

GRAPHES AVANCÉS

À la fin de ce chapitre, je sais :

- expliquer le concept d'arbre recouvrant
- expliquer l'intérêt d'un tri topologique
- schématiser le concept de forte connexité
- expliquer ce qu'est un arbre recouvrant
- expliquer l'intérêt d'un graphe biparti

A Rappels sur les graphes

■ **Définition 1 — Graphe.** Un graphe G est un couple $G = (S, A)$ où S est un ensemble fini et non vide d'éléments appelés **sommets** et A un ensemble de paires d'éléments de S appelées **arêtes**.

■ **Définition 2 — Sous-graphe.** Soit $G = (S, A)$ un graphe, alors $G' = \{S', A'\}$ est un sous-graphe de G si et seulement si $S' \subseteq S$ et $A' \subseteq A$.

■ **Définition 3 — Sous-graphe couvrant.** G' est un sous-graphe couvrant de G si et seulement si G' est un sous-graphe de G et $S' = S$.

■ **Définition 4 — Graphe connexe.** Un graphe $G = (S, A)$ est connexe si et seulement si pour tout couple de sommets (a, b) de G , il existe une chaîne d'extrémités a et b .

à

Théorème 1 — Condition nécessaire d'acyclicité d'un graphe. Soit un graphe $G = (S, A)$ possédant au moins une arête et acyclique alors G possède au moins deux sommets de degré un et on a :

$$|A| \leq |S| - 1 \quad (1)$$

Théorème 2 — Condition nécessaire de connexité d'un graphe. Si un graphe $G = (S, A)$ est connexe alors on a :

$$|A| \geq |S| - 1 \quad (2)$$

■ **Exemple 1 — Exercices types.** Soit $G = (S, A)$ un graphe non orienté dont l'ordre est n .

1. Montrer que si le G est connexe alors il comporte au moins $n - 1$ arêtes. (Procéder par récurrence)
2. Montrer que si G a tous les sommets de degré supérieur ou égal à 2 alors il possède un cycle. En déduire, qu'un graphe acyclique admet un sommet de degré 0 ou 1.
3. Montrer que si G est acyclique alors il possède au plus $n - 1$ arêtes. (Procéder par récurrence)

B Arbres recouvrants

■ **Définition 5 — Arbre.** Un arbre est un graphe non orienté connexe et acyclique.

(R) On déduit des deux théorèmes précédents qu'un arbre possède **exactement** $|S| - 1$ arêtes.

Théorème 3 — Caractérisation des arbres. Soit G un graphe non orienté à n sommets. Les propositions suivantes sont équivalentes :

- G est un arbre
- G est connexe et acyclique
- G est connexe et possède $n - 1$ arêtes
- G est connexe et la suppression de n'importe quelle arête le rend non connexe
- G est acyclique et possède $n - 1$ arêtes
- G est acyclique et l'ajout de n'importe quelle arête crée un cycle
- entre toute paire de sommets de G il existe une unique chaîne dont les extrémités sont ces sommets

■ **Définition 6 — Forêt.** Une forêt est un graphe non orienté sans cycle.

■ **Définition 7 — Arbre recouvrant un graphe.** Dans un graphe non orienté et connexe, un arbre est dit recouvrant s'il est inclus dans ce graphe et s'il connecte tous les sommets de ce graphe.

R On peut dire de manière équivalente qu'un arbre recouvrant est :

- un sous-graphe acyclique maximal,
- un sous-graphe recouvrant connexe minimal.

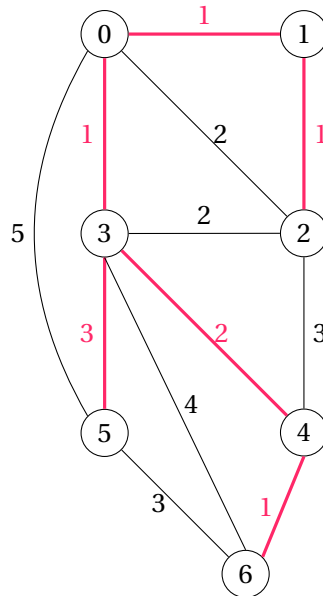


FIGURE 1 – Exemple d'arbre recouvrant dans un graphe non orienté et pondéré. En rouge, un arbre recouvrant minimal de poids total 9.

Les arbres recouvrants sont des éléments essentiels de monde réel. La figure 1 peut par exemple représenter un réseau de villes qu'on souhaite interconnecter par un chemin de fer. Si le coût de construction est proportionnel au poids porté par l'arête, comment choisir les arêtes pour minimiser le coût de la construction du réseau de voies ferrées tout en interconnectant toutes les villes? Les arêtes en rouges, qui forment un arbre recouvrant de poids minimal, permettent de répondre à cette question.

Les arbres recouvrants sont donc utilisés dans les domaines des réseaux d'énergie, des télécommunications, des réseaux de fluides mais également en intelligence artificielle et en électronique. Les deux algorithmes phares pour construire des arbres recouvrants sont l'algorithme de Kruskal [**kruskal_shortest_1956**] et l'algorithme de Prim [**prim_shortest_1957**].

R Si les poids du graphe sont tous différents, alors il existe un unique arbre recouvrant de poids minimal.

a Algorithme de Prim

L'algorithme de Prim est un algorithme **glouton optimal** qui s'applique aux **graphes pondérés connexes**. Pour construire l'arbre, l'algorithme part d'un sommet et fait croître l'arbre en choisissant un sommet dont la distance est la plus faible et n'appartenant pas à l'arbre, garantissant ainsi l'absence de cycle.

Algorithme 1 Algorithme de Prim, arbre recouvrant

```

1: Fonction PRIM( $G = (V, E, w)$ )
2:    $T \leftarrow \emptyset$  ▷ la sortie : l'ensemble des arêtes de l'arbre recouvrant
3:    $S \leftarrow s$  un sommet quelconque de  $V$ 
4:   tant que  $S \neq V$  répéter
5:      $(u, v) \leftarrow \min (w(u, v), u \in S, v \in E)$  ▷ Choix glouton!
6:      $S \leftarrow S \cup \{v\}$ 
7:      $T \leftarrow T \cup \{(u, v)\}$ 
8:   renvoyer  $T$ 

```

La complexité de cet algorithme, si l'on utilise un tas min pour le file et une liste d'adjacence pour représenter G , est en $O((n + m) \log n)$ si $m = |E|$ est le nombre d'arêtes du graphe et n l'ordre du graphe.

(R) Doit-on utiliser Dijkstra ou Prim? En pratique, l'algorithme de Dijkstra est utilisé lorsque l'on souhaite économiser du temps et du carburant pour se déplacer d'un point à un autre. L'algorithme de Prim, quant à lui, est utilisé lorsque l'on souhaite minimiser les coûts de matériaux lors de la construction de routes reliant plusieurs points entre eux.

Les algorithmes de Prim et de Dijkstra présentent trois différences majeures :

- Dijkstra trouve le chemin le plus court, mais l'algorithme de Prim trouve le l'arbre recouvrant minimal.
- Dijkstra s'applique sur les graphes orientés et non orientés mais Prim ne s'applique qu'à des graphes non orientés.
- Prim peut gérer des pondérations négatives alors que Dijkstra ne l'admet pas.

b Algorithme de Kruskal

L'algorithme de Kruskal (cf. algorithme 2) est un algorithme **glouton optimal** qui s'applique aux **graphes pondérés**. Le graphe peut ne pas être connexe et dans ce cas on obtient un **forêt** d'arbres recouvrants. Pour construire la forêt et effectuer un choix d'arête, il ordonne les arêtes du graphe par ordre de pondération croissante.

La complexité de cet algorithme, si l'on utilise un tas binaire, est en $O(n \log m)$ si $m = |E|$ est le nombre d'arêtes du graphe et n l'ordre du graphe.

Algorithme 2 Algorithme de Kruskal, arbre recouvrant

```

1: Fonction KRUSKAL( $G = (V, E, w)$ )
2:    $T \leftarrow \emptyset$  ▷ la sortie : l'ensemble des arêtes de l'arbre recouvrant
3:   pour  $k$  de 1 à  $|E|$  répéter
4:      $e \leftarrow$  l'arête de pondération la plus faible de  $E$  ▷ Choix glouton!
5:     si  $(S, T \cup \{e\}, w)$  est un graphe acyclique alors
6:        $T \leftarrow T \cup \{e\}$ 
7:   renvoyer  $T$ 

```

C Tri topologique d'un graphe orienté**a Ordre dans un graphe orienté acyclique**

Dans un graphe orienté acyclique, les arcs définissent un **ordre partiel**, le sommet à l'origine de l'arc pouvant être considéré comme le prédécesseur du sommet à l'extrémité de l'arc. Par exemple, sur la figure 2, a et b sont des prédécesseurs de d et e est un prédécesseur de g . Mais ces arcs ne disent rien de l'ordre entre e et h , l'ordre n'est pas total.

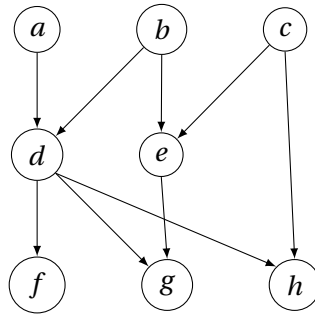


FIGURE 2 – Exemple de graphe orienté acyclique

L'algorithme de tri topologique permet de créer un ordre total \preceq sur un graphe orienté acyclique. Formulé mathématiquement :

$$\forall (v, u) \in V^2, (v, u) \in E \implies v \preceq u \quad (3)$$

Sur l'exemple de la figure 2, plusieurs ordre topologiques sont possibles. Par exemple :

- a,b,c,d,e,f,g,h
- a,b,d,f,c,h,e,g

b Tri topologique et détection de cycles dans un graphe orienté

L'algorithme de tri topologique (cf. algorithme 3) utilise le parcours en profondeur d'un graphe pour marquer au fur et à mesure les sommets dans l'ordre topologique. Une pile est utilisée pour enregistrer l'ordre chronologique de découverte des sommets.

Au cours de l'algorithme, un sommet change d'état : il peut passer de «non exploré» à «en cours d'exploration» et finalement à «exploré». Il est alors possible de détecter un cycle si on redécouvre un sommet dans l'état «en cours d'exploration». Des dates peuvent également être ajoutées au cours du traitement afin de pouvoir ordonnancer proprement les tâches parallélisables.

Algorithme 3 Tri topologique

```

1 : Fonction TRI_TOPOLOGIQUE( $G$ )                                ▷  $G$  est un graphe orienté
2 :    $L \leftarrow$  une liste vide
3 :   Marquer tous les sommets de  $G$  comme «non exploré»
4 :   pour chaque sommet  $s$  de  $G$  répéter
5 :     si  $s$  est «non exploré» alors
6 :       EXPLORER( $G, s, L$ )
7 :   renvoyer  $L$                                                 ▷ L'ordre topologique
8 : Procédure EXPLORER( $G, s, L$ )                                ▷ Parcours en profondeur depuis  $s$ 
9 :   Marquer  $s$  «en cours d'exploration»
10 :  pour chaque voisin  $u$  de  $s$  dans  $G$  répéter
11 :    si  $u$  est «non exploré» alors
12 :      EXPLORER( $G, u, L$ )
13 :    sinon si  $u$  est «en cours d'exploration» alors
14 :      Interrompre le programme car un cycle a été détecté
15 :    sinon
16 :      Ne rien faire                                            ▷ sommet déjà exploré
17 :  Marquer  $s$  «exploré»
18 :  Ajouter  $s$  en tête de la liste  $L$ 

```

D Composantes fortement connexes d'un graphe orienté et 2-SAT

■ **Définition 8 — Composante fortement connexe d'un graphe orienté** $G = (V, E)$. Une composante fortement connexe d'un graphe orienté G est un sous-ensemble S de ses sommets, maximal au sens de l'inclusion, tel que pour tout couple de sommets $(s, t) \in S$ il existe un chemin de s à t dans G .

Le calcul des composantes connexes d'un graphe est par exemple utilisé pour résoudre le problème 2-SAT. Dans le cadre de ce problème, on dispose d'une formule logique sous la forme conjonctive normale et chaque clause comporte deux variables. L'idée est de construire un graphe à partir de la formule de cette formule F . Supposons qu'elle soit constituée de m clauses et n variables (v_1, v_2, \dots, v_n) . On élabore alors un graphe $G = (V, E)$ à $2n$ sommets et $2m$ arêtes. Les sommets représentent les n variables v_i ainsi que leur négation $\neg v_i$. Les arêtes sont construites de la manière suivante : on transforme chaque clause de F de la forme $v_i \vee v_j$ en deux implications $\neg v_i \implies v_j$ ou $\neg v_j \implies v_i$. Cette transformation utilise le fait que la formule $a \implies b$ est équivalent à $\neg a \vee b$.

Théorème 4 F n'est pas satisfaisable si et seulement s'il existe une composante fortement connexe contenant une variable v_i et sa négation $\neg v_i$.

Démonstration. (\Leftarrow) S'il existe une composante fortement connexe contenant a et $\neg a$, alors cela signifie $F : (a \Rightarrow \neg a) \wedge (\neg a \Rightarrow a)$. Or cette formule n'est pas satisfaisable. En effet, si a est vrai alors $(a \Rightarrow \neg a)$ est faux, car du vrai on ne peut pas conclure le faux d'après la définition sémantique de l'implication. De même, si a est faux alors $(\neg a \Rightarrow a)$ est faux, pour la même raison. Dans tous les cas, la formule est fausse. F n'est pas satisfaisable.

(\Rightarrow) Par contraposée. Supposons qu'il n'existe pas de composante fortement connexe contenant a et $\neg a$. Cela peut se traduire en la formule $\neg F : \neg(a \Rightarrow \neg a) \vee \neg(\neg a \Rightarrow a)$. Or, cette formule F est toujours satisfaisable. En effet, $\neg F$ s'écrit

$$\neg(\neg a \vee \neg a) \vee \neg(a \vee a) = a \vee \neg a \quad (4)$$

ce qui est toujours vérifié. Par contraposée, F est donc satisfaisable s'il existe une composante fortement connexe. ■

On peut montrer que SAT-2 est un problème de décision polynomial. C'est une restriction à des clauses de deux variables du problème général SAT qui lui est NP-complet.

E Graphes bipartis et couplage maximum

■ **Définition 9 — Graphe biparti.** un graphe $G = (S, A)$ est biparti si l'ensemble S de ses sommets peut être divisé en deux sous-ensembles disjoints U et W tels que chaque arête de A ait une extrémité dans U et l'autre dans W .

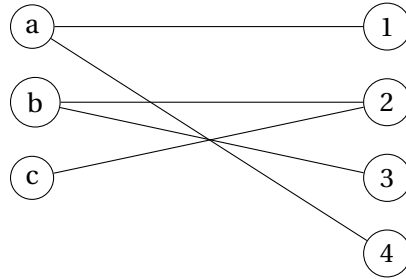


FIGURE 3 – Exemple de graphe biparti

Théorème 5 — Caractérisation des graphes bipartis. Un graphe est biparti si et seulement s'il ne possède aucun cycle de longueur impaire.

Démonstration. (\Rightarrow) Par l'absurde. Soit $G = (U, W, A)$ un graphe biparti. Soit $C = (s_0, s_1, s_2, \dots, s_k, s_0)$ un cycle de longueur impaire de G . On suppose sans perte de généralité que s_0 est dans U . Alors, comme G est biparti, $s_i \in U$ si i est pair et $s_i \in W$ sinon. Comme C est un cycle impair, k

est nécessairement pair. Donc s_k est dans U . Mais s_0 est dans U également, ce qui contredit le fait que G soit biparti. C'est donc absurde et il n'existe donc pas de cycle impair dans un graphe biparti.

(\Leftarrow) Soit un graphe $G = (S, A)$ ne possédant aucun cycle impair. Soit un T un arbre couvrant de racine r de G . On construit les ensembles U et W de la manière suivante :

- r appartient à U ,
- Tout sommet séparé de r par un nombre pair d'arêtes appartient à U ,
- Les autres sommets à W .

Les ensembles U et W sont disjoints par construction et recouvrent l'ensemble S de sommets de G . Il s'agit maintenant de montrer qu'aucune arête ne relie deux sommets de U ou de W en procédant par l'absurde. Supposons qu'il en existe une. Soit (a, b) une arête reliant deux sommets de U . il existe un chemin de a à b dans l'arbre couvrant T et ce chemin possède un nombre pair d'arêtes. Si on complète ce chemin par un l'arête (b, a) , on obtient un cycle de longueur impaire. Ce qui est absurde par hypothèse. Donc une telle arête n'existe pas et les ensemble U et W forment donc une bipartition de S . ■

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 6, c'est-à-dire son nombre chromatique est égal à 2.

a Couplage dans un graphe biparti

■ **Définition 10 — Couplage.** Un couplage Γ dans un graphe non orienté $G = (S, A)$ est un ensemble d'arêtes deux à deux non adjacentes. Formellement :

$$\forall (a_1, a_2) \in A^2, a_1 \neq a_2 \implies a_1 \cap a_2 = \emptyset \quad (5)$$

c'est-à-dire que les sommets de a_1 et a_2 ne sont pas les mêmes.

■ **Définition 11 — 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 12 — 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 13 — Couplage de cardinal maximum.** Un couplage de cardinal maximum est un couplage contenant le plus grand nombre d'arêtes possible.

■ **Exemple 2 — 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

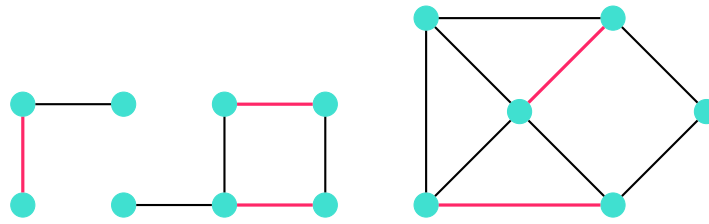


FIGURE 4 – Illustration de la notion de couplage maximal

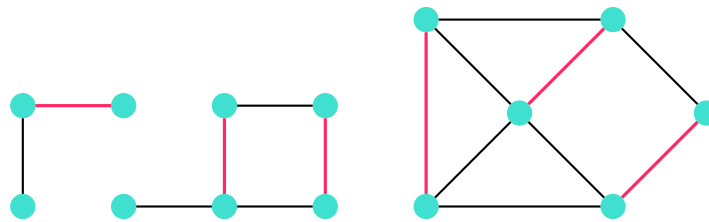


FIGURE 5 – Illustration de la notion de couplage de cardinal maximum

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 6. Dans ce cas précis, comme il y a quatre enfants qui ne veulent que les trois premiers cadeaux, il n'y a pas de solution. Mais si Enora avait choisit 4 et 5???

a. Ce papa est informaticien!

b Chemin augmentant

Pour résoudre le problème de trouver un couplage de cardinal maximum, on peut procéder en suivant l'algorithme 4. Il s'agit de construire un chemin augmentant pour atteindre un couplage de cardinal maximum.

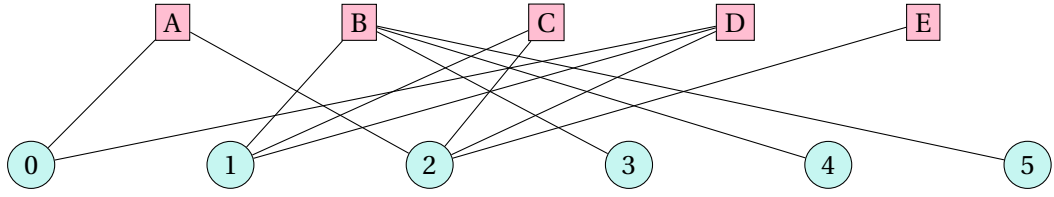


FIGURE 6 – Exemple de graphe biparti pour un problème d’affectation sans solution.

■ **Définition 14 — Chemin alternant.** Un 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 15 — 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 .

Théorème 6 — Lemme de Berge. Soit un couplage Γ dans un graphe. Γ est de cardinal maximum si et seulement s’il ne possède aucun chemin augmentant.

(R) Soit un couplage Γ dans un graphe. Alors il existe un couplage Γ' qui possède plus d’arêtes que Γ . Ce couplage vaut :

$$\Gamma' = \Gamma \Delta \pi$$

où Δ dénote la différence symétrique de deux ensembles.

$$\Gamma \Delta \pi = \{e \in E, e \in \{\Gamma \setminus \pi \cup \pi \setminus \Gamma\}\}$$

L’algorithme de recherche d’un couplage de cardinal maximum 4 s’appuie sur le lemme de Berge 6. La stratégie 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 7.

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 $\Gamma \Delta \pi$. 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.

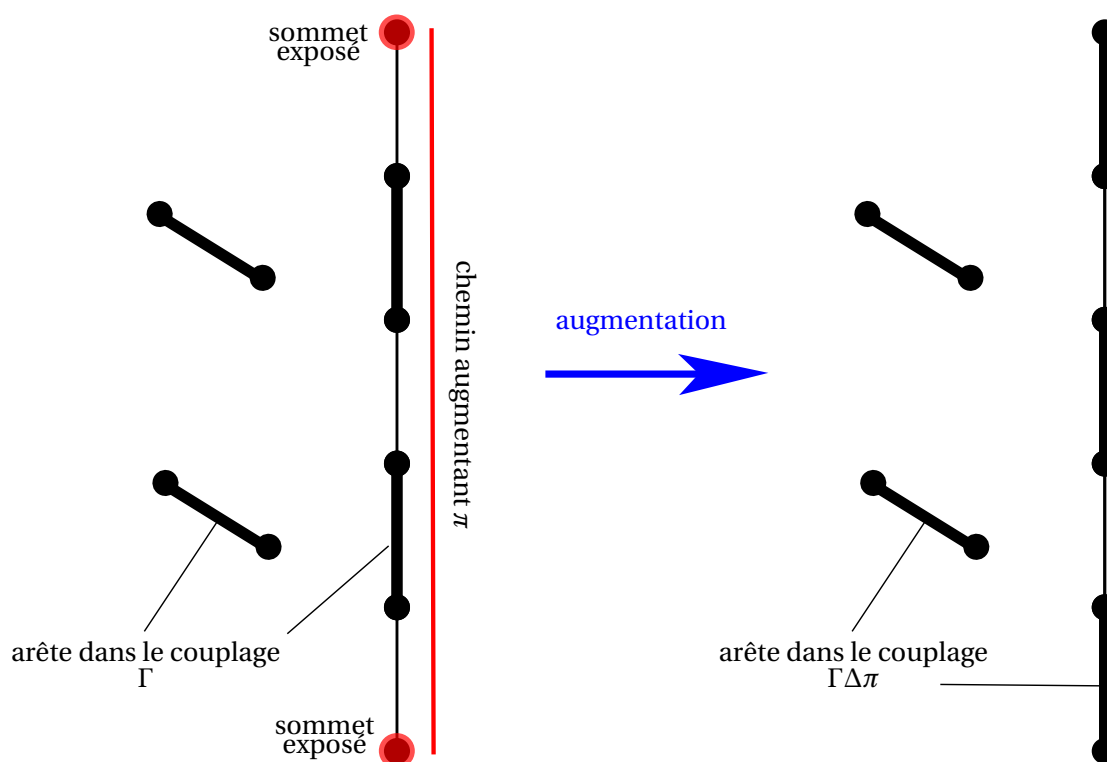


FIGURE 7 – Illustration de la construction d'un couplage de cardinal supérieur grâce à un chemin augmentant. (Source originale : Wikimedia Commons)

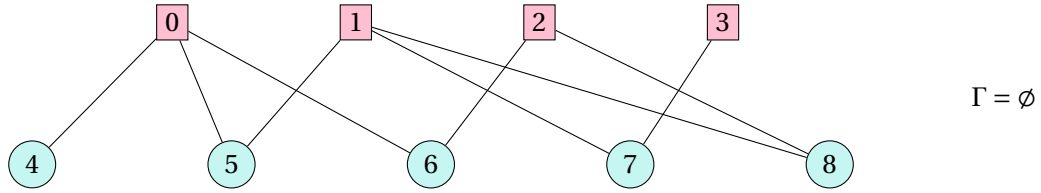
Pour trouver un chemin augmentant dans un graphe biparti $G = ((U, D), E)$, on cherche le plus court chemin entre deux sommets exposés dans un graphe orienté auxiliaire G_o construit de la manière suivante :

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

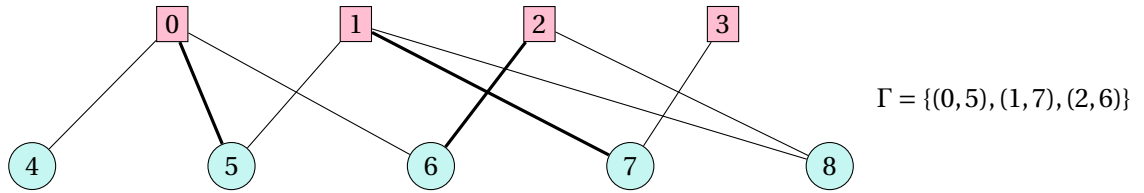
Le plus court chemin entre deux sommets exposés de G_o est nécessairement un chemin augmentant, son caractère alternant vient du fait que le graphe est biparti.

La figure 8 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.

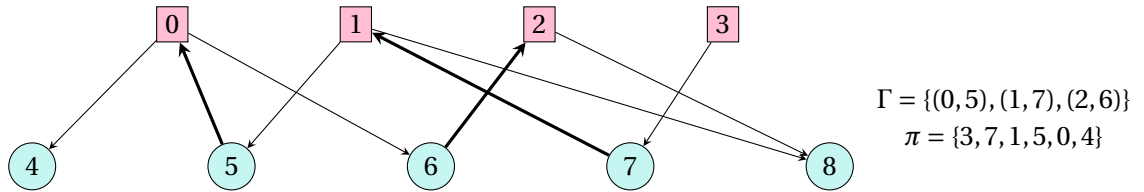
Le graphe de départ de l'algorithme est le suivant :



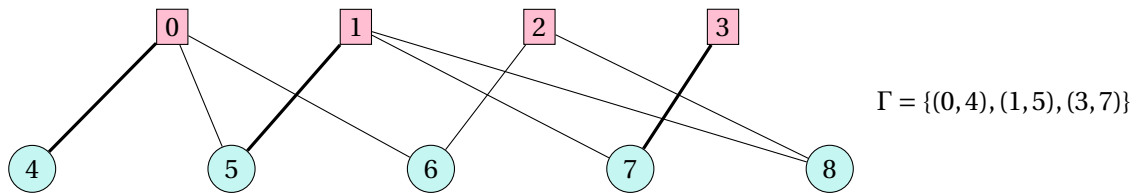
On effectue **trois appels récursifs** et, à chaque fois, on a trouvé une arête dont les sommets sont tous les deux exposés.



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 .



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



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.

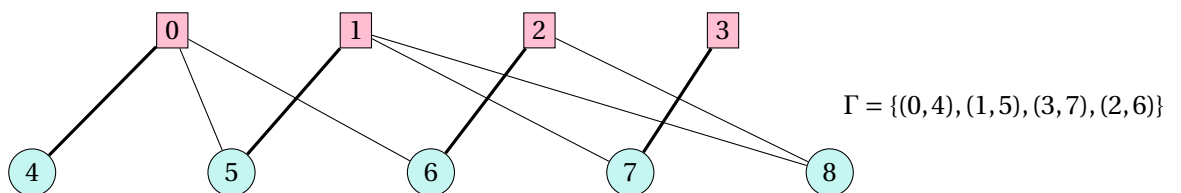


FIGURE 8 – Étapes de l'algorithme de recherche d'un couplage de cardinal maximum

Algorithme 4 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 C_MAX( $G = (V = (U, D), E), \Gamma, F_U, F_D$ )      ▷  $\Gamma$  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:     C_MAX( $G, \Gamma \cup \{e\}, F_U \setminus \{u\}, F_D \setminus \{v\}$ )
4:   sinon
5:     Créer le graphe orienté  $G_o$       ▷  $\forall e \in E, e$  de  $U$  vers  $D$  si  $e \notin \Gamma$ , l'inverse sinon
6:     Calculer le plus court chemin  $\pi$  entre un sommet de  $F_U$  et un de  $F_D$  dans  $G_o$ 
7:     si un tel chemin  $\pi$  n'existe pas alors
8:       renvoyer  $\Gamma$ 
9:     sinon
10:      C_MAX( $G, \Gamma \Delta \pi, F_U \setminus \{\pi_{\text{start}}\}, F_D \setminus \{\pi_{\text{end}}\}$ )
11:      ▷  $\pi_{\text{start}}$  et  $\pi_{\text{end}}$  : début et fin du chemin  $\pi$ 

```

F Pour aller plus loin ---> HORS PROGRAMME

a Affectation de ressources

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**. Le problème peut être énoncé de la manière suivante :

■ **Définition 16 — Problème d'affectation de ressources.** Soit un ensemble de personnes P et un ensemble de tâches T . Soit le graphe biparti $G = ((P, T), A)$ et une fonction de pondération sur les arêtes $w : A \rightarrow \mathbb{R}$. Le problème d'affectation consiste à trouver un couplage $\Gamma \subseteq A$ tel que :

- $|\Gamma| = |T|$,
- et la somme $\sum_{a \in \Gamma} w(a)$ est minimale.

L'algorithme hongrois résout ce problème en $O(n^4)$.

b Mariages stables ou appariement de ressources

Si les sommets du graphe biparti expriment des vœux de préférences, alors le problème considéré est celui des mariages stables. Ceci n'est pas au programme mais pourrait intéresser des rédacteurs d'épreuves de concours. Si vous avez du temps libre, les algorithmes d'acceptation différée [**gale_college_1962**] et des cycles d'échanges optimaux [**shapley_cores_1974**] sont à considérer.

La difficulté réside dans la capacité relative des algorithmes d'appariement de ressources de satisfaire les critères suivants :

Efficacité c'est-à-dire la satisfaction des préférences exprimées des candidats. Il s'agit de chercher à améliorer la satisfaction d'un candidat sans diminuer celle d'un autre.

Équité c'est-à-dire le respect des priorités émises par les institutions pour chaque candidat. Il n'est pas souhaitable en effet qu'un candidat plus prioritaire et désirant intégrer cette institution se voit refuser l'entrée alors qu'un autre candidat moins prioritaire y est admis.

Non manipulabilité c'est-à-dire la meilleure stratégie pour un candidat consiste toujours à classer ses vœux dans l'ordre réel de ses préférences, quels que soient les vœux soumis par les autres candidats. Il s'agit d'empêcher les stratégies individuelles de positionnement. Certains candidats pourraient être tentés par exemple de ne postuler qu'à certaines institutions pour lesquelles ils estiment avoir une chance. Il est possible aussi que les institutions manipulent l'algorithme.

Aucun algorithme n'est optimal sur ces trois critères, mais certains algorithmes comme [**gale_college_1962**] et [**shapley_cores_1974**] réalisent des compromis optimaux sur deux des trois critères et font au mieux sur le troisième. Le premier est non manipulable et respecte les priorités des candidats, mais ne produit pas nécessairement un appariement efficace. Le second algorithme est

également non manipulable mais produit un appariement efficace au prix d'une possible violation des priorités des candidats.