République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Université A/Mira de Béjaia
Faculté des Sciences Exactes
Département de Recherche Opérationnelle

Mémoire de Master

En
Recherche Opérationnelle

Option
Modélisation Mathématique et Techniques de Décision

Thème

Application de lex-bfs aux graphes triangulés

Présenté par :

AHFIR Massinissa & AKIL Thilleli

Soutenu le 08 Juillet 2019 devant le jury composé de :

Président	M^r K.KABYL	Université de Béjaia
Rapporteur	M^r DJ.TALEM	Université de Béjaia
Examinateur	M^r M.SOUFIT	Université de Béjaia
Examinateur	M^r O.BOUAROURI	Université de béjaia

Promotion 2018/2019.

⁂ *Remerciements* ⁂

Nous remercions Dieu (sans Lui rien n'aurait pu être possible) tout puissant de nous avoir accordé santé, courage et la volonté pour accomplir ce modeste travail.

Nous tenons également à remercier notre Promoteur M^r **TALEM DJAMEL** pour ses précieux conseils, sa disponibilité et ses encouragements qui nous ont poussés à donner le meilleur de nous-même tout au long de la préparation de ce mémoire.

Nous voudrions également remercier le président et les membres de jury d'avoir accepter de juger notre travail et de consacrer leur temps à la lecture et à la correction de ce mémoire.

Nos remerciements les plus vifs vont tout particulièrement à nos parents et à nos familles.

⁕ *Dédicaces* ⁕

Je dédie ce modeste travail :

A ma source de courage, mes trés chers parents,

A toutes ma famille,

A tous mes professeurs,

A tous mes amis.

M. ***Ahfir massinissa***

⁂ *Dédicaces* ⁂

Je dédie ce modeste travail :

A ma source de courage, mes trés chers parents.

A mes deux chers fréres ;KHALEF et AMINE .

A mes soeurs rosa et djamila et leurs familles .

A ma soeur WISSAM ,mes tantes ,ma grand mére et ma grande famille .

A mes ami(e)s en particulier HAYAT, LYDIA,FRAWSEN,SLIMANE.

Mlle. ***AKIL THILLELI***

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION GÉNÉRALE

Un mathématicien confronté à un problème de la vie réelle s'empresse de traduire celui-ci en un problème mathématique, qu'il peut alors tenter de résoudre. Dans sa boîte à outils mathématiques, les graphes peuvent s'avérer fort utiles.

Intuitivement, un graphe est un ensemble de points, dont certaines paires sont reliées. Plus formellement, un graphe est défini par deux ensembles : son ensemble de sommets et son ensemble d'arêtes, une arête étant une paire de sommets reliés. La théorie des graphes introduit ensuite de nombreuses classes de graphes, des familles de graphes qui vérifient certaines propriétés. Les graphes permettent de modéliser de nombreux problèmes dans le domaines des réseaux, par exemple un réseau de transport (un plan de métro est un graphe, où les sommets représentent les stations), un réseau social (chaque sommet représente une personne et une arête relie deux sommets si les personnes correspondantes se connaissent) ou plus simplement un réseau informatique ; mais également des problèmes de chimie ou de génétique. Ces nombreuses applications font de la théorie des graphes un sujet de recherche toujours prolifique, mais revenons d'abord aux origines de la théorie des graphes.

Le problème des ponts de Koenigsberg a été introduit en 1735 par Léonard Euler, considéré comme le fondateur de la théorie des graphes. La ville de Koenigsberg possède sept ponts enjambant la rivière Pregel (Fig. 1) et Euler s'interroge sur l'existence d'une promenade lui permettant de passer par tous les ponts de la ville une et une seule fois, et de revenir à son point de départ. Euler modélise ce problème par un graphe : un sommet est associé à chaque parcelle de terre délimitée par la rivière et une arête est associée à chaque pont les reliant (voir Fig. 2). Ainsi, une promenade passant par chacun des ponts une et une seule fois est alors un cycle eulérien dans ce cas.

FIGURE 1 – Plan de la ville Koenigsberg

FIGURE 2 – modélisation de problème de sept ponts Koenigsberg

le présent travail est réparti en cinq chapitres précédés d'une introduction générale :

- le premier chapitre est dédié aux notions de base utilisé dans ce mémoire sous forme de définitions et préliminaires.
- le deuxième chapitre, complexité algorithmique : rappelle des notions et des résultats de la complexité.
- au troisième chapitre, on a décrit les parcours dans les graphes parfaits ; parcours en largeur ; parcours en profondeur ; parcours lexicographique, et on expliqué par un exemple d'application pour chaque parcours.
  et l'utilisation de lexbfs à la reconnaissance des graphes triangulés.
- au quatrièmes chapitre, on rentre dans l'essentiel de travail ; dont on parle de la résolution de certains problème d'optimisations (clique et stabilité ,coloration)dans les graphes parfaits avec lexbfs
- le sixième chapitre, est réservé aux exemples d'applications (planning des examens),(transport des produits chimiques).

CHAPITRE 1

NOTIONS ET RÉSULTATS PRÉLIMINAIRES

Introduction

Un bon dessin vaut mieux qu'un long discours. Le langage des graphes essaie de mettre en pratique cette idée. bien souvent, en effet, c'est un réflexe naturel qui pousse à abstraire une situation donnée en tracant, sur une feuille de papier, des points pouvant représenter des individus, des localités, des corps chimiques,... et une relation entre eux par des lignes ou des fléches.
L'objet de ce chapitre est de présenter brièvement, certaines définitions de bases ou concepts généraux sur les éléments de la théorie des graphes qui nous seront utiles par la suite.

Le lecteur pourra se référer aux ouvrages «Graph Theory» de Diestel et «Graphs and Hypergraphs» de C. Berge pour plus de détails concernant les notions et généralités sur les graphes.

1.1 Le concept de Graphe

On distingue deux types de Graphes : les graphes orientés et les graphes non orientés. Notre travail se limite uniquement sur les graphes non orientés.
Un **graphe non orienté** G est défini par un couple de deux ensembles (V_G, E_G) : V_G est un ensemble non vide de **sommets** (vertices) de G et E_G désigne l'ensemble de ses **arêtes** (edges) ; on écrit $G = (V_G, E_G)$ ou simplement $G = (V, E)$ s'il n'y a pas d'ambiguité. Un sommet v est représenté par un point $p(v)$ dans le plan ou dans l'espace, tandis que une arête e est définie par un couple de deux sommets u et v , elle est représentée par une courbe reliant les deux points $p(u)$ et $p(v)$; on la note $e = \{u, v\}$ ou $e = uv$. Notons que uv et vu représentent la même arête. Les sommets u, v sont les extrémitś (endpoints) de l'arête $e = uv$; si $u = v$, l'arêtes $e = uu$ est une **boucle** (loop) ; si deux sommets sont reliés par plusieurs arêtes, G est un **multi graphe** (multigraph). Un **pseudo-graphe** (pseudograph) est un graphe ayant des arêtes multiples et des boucles ; un graphe G est **simple** (simple graph) lorsque il ne contient ni boucle ni arêtes multiples, ce qui signifie qu'un sommet n'est pas adjacent à lui même et entre deux sommets il y à au plus

une arête. Les nombres de sommets et d'arêtes de G sont notés $n(G)$ et $m(G)$; ces deux paramétres fondamentaux sont appelés l'**ordre** et la **taille** de G, respectivement. La figure 1.1 présente deux graphes : un pseudo-graphe G d'ordre 4 avec $V(G) = \{u, v, w, x\}$ et $E(G) = \{uv, uw, vx, ww, wv\}$ sont, respectivement, ses ensembles de sommets et d'arêtes ; uv est une arête double ; l'arête ww est une boucle et un graphe simple H d'ordre 5 avec $V(H) = \{1, 2, 3, 4, 5\}$ et $E(H) = \{14, 15, 13, 35, 34, 24, 25\}$ sont respectivement ses ensembles de sommets et d'arêtes.

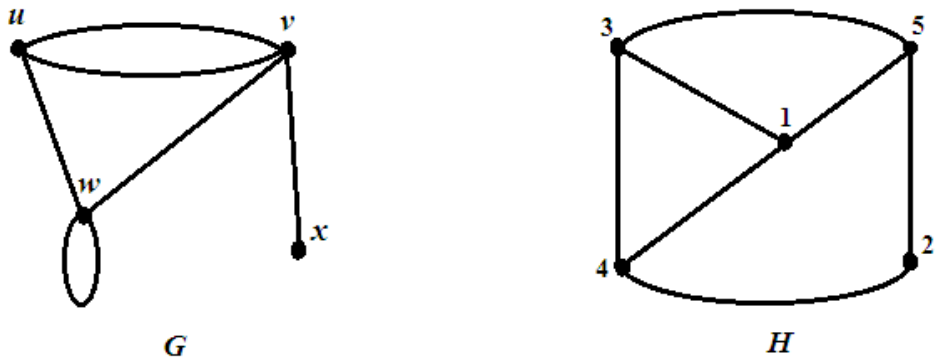


FIGURE 1.1 – Graphe et pseudo-graphe d'ordre 4

Les extrémités d'une arête sont dites **incidentes** à cette arête, et vice versa. Deux sommets incidents à une même arête sont **adjacents** (adjacent vertices) ou **voisins** (neighbors) ; de même, deux arêtes incidentes au même sommet sont dites adjacentes. L'ensemble des sommets adjacents á v, excepte lui-même, est appelé **voisinage ouvert** (open neighborhood) de v, il est noté $N(v) = \{u \in V, vu \in E\}$; Le voisinage **fermé** (closed neighborhood) d'un sommet v, noté $N[v] = N(v) \cup \{v\}$. Le **degré** (degree) d'un sommet v, noté $d_G(v)$, est égal au nombre $|N(v)|$ de ses voisins, c'est-á-dire $deg(v) = |N(v)|$. Un sommet de degré nul est dit **isolé** (isolated vertex). Le plus petit degré d'un graphe G est noté $\delta(G) = min_{v \in V} d(v)$ et son plus grand degré est est noté $\Delta(G) = max_{v \in V} d(v)$; ils sont, respectivement, égaux au degré d'un sommet ayant le moins et le plus de voisins. Un graphe est **régulier** si tous ses sommets ont le même degré, c'est-á-dire $\delta(G) = \Delta(G) = k$: on dit alors que le graphe est régulier de degré k ou $k - régulier$.

Le **complémentaire** (complement) d'un graphe G est le graphe $G^c = (V, E^c)$, c'est-á-dire G^c a les mêmes sommets que G et deux sommets sont adjacents dans G^c si et seulement s'il ne le sont pas dans G.

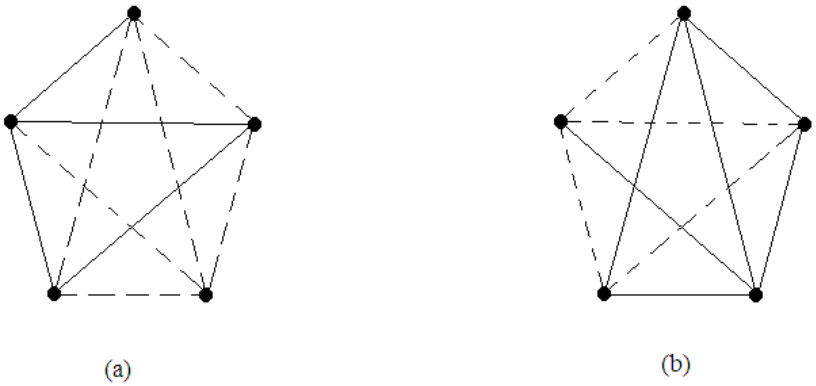


FIGURE 1.2 – (a) Un graphe G, (b) G^c, le complémentaire de G

Une **chaîne** dans un graphe $G = (V, E)$ est une séquence de sommets $\mu = (u_l, u_2, \ldots, u_q)$ telle que $u_i u_{i+1} \in E$. Si $u_1 = u_q$, μ est un **cycle**. Le nombre d'aêtes de la séquence est la longueur de la chaîne μ. Une chaîne et cycle qui ne rencontre pas deux fois le même sommet est dit **élémentaire** ; une chaîne et cycle qui n'utilise pas deux fois la même arête (resp.arc) est dite **simple**. Une *corde* dans une chaîne (resp. cycle) est une arête reliant deux sommets non consécutifs dans cette derniére. Un *trou* (hole) est un cycle d'au moins quatre sommets et sans corde. Un antitrou (antihole) est le complémentaire d'un trou.

Un graphe est **connexe** (connected) si, pour tout couple u, v de deux sommets, il y a une chaîne $\mu[u, v]$ (une chaîne allant de u á v) ; dans le cas contraire, le graphe est séparé ou non connexe. Autrement dit, un graphe est séparé si son ensemble de sommets peut être partitionné en sous-ensembles connexes appelés **composantes connexes** (connected component) de G. Un sommet (resp. ensemble) d'**articulation** (cutvertex (resp. cutset)) est un sommet (resp. ensemble de sommets) tel que sa suppression augmente le nombre de composantes connexes. Parfois, on dit **séparateur** au lieu d'ensemble d'articulations Un graphe est **fini** si son ensemble de sommets et son ensemble d'arêtes sont tous deux finis. Dans la suite, sauf la mention du contraire, nous supposons que tous les graphes qui seront considerés sont finis et simples.

Deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ sont **isomorphes** (isomorphic) s'il existe une bijection $f : V_G \longrightarrow V_H$ telle que : $\forall u, v \in V_G$, $uv \in E_G$ si et seulement si $f(u)f(v) \in E_H$, c'est-á-dire deux sommets u et v sont adjacents dans G si et seulement si leurs images $f(u)$ et $f(v)$ sont adjacentes dans H. On note alors $G \simeq H$.

Ainsi, deux graphes sont isomorphes s'ils sont identiques ou ils ont la même structure, c'est-á-dire ils différent seulement par les noms des sommets et des arêtes. Les deux graphes de la figure ? sont isomorphes. En effet, l'application f telle que $f(a) = 2, f(b) = 1, f(c) = 4, f(d) = 3, f(e) = 5$ est un isomorphisme qui conserve les relations d'adjacences.

FIGURE 1.3 – Deux graphes isomorphes

1.1.1 Quelques graphes particuliers

Nous présentons ici rapidement quelques classes de graphes particuliéres qui occupent une place majeure en théorie des graphes et que nous rencontrerons souvent par la suite, avec leurs propriétés et les liens entre elles.

Un graphe G est **complet** (complet graph) si entre deux sommets quelconques, il y a une arête ; K_n dénote un graphe complet d'ordre n. Un graphe **vide** est un graphe dans lequel l'ensemble d'arêtes est vide. Un graphe G est **biparti** (bipartite graph) si l'ensemble des sommets peut être partitionné en deux sous-ensembles V_1 et V_2 de sorte que toute arête ait une extrémité dans V_1 et une autre extrémité dans V_2 ; on le note $G = (V_1, V_2, E)$. Un graphe biparti est dit complet si tout sommet de V_1 est adjacent á tous les sommets de V_2. Un graphe biparti complet avec $|V_1| = p$, $|V_2| = q$ se dénote par $K_{p.q}$. Une étoile est un graphe biparti complet $K_{p,q}$ avec $p = 1$ ou $q = 1$. Pour deux graphes connexes G_1 et G_2.

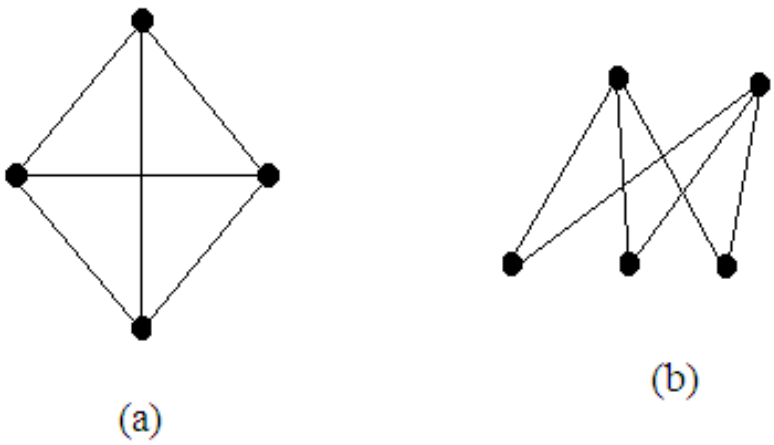


FIGURE 1.4 – (a) Un graphe complet K_4, (b) Un graphe biparti complet $K_{2,3}$.

Un cycle sans corde (ou trou) d'ordre $n \geqslant 3$ est un graphe connexe dont tous les sommets sont de degré 2 ; on le note C_n ou $n - cycle$. Une chaîne (path) d'ordre n est un graphe connexe dont tous les sommets sont de degré 2, sauf les deux extrémités de la chaîne qui sont de degré 1 ; on la note P_n. Une **forêt** (forest) est un graphe sans cycle. Un **arbre** (tree) est un graphe connexe sans cycle.

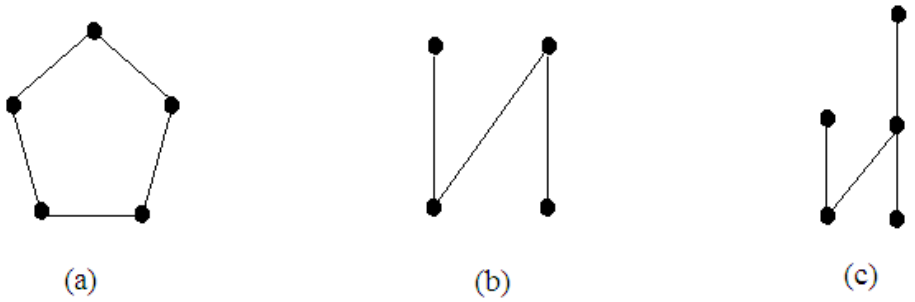


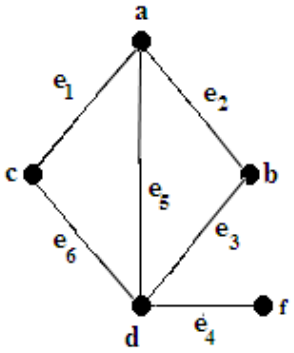
FIGURE 1.5 – (a) Un C_5, (b) Un P_4, (c) Un arbre

1.1.2 Représentations des graphes en machine

Il existe deux facons classiques de représenter un graphe $G = (V, E)$: par un ensemble de listes d'adjacences, ou par une matrice (matrice d'adjacences ou matrice d'incidence)

$$m_{ij} = \begin{cases} 1 & \text{s'il exite un arc orienté } (i,\, j)\,;\, i = \mathrm{I}(u)\,;\, j = \mathrm{T}(u) \\ 0 & \text{sinon} \end{cases}$$

. La représentation par listes d'adjacences est souvent préférée, car elle fournit un moyen



	a	b	c	d	f
a	0	1	1	0	0
b	1	0	0	1	0
c	1	0	0	1	0
d	1	1	1	1	1

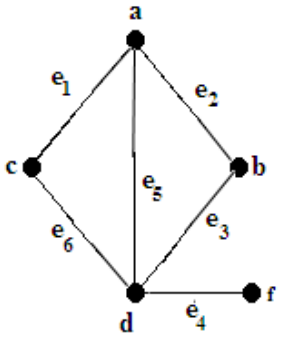
FIGURE 1.6 – Graphe non orienté et sa matrice d'adjacence

peu encombrant de représenter les graphes peu denses (ceux pour qui m est trés inférieur á n^2).
La **matrice d'adjacence** (adjacency matrix) d'un graphe d'ordre n est la matrice carrée A de taille $n \times n$ telle que $a_{ij} = 1$ s'il existe une arête entre les sommets i et j, et $a_{ij} = 0$ sinon. Une **liste d'adjacence** (adjacency list) est une structure de données dans laquelle on associe á chaque sommet sa liste de voisins.
On peut remarquer que cette matrice est toujour symétrique et á diagonale nulle pour un graphe simple. Le résultat principal concernant les matrices d'adjacence est le théoréme suivant :

Théoreme 1. *Soit G un graphe non-orienté de matrice d'adjacence A. Le nombre de chaines de longueur k (c'est á dire ayant k arêtes) joignant le sommet i_0 au sommet j_0 est donné par $a^k_{i_0 j_0}$ de la matrice $A^k = A \times A \ldots A$, k fois, de terme général (a^k_{ij}).*

La matrice d'incidence sommet-arêtes est une matrice $n \times m$ associée á un graphe non orienté et qui est définie par :

$$b_{ij} = \begin{cases} 1 & \text{si le sommet } i \text{ est l'une des extrémité de l'arête } u_j \\ 0 & \text{sinon} \end{cases}$$



	e_1	e_2	e_3	e_4	e_5	e_6
a	1	1	0	0	0	0
b	0	1	1	0	0	0
c	1	0	0	0	0	1
d	0	0	1	1	1	1
f	0	0	0	1	0	0

FIGURE 1.7 – Graphe non orienté et sa matrice d'incidence

1.1.3 Sous-graphe

Etant donné un graphe G, dans de nombreuses applications de théorie des graphes, on cherche á déterminer si un graphe donné a un **sous-graphe** (subgraph) avec certaines propriétés voulues. Il y a deux maniéres naturelles d'obtenir des graphes plus petits á partir de G : la suppression de sommets et la suppression des arêtes. Plus généralement, un graphe G' est un sous-graphe d'un graphe G si $V'_G \subseteq V_G$ et $E'_G \subseteq E_G$; G est alors un **sur-graphe** (supergraph) de G'. Ainsi nous disons que G contient G' ou que G' est contenu dans G.

Un graphe **partiel** (on dit aussi sous-graphe **couvrant**) (spanning subgraph) d'un graphe $G = (V, E)$ est un sous-graphe $G' = (V, E')$ tel que $E' \subset E$, c'est-á-dire G' est obtenu à partir de G par suppressions d'arêtes uniquement. Si S est l'ensemble des arêtes supprimées, on écrit $G' = G \setminus S$. Par exemple, tout graphe simple est un sous-graphe couvrant d'un graphe complet, les chaînes et cycles Hamiltoniens (c'est-á-dire les chaînes, respectivement, les cycles qui passent par tous les sommets de G)sont des sous-graphes couvrants.
Un sous-graphe **induit** (induced subgraph) par un sous-ensemble de sommets A de V_G est le sous-graphe noté G_A dont l'ensemble de sommets est A et l'ensemble d'arêtes est constitué de toutes les arêtes de G qui ont leur deux extrémités dans A. Le sous-graphe induit G_A est le sous-graphe obtenu par la suppression consécutive des sommets de $V_G \setminus A$.

Un **stable** (stable set ou independent set) dans G est un ensemble de sommets deux á deux non adjacents, c'est-à-dire un sous-graphe sans arêtes. Une **clique** (clique) dans G est un ensemble de sommets deux á deux adjacents, c'est-à-dire un sous-graphe complet. Une **biclique** (biclique) dans G est un sous graphe biparti complet.

Remarque 1. *Ainsi un graphe complet (resp. graphe biparti-complet) est aussi une clique (resp. biclique). Une clique dans un graphe est un stable dans son complémentaire et inversement.*

Notons que tout ensemble maximum est aussi maximal ; mais, un ensemble maximal n'est pas nécessairement maximum. En effet, dans la figure 2.8, l'ensemble des sommets $S = \{b, c\}$ induit un stable maximal d'ordre 2, car S n'est pas strictement inclue dans un autre stable ; cependant, ce dernier n'est pas maximum, car le stable induit par l'ensemble $S' = \{a, b, e\}$ est un stable d'ordre 3, donc contient plus d'éléments que S, et aucun autre stable ne peut avoir une cardinalité supérieure á 3 ; par conséquent $G_{S'}$ est un stable maximum.

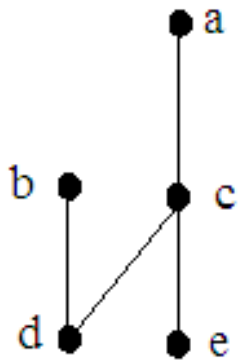


FIGURE 1.8 – Un stable maximal et un stable maximum

Définition 1.1.1. *On dit qu'une classe de graphes C est héréditaire (hereditary class) si tout sous-graphe induit d'un graphe de C appartient lui-même á la classe C (c'est-á-dire C est close par suppression de sommets).*

On peut voir facilement que tout sous graphe induit d'une clique d'un arbre ou d'un stable est une clique respectivement un arbre , un stable, donc chacun de ses classes de graphes est héréditaire.

Conlusion

On a introduit les définitions et les conceptes de bases de la théorie des graphes utilisés dans les prochains chapitres

CHAPITRE 2

PROBLÉME, ALGORITHME ET COMPLEXITÉ

Introduction

Nous introduisons ici dans ce deuxième chapitre, quelques notions de complexité. Pour toute notion non développée ici, nous renvoyons le lecteur intéressé vers l'excellent ouvrage de Michael R. Gary et David S.Johnson.

2.1 Probléme et algorithme

Un problème est une question générique, c'est-à-dire qui s'applique à un ensemble d'éléments. Une instance (une donnée) du probleme est une question posée pour un élément particulier de cette ensemble. Par exemple determiner si un entier naturel est premier ou calculer une clique de cardinalité maximal sont deux problemes. Dès qu'on fixe un entier, on aura une instance du premier probleme (connu sous le non du probleme de primalité) : 19 est-il un nombre premier ? est une instance ayant une reponse oui. De meme, la donnee d'un graphe G va constituer l'instance " calculer la clique maximum pour le graphe G". Un probléme est donc composé de deux éléments : une entrée (ou instance) et une question ou une tâche á réaliser. Les deux exemples précédents se reformulent ainsi :

1. Primalité :
   - entrée : un entier N ;
   - question : N est-il premier ?
2. clique de taille maximale :
   - entrée : Un graphe $G = (V, E)$;
   - tâche : Trouver une clique de G avec un maximum possible de sommets.

On distingue ainsi deux types de problémes : ceux qui consistent á répondre par oui ou par non á une question donnée (dans l'exemple précédent, déterminer si un entier est premier), qu'on appelle problémes de décision ; et ceux qui consistent á maximiser (ou minimiser) une certaine fonction sur un ensemble fini (dans l'exemple précédent, trouver $max(f)$ sur X, avec X est l'ensemble de toutes les cliques de G et f est une application sur X, qui á toute clique $C \in X$, $f(C)$ est le nombre de sommet de la clique C), qu'on appelle problémes d'optimisation. Ainsi, on obtient la définition suivante :

Définition 2.1.1.
- *Un probléme d'optimisation combinatoire consiste á chercher le minimum (resp. le maximum) x^* d'une certaine application f sur un ensemble fini X á valeur le plus souvent entiéres ou réelles : $f(x^*) = min_{x\in X} f(x)$ (resp ; $f(x^*) = max_{x\in X} f(x)$). La fonction f est une fonction-objectif ou économique, l'ensemble X est appelé ensemble des solutions réalisables.*
- *Un probléme de décision (on dit aussi langage) consiste á chercher dans un ensemble fini X s'il y a un élément x vérifiant une certaine propriété P. Ainsi un probléme de décision est une application sur X á valeur dans $\{0, 1\}$ telle que : $f(x) = 1$ si x vérifie P et $f(x) = 0$ sinon.*

Notons que tout probléme d'optimisation peut être transformé en un probléme de décision équivalent. Par exemple, le probléme du voyageur de commerce qui cherche, dans un graphe dont les arêtes sont étiquetées par des coûts, á trouver un cycle de coût minimum, passant une fois par chaque sommet, peut s'énoncer en un probléme de décision comme suit : Existe-t-il un cycle hamiltonien (passant une fois par chaque sommet) de coût inférieur á k ?

Comme on l'a signalé plus haut, résoudre un probléme P signifie trouver un algorithme (ou programme) A qui prend en entrée une donnée d du probléme P et produit en sortie un résultat $A(d)$.
Par exemple, un algorithme qui résout le probléme de du stable maximal, quand il s'applique sur le graphe de la figure ci-dessous, produit le résultat $S = \{b, e\}$ qui est un ensemble de sommets deux á deux non adjacents maximal.

Définition 2.1.2. *Un algorithme est une méthode permettant de résoudre un probléme donné en un temps fini. Il n'est pas un programme, il décrit une méthode qui sera ensuite implémentée dans un langage de programmation.*

Si un probléme doit être résolu par un algorithme, il faut que ses instances soient représentées d'une façon accessible á cet algorithme. Donc on a besoin de coder des entiers, des listes, des arbres, des graphes ... Le par des chaines de caractére qui soient compréhensibles par l'ordinateur. Le plus souvent, ces instances sont représentées par des chaines en 0 et 1.

2.2 complexité temporelle d'un algorithme

A tout algorithme est associée une fonction de complexité temporelle qui indique le temps de calcul á prévoir pour obtenir la solution. Pour une instance (une donnée) x d'un probléme donné p, le temps d'exécution de l'algorithme pour résoudre le probléme p sur l'entrée x ne s'exprime pas en seconde, mais il est donné par le nombre fini d'opérations élémentaires nécessaires jusqu'á l'affichage de la solution. Parmi les opérations élémentaires, on a les opérations arithmétiques ou logiques, les affectations, les comparaisons, ... Toute tâche qui se réalise en un temps constant par les calculateurs usuels indépendamment de leurs puissances est opération élémentaire.
En général, le temps d'exécution d'un algorithme sur l'entrée x dépend de la taille de x mais aussi de sa nature, et il peut varier sur des instances de même taille. En effet,

rechercher une valeur dans un tableau demande plus de temps dans un tableau dont les éléments sont désordonnés que dans le même tableau trié . Pour cette raison, on définit la complexité d'un algorithme en considérant la pire instance possible parmi toutes les instances de taille n, c'est-á-dire celle demandant le plus de ressources.

Définition 2.2.1. *Soient p un probléme et A un algorithme qui résout p. Notons* $I_{(p,n)} = \{x/x$ *est une instance de p et* $|x| = n\}$ *et* φ_A *est une application qui á toute instance x fait associer le temps d'exécution de A sur x . La fonction de (ou tout court la complexité) de l'algorithme A est une application* c_A *définie sur l'ensemble des entiers naturels par :*

$$c_A(n) = max_{I_{(P,n)}}\varphi(n).$$

$c_A(n)$ *est le nombre maximum d'opérations élémentaires pris par A sur les instances de taille n.*

Remarque 2. *Un algorithme de complexité polynômiale est quelquefois appelé bon ou efficace, et le probléme qu'il résout est dit facile.*

La complexité en temps d'un algorithme est habituellement exprimée á l'aide de la notation O. Sa définition est la suivante.

Définition 2.2.2. *(Notation O) Soient f et g deux fonctions* $f; g : N \to R_+$. *On dit que* $f \in O(g)$ *(on dit aussi f est un grand O de g) lorsqu'il existe un entier* n_0 *et une constante reelle c tel que pour tout* $n \geq n_0$, $f(n) \leq cg(n)$.

Intuitivement, cela signifie que g devient plus grande que f á partir d'un certain entier n_0, á une constante multiplicative prés. $O(g)$ est l'ensembles des fonctions d'ordre supérieur á g pour n assez grand. Par abus de langage, on écrira $f = O(g)$ lá oú on devrait écrire $f \in O(g)$.

Exemple 1. *: Soit* $f(n) = 6n^4 - 2n^3 + 5$. *Choisissons* $n_0 = 1$. *Alors pour tout* $n \geq n_0$, *on a* $6n^4 - 2n^3 + 5 \leq 6n^4 + 2n^4 + 5n^4 = 13n^4$ *Ainsi, en prenant* $c = 13$, *on a* $f = O(n^4)$. *Autrement dit, á un facteur constant prés,* $f(n)$ *ne croît pas plus rapidement que* n^4. *Il est facile de voir qu'un polynôme* $P(n)$ *de degré k est toujours en* $O(n^k)$.

Définition 2.2.3. *(Notations* Ω, Θ*)*
Soient f et g deux fonctions $f; g : \mathbb{N} \to \mathbb{R}_+$.

- *On note* $f = \Omega(g)$ *lorsque il existe un entier* n_0 *et une constante reelle* $c^{'}$ *tel que pour tout* $n \geq n_0$, $f(n) \geq c^{'}g(n)$.
- *On note* $f = \Theta(g)$ *lorsque* $f(n) = O(g)$ *et* $f(n) = \Omega(g(n))$, *c'est-á-dire lorsque il existe un entier* n_0 *et deux constantes reelles c et* $c^{'}$ *tel que* $cg(n) \leq f(n) \leq c^{'}g(n)$.

$\Omega(g)$ (respectivement $\Theta(g)$) est l'ensemble des fonctions d'ordre inférieur (respectivement équivalentes) á g pour n assez grand.

Exemple 2. *Soit* $f(n) = n^3 sinn$, *on a* $\forall n$, $-n^3 \leq f(n) \leq n^3$. *Ainsi* $f(n) = \Theta(n^3)$.

Définition 2.2.4. *Un algorithme A est dit polynômial si sa complexité est majorée par un polynôme en la taille des données, c'est-á-dire il existe un entier k tel que* $c_A \in O(n^k)$. *Dans le cas contraire, il est dit exponentiel.*

2.3 complexité d'un probléme

Nous allons maintenant nous intéresser á l'étude de la difficulté intrinsé que des problémes de décision, ce que l'on appelle complexité des problémes (non pas des algorithmes), et on va les classer selon la complexité des algorithmes les résolvant. Un grand nombre d'entre eux sont des problémes faciles car on connait des algorithmes polynômiaux pour les résoudre. Cependant, il existe aussi un grand nombre de problémes pour lesquels on ne connait pas d'algorithmes polynômiaux. On ne peut pas prouver qu'il n'en existe pas, mais on peut cependant montrer que l'existence d'un algorithme polynomial pour l'un d ?entre eux impliquerait l'existence d'un algorithme polynomial pour presque tous les problémes.

Définition 2.3.1. *La complexité d'un probléme est la complexité du meilleur algorithme qui permet de le résoudre. Si cet algorithme est polynômial, le probléme est dit facile, autrement le probléme est difficile.*

Définition 2.3.2. *La classe P est l'ensemble de tous les problémes de décision pour lesquels il existe un algorithme polynômial.*

Pour prouver qu'un probléme est dans P, on décrit un algorithme polynômial résolvant ce probléme.

Exemple 3. *Soit le probléme du plus court chemin entre deux sommets dans un graphe orienté et valué par des coûts positifs.*

- *Entrée : un graphe valué $G = (V, E, u)$, deux sommets s et t, et une constante k positive ;*
- *Sortie : exite-t-il un chemin allant de s á t de longueur au plus k.*

Ce probléme est dans la classe P car l'algorithme de Dijkstra, qui est un algorithme polynômial, peut résoudre ce probléme.

Cependant, on ne sait pas si les problémes de décision associés au probléme du cycle hamiltonien au problé du stable maximum appartiennent á P ou pas. Nous allons maintenant introduire une autre classe, notée NP, qui contient ces deux problémes. Pour les problèmes de la classe NP, nous n'exigeons pas un algorithme polynômial, en revanche nous demandons qu'il y ait, pour chaque instance "oui", un certificat (une solution devinée) qui puisse être vérifié en temps polynômial. Par exemple, pour le probléme du cycle Hamiltonien, la solution est un cycle Hamiltonien, donc un cycle hamiltonien constitue un certificat. Comme il est facile de vérifier si un ensemble d'arêtes donné forme un cycle hamiltonien, alors ce probléme est dans NP.
La même chose pour le probléme du stable de cardinalité au plus égale á k. En effet, ici la solution est un stable, et si l'on dispose d'un stable S (S est un certificat), on peut vérifier en un temps polynômial que S est un stable et aussi si $|S| \geq k$, ce qui implique que ce probléme est dans NP.

Définition 2.3.3. *La classe NP est l'ensemble de tous les problémes de décision pour lesquels toute solution proposée est vérifiable par un algorithme polynômial.*

Remarque 3.
- *Les problémes de la classe NP sont ceux que l'on peut rèsoudre par énumération compléte de toutes les solutions possibles (méthode "brutale") et en les testant á l'aide d'un algorithme polynômial.*

- *On a clairement $P \subseteq NP$. En effet, si on peut résoudre un problème par un algorithme polynômial, alors on peut aussi vérifier en temps polynômial que la solution fournie est bien une solution du même problème.*
- *La question de savoir si $P = NP$ est un probléme ouvert, le plus important, de la théorie de la complexité. Cela revient á savoir si le fait de chercher une solution est aussi simple que de vérifier une solution. De nombreuses personnes pensent que $P \neq NP$.*

Dans l'ensemble NP, on trouve un sous ensemble noté NP-complet constitué par les problèmes les plus difficiles de NP. Les problémes $NP - complet$ sont tous équivalents en termes de difficultés. Pour affirmer que certains problémes sont les plus difficiles, il faut pouvoir comparer les problèmes entre eux. On définit pour cela la notion de réduction polynômiale sur la classe NP.

Définition 2.3.4. *Soient p et p' deux problémes de décision. On dit que p se transforme (ou se réduit) polynômialement en p' s'il existe un algorithme polynômial transformant toute instance x de p en une instance x' de p' admettant la même réponse que x. On écrit alors $p \prec p'$. Autrement dit, les instances "oui" sont transformées en instances "oui", et les instances "non" sont transformées en instances "non".*

L'importance du concept de réduction polynômiale est principalement justifiée par la proposition suivante :

Proposition 1 (Optimisation combinatoire)**.** *Si p se réduit polynômialement á p' et s'il existe un algorithme polynômial pour p', alors il existe un algorithme polynômial pour p.*

Remarque 4.
- *La relation $\prec$ est transitive et $p \prec p'$ signifie que p n'est pas plus difficile que p'.*
- *Ainsi, "p est polynômialement réductible á p'" signifie que si l'on connait un algorithme polynômial résolvant p', on en déduit que $p \in P$. En effet, on traduit les instances de p en instances de p', puis on résout p' et enfin on retraduit les solutions de p' en solutions de p, tout cela se fait en temps polynômial.*

Définition 2.3.5. *Un probléme de decision q est dit $NP - complet$ si :*

1. $q \in NP$
2. $\forall p \in NP,\ p \prec q$

Il est facile de voir que la restriction de la notion de "réduction polynô-miale" sur les problémes $NP - complet$ est une relation d'équivalence, donc s'il existe un algorithme polynômial pour un seul élément de la classe $NP - complet$, on pourait en déduire un algorithme polynômial pour n'importe quel autre élément dans la même classe, et d'aprés la proposition précédente, on aurait aussi $P = NP$!

Il a été demontré que de nombreux problémes naturels réputés difficiles sont $NP - complets$. Cook [$Coo71$] et, indépendamment, Levin [$Lev73$] ont donné de tels problémes au début des années 1970, complétés ensuite par Karp [$Kar72$] notamment. Ici nous allons enoncé le théoréme du probléme SAT, la satisfaisabilité de formules booléennes.

2.4 Conclusion

Dans ce chapitre on introduit les définitions et concepts de base de théorie de complexité

CHAPITRE 3

PARCOURS DANS LES GRAPHES PARFAITS

Introduction

Introduite dans les années 60 par Claude Berge, la notion de graphe parfait a depuis été d'une grande importance en théorie des graphes. De nombreuses classes sont en effet incluses dans la classe des graphes parfaits. Elle permet de généraliser un certain nombre de résultats connus sur ces classes particulières. Les graphes parfaits ont une importance majeure du fait des nombreuses applications dans le monde réel qu'on leur trouve. De plus, les graphes parfaits ont étéétudiés intensivement dans le but de prouver les deux fameuses conjectures (faible et forte) des graphes parfaits. C'est chose faite depuis 2002, grace à la preuve de Chudnovsky, Robertson et Seymour. Nous allons ici survoler quelques bases de la théorie des graphes parfaits. Nous nous intéresserons à plusieurs classes différentes de graphes qui se trouvent être des graphes parfaits et regarderons plus précisément les propriétés de coloriage de ces graphes ; l'intérét de chacune de ces classes de graphes sera mis en évidence. Nous présenterons une preuve relativement courte de la conjecture faible des graphes parfaits, d'ecouverte en 1972 par Laszlo Lovasz. Enfin nous présenterons quelques résultats intéressants concernant les graphes parfaits [2].

3.1 Graphes parfaits

Depuis la formulation des conjectures jusqu'á la démonstration du théoréme fort, l'intérêt pour les graphes parfaits n'a cessé de croitre. Il n'est pas retombé non plus aprés la publication de la preuve puisque la trés grande technicité et la longueur de celle-ci laissent espérer l'existence d'une preuve plus courte et qui en renforce la compréhension.

Pour un graphe simple G, $\alpha(G)$ est le cardinal d'un stable maximum, appelé le nombre de **stabilité** de G ; $\theta(G)$ désigne le nombre minimum de cliques qui couvrent V ; $\gamma(G)$ est le nombre minimum de couleur nécessaire pour colorier les sommets de G de sorte que deux sommets adjacents aient deux couleurs différentes, on l'appelle le nombre **chromatique** de G ; $\omega(G)$ est le nombre maximum de sommets qui forment une clique.
Un graphe **parfait** (perfect graph) G est un graphe vérifiant $\gamma(H) = \omega(H)$ pour tout sous-graphe induit H de G. Il est aujourd'hui bien connu que le complémentaire d'un graphe parfait est parfait (*conjecture faible* des graphes parfaits formulée par *C. Berge* et

démontrée par *Lovász*).
Un graphe G est de **Berge** (Berge graph) s'il ne contient ni trou impair ni antitrou impair comme sous-graphe induit. Cette définition constitue la deuxiéme conjecture de Berge qui s'appelle conjecture forte des graphes parfaits et qui á êté demontrée en 2002 par :

Théoreme 2. *(Chudnovsky, Robertson, Seymour et Thomas (2002) [10]) Un graphe est parfait si et seulement s'il est de Berge.*

Un graphe **cordal** ou **triangulé** (chordal graph) est un graphe ne contenant pas de trou. De maniére équivalente, le graphe ne contient pas de cycle induit de longueur 4 ou plus. Ainsi tout sous-graphe induit d'un graphe cordal est cordal. Les graphes complets et les arbres sont des exemples simples de graphes cordaux. Le théoréme suivant, dû essentiellement á *Hajnal* et á *Suranyi* (1958), précise que les séparateurs minimaux dans un graphe cordal sont des cliques :

Théoreme 3. *(bondy) Soit G un graphe cordal connexe qui n'est pas complet, et soit S un séparateur minimal de G. Alors S est une clique de G.*

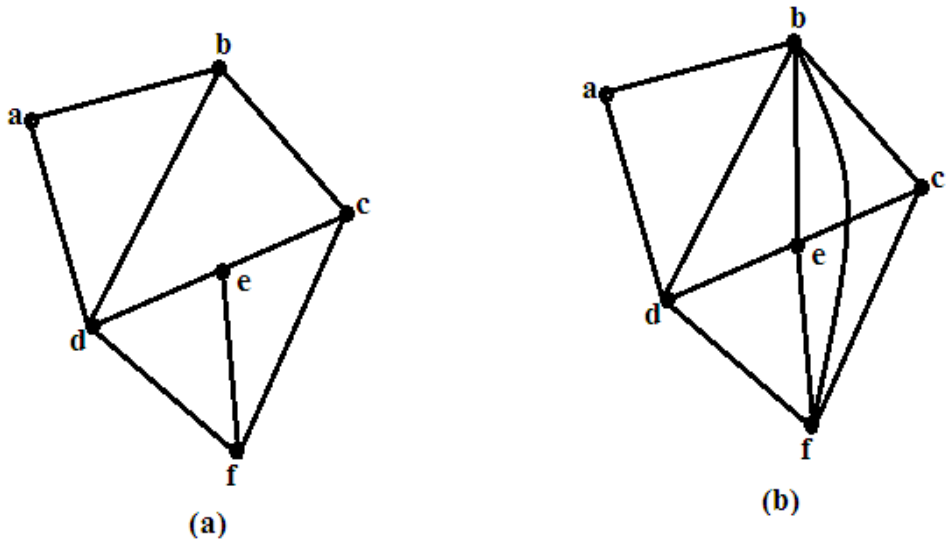


FIGURE 3.1 – Un graphe triangulé

Le graphe de la figure 3.1(a) n'est pas un graphe triangulé, car il contient deux trous : (b, d, f, c) et (b, d, e, c). Le graphe de la figure 3.1(b) qui est obtenu en rajoutant les deux arêtes bf et be á celui de la figure 3.1(a) est un graphe triangulé. Le sommet a est simplicial dans les deux figure, tandis que le sommet c n'est pas simplicial dans la figure (a) mais il est simplicial dans la figure (b).

Proposition 2. *(Diestel) Les graphes cordaux sont parfaits.*

Un graphe $G = (V, E)$ est un graphe de **comparabilité** s'il existe une orientation O transitive et anti-symétrique des arêtes de G, c'est-á-dire pour toute arête $uv \in E$, soit $(u, v) \in O$, soit $(v, u) \in O$; et si $(u, v) \in O$ et $(v, w) \in O$ alors $(u, w) \in O$.
Un graphe de comparabilité $G = (V, E)$ est dit graphe de **co-comparabilité** si son complémentaire G^c est aussi un graphe de comparabilité.

FIGURE 3.2 – Graphe de comparabilité

Il est facile de voir que le graphe obtenu aprés l'orientation d'un graphe de comparabilité est un graphe sans circuit dont la réduction transitive est un diagramme de Hass d'une certaine relation d'ordre définie sur V.

Théoreme 4. *(element de theorie p253) Les graphes de comparabilité sont parfaits.*

Un graphe $G = (V, E)$ est d'**intervalles** (interval graph) s'il existe une famille de n intervalles $I = \{I_1, I_2, ..., I_n\}$ dans $\mathbb{R}$ et une bijection de V dans I qui, á tout $v \in V$, fait associer un intervalle $I_v \in I$ telles que $uv \in E$ si et seulement si $I_u \cap I_v \neq \emptyset$. Autrement dit, G est le graphe d'intersection d'un ensemble d'intervalles de la droite réelle, c'est-á-dire chaque sommet représente un intervalle et une arête relie deux sommets lorsque les intervalles correspondants s'intersectent. La figure 4.2 montre un graphe d'intervalle.

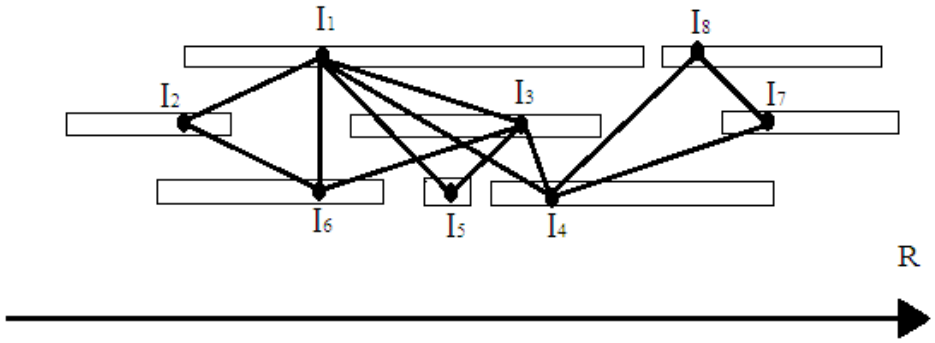


FIGURE 3.3 – La représentation d'un graphe d'intervalles

3.2 Les parcours DFS, BFS et LexBFS

Les parcours servent comme outils pour étudier une propriété globale d'un graphe, comme la connexité, la forte connexité (dans les graphes orientes) et reconnaissances de certaines classes graphes ....

Nous allons étudier dans cette section les deux principales stratégies d'exploration : le parcours en largeur consiste à explorer les sommets du graphe niveau par niveau, à partir d'un sommet donné et le parcours en profondeur consiste, à partir d'un sommet donné, à suivre un chemin le plus loin possible, puis à faire des retours en arrière pour reprendre tous les chemins ignorés précédemment.

Définition 3.2.1. *On appelle* ***parcours*** *d'un graphe,tout procédé déterministe qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter. Le problème de parcours consiste à déterminer un ordre sur les visites des sommets.*

Le sommet de départ, fixé à l'avance, dont on souhaite visiter tous les descendants est appelé **racine** de l'exploration.
Propriété :

Un parcours de racine r est une suite L de sommets telle que :
- r est le premier sommet de L.
- chaque sommet apparaît une fois et une seule dans L.
- tout sommet sauf la racine est adjacent à un sommet placé avant lui dans la liste.

Structures de données

1. **File** : Une File est une structure de données basée sur le principe du "Premier entré, premier sorti `FIFO`, cequi veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.
2. **Pile** : Une pile est une structure de données basée sur le principe du "dernier arrivé, premier sorti `LIFO` , ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.

Remarque 5. *La notion d'exploration peut être utilisée dans les graphes orientés comme non-orientés. Dans la suite, nous supposerons que le graphe est non-orienté. L'adaptation au cas des graphes orienté s'effectue sans aucune difficulté.*

3.2.1 Parcours en profondeur DFS (Depth-First Search)

La Recherche en Profondeur correspond à prendre une structure de PILE, c'est à dire une liste LIFO Last In/First Out (dernier entré/premier sortie). Le principe du parcours en profondeur d'un graphe (orienté ou non) est celui du parcours d'un labyrinthe : on va de sommet en sommet en marquant au fur et à mesure les sommets visités. La visite se poursuit le plus loin possible tant qu'il reste des sommets accessibles non encore marqués. Quand on atteint un sommet v dont tous les voisins ont été déjà marqués, alors on revient au sommet précédant v dans la visite. Autrement dit, On parcoure tous les sommets d'un graphe à partir d'un sommet v donné, à suivre un chemin le plus loin possible, puis à faire des retours en arrière pour reprendre tous les chemins ignorés précédemment [12].

Remarque 6. *Dans un parcours en profondeur on applique la règle "Dernier marqué, premier exploré", i.e on explore les sommets dans l'ordre inverse de celui utilisé pour les marquer. La complexité temporelle d'un parcours en profondeur est d'ordre* $O(n+m)$.

Dans l'algorithme de parcours en profondeur on procède au coloriage des sommets, on utilise une pile (P).

- Initialement, tous les sommets $v \in V$ sont coloriés à blanc (traduisant le fait qu'ils n'ont pas encore été découverts "parcourus").
- Lorsqu'un sommet v_i est "découvert" (autrement dit, quand on arrive pour la première fois sur ce sommet), il est colorié en gris. Le sommet reste gris tant qu'il reste des successeurs de ce sommet qui sont blancs (qui n'ont pas encore été découverts).
- Un sommet v_i est colorié en noir lorsque tous ses successeurs sont gris ou noirs (lorsqu'ils ont été tous découverts).

Pratiquement on va utiliser une pile au coloriage en noir dans laquelle on va stocker tous les sommets gris : un sommet est mis dans la pile dès qu'il est colorié en gris. Un sommet gris dans la pile peut faire rentrer dans la liste ses successeurs qui sont encore blancs (en les coloriant en gris). Quand tous les successeurs d'un sommet gris de la pile sont soit gris soit noirs, il est colorié en noir et il sort de la pile.

La structures des données utilisées :

• On utilise une pile P, pour laquelle on suppose définis les opérations :

- *init_pile (P)* qui initialise la pile P à vide.
- *empile* $(P; v)$ qui ajoute v au sommet de la pile P.
- *depile* $(P; v)$ qui enlève v du sommet de la pile P.
- *est_vide* (P) qui retourne vrai si la pile P est vide et faux sinon.
- *sommet* (P) qui retourne le sommet v au sommet de la pile P.

• On utilise deux tableaux :

- Un tableau T qui associe à chaque sommet le sommet qui l'a fait entrer dans la pile,
- Un tableau *couleur* qui associe à chaque sommet sa couleur (blanc, gris ou noir).

• On va en plus mémoriser pour chaque sommet si :

- $dec[v_i]$ = date de découverte de v_i (passage en gris).
- $fin[v_i]$ = date de fin de traitement de v_i (passage en noir) où l'unité de temps est un itération.
- *tps* la variable dont La date courante est mémorisée.

Programme du parcours en profondeur DFS :

Result:
T arborescence couvrante ;
dec tableau des dates de découverte ;
fin tableau des dates de fin de traitement ;
Input:
- *couleur* tableau des couleurs des sommets ;
tps date courante ;
P pile ;
- *init_pile*(P) ;
- $i \leftarrow 0$;
while *(tout sommet* $v_i \in V$*)* **do**
  $T[v_i] \leftarrow nil$;
  couleur $[v_i] \leftarrow$ **blanc** ;
  $d[v_i] \leftarrow \infty$;
  $i \leftarrow i + 1$;
end
$tps \leftarrow 0$;
$dec[v_0] \leftarrow tps$;
empile$(P; v_0)$;
couleur$[v_0] \leftarrow$ **gris**;
while *(est_vide(P) =* ***faux****)* **do**
  $tps \leftarrow tps + 1$;
  $v_i \leftarrow$ *sommet*(P);
  if *(*$\exists v_j \in$ succ(v_i) *tel que* couleur$[v_j] =$ ***blanc****)* **then**
    empile (P, v_j);
    couleur$[v_j] \leftarrow$ **gris** ;
    $T[v_j] \leftarrow v_i$;
    dec$[v_j] \leftarrow tps$;
  else
    /* tous les successeurs de v_i sont **gris** ou **noirs** */
    depile $(P; v_i)$;
    couleur$[v_j] \leftarrow$ **noir** ;
    fin$[v_i] \leftarrow tps$;
  end
end

Algorithm 1: Depth First Search DFS algorithm

Complexité de Programme du parcours en profondeur DFS

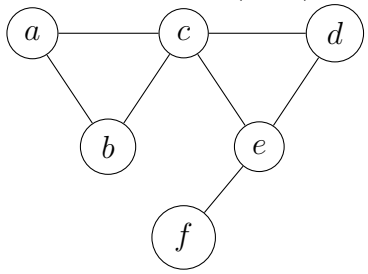
•*Espace* :

représentation du graphe : $\mathbf{O(n + m)}$.
structure pile : $\mathbf{O(n)}$.
tableau marque :$\mathbf{O(m)}$.
$\Rightarrow$: complexité dans l'espace : $\mathbf{O(n + m)}$.

•*Temps* : marquage : n opérations.
exploration : chaque sommet x nécessite d_x opérations donc en tout $\sum_{x \in V} d_x = 2 \mid E \mid = 2m$.
$\Rightarrow$: complexité en temps : $\mathbf{O(n + m)}$.

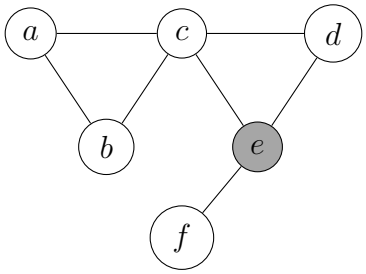
Exemple 4. *Soit le graphe* $G = (V, E)$ *donné dans la figure ci-dessous.*

$G = (V, E)$:



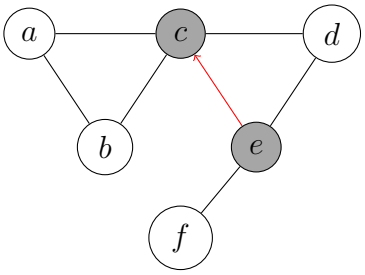
•***Itération 01*** **:**

$T = \{e : Non\}$
$P = [e]$
Découverts ***(gris ou noirs)***$=[e]$
Fermés $= [\_]$



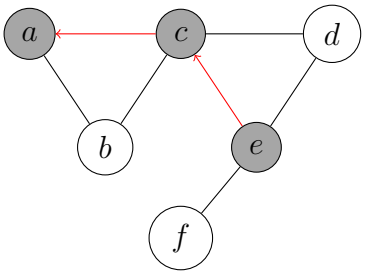
•***Itération 02*** **:**

$T = \{e : Non, c : e\}$
$P = [e, c]$
Découverts ***(gris ou noirs)***$=[e, c]$
Fermés $= [\_]$



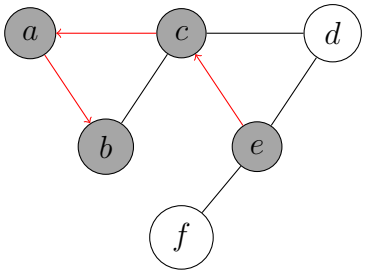
•***Itération 03*** **:**

$T = \{e : Non, c : e, a : c\}$
$P = [e, c, a]$
Découverts ***(gris ou noirs)***$=[e, c, a]$
Fermés $= [\_]$



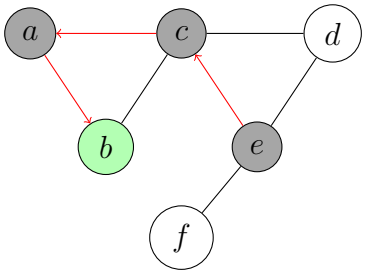
•***Itération 04*** :

$T = \{e : Non, c : e, a : c, b : a\}$
$P = [e, c, a, b]$
Découverts ***(gris ou noirs)***$=[e, c, a, b]$
Fermés $= [\_]$



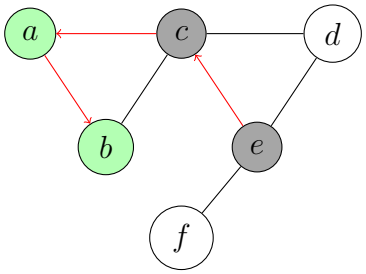
•***Itération 05*** :

$T = \{e : Non, c : e, a : c, b : a\}$
$P = [e, c, a]$
Découverts ***(gris ou noirs)***$=[e, c, a, b]$
Fermés $= [b]$



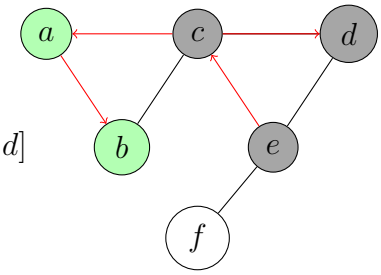
•***Itération 06*** :

$T = \{e : Non, c : e, a : c, b : a\}$
$P = [e, c]$
Découverts ***(gris ou noirs)***$=[e, c, a, b]$
Fermés $= [b, a]$



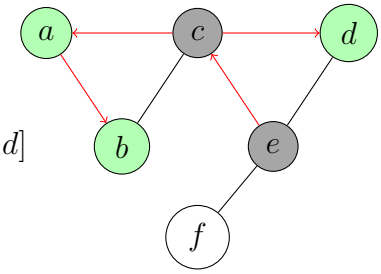
•***Itération 07*** :

$T = \{e : Non, c : e, a : c, a : b, d : c\}$
$P = [e, c, d]$
Découverts ***(gris ou noirs)***$=[e, c, a, b, d]$
Fermés $= [b, a]$



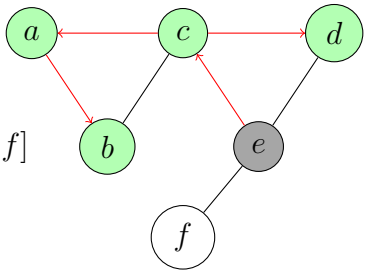
•***Itération 08*** :

$T = \{e : Non, c : e, a : c, a : b, \}$
$P = [e, c]$
Découverts ***(gris ou noirs)***$=[e, c, a, b, d]$
Fermés $= [b, a, d]$



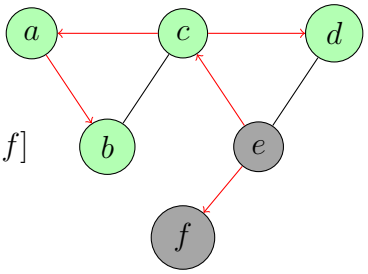
•***Itération 09*** **:**

$T = \{e : Non, c : e, a : c, a : b, f : e\}$
$P = [e]$
Découverts ***(gris ou noirs)***$=[e, c, a, b, d, f]$
Fermés $= [b, a, d, c]$



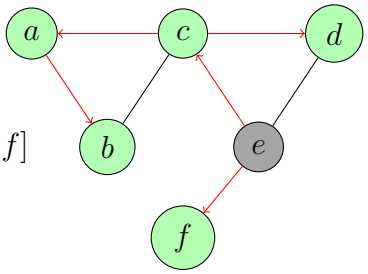
•***Itération 10*** **:**

$T = \{e : Non, c : e, a : c, a : b, f : e\}$
$P = [e, f]$
Découverts ***(gris ou noirs)***$=[e, c, a, b, d, f]$
Fermés $= [b, a, d, c]$



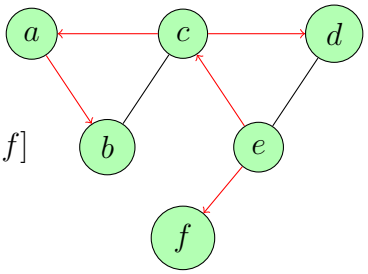
•***Itération 11*** **:**

$T = \{e : Non, c : e, a : c, a : b, f : e\}$
$P = [e]$
Découverts ***(gris ou noirs)***$=[e, c, a, b, d, f]$
Fermés $= [b, a, d, c, f]$



•***Itération 12*** **:**

$T = \{e : Non, c : e, a : c, a : b, f : e\}$
$P = [\_]$
Découverts ***(gris ou noirs)***$=[e, c, a, b, d, f]$
Fermés $= [b, a, d, c, f, e]$



3.2.2 Parcours en largeur BFS (Breadth-First Search)

Le principe dans ce parcours est de visiter tous les voisins d'un sommet avant de visiter le sommet suivant qui sera le premier voisin à avoir été visité auparavant. Autrement dit en parcoure tous les sommets d'un graphe a partir d'un sommet de départ v (racine),en commence par visiter tous les successeurs de v avant de visiter les autres descendants de v.
Le parcours en largeur est obtenu en gérant la liste d'attente au coloriage comme une file d'attente. Autrement dit, on enlève à chaque fois le plus vieux sommet gris dans la file d'attente,et on introduit tous les successeurs blancs de ce sommet dans la file, en les coloriant en gris.

•**Arborescence associée au parcours :**

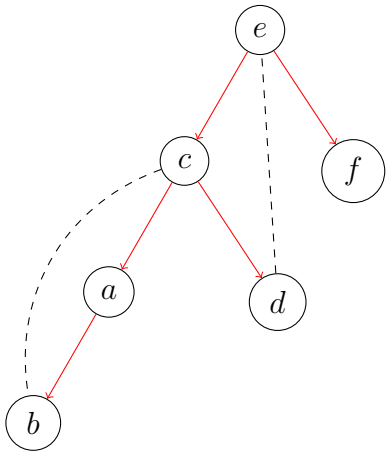


FIGURE 3.4 – Arborescence associée au graphe G

On utilise une file F, pour laquelle on suppose définie les opérations *initfile* (F) qui initialise la file F à vide, *ajoutefinfile* $(F; v)$ qui ajoute le sommet v à la fin de la file F, *estvide*(F) qui retourne **vrai** si la file F est vide et **faux** sinon, et *enlevedebutfile*$(F; s)$ qui enlève le sommet v au début de la file F.
On utilise un tableau T qui associe à chaque sommet le sommet qui l'a fait entrer dans la file, et un tableau *couleur* qui associe à chaque sommet sa couleur (**blanc**, **gris** ou **noir**) [12].

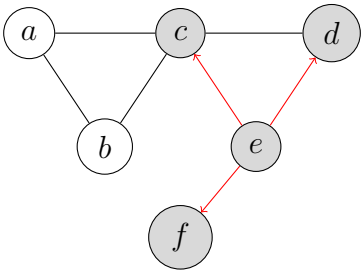
Complexité de programme du Parcours en largeur BFS (Breadth-First Search) •*Espace* :
représentation du graphe : $\mathbf{O(n + m)}$.
structure file : $\mathbf{O(n)}$.
tableau marque :$O(m)$.
$\Rightarrow$: complexité dans l'espace : $\mathbf{O(n + m)}$.

•*Temps* : marquage : n opérations.
exploration : chaque sommet x nécessite d_x opérations donc en tout $\sum_{x \in V} d_x = 2 \mid E \mid = 2m$.
$\Rightarrow$: complexité en temps : $\mathbf{O(n + m)}$.

Exemple 5. *On prend le graphe* $G = (V, E)$ *de l'exemple précédent :*

•***Première itération*** :

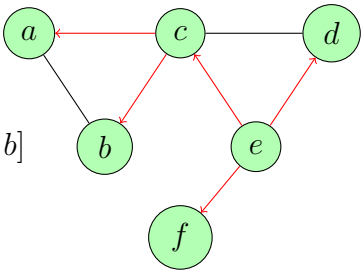
$T = \{e : Non, d : e, c : e, f : e\}$
$F = [e, d, f, c]$
Découverts ***(gris ou noirs)***$=[e, d, f, c]$
Fermés $= [\_]$



Le parcours en largeur du graphe $G = (V, E)$ précédent :

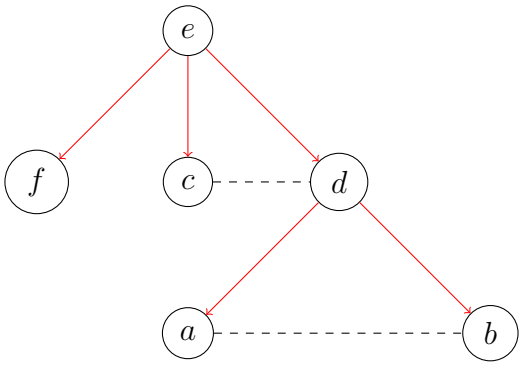
•***Dernière itération*** :

$T = \{e : Non, d : e, c : e, f : e, a : c, b : c\}$
$F = [\_]$
Découverts ***(gris ou noirs)***$=[e, d, f, c, a, b]$
Fermés $= [e, d, f, c, a, b]$



- *Les sommets sont visités depuis e dans l'ordre* { ***e,c,a,b,d,f*** }

•***Arborescence associée*** :



3.2.3 Parcours en largeur lexicographique LexBFS

Parcours en largeur lexicographique dans un graphe (lexBFS pour Lexicographic Breadth First Search en anglais) est un algorithme qui a été introduit par Rose Tarjan et Lueker [3] en 1976. Pour un graphe donné $G = (V, E)$ d'ordre n, l'algorithme lexbfs numérote les sommets de V de n á 1, c'est-á-dire il produit un ordre total sur les sommets de G.

Initialement chaque sommet v a une étiquette $l(v)$ vide, un sous-ensemble de $\{1, 2, \ldots, n\}$ décroissant. Sur l'ensemble des étiquette, est défini un ordre lexicographique noté "$\leq_{lex}$" : pour deux étiquettes $l_1 = \{a_1, a_2, \ldots, a_p\}$ et $l_2 = \{b_1, b_2, \ldots, b_q\}$, $l_1 <_{lex} l_2$ si $\exists i_0$ tel que $a_i = b_i$ pour $i < i_0$ et $a_{i_0} < b_{i_0}$ ou $p < q$ et $a_i = b_i$ pour $i = 1, \ldots p$. $l_1 = l_2$ si $p = q$ et $a_i = b_i$, $1 \leq i \leq p$. A chaque étape, on choisit le plus grand entier i qui n'est pas encore attribué ; on cherche un sommet v qui n'est pas encore numéroté et ayant une

étiquette $l(v)$ maximale lexicographiquement. On pose $v = v_i$ et pour tout voisin u de v qui n'est pas encore numéroté, on rajoute l'indice i á $l(u)$.
L'ordre produit par le parcours lexBFS est appelé ordre lexBFS, on le note $\sigma = \{v_1, v_2, \ldots, v_n\}$ avec $v_1 <_\sigma v_2 <_\sigma \ldots <_\sigma v_n$.
Au cours de l'éxicution de lexBFS, pour attribuer l'indice i, plusieurs sommets peuvent avoir la même étiquette maximale. Dans ce cas, choisir l'un ou l'autre parmi eux n'importe pas.
Il est connu que lexBFS calcule une arborescence des plus courtes distances par rapport á la source v_n, donc l'ensemble des sommets est partitionné en niveaux $N_0, N_1, \ldots N_k$ [ ] tel que $N_0 = \{v_n\}$, et pour $i = 1$ á k, $v \in N_i$ si et seulement si $d(v, v_n) = i$. De plus, lexBFS découvre et énumére tous les sommets d'un même niveau N_i avant de passer aux éléments de N_{i+1}. Ainsi, en général, une arête relie deux sommets qui sont, soit dans un même niveau soit dans deux niveaux consécutifs.

Algorithme lex-BFS :
Donnée : Un graphe $G = (V, E)$.
Résultat : Un ordre σ des sommets de G.

1. Pour chaque sommet $x \in V$ faire :
   marque (x) $\leftarrow \varnothing$
2. Pour $i = n$ à 1 faire :
   - Choisir un sommet x non numéroté de marque maximum dans l'ordre lexicographique ;
     $\sigma(i) \leftarrow x$
   - Pour chaque voisin non numéroté y de x faire :
     $marque(y) \leftarrow marque(y) \cup i$

Théoreme 5. *[3] Si σ est un ordre lex-BFS sur un graphe G, alors la trace de σ sur tout sous graphe induit est aussi un ordre lex-BFS.*

3.3 Reconnaissance des graphes triangués

Et pour la reconnaissance d'un graphe cordal, on utilise l'algorithme de parcours en largeur lexicographique ou Lex-BFS (Lexicograpic Breath First Search) donné par *Rose, Tarjan et Luekerd*ans [3].

3.3.1 Les graphes triangulés

Rappelons qu'un graphe est cordal ou triangulé s'il ne contient pas de trou, c'est á dire un cycle de longueur superieur á 3 sans corde.

Un sommet v est **simpliciale** si l'ensemble de ses voisins $N(v)$ induit une clique. Un ordre $(v_1, v_2, ..., v_n)$ sur les sommets de G est un **ordre d'élimination simplicial**(perfect elimination ordering) si pour tout $i \in \{1, 2, ..., n\}$, le sommet v_i est simplicial dans le sous-graphe induit par $\{v_i, v_{i+1},$
$..., v_n\}$. En s'appuyant sur ces deux notions, *Dirac* [7] (sans p5 117) affirme qu'on peut toujours trouver un sommet simplicial dans un graphe cordal (et même deux, si le graphe n'est pas une clique). On aussi une autre caractérisations de cette classe de graphes :

Théoreme 6. *(Fulkerson et Gross (1965) [4])(p5) Un graphe est cordal si et seulement s'il admet un ordre d'élimination simplicial.*

Autrement dit, dans un graphe triangulé tout cycle de longueur au moins 4 contient une corde (une arête joignant deux sommets non consécutifs du cycle), i.e. les seuls cycles induits sont des trianglés.

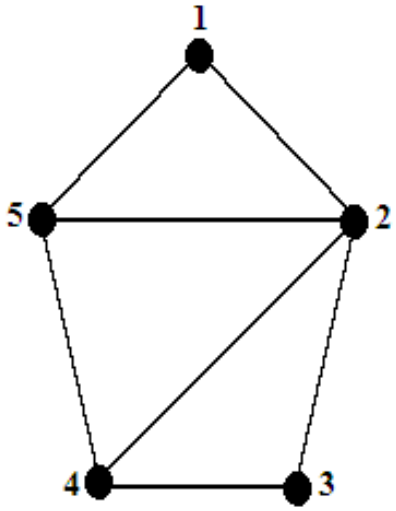


FIGURE 3.5 – Graphe triangulé

Les sommets 1 et 3 sont simpliciaux dans le graphe G alors que les autres sommets ne le sont pas.

Remarque 7. *D'apres la figure ci-dessus, on ramarque qu'un sommet simpliciale v appartient á une unique clique maximale qu'on va noter par la suite B_v.*

Lemme 1. *La classe des graphes triangulés est héréditaire. Autrement dit tout sous graphe induit dans un graphe triangulé est triangulé.*

Il est évident que si tous les cycles élémentaires d'un graphe G sont des triangles, alors les cycles élémentaires de tout sous graphe induit de G sont aussi des triangles.

Un célèbre théorème de Dirac affirme qu'on peut toujours trouver un sommet simplicial dans un graphe cordal (et même deux, si le graphe n'est pas une clique).

Théoreme 7. *(Dirac [7]) Tout graphe triangulé G autre qu'une clique contient au moins deux sommets simpliciaux non adjacents.*

Fulkerson et Gross ont donné la caractérisation suivante des graphes cordaux :

Théoreme 8. *[4] (Fulkerson et Gross (1965)) Un graphe est cordal si et seulement s'il admet un ordre d'élimination simplicial.*

[3] Rose, Tarjan et Lueker ont montré que l'algorithme Lex-BFS permet de trouver efficacement un ordre d'élimination simplicial d'un graphe cordal, et qui permet par conséquent de déterminer rapidement si un graphe est cordal ou non.

Théoreme 9. *[7] Un graphe est triangulé si et seulement si il admet un schéma d'élimination simplicial.*

Lex-BFS calcule un ordre d'élimination simplicial pour un graphe triangulé avec une complexité temporelle $O(n + m)$.

Exemple 6. *1. L'ordre obtenu dans la figure 3.6 par l'algorithme Lex-BFS est* $\sigma = (1, 2, 3, 4, 5, 6)$. *Nous pouvons vérifier facilement que* σ *est un ordre d'élimination simplicial pour le graphe* G *et donc le graphe* G *est triangulé.*

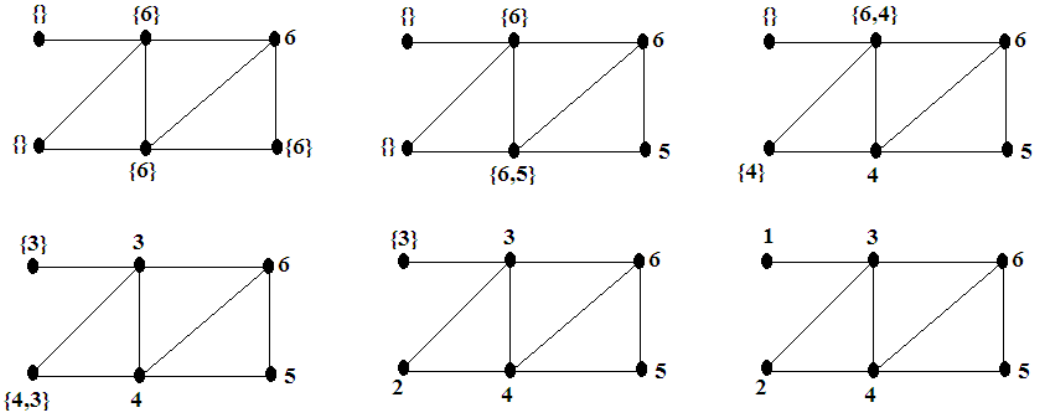


FIGURE 3.6 – Exécution de Lex-BFS

2. *Par contre, le graphe de la figure 3.7 n'est pas triangulé. En effet, après application de Lex-BFS, on trouve le graphe de la figure 3.8(a). On voit bien que le sommet* F *qui porte le numéro 1 est simplicial dans* G*, mais dans la figure (b) qui présente le graphe* $G - \{F\}$*, le sommet* E *qui porte le numéro 2 n'est pas simplicial dans ce graphe. Donc d'après le Théorème 3.6,* G *n'est pas triangulé.*
Puisque le graphe G *n'est pas triangulé, alors il y à un trou quelque part. Effectivement* $\mu = (A, B, D, E, A)$ *est un trou dans* G.

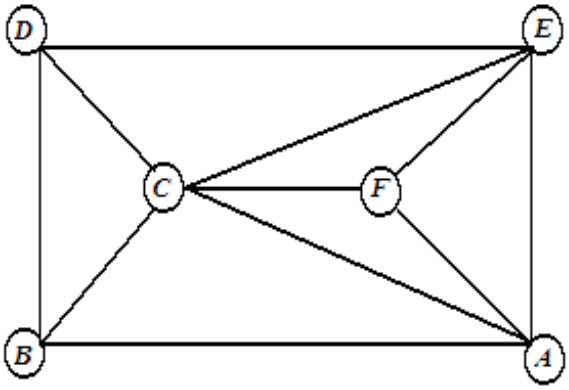


FIGURE 3.7 – Un graphe qui n'est pas triangulé

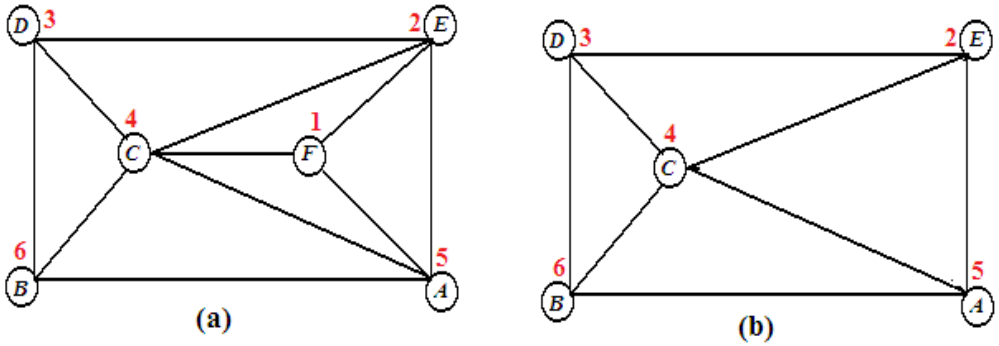


FIGURE 3.8 – Un graphe qui n'est pas triangulé

Conclusion

Dans ce chapitre on a définit certains graphes parfaits, et les parcours, BFS, DFS et LexBFS et on a présenté la reconnaissance de graphe triangulé avec le parcours Lexicographique

CHAPITRE 4

LEXBFS ET OPTIMISATION

4.1 Probléme de stabilité et clique

Rappelons qu'un stable (on dit aussi ensemble indépendant) dans un graphe G est un ensemble de sommets deux á deux non-adjacents ; une clique est un ensemble de sommets deux á deux adjacents. Un stable (resp. clique) ayant le maximum possible d'éléments est dit maximum ; un stable maximal (rssp. clique maximale) est un stable (resp. clique) qui n'est pas inclus(e) dans un autre stable (resp. clique) : un stable (resp. clique) maximum est nécessairement maximal(e), mais pas l'inverse (voir figure ?). Rappelons aussi que la taille d'un stable maximum est noté par $\alpha(G)$ tandis que celui de la clique maximum par $\omega(G)$. Les problemes de decision liés aux probleme du stable et clique maximums sont :
PROBLÉME DU STABLE
Instance Un graphe G et un entier k.
Question G posséde-t-il un stable de cardinalité k ?

PROBLÉME DE LA CLIQUE
Instance Un graphe G et un entier k.
Question G posséde-t-il une clique de cardinalité k ?
Les stables et les cliques sont reliés de maniére trés simple : On a clairement, un ensemble de sommets S est une stable d'un graphe G si et seulement si c'est un stable du complémentaire G^c de G. D'oú, $\alpha(G) = \omega(G^c)$.
Ainsi tout énoncé sur les stables peut être reformulé en termes de cliques.
Un ensemble **dominant** (*dominating set*) est un sous-ensemble $D \subseteq V$ tel que pour tout sommet $u \in V \setminus D$, $N(u) \cap D \neq \emptyset$, c'est-á-dire u est adjacent á au moins un sommet dans D. Ici, le probléme est de déterminer un plus petit dominant dans un graphe G :
PROBLÉME DE DOMINANT MINIMUM
(Minimum Dominating Set)
Instance : Un graphe $G = (V, E)$
Objectif : Déterminer un dominant minimum de G.
La taille d'un plus petit dominant est désignee par le parametre $\gamma(G)$, appelée nombre de domination (*domination number*) de G.

La proposition ci-dessous donne la relation entre les stables et les ensembles dominants :

Proposition 3. *Un stable S est maximal si et seulement s'il est dominant minimal.*

Preuve 1. *Si S n'est pas maximal, alors, il existe $u \in V \setminus S$ tel que $S \cup \{u\}$ est un stable, ce qui implique que $N(u) \cap S = \emptyset$, donc S n'est pas dominant. Par conséquent S n'est pas dominant minimal. Réciproquement, si S est maximal, alors $\forall u \in V \setminus S$, u est adjacent á au moins un élément dans S, ce qui implique que S est un dominant. De plus, $\forall u \in S$, on a $N(u) \cap (S \setminus \{u\}) = \emptyset$, c'est-á-dire $S \setminus \{u\}$ n'est pas un dominant ; par conséquent, S est un dominant minimal.*

Notons que les éléments d'un dominant peuvent être adjacents, ceci montre qu'un dominant n'est pas forcement un stable comme le montre la figure .

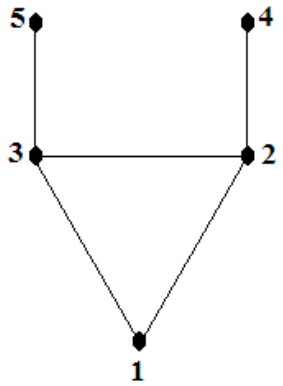


FIGURE 4.1 – $\alpha(G) = |\{1,4,5\}| = 3$, $\omega(G) = |\{1,2,3\}| = 3$, $\gamma(G) = |\{2,3\}| = 2$. $D = \{2,3\}$ est un dominant minimal qui n'est pas stable.

Si on désigne par $i(G)$ la taille d'un plus petit stable maximal dans G, on aura, d'aprés ce qui précéde, la relation évidente : $\gamma(G) \leq i(G) \leq \alpha(G)$. D'oú la naissance d'un autre probléme, appelé problème du dominant stable minimum qui consiste à trouver le plus petit des stables maximaux connu dans la littérature comme Minimum Maximal Independent Set), que nous noterons MIDS :
PROBLÉME DU DOMINANT STABLE MINIMUM (Minimum Independent Dominating Set)
Instance : Un graphe $G = (V, E)$
Objectif : Déterminer un dominant stable minimum de G.
L'algorithme ci-dessous calcule un stable maximum dans un graphe triangulé :
Entré : un graphe triangulé $G = (V, E)$. et ordre lexbfs $\sigma = \{v_1, v_2, \ldots, v_n\}$
Sorté : S qui est un stable maximum .
début $S = \emptyset$
Pour $i = 1$ **à** n **faire**
Si v_i n'est pas marqué alors **faire**
$S = S \bigcup \{V_i\}$
marguer les sommets de $N(v_i)$
Fin si
Fin pour
Fin

Lemme 2. *Soient G un graphe triangulé et v un sommet simplicial. Posons $G' = G - B_v$, le sous graphe obtenu par la suppression des sommet de la biclique maximale B_v. Alors $\alpha(G') = \alpha(G) - 1$.*

Théoreme 10. *L'algorithme ci-dessus calcule un stable maximum pour un graphe triangulaire en un temps linéaire.*

Définition 4.1.1. *Dans un graphe G muni d'un ordre Lex-BFS σ, pour tout sommet v, $d^+(v) =\mid \{u \in V, uv \in E, v <_\sigma u\} \mid$, c'est á dire $d^+(v)$ est égale au nombre de sommets qui sont adjacents et supérieurs á v.*

Entré : Un graphe triangulé $G = (V, E)$. et ordre lexbfs $\sigma = \{v_1, v_2, \ldots, v_n\}$
Sorté : $\theta(G)$ qui est la taille d'une clique maximum.
début $\theta = 0$
Pour $i = 1$ **à** n **faire**
Si $\theta < d^+(v_i)$alors **faire**
$\theta = d^+(v_i)$
Fin si
Fin pour
Fin

4.2 probleme de coloration

Etant donné un graph $G = (V, E)$. On définit une k-coloration propre de G comme une application $c : V \longrightarrow \{1, 2, ..., k\}$ qui associe á chaque sommet de G une couleur tel que pour toute arête $uv \in E$, on a $c(u) \neq c(v)$, c'est-á-dire deux sommets adjacents ne peuvent pas être colorés par une même couleur. Pour $i = 1, 2, ..., k$, $c^{-1}(i)$ est un sous-ensemble de sommets ayant une même couleur, donc qui sont deux á deux non adjacent ; ainsi, une k-coloration partitionne l'ensemble des sommets en k stables $V_1, V_2, ..., V_k$. Les ensembles V_i sont appelés les classes de couleur de la coloration.
On appelle nombre chromatique le plus petit k pour lequel G est k- colorable, noté $\chi(G)$. Il vient que un graphe 1-colorable est un graphe sans arete, donc vide ; un graphe 2-colorable est un graphe biparti ; tout graphe complet K_n est n-colorable. Une coloration optimale est celle qui utilise le moins de couleur possible :
PROBLÉME DE LA COLORATION DES SOMMETS
Instance : Un graphe non orienté G.
Tâche : Trouver une coloration des sommets $f : V(G) \longrightarrow \{1, ..., k\}$ de G avec k minimum.

Des problémes de coloration apparaissent naturellement dans beaucoup de situations pratiques oú il est question de répartir les objets d'un ensemble donné en groupes de telle sorte que les membres de chaque groupe soient mutuellement compatibles suivant certains critéres.

Exemple 7. *(tiré Murty [5])* PLANIFICATION D'EXAMENS.
Les étudiants d'une université ont des examens annuels dans tous les cours auxquels ils sínscrivent. Naturellement, les examens de deux cours différents ne peuvent avoir lieu en même temps s'il y a des étudiants inscrits á ces deux cours. Comment doit-on organiser les examens pour qu'il y ait le moins de sessions possibles ? Pour trouver un tel planning, considérons le graphe G dont l'ensemble de sommets est l'ensemble de tous les cours, deux cours étant reliés par une arête s'il font l'objet d'un conflit. Clairement, les stables de G correspondent aux groupes de cours sans conflit. Ainsi le nombre minimum de sessions requis est le nombre chromatique de G.

La proposition ci-dessous donne une relation simple entre le nombre chromatique et les invariants $\alpha(G)$, $\omega(G)$:

Proposition 4. *Pour tout graphe G, on a :*

1. $\chi(G) \geqslant \dfrac{n}{\alpha(G)}$;
2. $\chi(G) \geqslant \omega(G)$.

Preuve 2.
1. *On a $\sum_{i=1}^{\chi(G)} |V_i| = n$, et $|V_i \leq \alpha(G)|$ pour tout $i = 1, ..., \chi(G)$; ceci implique que $\alpha(G)\chi(G) \geqslant n$; d'oú $\chi(G) \geqslant \dfrac{n}{\alpha(G)}$.*
2. *Il est claire que si H est un sous- graphe induit de G, alors $\chi(H) \leq \chi(G)$; il est aussi claire que pour un graphe complet K_n, $\chi(K_n) = n$. En particulier si H est une clique maximum dans G alors : $\chi(H) = \omega(G) \leq \chi(G)$.*

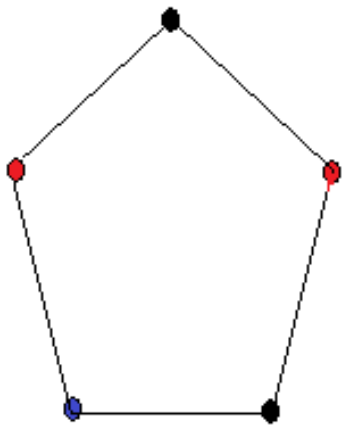


FIGURE 4.2 – Le nombre chromatique

Le cycle C_5 présenté par la figure ? vérifie $\omega = 2 < \chi = 3 = \Delta(C_5) + 1$, ce qui montre que cette borne sur le nombre chromatique n'est pas toujours atteinte.

Proposition 5. *Pour tout graphe $G = (V, E)$, on a : $\chi(G) \leq \Delta(G) + 1$.*

Preuve 3. ***Element de theorie-page 217–a revoir*** *Raisonnons par récurrence sur le nombre n de sommets du graphe. Si $n = 1$, G est reduit á un seul sommet de degre 0 et á qui il faut une seule couleur ; ainsi $\chi(G) = 1 = \Delta(G) + 1$.*
Supposons l'assertion vraie pour tout graphe ayant au plus $n - 1$ sommets, $n = 2$, et soit G un graphe avec n sommets. Si on supprime un sommet u, obtient le graphe G' avec $\Delta(G') \leq \Delta(G)$. Par hypothése de récurrence, ce graphe est $(?(G) + 1)$-colorable. Une $(?(G) + 1)$-coloration de G est obtenue en coloriant le sommet u avec une couleur différente des couleurs déjà affectées aux sommets adjacents à u (il y en a au plus $?(G)$).

En combinant la proposition ? et la proposition ? ? nous obtenons un encadrement du nombre chromatique d'un graphe.

Pour tout graphe $G = (V; E)$ on a :

$$\frac{|V|}{\alpha(G)} \leq \chi(G) \leq \Delta(G) + 1$$

Le cycle C_5 présenté par la figure ? vérifie $\chi(C_5) = 3 = \Delta(C_5)+1$, ce qui montre que la borne superieure du nombre chromatique atteinte. *KOOKS 1941* a montré que les seuls graphes qui atteint la borne superieure de l'invariant $\chi(G)$ sont les graphes complets et les trous impairs :

Théoreme 11. *(*BROOKS, 1941*) elemnt de la theorie Soit* $G \cdot = (V; E)$ *un graphe simple connexe. Si* G *n'est ni un graphe complet, ni un cycle de longueur impaire, alors* $\chi(G) \leq \Delta(G)$.

Le probléme de trouver le nombre chromatique est $NP - difficile$, cependant pour resoudre des cas pratiques, il est possible de developper des heuristiques efficaces qui, pour colorier les sommets d'un graphe, fonctionnent raisonablement bien. l'approche la plus naturelle consiste á colorier les sommets de maniere gloutonne : supposons que les sommets soient ordonnés par un ordre total $u_1, u_2, u_3, ..., u_n$. Nous allons colorier les sommets un par un en utilisant toujours la plus petite couleur possible, les couleurs étant représentées par des nombres entiers 1, 2, 3, . . .

HEURISTIQUE DE COLORATION GLOUTON
Entrée : un graphe G.
Sortie : une coloration de G.

1. Ranger les sommets de G suivant un ordre total : $v_1, v_2, ..., v_n$.
2. Colorier les sommets l'un aprés l'autre suivant cet ordre, en attribuant á v_i le plus petit entier strictement positif qui n'est attribué á aucun de ses voisins déjá coloriés.

L'heuristique ci-dessus construit bien une coloration, sa complexité est polynomiale. Par exemple dans la figure ?(a), si l'on applique cet heuristique sur un P_4 ordonné comme suit : $v_1 < v_2 < v_3 < v_4$, on obtient une $2 - coloration$, ce qui correspond á une solution optimale puisqu'un P_4 est un graphe biparti (car un graphe est biparti si et seulement si il est $2 - colorable$). En revanche, si l'on exécute le même algorithme sur le même P_4 dont l'ordre est : $v_1 < V_2 < v_4 < v_3$ (voir la figure ?(b)), on obtient une $3 - coloration$ qui n'est donc pas optimale. Ainsi le nombre de couleurs utilisées pour colorier un graphe grâce á cet heuristique gloutonne dépend de l'ordre des sommets (il y en a $n!$ possibilités pour un tel ordre) ; d'oú la difficulté de savoir á l'avance lequel de ces ordres produira une coloration optimale.

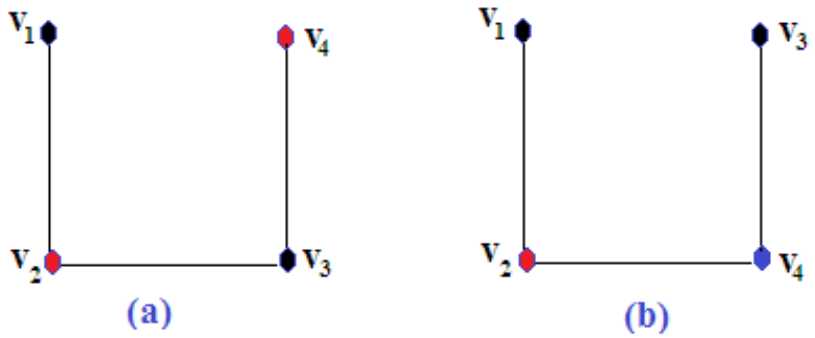


FIGURE 4.3 – coloration gloutonne

CHAPITRE 5

APPLICATION

5.1 Planification des examens

dans une classe master 2 option MMTD neuf étudiants sont concernés par la session rattrapage : amine , Lila, Amel, Omar, Sara, massinissa, Adam, Karim et Amina. Les modules qui n'ont pas étaient validés par les neuf étudiants sont :théorie de jeux, processus aléatoire avancée, gestion de projet, organisation industrielle, gestion de stock et anglais ,chaque étudiants est concerné par quelques modules comme s'est expliqué par le tableau suivant

Etudiant\module	TJ	PAA	GP	OI	GS	ANGLAIS
Amine	X	X		X	X	
Lila		X		X	X	X
Amel	X		X	X	X	
Omar	X		X	X	X	
Sara		X		X	X	
Massinissa	X			X	X	
Adam				X	X	X
Karim			X	X	X	
Amina				X	X	

FIGURE 5.1

On suppose que chaque examen dure deux heures et que les étudiants n'ont pas les mêmes modules non validés. Le symbole "x" dans le tableau indique les modules dont l'étudiant est concerné par exemple amine est concerné par les modules : théorie de jeux, processus aléatoires avancées, organisation industrielle et gestion de stocks mais pas les autres modules. L'objectif est d'organiser une session de rattrapage qui dure le moins longtemps possible de façons à ce que tout étudiant passent les examens indiqués dans le plan.

La solution la plus évidente pour planifier tout ça est de programmer les modules l'un après l'autre. Ainsi la session de rattrapage durera 12 heures puisque il y a six examens de deux heures. La question maintenant est de savoir si on peut faire mieux en tenant compte du fait que les modules dont un même étudiant est concernés ne peuvent pas ˆêtre programmés au même temps. Pour résoudre ce problème, associons le graphe d'incompatibilité $G = (V, E)$ tel que les sommet représentent les modules et deux sommet sont relies par une arête si et seulement si les deux examens ne peuvent pas avoir lieu au même temps. Le graphe de la figure donne une modélisation de ce plan

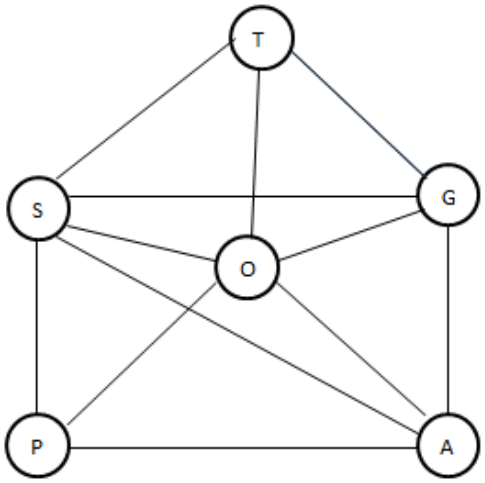


FIGURE 5.2 – Modélisation de plannig

D'après ce graphe on voit bien que l'examen de processus aléatoire avancée et celui de gestion de projet peuvent étre programmés au même temps car ils sont pas adjacents dans G. Ainsi il est naturel de penser qu'une partition minimale en stable donne un planning optimale pour le déroulement des examens c'est-à-dire ce planning sera donné par le nombre chromatique de G. Commençons par l'application de l'algorithme Lex-BFS sur G.

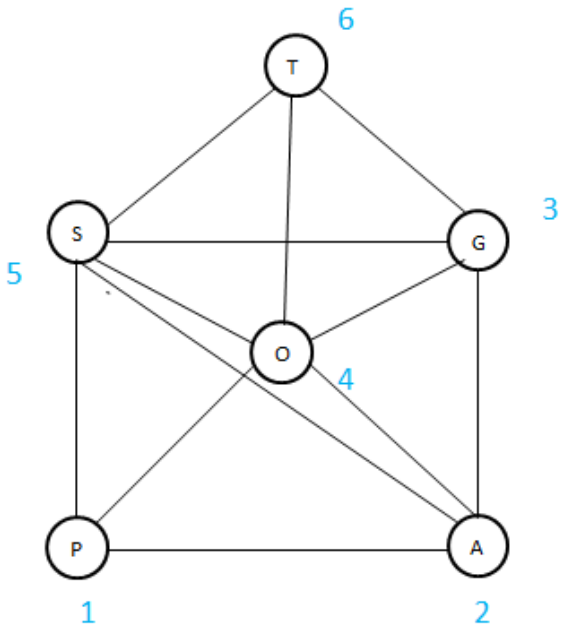


FIGURE 5.3 – l'ordre des sommets de G

La figure montre l'ordre sur les sommets donné par cet algorithme.
On peut vérifier facilement que l'ordre des sommets de G est un ordre d'élimination parfait (simplicial), ce qui implique que le graphe G est triangulé. Par conséquent l'algorithme LexBFSCOLOR donne une coloration optimale pour les sommets de G comme est montré sur la figure ?

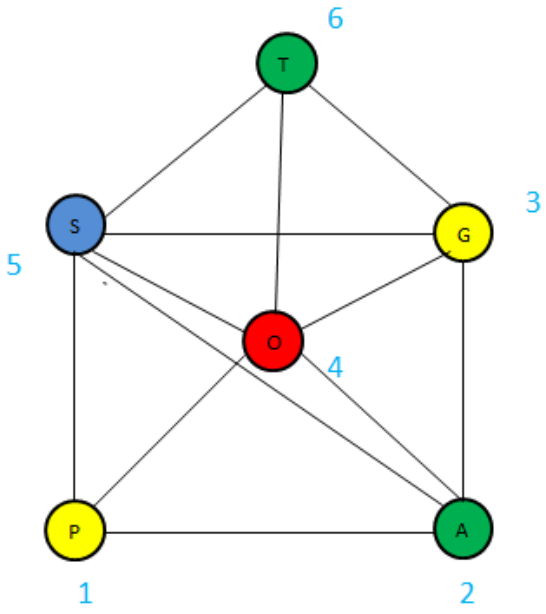


FIGURE 5.4 – Graphe triangulé

On voit bien que le nombre de couleurs minimum est quatre, c'est-'a-dire $X(G) = 4$.

A partir de la figure 5.3, on déduit un planning optimal de la manière suivante :
A les premières deux heure on programme le module de théorie de jeux et Anglais, au deux heures suivantes nous pouvons mettre l'examen de gestion de stocks qui Intéresse tout le monde, au deux heures suivantes correspond l'examen de organisation industrielle qui intéresse aussi tout le monde et enfin nous avons l'examen de processus aléatoire avancé et gestion de projet .

5.2 Transport des produits chimiques

On veut transporter des produits chimiques par le rail. A, B, C, D, E, F, G et H désignent huit produits chimiques. Dans le tableau ci-dessous, une croix signifie que les produits ne peuvent pas être entreposés dans le même wagon, car il y aurait risque d'explosion :

	A	B	C	D	E	F	G	H
A		x		x				
B	x		x	x	x			
C		x		x	x	x		
D	x	x			x		x	
E		x	x	x		x	x	
F			x		x			x
G				x	x			
H			x			x		

FIGURE 5.5

modélisant les données du tableu par le graphe suivant :

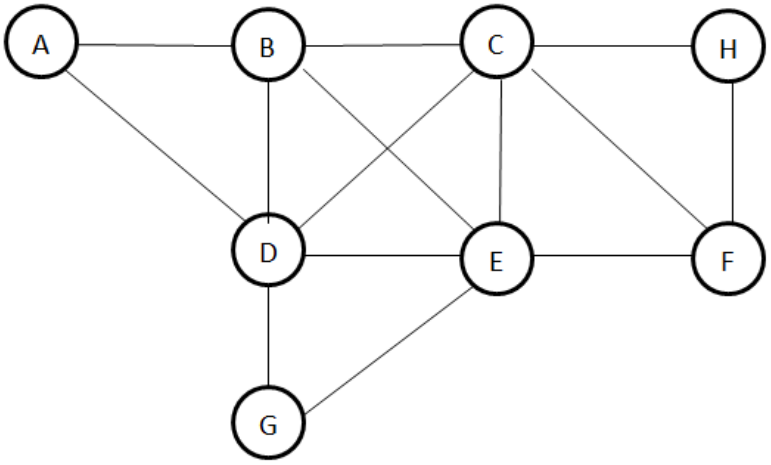


FIGURE 5.6

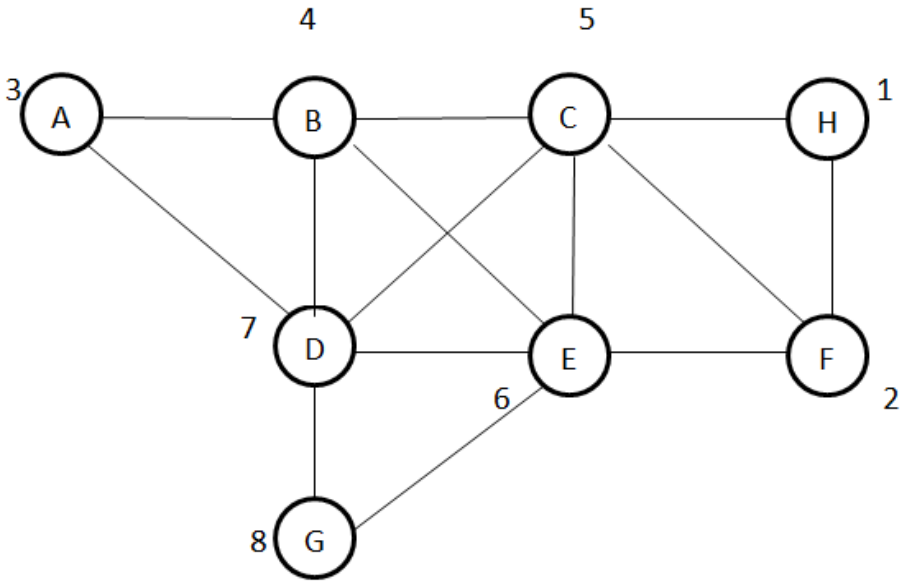


FIGURE 5.7 – Ordre des sommets de G

On vérifie facilement que l'ordre $\sigma = (H, F, A, B, C, E, D, G)$ des sommets de G est un ordre d'élimination simplicial , ce qui implique que G est triangulé.
Maintenant,on applique l'algorithme de coloration LEXBFS COLOR à l'ordre inverse de sigma ,c'est-à-dire l'ordre $\sigma' = (G, D, E, C, B, A, F, H)$

- à la première étape on attribue la couleur rouge aux sommets G, C et A.
- à la deuxième étape ,on attribue la couleur jaune aux sommets D et F.
- à la troisième étape ,on attribue la couleur bleu aux sommets C et H.
- à la quatrième étape ,on attribue la couleur verte au sommet B.
- à la quatriéme étape ,on attribue la couleur verte au sommet B.

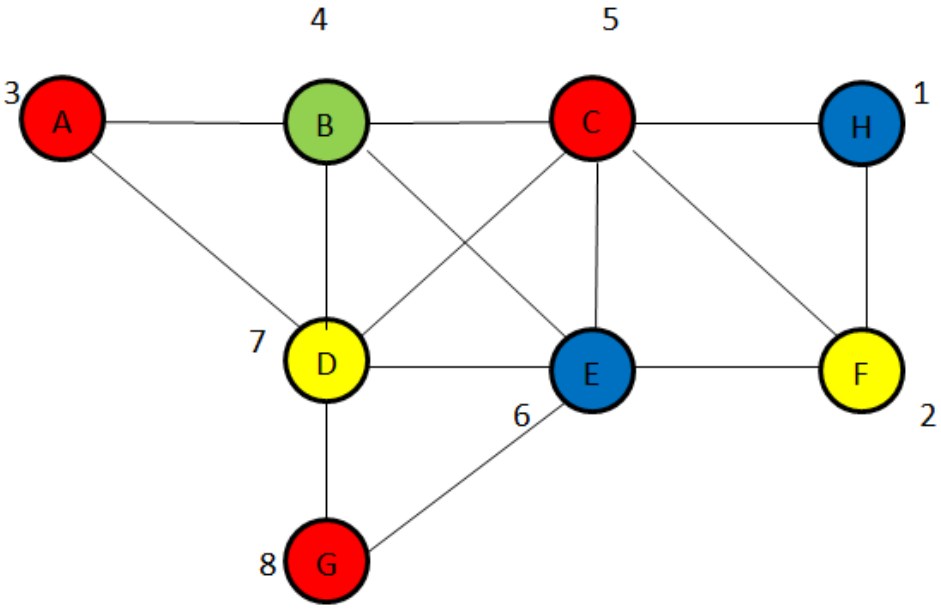


FIGURE 5.8 – Graphe triangulé

On voit bien que le nombre de couleurs minimum est quatre ,c'est-à-dire $X(G) = 4$,on déduit que le nombre optimale des wagon est quatre .

On peut transporter les produits G ,C et A dans le wagon ,et les produits D et F ensemble dans un autre wagon, et C et H ensemble dans un autre wagon et le produit B tout seul dans un autre wagon .

CONCLUSION GÉNÉRALE

Ce travail présente à ses lecteurs une vision globale sur la théorie des graphes qui englobe un sujet trés intéressant qui est le parcours lexicographique. L'importance de la théorie des graphes vient aussi du fait qu'elle fournit un cadre conceptuel adéquat pour l'analyse et la résolution de nombreux problèmes. Elle constitue l'un des instruments les plus courants et les plus efficaces pour résoudre des problèmes discrets posés en Recherche Opérationnel(RO).

Notre travail consiste a donné au lecteur la définition et les caractéristiques du parcours lexicographique ; l'utilisation de lexbfs à la reconnaissance des graphes triangulés ; et l'exactitude de notre objectif consisite à la résolution de certain problèmes d'optimisation dans les graphes triangulés avec lexbfs comme le problème de colorations.

BIBLIOGRAPHIE

[1] B.Lèveque«Thèse Doctorat Coloration de graphes = structure et algorithme»

[2] C.Berge. Farbung von graphan, deren Samtiliche bzw,deren unegrade kreise starr sind.

[3] D.J.Rose, R.E.Tarjan, G.S.Lueker, Algorithmics aspects of vertex elimination of graphs, SIAM Journal on computing (1976) 266-283

[4] D.R. Fulkerson et O.A. Gross : Incidence matrices and interval graphs. Pacific J. Math., 15 :835-855, 1965.

[5] J.A. Bondy and U.S.R. Murty. Théorie des Graphes. 2008.

[6] J. L. Ramirez Alfonsin and B. A. Reed, editors. Perfect graphs. Series in Discrete Mathematics and Optimization. Wiley-Interscience, 2001.

[7] G.A. Dirac : On rigid circuit graphs. In Abhandlungen aus dem Mathematischen Seminar der Universit ”at Hamburg, volume 25, pages 71-76. Springer, 1961.

[8] G.Paul«thèse Doctorat parcours en largeur lexicographique»

[9] K.S. Booth et G.S. Lueker : Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. Journal of Computer and System Sciences, 13(3) :335-379, 1976. A.Brunel«Rapport de cours de recherche 2009»

[10] Maria Chudnovsky, Neil Robertson, Paul Seymour et Robin Thomas, « The strong perfect graph theorem », Annals of Mathematics, vol. 164, no 1, ? 2006, p. 51-229

[11] M. Habib, R. McConnell, C. Paul et L. Viennot : Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theoretical Computer Science, 234(1-2) :59-84, 2000.

[12] R.Perrot«La théorie des graphes 2002/2006»

[13] S.A. Cook : The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing, pages 151-158. ACM, 1971. (Cit´e page 182.)

Résumé

Dans ce mémoire, nous nous intéressons à abordé certainement l'un des plus fameux sujets de la théorie des graphes ; le parcours lexicographique et les graphe parfaits, en particulier les graphes triangulés, nous parlons de quelques problèmes d'optimisation dans les graphes triangulés résolus par lexbfs afin d'obtenir des meilleurs résultats.
Après avoir montré quel genre de résultat nous pouvions attendre, nous étudions comment adapter les méthodes connus à ce jour à savoir l'algorithme color, lexbfs color... etc pour la résolution des problèmes concrets, les limites de ces algorithmes sont utilisés dans le cadre de l'optimisation .

Mots-clés : Graphe triangulé, parcours lexicographique , problème d'optimisation , Coloration.

Abstract

In this thesis, we are interested in certainly addressing one of the most famous subjects of graph theory : the lexicographic course and the perfect graph, in particular the triangulated graphs, we talk about some optimization problems in triangulated graphs solved by lexbfs for better results.
After showing what kind of results we could expect, we study how to adapt the known methods to date namely the color algorithm, lexbfs color ... etc for the resolution of concrete problems, the limits of these algorithms are used in the framework of optimization.

Keywords : Triangulated graph, lexicographic path, optimization problem, coloration
.