biparti

Théorème 1 — Caractérisation des graphes bipartis. Un graphe est biparti si et seulement s'il ne possède aucun cycle de longueur impair.

Pour savoir si un graphe est biparti on pourrait donc rechercher les cycles et évaluer leurs longueurs. Il existe cependant une autre approche. Un graphe biparti est un graphe bicolorable comme le montre la figure 1.

B Coloration et bi-coloration de graphes

■ **Définition 2 — Coloration.** Une coloration d'un graphe simple est l'attribution d'une couleur aux sommets de ce graphe.

■ **Définition 3 — Coloration valide.** Une coloration est valide lorsque deux sommets adjacents n'ont jamais la même couleur.

■ **Définition 4 — Nombre chromatique.** Le nombre chromatique d'un graphe G est le plus petit nombre de couleurs nécessaires pour obtenir une coloration valide de ce graphe. On le note généralement $\chi(G)$.

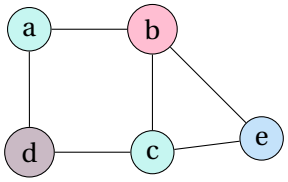


FIGURE 2 – Exemple de 4-coloration valide d'un graphe. Cette coloration n'est pas optimale.

■ **Définition 5 — k-coloration.** Lorsqu'une coloration de graphe utilise k couleurs, on dit d'elle que c'est une k-coloration.

■ **Définition 6 — Coloration optimale.** Une $\chi(G)$-coloration valide est une coloration optimale d'un graphe G.

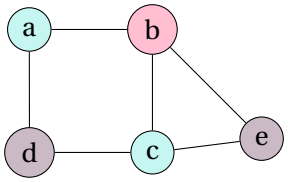


FIGURE 3 – Exemple de 3-coloration valideă d'un graphe. Cette coloration est optimale.

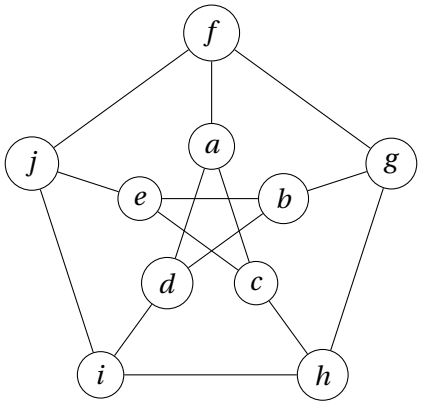


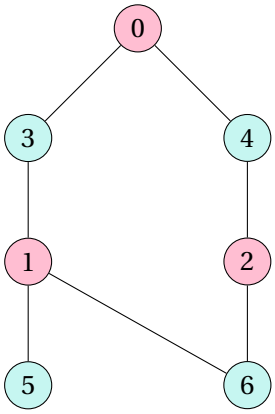
FIGURE 4 – Graphe de Petersen : saurez-vous proposer une coloration optimale de ce graphe sachant que son nombre chromatique vaut trois ?

C Algorithme de test de la bi-colorabilité d'un graphe

Pour tester la bi-colorabilité d'un graphe, il suffit de parcourir le graphe en largeur et de colorer les sommets d'un même niveau de la même couleur. L'algorithme s'arrête si deux sommets adjacents possèdent la même couleur.

C1. Effectuer à la main un parcours en largeur du graphe de la figure 1 en partant du sommet 0. En déduire un dessin du graphe faisant apparaître quatre niveaux. Ce graphe est-il bi-colorable ?

Solution : Le parcours en largeur à partir de 0 donne `[0; 3; 4; 1; 2; 5; 6]`. On peut donc redessiner le graphe comme suit pour faire apparaître quatre niveaux :



Ce graphe est clairement bi-colorable !

C2. Créer un type somme `color_type` sont les valeurs sont `White`, `Black` ou `Orange`.

Solution :

```
type color_type = White | Black | Orange;;
```

C3. Créer trois variables globales `cw`, `cb` et `co` de type `color_type` dont les valeurs sont respectivement `White`, `Black` et `Orange`.

Solution :

```
let co = Orange;;
let cb = Black;;
let cw = White;;
```

C4. Créer une fonction de signature `color_to_string : color_type -> string` qui transforme un type `color_type` en chaîne de caractère en utilisant le pattern matching. Par exemple, `White` devient `"White"`.

Solution :

```
let color_to_string color =  match color with
  | White -> "White"
  | Black -> "Black"
  | Orange -> "Orange";;
```

C5. Créer une fonction de signature `next_color : color_type -> color_type` qui renvoie la couleur du prochain niveau en utilisant le pattern matching. Si le paramètre d'entrée est `Black` elle renvoie `Orange` et inversement. Si le paramètre d'entrée est `White`, elle échoue en renvoyant le message `"Can not color in white !"` : on ne peut pas colorer en blanc dans l'algorithme, c'est juste la couleur initiale des sommets non explorés.

Solution :

```
let next_color c = match c with
    | Black -> Orange
    | Orange -> Black
    | White -> failwith "Can not color in white !";;
```

C6. Créer un exception `Not2colorable`.

Solution :

```
exception Not2colorable;;
```

C7. Créer une fonction récursive de signature `color_neighbours : 'a list -> 'b -> ('a * 'b)list` qui prend comme paramètre une liste de sommets et une couleur et qui renvoie la liste des tuples de chaque sommet adjoint à la couleur. Par exemple, avec les paramètres `[1;2;4]` et `Blue`, la fonction renvoie `[(1,Blue);(2,Blue);(4,Blue)]`.

Solution :

```
let rec color_neighbours neighbours c = match neighbours with
    | [] -> []
    | v::t -> (v,c)::(color_neighbours t c);;
```

C8. Coder un algorithme de parcours en largeur afin de statuer sur la bi-colorabilité d'un graphe. Cette implémentation récursive renvoie `false` si l'exception `Not2colorable` est levée. Elle renvoie `true` sinon. On remarquera qu'on peut parcourir en largeur un graphe sans pour autant renvoyer la liste des sommets parcourus : dans le cas présent, on n'en a pas besoin. Enfin, la fonction `color_neighbours` est utile pour construire une file de voisins colorés à l'identique. La fonction `next_color` est utile pour choisir la couleur du niveau suivant. La fonction `color_to_string` est utile pour afficher sur la console l'évolution de l'algorithme. Suivre le squelette de code suivant :

```
let bicolorable g v0 =
  let color = Array.make (Array.length g) White in
```

```
    let rec explore queue = (* queue -> file en anglais FIFO *)
      match queue with
        | ... (* End case, we don't need the path list so return unit () *)
        | ... (* color vertex and neighbours and keep on exploring *)
        | ... (* already colored OK, exploring remaining vertices *)
        | ... (* stop exploring, raise execption Not2colorable  *)
  in try explore [(v0,Black)]; true with (* return true * )
         | Not2colorable -> false;; (* catching exception, return false *)
```

C9. Tester cette implémentation sur le graphe de la figure 1 `let bg = [| [3;4] ; [3;5;6] ; [4;6] ; [0;1] ; [0;2] ; [1] ; [1;2] |]`.

C10. Tester l'implémentation sur le graphe `let bg_mod = [| [2;3;4] ; [3;5;6] ; [4;6;0] ; [0;1] ; [0;2] ; [1] ; [1;2] |]`.

Solution :

```
let bg = [| [3;4] ; [3;5;6] ; [4;6] ; [0;1] ; [0;2] ; [1] ; [1;2] |] ;;
let bg_mod = [| [2;3;4] ; [3;5;6] ; [4;6;0] ; [0;1] ; [0;2] ; [1] ; [1;2] |]
     ;;

let bicolorable g v0 =
  let color = Array.make (Array.length g) White in
  let rec explore queue = (* queue -> file en anglais FIFO *)
    match queue with
      | [] -> ()  (* we dont need he path list --> unit () *)
      | (v,c)::t when color.(v) = White ->
        color.(v) <- c;
        Printf.printf "%i from white to %s\n" v (color_to_string c);
        explore (t @ (color_neighbours g.(v) (next_color c)))
      | (v,c)::t when color.(v) = c ->
        Printf.printf "Vertex %i will remain %s\n" v (color_to_string c);
        explore t    (* keep on exploring *)
      | (_,_)::_ -> raise Not2colorable  (* stop exploring *)
  in try explore [(v0,Blue)]; true with
      |  Not2colorable -> false;;

bicolorable bg 0 ;;
bicolorable bg_mod 0 ;;
```

D Couplage dans un graphe biparti

■ **Définition 7 — Couplage.** Un couplage Γ dans un graphe non orienté $G = (V, E)$ est un ensemble d'arêtes deux à deux non adjacentes. Formellement :

$$\forall (e_1, e_2) \in E^2, e_1 \neq e_2 \Longrightarrow e_1 \cap e_2 = \emptyset \tag{1}$$

c'est à dire que les sommets de e_1 et e_2 ne sont pas les mêmes.

■ **Définition 8 — Sommets couplés, sommets exposés.** Un sommet est couplé s'il fait parti d'une arête de Γ. Un sommet est exposé s'il ne fait pas parti des arêtes de Γ, c'est à dire il n'est pas couplé.

■ **Définition 9 — Couplage maximal.** Un couplage maximal est tel que si on lui ajoute une arête, ce n'est plus un couplage. Il est donc maximal au sens de l'inclusion.

■ **Définition 10 — Couplage de cardinal maximum.** Un couplage de cardinal maximum est un couplage contenant le plus grand nombre d'arêtes possible.

■ **Exemple 1 — Affectation des cadeaux sous le sapin .** Au pied du sapin de Noël, un papa a disposé six cadeaux dont les paquets sont tous différents et numérotés de 0 à 5 [a]. Il a décidé que les cadeaux seraient répartis en fonction des paquets que les enfants préfèrent.

Ses cinq enfants expriment donc leurs préférences. Le papa pourra-t-il affecter un cadeau à chaque enfant et faire en sorte que ce cadeau soit un de leurs préférés ?

Évidemment la réponse à cette question dépend des préférences émises par les enfants. Supposons qu'ils se soient exprimés ainsi :

Alix 0,2

Brieuc 1,3,4,5

Céline 1,2

Dimitri 0,1,2

Enora 2

On peut représenter par un graphe biparti cette situation comme sur la figure 5. Dans ce cas précis, comme il y a quatre enfants qui ne veulent que trois les trois premiers cadeaux, il n'y a pas de solution. Mais si Enora avait choisit 4 et 5 ???

a. Ce papa est informaticien !

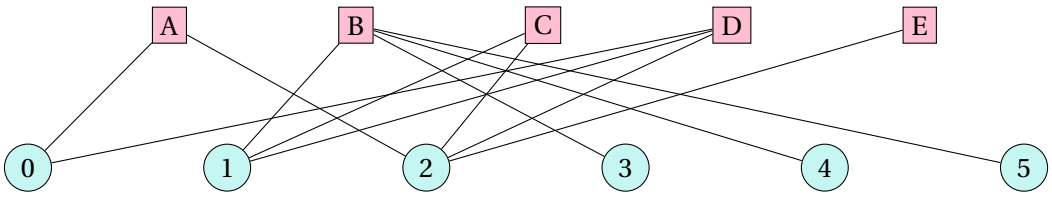


FIGURE 5 – Exemple de graphe biparti pour un problème d'affectation sans solution.

R Si les arêtes du graphe sont valuées (graphe pondéré), alors la recherche d'un couplage de cardinal maximum et de poids minimum dans un graphe biparti est en fait le problème de l'**affectation des ressources**. Ceci n'est pas au programme.

E Chemin augmentant

Pour résoudre le problème de trouver un couplage de cardinal maximum, on peut procéder en suivant l'algorithme 1. Il s'agit de construire un chemin augmentant pour atteindre un couplage de cardinal maximum.

■ **Définition 11 — Chemin alternant.** Une chemin alternant dans un graphe non orienté G et pour un couplage Γ est tel que les arêtes appartiennent successivement à Γ et $E \setminus \Gamma$.

■ **Définition 12 — Chemin augmentant.** Un chemin augmentant est un chemin alternant dont les extrémités sont des sommets exposés, c'est à dire qui n'appartiennent pas au couplage Γ.

La stratégie de l'algorithme de recherche d'un couplage de cardinal maximum est la suivante : à partir d'un couplage Γ, on construit un nouveau couplage de cardinal supérieur à l'aide d'un chemin augmentant comme le montre la figure 6.

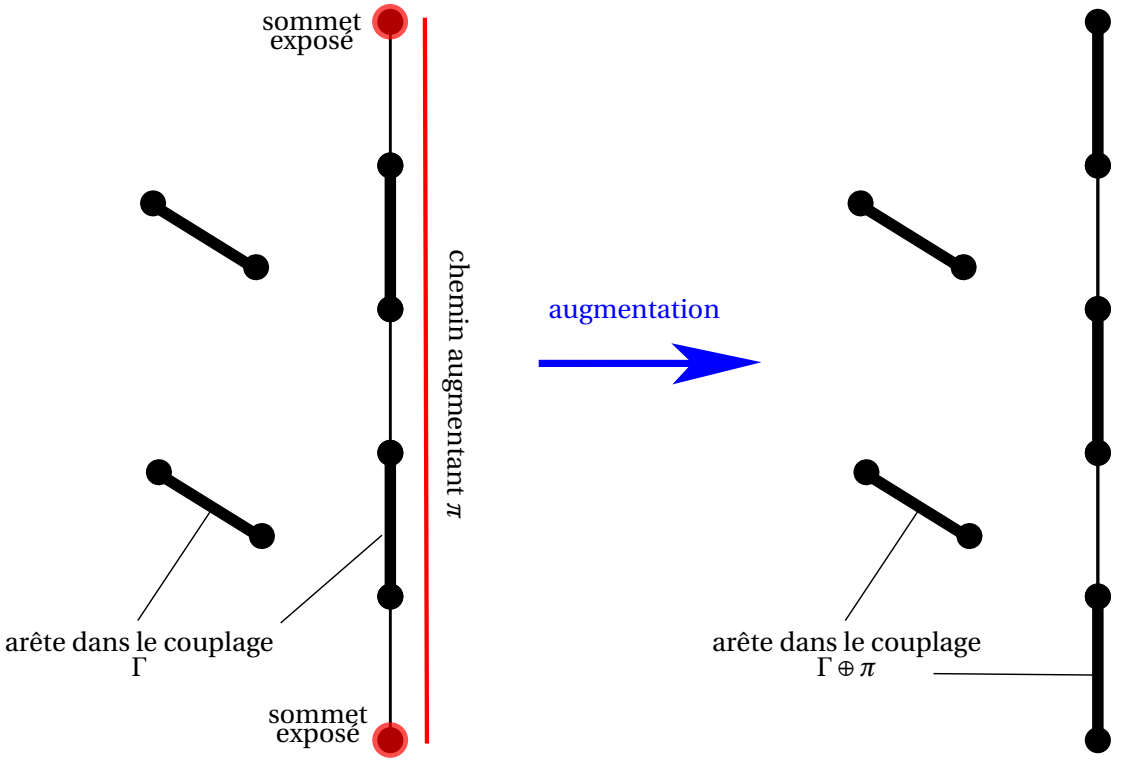


FIGURE 6 – Illustration de la construction d'un couplage de cardinal supérieur grâce à un chemin augmentant. (Source originale : Wikimedia Commons)

Dans un graphe **biparti**, il est facile d'augmenter la taille d'un couplage jusqu'au cardinal maximum :

1. s'il existe deux sommets exposés reliés par une arête, il suffit d'ajouter cette arête au couplage. Puis, on appelle récursivement l'algorithme sur ce nouveau couplage.
2. sinon il faut trouver un chemin augmentant π dans le graphe.
   (a) s'il n'y en a pas, l'algorithme est terminé.
   (b) sinon on effectue la différence symétrique entre le couplage Γ et l'ensemble des arêtes du

chemin augmentant π pour obtenir le nouveau couplage : $\Gamma \oplus \pi = \{e \in E, e \in \Gamma \setminus \pi \cup e \in \pi \setminus \Gamma\}$. Puis, on appelle récursivement l'algorithme avec ce nouveau couplage.

Il faut noter que le cardinal du couplage n'augmente pas nécessairement lorsqu'on effectue la différence symétrique mais il ne diminue pas.

Pour trouver un chemin augmentant dans un graphe biparti $G = ((U, D), E)$, on cherche le plus court chemin entre deux sommets exposés dans un graphe orienté auxiliaire G_o construit de la manière suivante :

1. toutes les arêtes de E qui n'appartiennent pas au couplage Γ sont orientés de U vers D.
2. toutes les arêtes de Γ sont orientées de D vers U.

Le plus court chemin entre deux sommets exposés de G_o est nécessairement un chemin augmentant, son caractère alternant vient du fait que le graphe est biparti.

La figure 7 illustre les différentes étapes de l'algorithme. On vérifie bien que le résultat est correct : chaque enfant aura bien un cadeau qu'il avait mis sur sa liste.

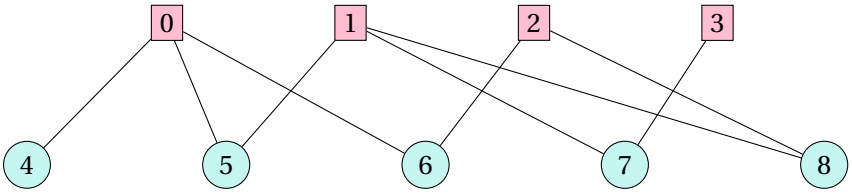
Algorithme 1 Recherche d'un couplage de cardinal maximum

Entrée : un graphe biparti $G = ((U, D), E)$
Entrée : un couplage Γ initialement vide
Entrée : F_U, l'ensemble de sommets exposés de U initialement U
Entrée : F_D, l'ensemble de sommets exposés de D initialement D

```
1 : Fonction CHEMIN_AUGMENTANT(G = (V = (U, D), E), Γ, F_U, F_D))      ▷ M est le couplage, vide initialement
2 :     si une arête e = (u, v) entre un sommet de F_U et un sommet de F_D existe alors
3 :         CHEMIN_AUGMENTANT(G, Γ ∪ {e}, F_U \ {u}, F_D \ {v})
4 :     sinon
5 :         Créer le graphe orienté G_o                    ▷ ∀e ∈ E, e de U vers D si e ∉ Γ, l'inverse sinon
6 :         Calculer le plus court chemin π entre un sommet de F_U et un de F_D dans G_o
7 :         si un tel chemin π n'existe pas alors
8 :             renvoyer M
9 :         sinon
10 :            CHEMIN_AUGMENTANT(G, Γ ⊕ π, F_U \ {π_start}, F_D \ {π_end})
11 :                ▷ π_start début du chemin π, π_end fin du chemin π et Γ ⊕ π = {e ∈ E, e ∈ Γ \ π ∪ e ∈ π \ Γ}
```

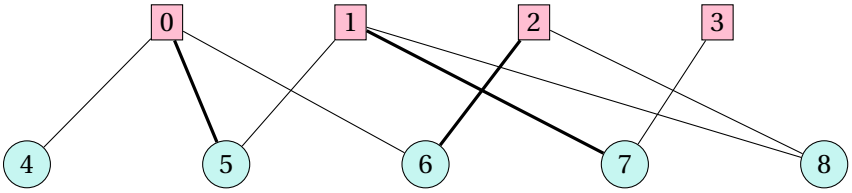
E1. L'implémentation de cet algorithme est un excellent exercice de révision ! Plus un point au DS à celle ou celui qui me produit un code fonctionnel et clair.

Le graphe de départ de l'algorithme est le suivant :



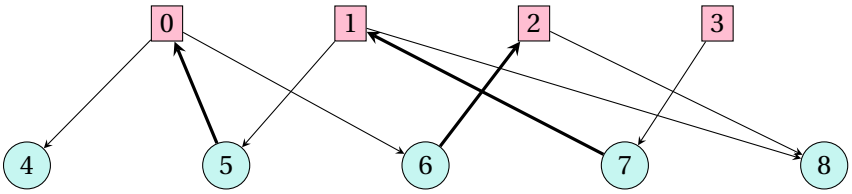
$\Gamma = \emptyset$

On effectue **trois appels récursifs** et, à chaque fois, on a trouvé une arête dont les sommets sont tous les deux exposés.



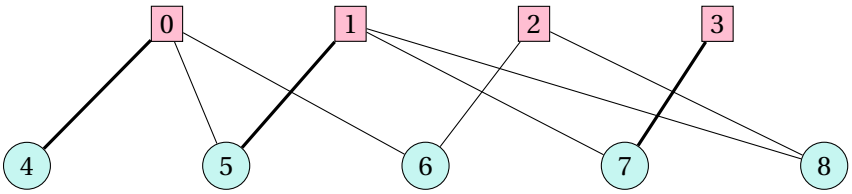
$\Gamma = \{(0,5),(1,7),(2,6)\}$

À ce stade de l'algorithme, aucun sommet exposé n'est relié par une arête à un autre sommet exposé. Donc, on construit le graphe G_o d'après le couplage Γ. On trouve le chemin le plus court entre les deux premiers sommets exposés 3 et 4 : π.



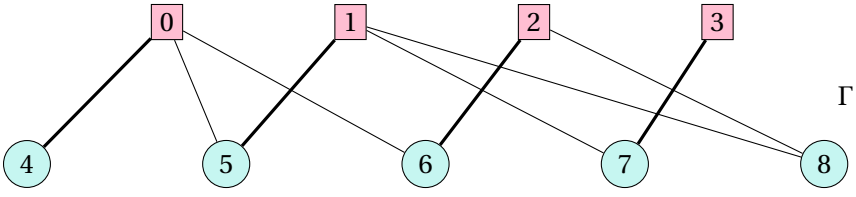
$\Gamma = \{(0,5),(1,7),(2,6)\}$
$\pi = \{3,7,1,5,0,4\}$

On en déduit un nouveau couplage $\Gamma = \Gamma \oplus \pi$:



$\Gamma = \{(0,4),(1,5),(3,7)\}$

On effectue **un appel récursif** et on trouve une arête dont les sommets sont tous les deux exposés : $(2,6)$. On effectue un dernier appel récursif et l'algorithme se termine car un seul sommet est non couplé.



$\Gamma = \{(0,4),(1,5),(3,7),(2,6)\}$

FIGURE 7 – Étapes de l'algorithme de recherche d'un couplage de cardinal maximum