GRAPHES AVANCÉS

À la fin de ce chapitre, je sais :

- ☞ expliquer le concept d'arbre recouvrant
- ☞ connaître les algorithmes de Prim et Kruskal
- ☞ connaître la notion de tri topologique et son lien avec le parcours en profondeur
- ☞ expliquer le concept de forte connexité
- ☞ expliquer l'intérêt d'un graphe biparti

A Graphes connexes et acycliques

■ **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 $S' = S$.

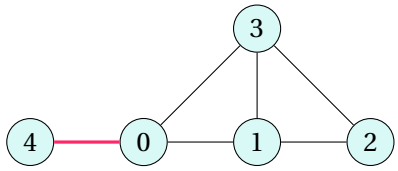
■ **Définition 4 — Graphe connexe.** Un graphe $G = (S, A)$ est connexe si et seulement si pour tout couple de sommets (a, b) de G, il existe une chaîne d'extrémités a et b.

■ **Définition 5 — Composantes connexes d'un graphe.** Un graphe non connexe est l'union de plusieurs sous-graphes connexes qui n'ont de sommet en commun. Les sous-graphes connexes disjoints sont appelés les composantes connexes du graphe.

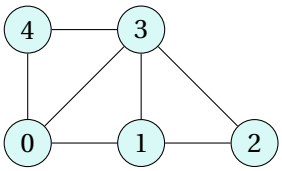
Théorème 1 — Un graphe connexe d'ordre n possède au moins $n-1$ arêtes..

Démonstration. Par récurrence sur l'ordre du graphe.

(Initialisation) Soit G un graphe d'ordre 1. Ce graphe est trivialement connexe.



Cas où un sommet au moins est de degré 1.



Cas où tous les sommets sont de degré supérieur ou égal à deux.

FIGURE 1 – Disjoinction des cas pour la démonstration de du théorème 1

(Hérédité) Supposons la propriété vraie pour les graphes d'ordre $n-1$. Soit $G=(S,A)$ un graphe d'ordre n. La figure 1 illustre la disjonction des cas ci-dessous.

- Si G possède un sommet d'ordre 1, alors en supprimant l'arête qui lui est connectée, on obtient deux commposantes connexes, dont une à $n-1$ sommets. Par hypothèse de récurrence, cette composante connexe à $n-1$ sommets possède au moins $n-2$ arêtes. Donc G possède au moins $n-1$ arêtes.
- Si G ne possède pas de sommets d'ordre 1, cela signifie qu'il possède des sommets qui sont au moins d'ordre 2. D'après le lemme des poignées de main, cela signifie que le nombre d'arêtes m de G vaut :

$$m = \frac{\sum_{s\in S} deg(s)}{2} \geqslant \frac{2n}{2} = n$$

Le graphe G possède donc au moins n arêtes.

(Conclusion) Comme la proprité est vraie pour $n=1$ et que l'hérédité est vérifiée, on en conclut qu'elle est vraie pour tous les graphes connexes.

■

Lemme 1 — Un graphe dont tous les sommets sont de degré supérieur ou égal à deux possède au moins un cycle.

Démonstration. Soit G un graphe d'ordre n dont tous les sommets sont au moins de degré 2. D'après le théorème précédent et le lemme des poignées de mains, le nombre d'arêtes de G est supérieur à n et donc G est connexe.

Soit s_0 un sommet quelconque. On parcours G en profondeur à partir de s_0. Ce parcours $\mathcal{P}$ est une suite finie de sommets, chaque sommet étant découvert par adjacence et visité une seule fois. Tous les sommets, quelque soit s_0, sont visités une fois car le graphe est connexe.

Comme l'ordre d'un graphe est fini, $\mathcal{P}$ va se terminer, par exemple sur le sommet z. On a découvert z par une arête du graphe G, mais on n'en est pas sorti. Comme tous les sommets

sont au moins de degré 2, cela signifie qu'il existe une autre arête incidente à z qui connecte ce sommet à un autre sommet x de G, sommet déjà découvert par le parcours en profondeur. L'arête (z,x) forme donc un cycle dans le graphe. G possède nécessairement un cycle. ■

Théorème 2 — Un graphe acyclique possède au plus $n-1$ arêtes.

Démonstration. Par récurrence sur l'ordre du graphe.

(Initialisation) Un graphe d'ordre 1 est trivialement acyclique.

(Hérédité) Supposons la propriété vraie pour un graphe d'ordre $n-1$. Soit G un graphe acyclique à n sommets. D'après le lemme précédent, G possède au moins un sommet dont le degré est 0 ou 1.

- S'il est de degré 0, alors le graphe n'est pas connexe et les composantes acycliques restantes comportent $n-1$ sommets et au plus $n-2$ arêtes, d'après notre hypothèse de récurrence.
- S'il est de degré 1, en retirant l'arête qui le relie au reste du graphe, on obtient au moins deux composantes acycliques dont une possédant $n-1$ sommets, qui par hypothèse de récurrence comporte donc au plus $n-2$ arêtes.

Dans tous les cas, G possède donc au plus $n-1$ arêtes.

(Conclusion) La propriété étant vérifié à l'ordre 1 et l'hérédité étant conservée, la propriété est vraie pour tous les graphes acycliques.

■

■ **Définition 6 — Arbre.** Un arbre est un graphe **non orienté connexe et acyclique**.

Théorème 3 — Un arbre à n sommets possède exactement $n-1$ arêtes.

Démonstration. On utilise les résultats précédents : un arbre possède au moins $n-1$ arêtes et au plus $n-1$ arêtes. Il en possède donc exactement $n-1$. ■

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

1. Soit $G=(S,A)$ un graphe non orienté d'ordre n. Montrer que si G est connexe alors il comporte au moins $n-1$ arêtes. (Procéder par récurrence)
2. Soit $G=(S,A)$ un graphe non orienté d'ordre n. Montrer que si G est connexe et 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. Soit $G=(S,A)$ un graphe non orienté d'ordre n. Montrer que que si G est acyclique alors il possède au plus $n-1$ arêtes. (Procéder par récurrence)
4. Montrer que la modélisation planaire des alcanes de formule C_nH_{2n+2} est un arbre. (Utiliser le lemme de l'étoile)

5. Montrer qu'un graphe d'ordre n non connexe possède au plus $\frac{(n-1)(n-2)}{2}$ arêtes.
6. On souhaite interconnecter les sept plus grandes villes de Bretagne à l'aide d'un réseau à très haut débit. Trois villes possèdent au moins deux liens les reliant à deux autres autres villes. Toutes les villes sont-elles interconnectées ?

Théorème 4 — Caractérisation des arbres. Soit G un graphe non orienté à n sommets. Les propositions suivantes sont équivalentes :

1. G est un arbre.
2. G est connexe et possède $n-1$ arêtes.
3. G est acyclique et possède $n-1$ arêtes.

Démonstration. Soit G un graphe non orienté à n sommets.

- Si G est un arbre, c'est un graphe connexe qui possède $n-1$ arêtes, car également acyclique. Donc 1. implique 2.
- Supposons que 2. est vraie. Supposons que G possède un cycle. Alors on peut retirer une arête et obtenir un graphe à n sommet connexe qui possède $n-2$ arêtes. Ceci est absurde d'après le théorème 1. Donc, G est acyclique et 2. implique 3.
- Supposons que 3. est vraie. S'il ne possède qu'une seule composante connexe, alors G est acyclique et connexe. C'est donc un arbre et 1. est vraie. S'il possède m composantes connexes $G_i = (S_i, A_i)$, alors comme les G_i sont connexes et acycliques, on peut écrire :

$$|A| = \left(\sum_{i=1}^{m} |S_i|\right) - m = |S| - m = n - 1$$

  Par identification, on trouve que m vaut 1. Le graphe G est donc connexe et acyclique : c'est un arbre.

Comme 1. $\Rightarrow$ 2. $\Rightarrow$ 3. $\Rightarrow$ 1., on en déduit que ces trois propositions sont équivalentes. ■

■ **Définition 7 — Forêt.** Une forêt est un graphe non orienté sans cycle dont chaque composante connexe est un arbre.

B Arbres recouvrants

■ **Définition 8 — 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.

■ **Définition 9 — Arbre recouvrant minimal.** Un arbre recouvrant minimal est un arbre recouvrant un graphe dont la somme des poids des arêtes est minimale.

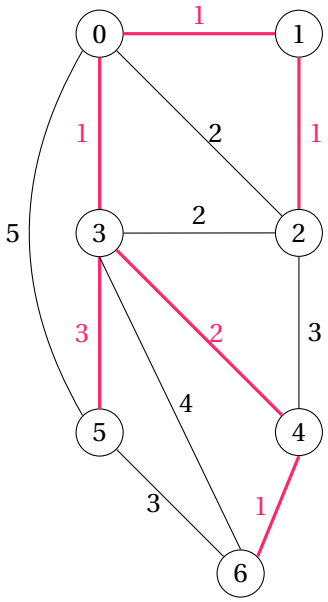


FIGURE 2 – Exemple d'arbre recouvrant dans un graphe non orienté et pondéré. En rouge, un arbre recouvrant minimal de poids total 9.

Les arbres recouvrants sont des éléments essentiels de monde réel car les applications sont nombreuses. La figure 2 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 utilisés dans les réseaux d'énergie, de télécommunications, de fluides, de transport, de construction mais également en intelligence artificielle et en conception de circuits électroniques. Les deux algorithmes phares pour construire des arbres recouvrants **minimaux** sont l'algorithme de Kruskal [**kruskal_shortest_1956**] et l'algorithme de Prim [**prim_shortest_1957**]. L'algorithme de Wilson permet quant à lui de générer des arbre recouvrants aléatoires.

■ **Définition 10 — Coupe de graphe.** On appelle coupe d'un graphe $G = (S, A)$

- une partition de l'ensemble des sommets S en deux sous-ensembles S_1 et S_2 non vides tels que $S = S_1 \cup S_2$ et $S_1 \cap S_2 = \emptyset$,

- ou bien l'ensemble des arêtes possédant une extrémité dans chaque sous-ensemble de la partition.

Le poids de la coupe est :

- soit la somme des poids respectifs des arêtes de la coupe si G est pondéré,
- soit le nombre d'arêtes dans la coupe.

Une arête de la coupe est dite traversante.

Théorème 5 — Propriété de la coupe. Soit $G = (S, A, w)$ un graphe pondéré. Soit C une coupe de ce graphe. Alors l'arête de poids minimum de la coupe fait partie de tous les arbres recouvrants minimum du graphe.

Démonstration. Par l'absurde. La figure 3 illustre l'objet de la propriété.

Soit $G = (S, A, w)$ un graphe pondéré. Soit $C = \{S1, S2\}$ une coupe de ce graphe. Soit $a_{\min} = (s_1, s_2)$, l'arête de poids minimum de la coupe. Supposons qu'il existe un arbre recouvrant minimum $\mathcal{A}$ de G ne contenant pas $a_{\min}$.

$\mathcal{A}$ est un graphe connexe et donc on peut y trouver un chemin qui connecte s_1 et s_2. Comme $a_{\min}$ n'est pas une arête de $\mathcal{A}$, il existe sur ce chemin de s_1 à s_2 une arête $e = (u, v)$ telle que $u \in S_1$ et $v \in S_2$ et que $w(e) > w(a_{\min})$. Alors $(\mathcal{A} \setminus \{e\}) \cup \{a_{\min}\}$ est un arbre revouvrant de G dont le poids est inférieur à celui de $\mathcal{A}$. Ce qui est absurde puisqu'on a supposé que $\mathcal{A}$ était un arbre recouvrant minimum. ■

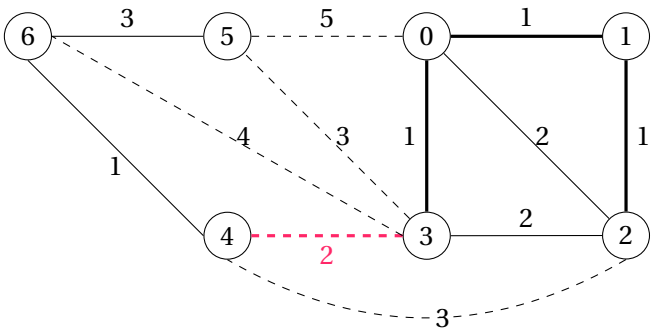


FIGURE 3 – Illustration de la propriété de la coupe. En pointillé, les arêtes de la coupe $(\{0,1,2,3\},\{4,5,6\})$. L'arête $(3,4)$ en rouge fait partie de tous les arbres recouvrants minimaux du graphe.

Théorème 6 — Propriété du cycle. Soit $G = (S, A, w)$ un graphe pondéré possédant un cycle. L'arête de poids maximum de ce cycle ne peut pas faire partie d'un arbre recouvrant minimum du graphe.

Démonstration. Par l'absurde.

Soit $G = (S, A, w)$ un graphe pondéré possédant un cycle dont l'arête de poids maximum est $a_{\max} = (u, v)$. Supposons qu'il existe un arbre recouvrant minimum $\mathcal{A}$ de G contenant $a_{\max}$.

L'arête $a_{\max}$ fait partie d'un cycle dans G. Pour construire $\mathcal{A}$, on a ôté une arête e de ce cycle. Considérons $\mathcal{A} \setminus \{a_{\max}\} \cup \{e\}$: ce graphe est un arbre dont le poids est inférieur à celui de $\mathcal{A}$, car $w(e) < w(a_{\max})$. Ce qui est absurde puisqu'on a supposé que $\mathcal{A}$ était un arbre recouvrant minimum. ■

■ **Exemple 2 — Exercices types.**

- Montrer que l'arbre recouvrant minimal n'engendre pas nécessairement le plus cours chemin entre deux sommets donnés. (Donner un exemple)
- Caractérisation de la suite des degrés d'un arbre. Soit G un graphe connexe non orienté à $n \geq 2$ sommet.
  1. Montrer que G est un arbre si et seulement si la suite des degrés $(d_1, d_2, \ldots, d_n) \in (\mathbb{N}^*)^n$ des sommets de l'arbre vérifie :
  $$\sum_i d_i = 2(n-1)$$
  2. Représenter tous les arbres d'ordre inférieur à 5.

a Algorithme de Prim

L'algorithme de Prim est un algorithme **glouton optimal** qui s'applique aux **graphes pondérés connexes** et qui construit un arbre recouvrant minimal. Pour construire l'arbre, l'algorithme part d'un sommet et fait croitre l'arbre en choisissant une arête dont l'extrémité de départ appartient à l'arbre et mais pas l'extrémité d'arrivée **et** dont le poids est le plus faible, garantissant ainsi l'absence de cycle et la minimalité.

Algorithme 1 Algorithme de Prim, arbre recouvrant

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

Démonstration.
TERMINAISON. Soit v la fonction définie par :

$$\begin{aligned} v : \mathbb{N} &\longrightarrow \mathbb{N} \\ |S| &\longmapsto |V| - |S| \end{aligned}$$

La fonction v est un variant de boucle pour la boucle *tant que* de l'algorithme de Prim. En effet, un graphe possède toujours un nombre fini de sommets, donc $|V|$ est un entier naturel. Au premier tout de boucle, $|S| = 1$ et à chaque tour le cardinal de S augmente de 1. Donc, v est une fonction à valeurs entières **strictement** décroissante. Elle atteint donc 0 (théorème de la limite monotone) lorsque la condition de boucle est invalidée, c'est-à-dire lorsque $|S| = |V|$. L'algorithme de Prim se termine.

CORRECTION. Soit $G = (V, E, w)$ un graphe pondéré connexe d'ordre n.

Le choix de v se fait dans $V \setminus S$ ce qui exclue la construction de cycles dans (S, T) et garantit qu'on n'ajoute un sommet qu'une seule fois à S. Par ailleurs, le choix de l'arête se fait au départ de S, ce qui garantit la connexité dans la construction de (S, T). (S, T) est donc un graphe connexe acyclique, c'est-à-dire un arbre. Cet arbre est recouvrant puisque, à la fin de la boucle, $S = V$.

Il reste à démontrer que l'arbre est poids **minimal**. On peut le montrer en utilisant la propriété de la coupe 5 et en exhibant un invariant de boucle

On considère la propriété $\mathcal{J}$ suivante : « Le sous-graphe $G' = (S, T, w)$ est un arbre recouvrant minimal de (S, E, w). » On montre que c'est un invariant pour la boucle **tant que**.

À l'entrée de la boucle $G' = (\{s\}, \emptyset, w)$, on a $|G'| = 1$, T est vide et on a $|T| = 0 = 1 - 1$. Donc, G' est minimal car le poids de l'arbre est nul.

Conservation Supposons que la propriété soit vraie au début d'une itération.

Soit $a_{min} = (u, v)$ l'arête de poids le plus faible dans la coupe $(S, V \setminus S)$.

À la fin de l'itération, $G'' = (S \cup \{v\}, T \cup \{u, v\}, w)$. G'' est un graphe connexe et acyclique car on choisit de relier un sommet de S à un autre sommet de $V \setminus S$, donc G'' est un arbre recouvrant.

On considère la coupe $(S, \{v\})$ de $(S \cup \{v\}, E, w)$. a_{min} est également l'arête de poids le plus faible de $(S \cup \{v\}, E, w)$. D'après la propriété de la coupe, l'arête a_{min} fait partie de tous les arbres recouvrants minimaux de $(S \cup \{v\}, E, w)$. Comme G' est un arbre couvrant minimal de (S, E, w) à l'entrée de la boucle et qu'on a ajouté une arête faisant parti de tous les arbres minimaux de $(S \cup \{v\}, E, w)$, alors G'' est une arbre couvrant minimal de de $(S \cup \{v\}, E, w)$.

Conclusion La propriété est donc vraie à l'entrée de la boucle et invariante par les instructions de la boucle. On en conclut que tous les sous-graphes $G' = (S, T, w)$ sont des arbres revouvrants minimaux de de (S, E, w). À la sortie de la boucle, $S = V$: $G' = (V, T, w)$ est donc un arbre recouvrant minimal de $G = (V, E, w)$.

■

R L'algorithme de Prim est une preuve constructive que tout graphe connexe possède au moins un arbre recouvrant minimal.

R La connexité de l'arbre est garanti par le choix de v dans $V \setminus S$. C'est en même temps une limitation : cet algorithme ne fonctionnerait pas si le graphe n'était pas connexe car certains sommets seraient inaccessibles.

R Si tous les poids du graphe sont différents, T est unique car il n'y a pas d'ambiguité dans le choix de l'arête de poids le plus faible.

R La complexité de cet algorithme, si l'on utilise un tas binaire pour la file de priorités 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 Quelles sont les différences entre Dijkstra et 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. On peut le vérifier rapidement sur la figure 2.
- Dijkstra s'applique sur les graphes orientés et non orientés mais Prim ne s'applique qu'à des graphes pondérés 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** et qui construit une forêt d'arbres recouvrants minimaux. Le graphe peut ne pas être connexe. Pour construire la forêt et effectuer un choix d'arête, il ordonne les arêtes du graphe par ordre de pondération croissante.

Démonstration. Il est facile de montrer que (S, T, w) :

- est une forêt, car aucun cycle créé,
- est une forêt recouvrante : s'il existait un sommet non recouvert, c'est-à-dire non connecté à (S, T, w), cela signifierait qu'il n'est connecté par aucune arête au reste du graphe (sommet isolé, c'est donc une composante connexe de la forêt) ou que l'arête qui le relie n'a pas été considérée. Cette dernière solution est impossible puisqu'on les considère toutes en m itérations.

En utilisant la propriété de la coupe 5, on montre qu'à chaque itération, (S, T, w) est inclus dans une forêt recouvrante minimale de G. ■

Algorithme 2 Algorithme de Kruskal, arbre recouvrant

```
1 : Fonction KRUSKAL(G = (V, E, w))
2 :     T ← ∅                          ▷ la sortie : l'ensemble des arêtes de la forêt recouvrante
3 :     pour k de 1 à |E| répéter
4 :         e ← l'arête de pondération la plus faible de E          ▷ Choix glouton !
5 :         si (S, T ∪ {e}, w) est un graphe acyclique alors
6 :             T ← T ∪ {e}
7 :         E ← E \ e                  ▷ On n'a plus à tester cette arête
8 :     renvoyer T
```

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

c Unir et trouver --→ HORS PROGRAMME

La principale difficulté de l'algorithme de Kruskal réside au niveau du test d'acyclicité : comment savoir si le nouvel arbre qu'on projette de construire est bien acyclique ? On pourrait utiliser un parcours en profondeur car dès qu'on trouve un sommet déjà découvert un cycle est détecté. Cependant ce n'est pas optimal au niveau de la complexité. C'est pourquoi il est préférable d'utiliser une structure de type Union-Find (UF) : c'est une structure très efficace qui permet de réunir des ensembles disjoints en les étiquetant sous la même étiquette. On distingue ainsi les sommets du graphe qui sont connexes dans l'arbre en cours de construction et des autres.

■ **Définition 11 — Unir-trouver.** Unir-trouver est une structure de données qui représente une partition d'un ensemble fini ou de manière équivalente une relation d'équivalence.

Un ensemble est composé de n éléments représentés par des entiers de 0 à $n-1$. Une partie de l'ensemble est identifiée par un des éléments qui la compose. Au départ, une structure unir-trouver est initialisée de telle manière que chaque élément de l'ensemble appartient à sa propre partie.

La structure unir-trouver possède deux opérations :

Trouver (find) détermine à quelle partie appartient un élément. Cette opération permet de déterminer si deux éléments appartiennent à la partie.

Unir (union) réunit deux parties de la partition en une seule.

Pour être efficace, cette structure est composite et composée :

- d'un tableau `parent` dans laquelle on maintient l'identité de la partie à laquelle appartient l'élément : `parent[i]` est un entier qui représente la partie à laquelle appartient `i`.
- d'un tableau `rank` dans laquelle on maintient un ordre entre les parties : leur rang. Ce rang représente d'une certaine manière la taille de la partie. `rank[i]` est un entier qui vaut 0 à l'initialisation de la structure et qui croit au fur et à mesure des unions de parties.

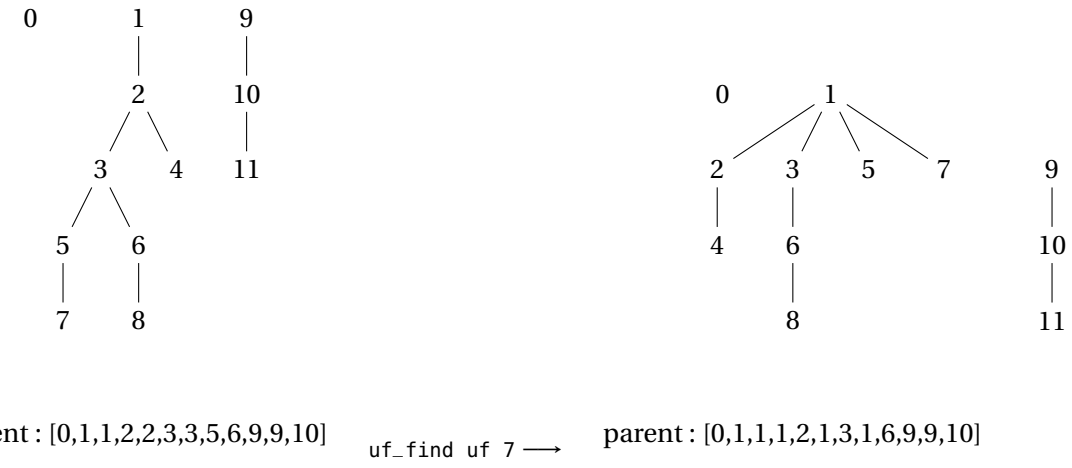


FIGURE 4 – Illustration de la compression de chemin lorsqu'on cherche le parent de l'élément 7. Cet ensemble est divisé en trois parties identifiées par les racines 0, 1 et 9.

■ **Exemple 3 — Exemple de tructure unir-trouver.** Par exemple, pour un ensemble à 5 éléments, `parent = [1,1,4,1,4]` signifie que les éléments d'indice 0,1 et 3 sont dans une même partie représentée par 1 et que 2 et 4 sont dans une même partie représentée par 4. `rang =[1,1,2,1,2]` signifie que le rang de la partie 1 est 1 et celui de la partie 4 est 2.

R Pour que la complexité soit optimale, il est important d'implémenter la compression de chemin dans les opérations de la structure unir-trouver. Ce concept est illustré sur la figure 4. Au fur que l'on interroge (trouver) la structure, on relie directement un élément à l'entier qui représente sa partie.

C Tri topologique d'un graphe orienté

a Ordre dans un graphe orienté acyclique

Dans un graphe orienté acyclique (Acyclic Directed Graph ou DAG en anglais), 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 5, 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.

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

$$\forall (v,u) \in V^2, (v,u) \in E \Longrightarrow v \preceq u \tag{1}$$

Sur l'exemple de la figure 5, plusieurs ordre topologiques sont possibles. Par exemple :

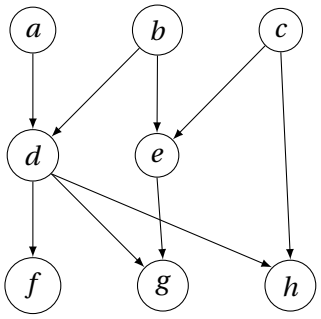


FIGURE 5 – Exemple de graphe orienté acyclique

- $a \preceq b \preceq c \preceq d \preceq e \preceq f \preceq g \preceq h$
- $a \preceq b \preceq d \preceq f \preceq c \preceq h \preceq e \preceq g$

b Existence d'un tri topologique

Théorème 7 — Existence d'un tri topologique. Tout graphe orienté acyclique possède un tri topologique.

Démonstration. Par récurrence sur le nombre de sommets du graphe.

- Initialisation : Pour un graphe orienté acyclique à un seul sommet, ce sommet est trivialement trié.
- Hérédité : on suppose que tout graphe orienté acyclique d'ordre strictement inférieur à n possède un tri topologique. Considérons un graphe G d'ordre n, acyclique et orienté. **Il existe au moins un sommet s qui ne possède aucun arc entrant, sinon cela aurait pour effet de créer un cycle dans le graphe** [1]. Si on retire s et ses arcs sortants du graphe, on obtient deux composantes dont un graphe orienté acyclique à $n-1$ sommets qui possède, par hypothèse de récurrence, un tri topologique. Comme s n'a pas de prédecesseurs dans le graphe, il suffit alors de le placer en tête de ce tri topologique pour obtenir un tri topologique de G.
- Conclusion : comme la propriété 7 est vraie pour un graphe à un sommet et que l'héridité est démontrée, alors elle est vraie pour tout graphe acyclique orienté.

■

R La clef des démonstrations autour des graphes acyclique est qu'il existe au moins un sommet s qui ne possède aucun arc entrant.

1. On peut démontrer par l'abusrde cette propriété.

c Algorithmes de tri topologique

Pour construire un tri topologique, on peut utiliser l'algorithme de Kahn (cf. algorithme 3) ou le parcours en profondeur (cf. algorithme 4). Tous les deux permettent de détecter des cycles.

Algorithme 3 Algorithme de Kahn pour le tri topologique

```
1 : Fonction TT_KAHN(G)                          ▷ G = (S, A) est un graphe acyclique orienté
2 :     Créer une liste vide ordre
3 :     Créer une file file pour stocker les sommets dont le degré entrant vaut 0
4 :     Créer un dictionnaire degré_entrant pour stocker le degré entrant de chaque sommet
5 :     pour chaque sommet v ∈ S répéter
6 :         Calculer le degré entrant de v
7 :         degré_entrant[v] ← nombre d'arcs entrants pour v
8 :         si degré_entrant[v] = 0 alors
9 :             ENFILER(file, v)
10 :    tant que file n'est pas vide répéter
11 :        u ← DÉFILER(file)
12 :        AJOUTER(ordre,u)
13 :        pour pour chaque arc u → v répéter
14 :            Réduire le degré entrant de v de 1
15 :            si degré_entrant[v] = 0 alors
16 :                ENFILER(file, v)
17 :    si la taille de ordre vaut le cardinal de S alors
18 :        renvoyer ordre
19 :    sinon
20 :        Lever une exception : il y a un cycle dans le graphe et aucun un ordre topologique.
```

L'algorithme de tri topologique (cf. algorithme 4) 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 visité» à «en cours» et finalement à «terminé». Il est alors possible de détecter un cycle si on redécouvre un sommet dans l'état «en cours». Des dates peuvent également être ajoutées au cours du traitement afin de pouvoir ordonnancer proprement les tâches parallélisables.

Algorithme 4 Tri topologique

```
1 : Fonction TRI_TOPOLOGIQUE(G)                    ▷ G = (V, E) est un graphe orienté acyclique
2 :     Créer une liste vide ordre_topologique
3 :     Créer un ensemble visités pour les sommets visités
4 :     Créer un dictionnaire état pour l'état des sommets (non_visité, en_cours, terminé)
5 :     Fonction DFS(s)                            ▷ DFS : parcours en profondeur en anglais
6 :         si s n'est pas dans visités alors
7 :             Ajouter s à visités
8 :             état[s] ← en_cours
9 :             pour chaque arc s → v répéter
10 :                si état[v] = non_visité alors
11 :                    DFS(v)
12 :                sinon si état[v] = en_cours alors
13 :                    Lever une exception : cycle détecté
14 :            état[s] ← terminé
15 :            Ajouter s au début de ordre_topologique
16 :    pour chaque sommet v ∈ V répéter
17 :        si v n'est pas dans visités alors
18 :            DFS(v)
19 :    renvoyer ordre_topologique
```

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

■ **Définition 12 — 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.

■ **Définition 13 — Fortement connexe, une relation binaire** $\mathcal{C}$**.** Soit u et v deux sommets de G et $\mathcal{R}$ la relation binaire définie sur les sommets d'un graphe orienté G par : $u\mathcal{R}v$ si et seulement si u et v font partie d'une même composante fortement connexe $\mathcal{C}$.

Théorème 8 — $\mathcal{R}$ est une relation d'équivalence.

Démonstration.
- Réflexivité : soit u un sommet de G. On a bien $u\mathcal{R}u$, car u est trivialement connecté à u.
- Symétrie : si $u\mathcal{R}v$ alors il existe un chemin de u à v, et v et u appartiennent à la même composante fortement connexe. D'après la définition d'une composante fortement connexe, on a donc $v\mathcal{R}u$.
- Transitive : si $u\mathcal{R}v$ et si $v\mathcal{R}w$, alors u et w appartiennent à la même composante fortement connexe que v. On a donc $w\mathcal{R}v$ et $v\mathcal{R}u$: il existe donc un chemin de w à u, c'est-à-dire $w\mathcal{R}u$.

$\mathcal{R}$ est donc une relation d'équivalence. ■

Théorème 9 — L'ensemble des composantes connexes d'un graphe orienté $G = (S, A)$ **forme une partition de** S**.**

Démonstration. Montrer :
- que les composantes fortement connexes sont disjointes par l'absurde,
- que chaque sommet appartient au moins à une composante, fût-elle réduit à un seul élément.

■

■ **Définition 14 — Graphe quotient.** Le graphe quotient d'un graphe orienté $G = (S, A)$ est le graphe $G_q = (S_q, A_q)$ où :
- S_q est l'ensemble des composantes fortement connexes de G, c'est-à-dire chaque sommet de G_q est une composante connexe de G.
- $A_q = \left\{(c_1, c_2) \in S_q^2, c_1 \neq c_2 \text{ et } \exists (u, v) \in c_1 \times c_2, (u, v) \in A\right\}$

Théorème 10 — Le graphe quotient est acyclique.

Démonstration. On procède par l'absurde. Supposons que G_q, le graphe quotient de G soit cyclique. Cela signifierait qu'il existerait un cycle dans G_q et donc on pourrait trouver un sommet c_i de S_q tel qu'il existerait un chemin dans G_q tel que $c_i \rightarrow^* c_i$. Autrement dit, il existerait un cycle qui relierait des composantes connexes de G. Mais alors, ce cycle serait lui-même une composante connexe de G et devrait donc être un sommet de G_q. On aboutit alors une contradiction : G_q ne serait pas le graphe quotient de G.

C'est pourquoi un graphe orienté est un graphe acyclique de ses composantes fortement connexes. ■

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 $(v_1, v_2, \ldots, v_n)$. 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 $\neg v_1 \implies v_2$ ou $\neg v_2 \implies v_1$. Cette transformation utilise le fait que la formule $a \implies b$ est équivalent à $\neg a \vee b$.

Théorème 11 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. ($\Longleftarrow$) S'il existe une composante fortement connexe contenant a et $\neg a$, alors cela signifie $F : (a \implies \neg a) \wedge (\neg a \implies a)$. Or cette formule n'est pas satisfaisable. En effet, si a est vrai alors $(a \implies \neg a)$ 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 $(\neg a \implies a)$ est faux, pour la même raison. Dans tous les cas, la formule est fausse. F n'est pas staisfaisable.

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

$$\neg(\neg a \vee \neg a) \vee \neg(a \vee a) = a \vee \neg a \tag{2}$$

ce qui est toujours vérifié. S'il n'existe pas de composante fortement connexe, alors F est satisfaisable. Par contraposition, si F n'est pas satisfaisable alors 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

a Caractérisation des graphes bipartis

■ **Définition 15 — 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.

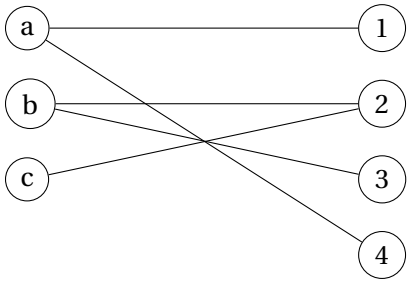


FIGURE 6 – Exemple de graphe biparti

Théorème 12 — Caractérisation des graphes bipartis par les cycles. Un graphe est biparti si et seulement s'il ne possède aucun cycle de longueur impaire,

Démonstration. ($\Longrightarrow$) Par l'absurde. Soit $G = (U, W, A)$ un graphe biparti. Soit $C = (s_0, s_1, s_2, \ldots, s_k, s_0)$ 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.

($\Longleftarrow$) 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.

■

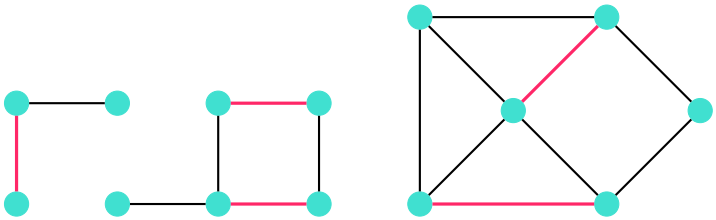


FIGURE 7 – Illustration de la notion de couplage maximal

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

Théorème 13 — Caractérisation des graphes bipartis et coloration. Un graphe est biparti si et seulement si est bi-colorable.

b Couplage dans un graphe biparti

■ **Définition 16 — Couplage.** Un couplage Γ dans un graphe non orienté $G = (S, A)$ est un ensemble d'arêtes deux à deux non adjacentes. Formellement :

$$\forall (a_1, a_2) \in A^2, a_1 \neq a_2 \Longrightarrow a_1 \cap a_2 = \emptyset \tag{3}$$

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

■ **Définition 17 — 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 18 — 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 19 — Couplage de cardinal maximum.** Un couplage de cardinal maximum est un couplage contenant le plus grand nombre d'arêtes possible.

■ **Exemple 4 — 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 :

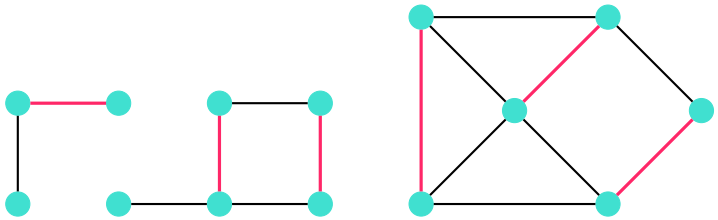


FIGURE 8 – Illustration de la notion de couplage de cardinal maximum

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 9. 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 !

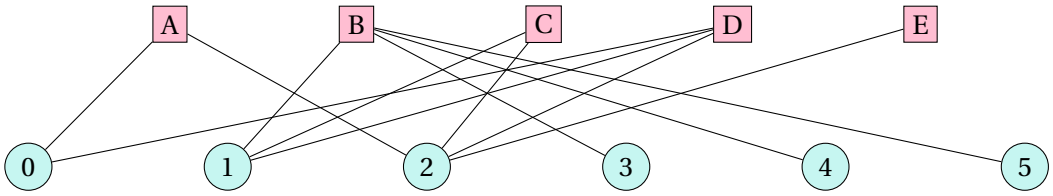


FIGURE 9 – Exemple de graphe biparti pour un problème d'affectation sans solution.

c Chemin augmentant

Pour résoudre le problème de trouver un couplage de cardinal maximum, on peut procéder en suivant l'algorithme 5. Il s'agit de construire un chemin augmentant pour atteindre un couplage de cardinal maximum.

■ **Définition 20 — 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 21 — 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 14 — Lemme de Berge. Soit un couplage Γ dans un graphe. Γ est de cardinal maximum si et seulement s'il ne possède aucun un chemin augmentant.

Démonstration. ($\Rightarrow$) Soit Γ une couplage de cardinal maximum. Si un chemin augmentant existait, on pourrait améliorer Γ en augmentant le nombre d'arêtes de ce couplage, ce qui contredit l'hypothèse que Γ est maximum.

($\Leftarrow$) Soit Γ une couplage ne possédant aucun un chemin augmentant. Si Γ n'est pas maximum, cela signifie qu'on peut ajouter une arête au couplage et le faire croitre. Mais cela signifie également qu'il existe un chemin augmentant. Ce qui est contredit notre hypothèse.

■

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 5 s'appuie sur le lemme de Berge 14. 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 10.

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.

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 11 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.

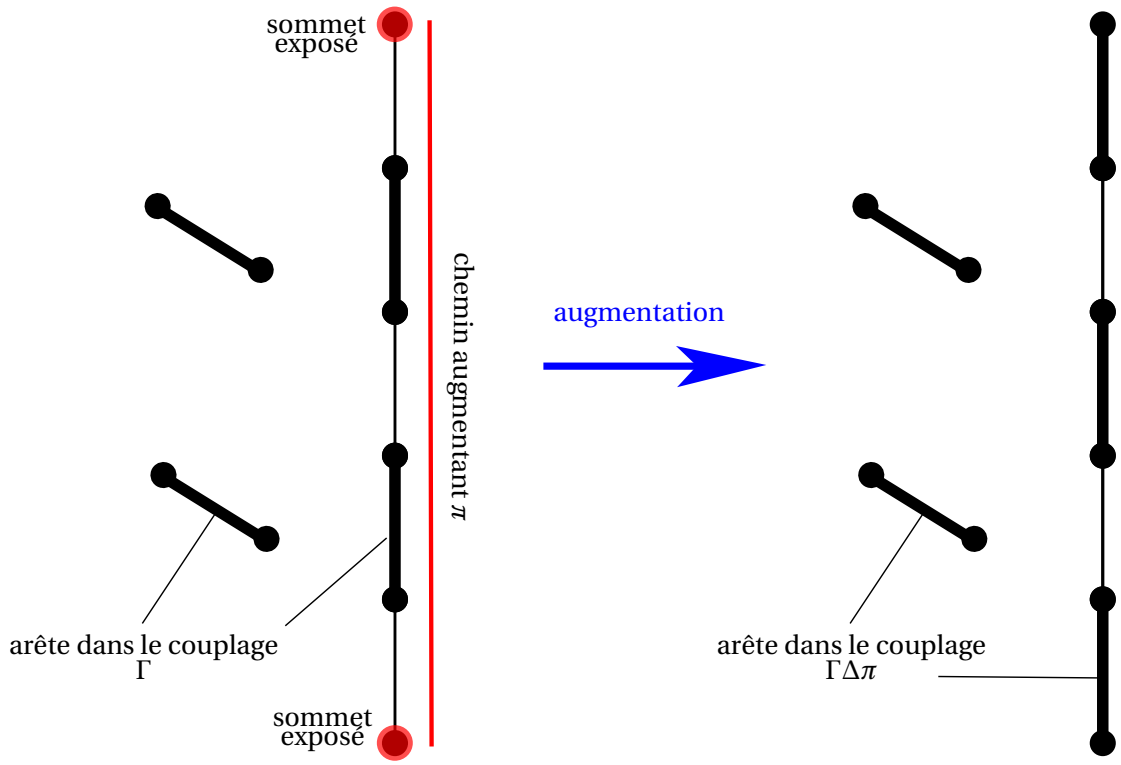


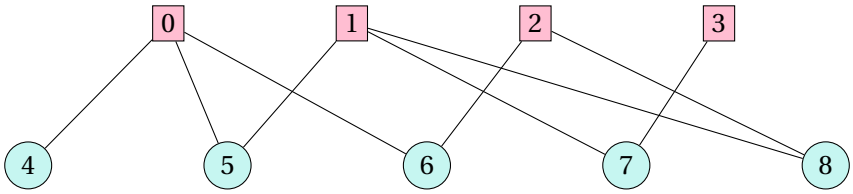
FIGURE 10 – Illustration de la construction d'un couplage de cardinal supérieur grâce à un chemin augmentant. (Source originale : Wikimedia Commons)

Algorithme 5 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

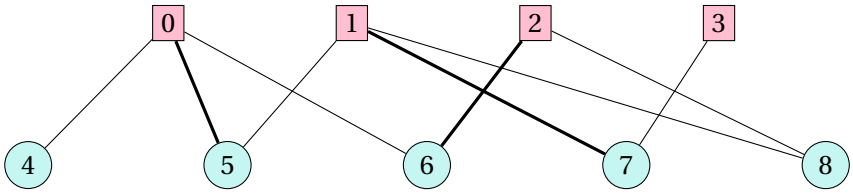
```
1 : Fonction C_MAX(G = (V = (U, D), E), Γ, F_U, F_D))        ▷ Γ est le couplage, vide initialement
2 :     si une arête e = (u, v) entre un sommet de F_U et un sommet de F_D existe alors
3 :         C_MAX(G, Γ ∪ {e}, F_U \ {u}, F_D \ {v})
4 :     sinon
5 :         Créer le graphe orienté G_o                  ▷ ∀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 Γ
9 :         sinon
10 :            C_MAX(G, ΓΔπ, F_U \ {π_start}, F_D \ {π_end})
11 :                                  ▷ π_start et π_end : début et fin du chemin π
```

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



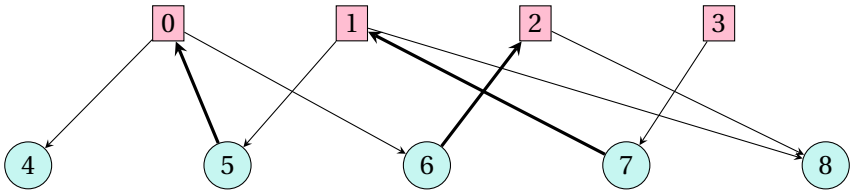
$\Gamma = \emptyset$

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



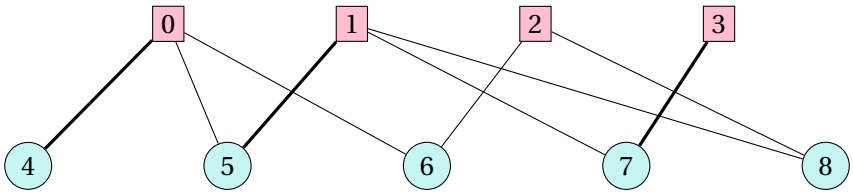
$\Gamma = \{(0,5),(1,7),(2,6)\}$

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.



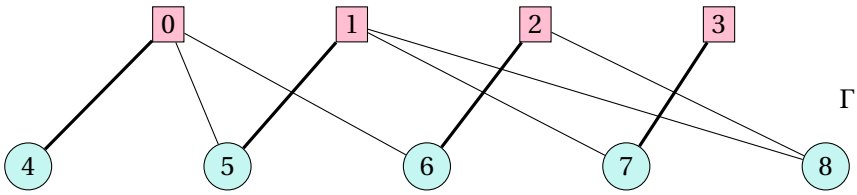
$\Gamma = \{(0,5),(1,7),(2,6)\}$
$\pi = \{3,7,1,5,0,4\}$

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



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

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

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 22 — 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 $w : A \to \mathbb{R}$. 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 stratégie 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.