République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieure et de la recherche scientifique

Université MOULOUD MAMMERI de Tizi Ouzou
Faculté des sciences
Département de mathématiques

MÉMOIRE DE FIN D'ÉTUDES

En vue d'obtention du diplôme de master
Spécialité

RECHERCHE OPÉRATIONNELLE

Thème

L'ensemble du Feedback minimum
Combinatoire et Sécurisation

Présenté par :
ISSAAD Malika
BELKHIR Romaissa

Devant le jury d'examen composé de :

Mr. SADI Bachir	Professeur	Encadreur
Mr. AOUANE Mohouhand	M.A.A	Président
Mr. KASDI Kamel	M.A.A	Examinateur

Promotion 2020/2021

Remerciements

Avant tout, nous remercions le Bon Dieu, le Miséricordieux de nous avoir donné la force, la patience, la volonté et la santé pour mener à bien ce travail.

Nos sincères remerciements s'adressent à notre directeur de mémoire Mr Sadi Bachir d'avoir accepté de diriger ce travail, pour sa patience, sa disponibilité, son soutien, et surtout ses judicieux conseils qui ont été d'une aide inestimable et qui ont contribué à alimenter notre réflexion.

Nous tenons à remercier sincèrement les membres du jury pour l'intérêt manifesté à examiner ce travail.

Un grand merci à nos chers parents, pour leur amour, leurs conseils ainsi que leur soutien inconditionnel, à la fois moral et économique qui nous a permis de réaliser les études que nous voulions et par conséquent, ce mémoire.

Dédicaces

Ce modeste travail est dédié à mon défunt père, décédé très récemment, qui m'a toujours soutenue et motivée dans mon parcours, je ne saurais jamais exprimer mon regret de ne pas l'avoir parmi nous aujourd'hui, j'espère qu'il aurait été fier de moi.
Son absence, désormais rien ne saura la combler, sa présence est à jamais dans mon coeur. Que Dieu le tout puissant lui accorde sa sainte Miséricorde et l'accueille dans son vaste paradis. Repose en paix mon cher papa.

A ma très chère mère, pour les sacrifices qu'elle a consentit pour mon instruction et mon bien être, pour son soutien et son amour inconditionnel. Puisse Dieu, lui accorder santé, bonheur et longue vie.

A mon cher et unique frère Ghiles, qui est la source de mon bonheur et ma joie de vivre. Que Dieu le garde pour nous.

A toute ma famille et tous mes proches.

A tous mes amis et tous ceux qui me sont chers.

Malika

Dédicace

Je dédie ce modeste travail à tous ceux qui me sont chers.

Romaissa.

Table des matières

1 Généralités **9**
- 1.1 **Introduction** . . . 9
- 1.2 **Définitions et notions de base** . . . 9
  - 1.2.1 Graphe non orienté . . . 9
  - 1.2.2 Graphe orienté . . . 10
  - 1.2.3 Arcs adjacents, arêtes adjacentes . . . 10
  - 1.2.4 Graphe valué . . . 10
  - 1.2.5 Chaîne . . . 11
  - 1.2.6 Cycle . . . 11
  - 1.2.7 Chemin . . . 11
  - 1.2.8 Circuit . . . 11
  - 1.2.9 Représentation de graphe en machine . . . 12
  - 1.2.10 Graphe complémentaire . . . 13
  - 1.2.11 Graphe partiel . . . 13
  - 1.2.12 Sous-graphe . . . 13
  - 1.2.13 Graphe transitif . . . 14
  - 1.2.14 Graphe symetrique . . . 14
  - 1.2.15 Graphe complet . . . 14
  - 1.2.16 Cliques et stables . . . 14
- 1.3 Quelques classes de graphes . . . 15
  - 1.3.1 Graphe triangulé . . . 15
  - 1.3.2 Graphe de comparabilité . . . 15
  - 1.3.3 Graphe d'intervalles . . . 16
- 1.4 Notions de complexité algorithmique . . . 17
  - 1.4.1 Classes des problèmes . . . 18
  - 1.4.2 Exemple de problème difficile . . . 18
  - 1.4.3 **La conjecture** $P \neq NP$ . . . 19

2 Minimum Feedback Vertex Set dans un graphe d'intervalles **20**
- 2.1 Introduction . . . 20
- 2.2 **Définition du feedback** . . . 20
- 2.3 Structure de données et préliminaires . . . 21
  - 2.3.1 **Vérification du minimum** . . . 22
  - 2.3.2 **Lemme 1** [2] . . . 22
  - 2.3.3 **Théorème 1** [2] . . . 24
- 2.4 **Complexité de l'algorithme MFVS** . . . 24
  - 2.4.1 **Lemme 2** [2] . . . 24
  - 2.4.2 **Théorème 2** [2] . . . 24
- 2.5 Sécurisation du feedback minimum . . . 25

2.5.1 **Définition** . . . 25
2.5.2 Comment vérifier la sécurisation de l'ensemble F_{min} . . . 25
2.5.3 Remarques . . . 26
2.5.4 Lemme 3 [2] . . . 27
2.5.5 Lemme 4 [2] . . . 28
2.6 **Comment sécuriser un Feedback non sécurisé ?** . . . 28
2.6.1 Couverture minimale par sommets(transversal minimal) . . . 28
2.6.2 Ajout de sommets . . . 28
2.6.3 Exemple d'application des graphes d'intervalles dans la vie réelle . . . 29

3 Algorithme et implémentation **32**
3.1 Introduction . . . 32
3.2 Le langage python . . . 32
3.2.1 Définition . . . 32
3.2.2 Histoire de Python . . . 32
3.2.3 Avantages de Python . . . 33
3.2.4 Inconvénients de Python . . . 34
3.3 Algorithme et sa complexité . . . 35
3.3.1 **Algorithme SMFVS** . . . 35
3.3.2 **Algorithme MARK** . . . 36
3.3.3 Lemme 1 [2] . . . 36
3.3.4 Lemme 2 [2] . . . 36
3.3.5 Théorème [2] . . . 36
3.4 Organigramme de l'algorithme MARK . . . 37
3.5 Implémentation . . . 37
3.6 Déroulement de l'algorithme . . . 38
3.6.1 Programmation de l'algorithme MARK sur Python . . . 39

Introduction générale

La recherche opérationnelle (RO) peut se définir comme la mise en oeuvre de méthodes scientifiques essentiellement mathématiques, en vue de prendre la meilleure décision possible. Elle fournit des outils pour rationaliser, simuler et optimiser le fonctionnement des systèmes industriels et économiques. La RO propose des modèles pour analyser des situations complexes et permet aux décideurs de faire des choix efficaces.

Le transport aérien est une illustration parfaite de l'importance stratégique de la recherche opérationnelle. La RO a été au coeur de la révolution des compagnies aériennes dans les dernières années, en particulier en introduisant des approches sophistiquées d'adaptation des prix à la demande, sans lesquelles, elles n'auraient pas pu mettre en oeuvre les transformations nécessaires à leur compétitivité et à leur survie.

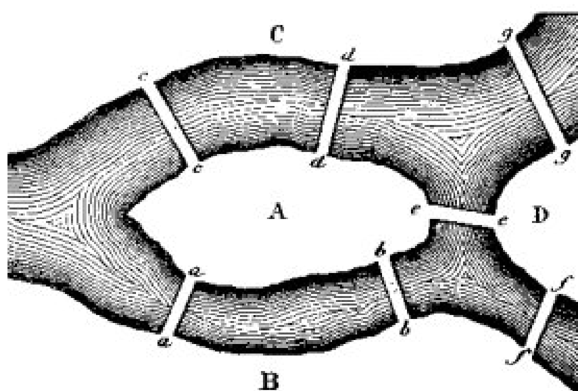
La RO est utilisée dans bien d'autres domaines que le transport, elle est présente casi-partout comme, dans l'organisation des lignes de production d'automobiles, la planification des missions spatiales, mais aussi dans la vie de tous les jours pour le recyclage des déchets, l'organisation des ramassages scolaires, la couverture des satellites des téléphones portables etc...

Faire de la RO, ce n'est pas simplement résoudre un problème combinatoire dans un sous-ensemble d'une entreprise, c'est aussi fournir aux décideurs des outils quantitatifs et qualitatifs de leur activité. La RO doit devenir plus encore le partenaire privilégié du dirigeant et des contrôleurs de gestion, ainsi que des services à valeur ajoutée de l'entreprise.

La notion de graphes est relativement une science jeune, car elle n'est apparue formellement qu'au cours de XXe siècle, avec une accélération particulière dans la deuxième moitié du siècle, et s'est vite imposée par ses liens avec d'autres sciences. La théorie des graphes nourrit en effet des liens étroits avec les mathématiques et l'informatique, en particulier avec l'algorithmique et la récente théorie de la complexité.

Un graphe peut être vu, intuitivement car il s'agit plutôt d'une représentation de la notion abstraite, comme un ensemble de points et de lignes reliant certains de ses points. On comprend que, par sa simplicité même, cette situation peut modéliser, des situations concrètes très variées : réseaux de communication, états possibles d'un jeu, organisation d'un emploi du temps, comme il existe plusieurs exemples moins évidents de modélisation par les graphes, qui attestent encore l'utilité pour les applications de la notion de graphe et de l'intérêt d'en développer la théorie.

L'origine de la théorie des graphes est issue du premier écrit qui s'y rapporte au mathématicien *Euler*, en 1736, sur le problème devenu célèbre des ponts de Koenigsberg. Dans cette ville il y a sept ponts disposés comme suit :



Les ponts de Konigsberg en 1759

Les habitants à l'époque se demandaient s'il était possible, lors d'une promenade, de traverser tous les ponts, chacun une seule fois. La réponse est non comme on peut s'y attendre, car si non, vu le petit nombre de possibilités, un habitant plus futé que les autres aurait fini par trouver une solution. Mais le problème était de s'assurer, de démontrer mathématiquement, que ce n'était pas possible, ce qu'a fait *Euler*. Vu comme problème de graphe, ce petit problème amusant est le même qu'un autre, bien connu des écoliers, qui concerne ces figures qu'on peut tracer sur le plan d'un seul trait sans repasser une deuxième fois sur un même trait. Ces questions relèvent de ce qu'on appelle aujourd'hui les cycles euleriens dans les graphes qu'on définira dans la suite.

Pour bien mener ce mémoire, nous l'avons structuré en trois chapitres :

Le premier chapitre est dédié aux définitions fondamentales de la théorie des graphes et quelques rappels sur la théorie de la complexité algorithmique, qu'on utilisera dans la suite de ce document.

Dans le deuxième chapitre, on s'intéresse à l'algorithme proposé par Pal et Saha, on essaie de trouver des méthodes pour vérifier la sécurisation de l'ensemble trouvé en utilisant cet algorithme. En dernier, nous abordons une situation réelle qui permet de sécuriser le système d'éclairage d'un supermarché.

Le troisième chapitre est la pratique (programmation) de la situation réelle donnée au deuxième chapitre, en langage Python.

On termine notre travail par une conclusion générale.

Chapitre 1

Généralités

1.1 Introduction

La théorie des graphes est un outil puissant de modélisation et de résolution de problèmes purement abstraits. Elle s'est essentiellement développée au cours du XXe siècle et s'est vite imposée par ses liens avec d'autres sciences et par l'étendue de ses applications.
Ce premier chapitre est consacré aux notions générales et introductives à la théorie des graphes et à la théorie de la complexité algorithmique.

1.2 Définitions et notions de base

1.2.1 Graphe non orienté

Un graphe G est un couple (X,E) formé de deux ensembles finis et disjoints $X=\{x_1,x_2,...,x_n\}$ où $n \geq 1$ et $E=\{e_1,e_2,...,e_m\}$ avec $m \geq 0$, tel que, pour tout i, e_i est une paire d'éléments de X.
X est appelé l'ensemble des sommets et E, l'ensemble des arêtes de G.
L'ordre de G est le nombre de ses sommets. On note alors le graphe $G=(X,E)$.

Une arête e dont les extrémités sont x et y est notée $e=[x,y]$ ou $e=xy$.
- Lorsque $x=y$, on dit que e est une boucle.
- Deux arêtes parallèles ont les mêmes extrémités.
- On appelle multigraphe, un graphe non orienté.
- Un graphe est dit simple s'il est multigraphe sans boucle et sans arêtes parallèles.

Exemple 1.1

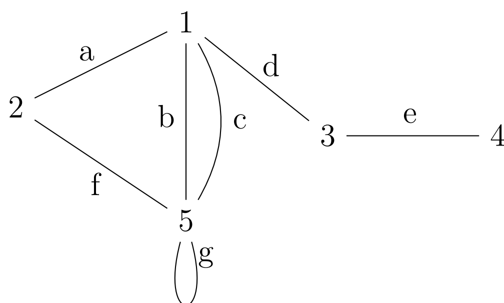


Fig 1.1. Graphe non orienté.

$X=\{1,2,3,4,5\}$; $E=\{a,b,c,d,e,f,g\}$; $\{g\}$ est une boucle ; $\{b\}$ et $\{c\}$ sont deux arêtes parallèles.

1.2.2 Graphe orienté

Un graphe orienté est un couple (X,U), où X est un ensemble fini non vide de sommets et U est un ensemble fini d'arcs notés $\{u_1, u_2,...,u_m\}$.
L'arc est une relation entre deux sommets, dotée d'une orientation telle que si $u=(x,y)$ est un arc de U, avec x, $y \in U$, l'orientation est de x vers y.
On obtient donc un graphe noté $G=(X,U)$.

Exemple 1.2

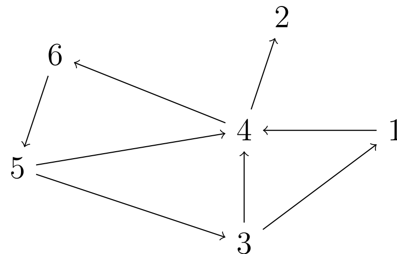


Fig 1.2. Graphe orienté.

Successeur d'un sommet
On dit que y est **successeur** de x s'il existe un arc ayant son extrémité initiale en x et son extrémité terminale en y. L'ensemble des successeurs de x est $\Gamma^+(x)$.
$\Gamma^+(x)=\{y \in X \ / \ (x,y) \in U\}$

Prédécesseur d'un sommet
On dit que y est **prédécesseur** de x s'il existe un arc de la forme (y,x). L'ensemble des prédécesseurs de x est noté $\Gamma^-(x)$.
$\Gamma^-(x) =\{y \in X \ /(y,x) \in U\}$

1.2.3 Arcs adjacents, arêtes adjacentes

Deux arcs (ou deux arêtes) sont dits adjacents si ils ont au moins une extrémité commune.

1.2.4 Graphe valué

Soit $G=(X,U)$ un graphe et v, une application sur U telle que

$$\begin{aligned} v : U &\rightarrow \mathbb{R} \\ u &\rightarrow v(u) \end{aligned}$$

Où $v(u)$ est la valeur portée par l'arc u.
La valeur de l'arc u peut être un coût, un poids, une distance, une longueur, un temps, une probabilité, etc...
G muni de l'application v est dit ***graphe valué***.

Exemple 1.3

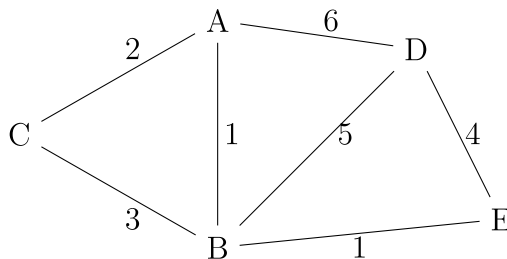


Fig 1.3. Graphe à arêtes valuées.

1.2.5 Chaîne

Une chaîne dans G, reliant x à y est une séquence altérnée de sommets et d'arêtes notée C=$\{x_1$, e_1, x_2, e_2,..., x_{k-1}, e_{k-1}, $x_k\}$; $k \leq n$, telle que chaque arête de la séquence ait une extrémité en commun avec l'arête précédente, et l'autre extrémité en commun avec l'arête suivante.
Le nombre d'arêtes de la séquence est la longueur de la chaîne C.

1.2.6 Cycle

C'est une chaîne C =$\{x_1$, e_1, x_2, e_2,..., x_{k-1}, e_{k-1}, x_k=$x_1\}$ dont les deux extrémités sont communes.

1.2.7 Chemin

Un chemin de x à y dans G est une suite C alternée de sommets et d'arcs du graphe, telle que : C=$\{x$=x_1, u_1, x_2, u_2,..., x_{p-1},u_{p-1}, x_p=$y\}$; $p \leq n$ avec : x_i=$I(u_i)$ et x_{i+1}=$T(u_i)$.

1.2.8 Circuit

C'est une chaîne dont toutes les arêtes sont orientées dans le même sens, tel que l'extrémité terminale de u_i coincide avec l'extrémité initiale de u_{i+1}

Remarques
Dans ce qui suit, C désigne "chaîne, cycle, chemin ou circuit".
• C est dit simple s'il ne passe par aucun arc(ou arête) plus d'une fois. De plus, si C emprunte tous les arcs du graphe, il sera dit eulérien.
• C est dit élémentaire s'il ne passe par aucun sommet plus d'une fois. De plus, si C emprunte tous les sommets du graphe une et une seule fois, il sera dit hamiltonien.

Exemple 1.4

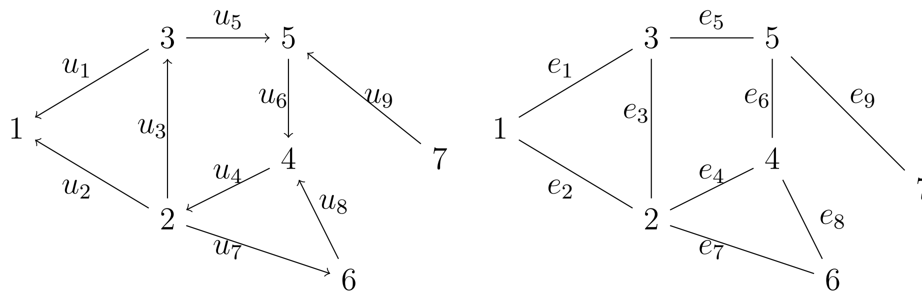


Fig 1.4.

Chaîne $=\{1, e_1, 3, e_3, 2, e_4, 4, e_6, 5, e_9, 7\} = \{1, 3, 2, 4, 5, 7\}$
Cycle $=\{1, e_2, 2, e_3, 3, e_1, 1\} = \{1, 2, 3, 1\}$
Chemin $=\{2, u_3, 3, u_5, 5, u_6, 4\} = \{2, 3, 5, 4\}$
Circuit $=\{2, u_3, 3, u_5\ 5, u_6, 4, u_4, 2\} = \{2, 3, 5, 4, 2\}$

1.2.9 Représentation de graphe en machine

On représente un graphe en machine en utilisant sa matrice d'adjacence.
Elle se définit par $A=(a_{ij})$; $i, j = 1,...,n$.
Pour un graphe orienté, on a :

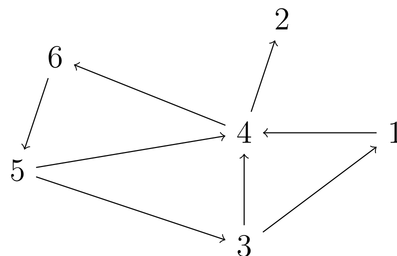
$$a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in U\,; \\ 0 & \text{sinon .} \end{cases}$$

Pour un graphe non orienté, on a :

$$a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E\,; \\ 0 & \text{sinon .} \end{cases}$$

• La matrice d'adjacence d'un graphe non orienté est symétrique.

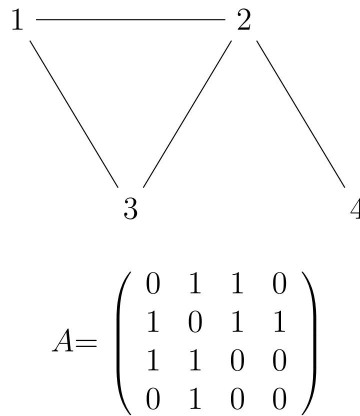
Exemple 1.5



$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Fig 1.5. Graphe orienté et sa matrice d'adjacence.

Exemple 1.6



$$A=\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Fig 1.6. Graphe non orienté et sa matrice d'adjacence

• Une matrice d'adjacence est dite creuse si elle contient beaucoup de zéros. En cas de matrice creuse, il est plus judicieux d'utiliser le tableau des successeurs ou des prédécesseurs des sommets du graphe pour faire connaître le graphe à la machine.
L'espace utilisé par les zéros de la matrice est inutilement occupé dans l'espace-mémoire de la machine.

1.2.10 Graphe complémentaire

Le graphe complémentaire de G=(X,E) est le graphe $\bar{G}$=(X,$\bar{E}$) ayant les mêmes sommets que G, et dans lequel deux sommets sont reliés par une arête si et seulement s'ils ne le sont pas dans G. Autrement dit, $xy \in \bar{G} \Leftrightarrow xy \notin G\,;\ \forall\ x, y \in X$.

1.2.11 Graphe partiel

Un graphe partiel de G=(X,E) engendré par $V \subseteq E$, est le graphe G_V=(X,V) dont les sommets sont ceux de G, et dont les arêtes sont celles de V.

1.2.12 Sous-graphe

Soit G=(X,E) un graphe et $X^{'} \subseteq X$. Le sous-graphe de G, engendré par $X^{'}$, est le graphe $G_{X'} = (X^{'},E^{'})$ avec $E^{'}= \{\ e = (x,y) \in E\ /\ x \in X^{'}$ et $y \in X^{'}\ \}$.

Sous-graphe induit de $G = (X, E)$ est un graphe $G^{'}$ tel que :
$G^{'} = (X^{'}, E^{'})$ avec $X^{'} \subset X$ et $E^{'}=\{(x, y) \in E,\ x \in X^{'}$ et $y \in X^{'}\}$.

Sous-graphe partiel de $G = (X, E)$ est un graphe $G^{'}$ tel que :
$G^{'} = (X^{'}, E^{'})$ avec $X^{'} \subset X$ et $E^{'} \subset \{(x, y) \in E,\ x \in X^{'}$ et $y \in X^{'}\}$.

• Tout sous-graphe induit est donc partiel mais le contraire n'est pas nécessairement vrai.

Exemple 1.7

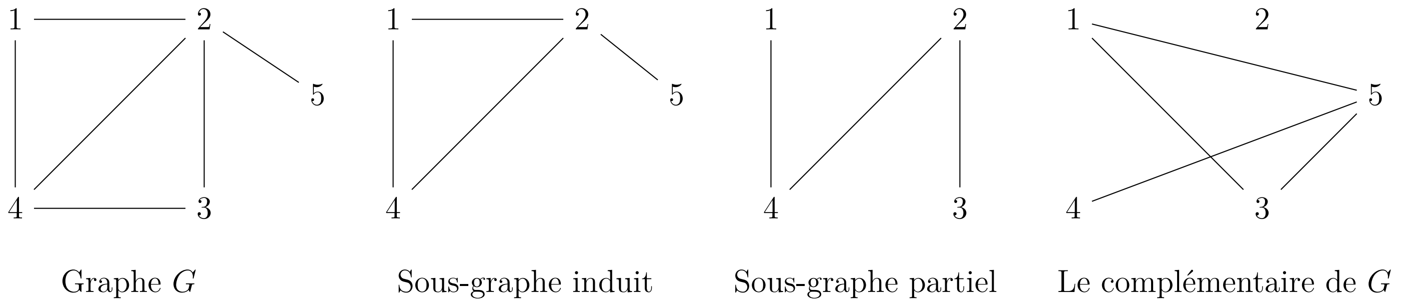


Fig 1.7.

1.2.13 Graphe transitif

Soit $G = (X, U)$ un graphe orienté, G est transitif si
$(x, y) \in U$ et $(y, z) \in U \Rightarrow (x, z) \in U$, $\forall\ x, y, z \in X$

1.2.14 Graphe symetrique

$G = (X, U)$ est dit symetrique si : $\forall\ x, y \in X$,$(x, y) \in U \Rightarrow (y, x) \in U$.
Il sera appelé anti-symetrique si $(x, y) \in U \Rightarrow (x, y) \notin U$.

1.2.15 Graphe complet

Un graphe $G = (X, E)$ est dit complet si tous les sommets sont adjacents, deux à deux, soit :
$\forall\ x$,$y \in X$, $xy \in E$.

Exemple 1.8

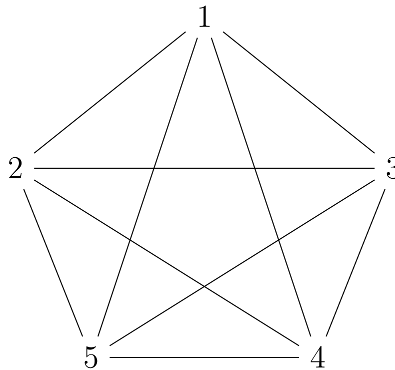


Fig 1.8. Graphe complet.

1.2.16 Cliques et stables

Une clique est un sous-graphe complet.

Les notions de stable ou ensemble indépendant et graphe complet sont complémentaires.
$G = (X, E)$ un graphe. $S \subset X$ est un stable de G ou ensemble indépendant si $\forall\ x, y \in S$, $xy \notin E$.

- Une clique est dite maximale si elle n'est pas contenue dans une autre clique.
- Une clique d'ordre n est notée K_n (le graphe précédent est un K_5).

Exemple 1.9

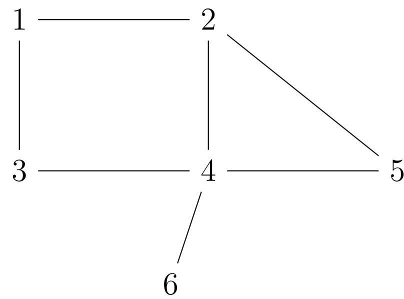


Fig 1.9.

$Clique = \{2, 4, 5\}$
$Stables = \{1, 5, 6\}; \{2, 3, 6\}; \{3, 5, 6\}$

1.3 Quelques classes de graphes

1.3.1 Graphe triangulé

• Une corde est une arête reliant deux sommets non consécutifs du cycle ou d'une chaîne. Un graphe G est dit triangulé si tout cycle de longueur > 3 admet une corde.

Exemple 1.10

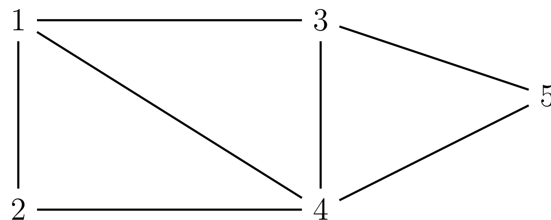


Fig 1.10. Graphe triangulé.

1.3.2 Graphe de comparabilité

G est de comparabilité s'il admet une orientation transitive de ses arêtes ; c'est-à-dire, $\forall\ x,y,z \in U$, $(x,y) \in U$ et $(y,z) \in U$ alors $(x,z) \in U$

Exemple 1.11

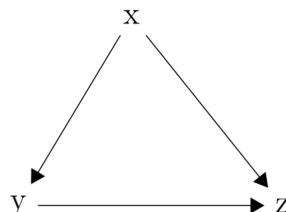


Fig 1.11. Graphe de comparabilité.

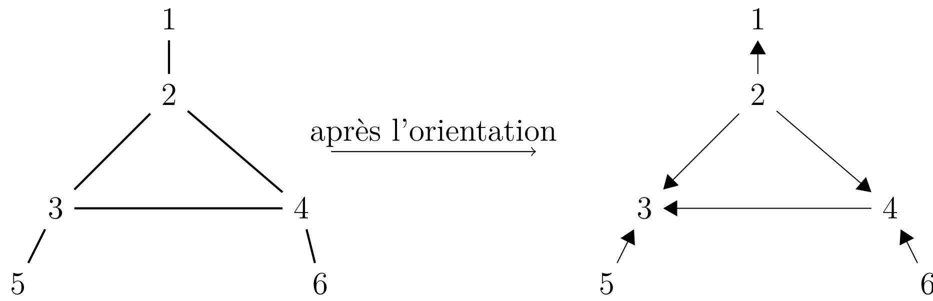


Fig 1.12.

Ce graphe n'est pas de comparabilité car si on essaye d'orienter les deux arêtes 34 et 35 d'une façon transitive, on ne pourra pas car il y a une arête qui manque qui est l'arrête 45, même chose pour les deux arêtes 46 et 34.

1.3.3 Graphe d'intervalles

Soit G=(X,E) un graphe.
$x \in X \longrightarrow I_x$ un intervalle de $\mathbb{R}$
$y \in X \longrightarrow I_y$ un intervalle de $\mathbb{R}$
G est d'intervalles si $\forall$ x,$y \in X$, $xy \in E \Leftrightarrow I_x \cap I_y \neq \emptyset$ (I_x et I_y se chevauchent)

Exemple 1.13

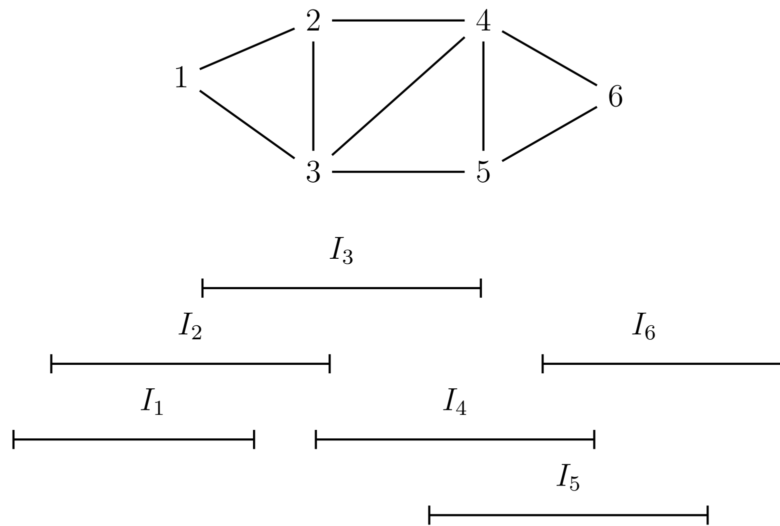


Fig 1.13. Graphe d'intervalles et sa représentation intervallaire.

De cette représentation on constate entre autre que.
$I_3 \cap I_2 \neq\emptyset$ et $23 \in E$.
$I_1\cap$ $I_3 \neq \emptyset$ et $13 \in E$.
$I_2 \cap I_4 \neq \emptyset$ et $24 \in E$.

Par contre
$I_1 \cap I_4 = \emptyset$ et $14 \notin E$.
$I_3 \cap I_6 = \emptyset$ et $36 \notin E$.
$I_2 \cap I_5 = \emptyset$ et $25 \notin E$.

Théorème 2(*Gilmore,Hoffman,1964*)[6]
Un graphe simple G est représentatif d'une famille d'intervalles si et seulement si on a les deux conditions suivantes :
(1) G est triangulé,
(2) $\bar{G}$ est de comparabilité.

Exemple 1.14

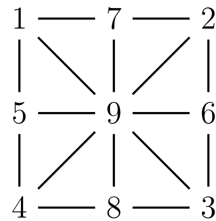


Figure 1.14.

Ce graphe n'est pas d'intervalles car il n'est pas triangulé, on a C={1,7,2,6,3,8,4,5,1}.

1.4 Notions de complexité algorithmique

La complexité algorithmique est une branche des mathématiques et de l'informatique, ayant pour cadre l'étude de la difficulté intrinsèque des problèmes algorithmiques, et qui vise à classer ces problèmes en fonction de cette difficulté.

Pour évaluer et comparer les algorithmes, on commence par les programmer, puis mesurer leurs durées d'exécution. En fait, le temps de calcul est une mesure imparfaite car elle dépend trop de la machine, du langage de programmation, du compilateur particulier utilisé pour ce langage (constructeur, version, options de compilateur choisi), et des données.
En pratique, on préfère compter le nombre d'opérations élémentaires de l'algorithme ; ce nombre ne dépend ni de la machine, ni du langage, et peut s'évaluer sur le papier.

Opération élémentaire
La notion d'opération élémentaire dépend en fait de la nature du problème traité et des calculs effectués. En général, il s'agit d'opérations arithmétiques $(+, -, \times, \div, \cap, \cup, ...)$ pour un algorithme de tri, la comparaison est considérée comme une opération élémentaire.

Définition
La complexité d'un algorithme A est une fonction $C_A(n)$, donnant le nombre d'instructions caractéristiques exécutées par A dans le pire des cas, pour une donnée de taille n. Par exemple, quand on passe de petits jeux d'essai à de grands problèmes.
Ceci permet aussi d'estimer la taille maximale des problèmes qu'on peut résoudre.

- **Complexité temporelle**

C'est le nombre d'opérations élémentaires qu'effectue un algorithme ; c'est un temps théorique dépendant de la taille du problème.

- **Complexité spaciale**

Elle mesure l'espace mémoire que l'algorithme utilise au cours de son exécution.
L'idéal pour un algorithme est qu'il soit rapide et utilise le minimum d'espace mémoire.

La théorie de la complexité ne s'intéresse qu'aux problèmes dont la solution est l'écriture d'un algorithme. Elle ne considère pas les problèmes dont la solution est trouvée par hasard ou à la suite d'un choix aléatoire.

1.4.1 Classes des problèmes

Problème de décision ou problème de reconnaissance
Ceci consiste à chercher dans un ensemble fini S, l'existence d'un élément s vérifiant une propriété p. Les problèmes de décision sont des problèmes auxquels la réponse est oui ou non.

Exemple de problème de décision
Donnée : Soit un graphe orienté G=(X,U) et deux sommets x_1 et x_2 de X.
Question : Existe-t-il un chemin de x_1 à x_2 ?

La classe P (ou determinist polynomial)
Un problème de décision est dans la classe P si on peut le résoudre en temps polynomial en fonction de la taille de son entrée, donc l'algorithme peut être résolu en temps de complexité temporelle $O(n^t)$, où n est la taille du problème et t, une constante.
Quelques problèmes appartenant à cette classe : Problème du flot maximum, problème du plus court chemin,etc...

La classe NP
C'est la classe des problèmes pour lesquels une solution est vérifiable en temps polynomial .
NP inclut P, on peut facilement vérifier cette inclusion, comme le problème dispose d'un algorithme de résolution polynomiale, le superviseur pourra exécuter cet algorithme pour voir s'il obtient le même résultat que nous.

La classe des problèmes NPC (ou non determinist polynomial)
Il s'agit des problèmes les plus difficiles de NP, la notion de problèmes NPComplets est basée sur celle de la transformation polynomiale d'un problème. La technique utilisée pour montrer qu'un problème est NPComplet est de comparer ce dernier à un autre problème connu dans la même classe.
C'est-à-dire, pour prouver que P_1 de NP est NPC, on montre qu'un problème NPComplet P_2 connu peut se transformer polynomialement en P_1, donc P_2 étant dans NPC, tous les problèmes de NP s'y transforment polynomialement.

Les problèmes difficiles sont dits aussi exponentiels ; leur complexité peut être a^n, n^n, $\exp(n)$,etc... En général, un algorithme polynomial est plus rapide qu'un algorithme non polynomial. Très peu de cas font exception. Prenons l'exemple de la programmation linéaire, l'algorithme du simplexe, connu pour être exponentiel, peut faire mieux qu'un algorithme polynomial en pratique.

En 1971, Stephen A.Cook fut le premier à avoir prouvé l'existence de tels problèmes, comme : le problème du stable maximum, du transversal minimum, de la coloration minimale,etc...

1.4.2 Exemple de problème difficile

Problème de satisfiabilité ou problème SAT
Le problème SAT est le premier problème difficile. Il contient n variables booléennes x_i et un ensemble de clauses (unions de variables complémentées ou non comme $x_1 \vee x_3 \vee \bar{x}_5$)
Soit l'équation booléenne suivante :
$S= C_1 \wedge C_2 \wedge C_3 \wedge ... \wedge C_m$

Peut-on affecter à chaque variable une valeur vrai ou faux (1 ou 0) de façon à rendre vraies toutes les clauses ?

Exemple 1.14
S=($x_1 \vee \bar{x}_2 \vee x_3$)$\wedge$ (x_2)$\wedge$ ($x_2 \vee \bar{x}_3$)
S=1 a une solution ?
Pour x_1=x_2=x_3=1, S a une solution .
Donc la solution de notre problème est (1,1,1).
Le problème SAT équivaut à tenter de rendre vraie l'intersection des clauses. Il est crucial pour l'informatique théorique et l'intelligence artificielle. Par exemple, dans les systèmes experts, le problème de savoir si un but donné peut être prouvé en partant d'un certain nombre de faits peut se ramener à SAT.

1.4.3 La conjecture $P \neq NP$

Une question cruciale est de savoir si un problème NPComplet peut être résolu polynomialement, au cas où P=NP, ou bien si on ne lui trouvera jamais d'algorithme polynomial, au cas où P≠NP. Cette question n'est pas définitivement tranchée. La découverte d'un algorithme polynomial pour un seul problème NPComplet permettrait de les résoudre tous facilement. Pour ne pas détruire la théorie de la complexité connue, on préfère conjecturer P≠NP. La preuve de cette conjecture nécessitera probablement l'invention de nouvelles mathématiques, ou concevoir de nouveaux calculateurs comme l'ordinateur quantique.

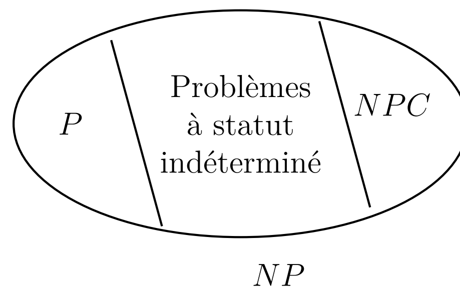


Fig 1.15. Représentation des classes de complexité.

Chapitre 2

Minimum Feedback Vertex Set dans un graphe d'intervalles

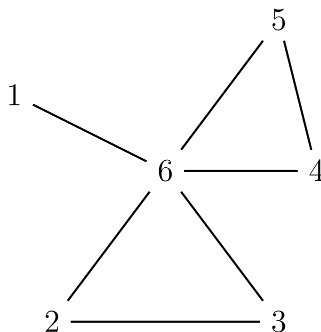
2.1 Introduction

Les graphes d'intervalles surviennent lors de modélisation de diverses situations réelles, impliquant en particulier des dépendances temporelles ou d'autres restrictions de nature linéaire.
On peut citer quelques exemples où les graphes d'intervalles interviennent :
La biologie moléculaire, la génétique, la planification de trafic, l'organisation de fichiers,...
Dans ce chapitre, nous travaillons sur les graphes d'intervalles non orientés et non valués, où l'on cherche un ensemble minimum de sommets qu'on doit supprimer afin de détruire tous les cycles (le graphe devient acyclique). Ce problème est connu pour être NPComplet dans les graphes en général.

2.2 Définition du feedback

Soit $G = (X, E)$ un graphe ; $F \subset X$.
L'ensemble F est un feedback (**FVS**) de G si : $G(X - F)$ n'a pas de cycle .
F est un feedback minimum (**MFVS**) de G si $|F| \leq |F^{'}|$, $\forall$ $F^{'}$ feedback de G. On le note F_{min}.

Exemple 2.1
Soit le graphe suivant :



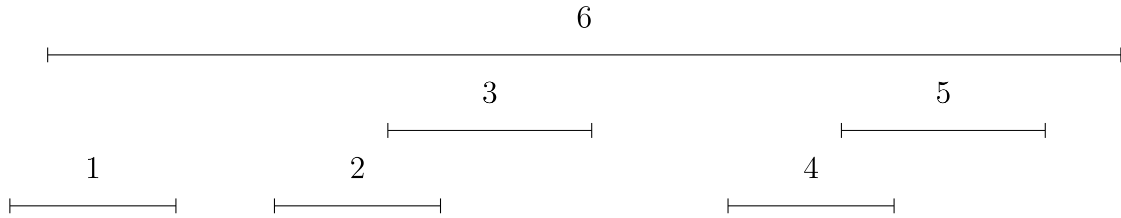


Fig 2.1. Graphe ayant un seul MFVS.

Pour cet exemple on a F_1={2,5} ; F_2={3,4} ; F_3={2,4} ; F_4={3,5,6} ; ...
On a F_{min}={6}.

2.3 Structure de données et préliminaires

Anita Saha et Madhumangal Pal ont travaillé sur un algorithme qui détermine le feedback minimum dans un graphe d'intervalles G, ainsi que sa complexité dans le pire des cas qui est lineaire (O(n+m)).

Dans ce qui suit, $G = (X, E)$ est un graphe d'intervalles non orienté, non valué.
Soit I={I_1,I_2,...,I_n}, la représentation intervallaire de G où :
a_i est l'extrémité gauche et b_i l'extrémité droite d'un intervalle du sommet i tel que, I_i=[a_i,b_i] ; $\forall$ i=1,2,...,n.
Sans perte de généralités, on accepte ce qui suit :

1. Les intervalles dans I sont indexés par les extrémités à droite en ordre croissant i.e.,$b_1 < b_2 < ... < b_n$.
2. Les intervalles sont fermés i.e, ils contiennent toutes les extrémités et il n'y a pas deux intervalles qui partagent la même extrémité.
3. Les sommets d'un graphe d'intervalles et les intervalles représentés sur la droite réelle sont équivalents.
4. Un graphe d'intervalles est connexe et la liste de ses extrémités finales est ordonnée.

Afin de déterminer le **MFVS** d'un graphe G, on définit d'abord toutes les cliques maximales. Une brève description est donnée ci-dessous.
Traçons des lignes verticales fictives en chaque extrémité "b" puis on définit tous les intervalles coupés par ces lignes, et de là qu'on forme toutes les cliques maximales.
Si une ligne verticale tracée passe par un point quelconque qui est entre deux "b" consécutifs, alors le nombre d'intervalles coupant cette ligne est soit le même ou bien inférieur au nombre d'intervalles dessinés sur le point "b".

Soit B_i l'ensemble de sommets correspondant à l'intervalle i et aux intervalles coupés par les lignes verticales fictives, $\forall$ i=1,2,..,k.
Le plus petit élément de chaque clique maximale noté et ordonné par ordre croissant, disant P_1,P_2,...,P_k où k est le nombre totale de cliques.
Soit S_j l'ensemble de sommets formant une clique où P_j est son plus petit élément.
Alors, les cliques maximales sont S_1,S_2,...,S_k.

Pour chaque j, tous les sommets appartenant à B_{pj} sont dans un ordre croissant.

Posons :

S_j : clique maximale, $|S_j|=N_j$, $\forall$ j=1,..,k.

$S_j(l)$ le l^{eme} élément de la j^{eme} clique maximale, où l=1,2,..,N_j.

F_{min}= C_k, $\forall$ i=**1...**k**.**

C_1 : l'ensemble de tous les sommets de S_1 autres que les deux premiers tels que C_1=S_1-$\{x_1,x_2\}$.

C_2 : l'ensemble de sommets dont les éléments sont l'union de C_1 et S_2 autres que les deux premiers sommets.

En général C_i=$C_{i-1}\cup$**{**S_i-C_{i-1}-**{**x_1**,**x_2**}}**
où x_1**,** x_2 sont les deux premiers sommets de S_i-C_{i-1}.

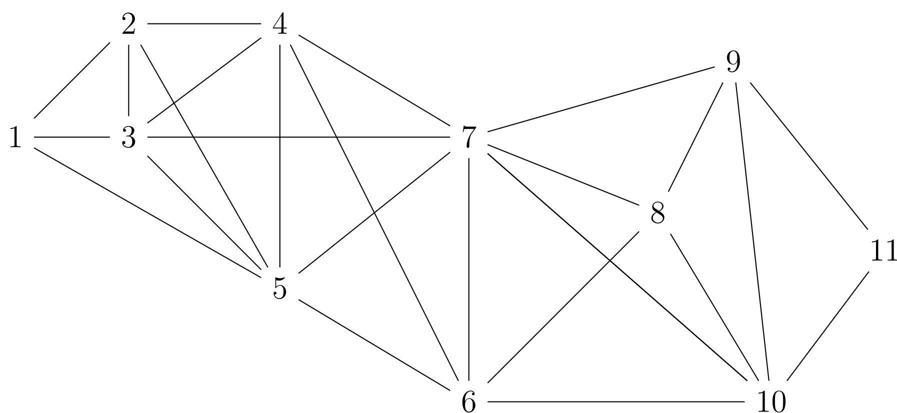
2.3.1 Vérification du minimum

Les arguments suivants montrent que l'ensemble de sommets F obtenu par la méthode ci-dessus est minimum.
Pour la sélection des éléments de l'ensemble du Feedback F, on considère le sommet correspondant à l'intervalle dont la longueur est étendue le plus à droite de "b" tout en respectant l'ordre des "b_i", et est connecté au sommet correspondant au point final droit. La sélection des intervalles qui sont plus à droite nous donne F_{min}.

2.3.2 Lemme 1 [2]

Pour que F soit un Feedback minimum d'un graphe d'intervalles $G(X, E)$, il faut que $G(X - F)$ devienne sans cycle, et cela se produit quand la représentation intervallaire $I(X - F)$ a au plus deux intervalles qui se coupent sur la ligne verticale fictive, à n'importe quel point de la droite réelle, où $I(X)$ est la représentation d'intervalles de G.

Exemple 1.2



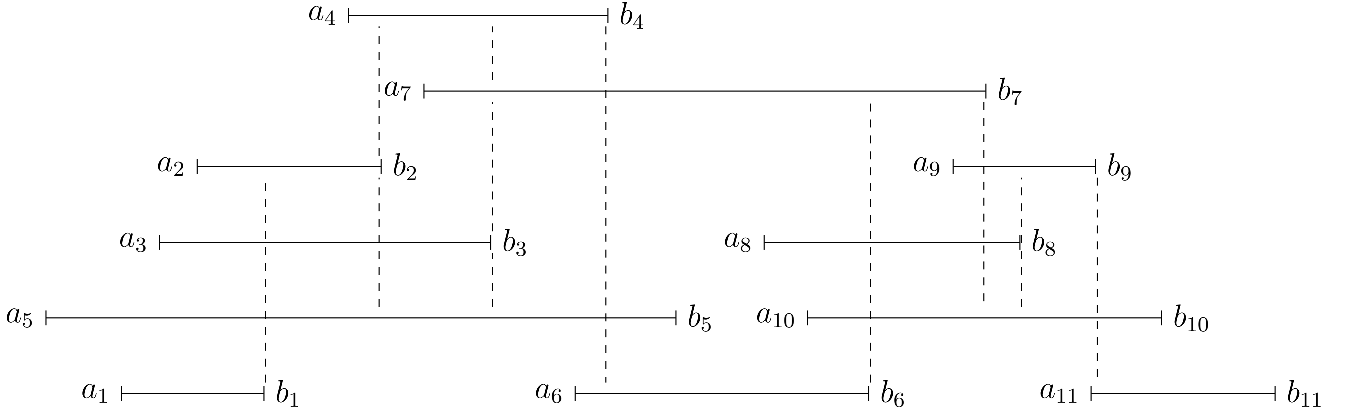


Fig 2.2. Graphe d'intervalles et sa représentation intervallaire.

$B_{p1}=S_1=\{1,2,3,5\}$; $B_{p2}=S_2=\{2,3,4,5\}$; $B_{p3}=S_3=\{3,4,5,7\}$; $B_{p4}=S_4=\{4,5,6,7\}$; $B_{p5}=S_5=\{6,7,8,10\}$; $B_{p6}=S_6=\{7,8,9,10\}$; $B_{p7}=S_7=\{9,10,11\}$

où $P_1=1$; $P_2=2$; $P_3=3$; $P_4=4$; $P_5=6$; $P_6=7$; $P_7=9$.

Les cardinalités N_j de S_j de ce graphe sont : $N_1=4$; $N_2=4$; $N_3=4$; $N_4=4$; $N_5=4$; $N_6=4$; $N_7=3$.

On a aussi $S_1(3)=3$; $S_1(4)=5$; $S_2(2)=3$,....

Soient :
C_1 l'ensemble des sommets de S_1 sauf les deux premiers éléments,
$C_1=\{1,2,3,5\}-\{1,2\}=\{3,5\}$.
C_2 l'ensemble des sommets de $C_1\cup(S_2-C_1)$ sauf les deux premiers éléments,
$C_2=C_1\cup\{S_2-C_1-\{x_1,x_2\}\}=\{3,5\}\cup\{\{2,3,4,5\}-\{3,5\}-\{x_1,x_2\}\}=\{3,5\}$.

Calculons avec cette formule $C_k=C_{i-1}\cup\{S_i-C_{i-1}-\{x_1,x_2\}\}$, C_3,C_4,C_5,C_6,C_7.

$C_3=C_2\cup\{S_3-C_2-\{x_1,x_2\}\}=\{3,5\}\cup\{\{3,4,5,7\}-\{3,5\}-\{x_1,x_2\}\}=\{3,5\}\cup\{\{4,7\}-\{4,7\}\}=\{3,5\}$

$C_4=C_3\cup\{S_4-C_3-\{x_1,x_2\}\}=\{3,5\}\cup\{\{4,5,6,7\}-\{3,5\}-\{x_1,x_2\}\}=\{3,5\}\cup\{\{4,6,7\}-\{4,6\}\}=\{3,5,7\}$

$C_5=C_4\cup\{S_5-C_4-\{x_1,x_2\}\}=\{3,5,7\}\cup\{\{6,7,8,10\}-\{3,5,7\}-\{x_1,x_2\}\}=\{3,5,7\}\cup\{\{6,8,10\}-\{6,8\}\}=\{3,5,7,10\}$

$C_6=C_5\cup\{S_6-C_5-\{x_1,x_2\}\}=\{3,5,7,10\}\cup\{\{7,8,9,10\}-\{3,5,7,10\}-\{x_1,x_2\}\}=\{3,5,7,10\}\cup\{\{8,9\}-\{8,9\}\}=\{3,5,7,10\}$

$C_7=C_6\cup\{S_7-C_6-\{x_1,x_2\}\}=\{3,5,7,10\}\cup\{\{9,11\}-\{9,11\}\}=\{3,5,7,10\}$

$F=\{3,5,7,10\}$.

Vérifions que $|F|$ est minimum
A l'extrémité b_1 l'intervalle le plus étendu à droite est I_5, il nous reste encore quelques cycles non détruits donc on prend l'intervalle qui le précède qui est I_3.
En b_4 le plus étendu à droite est I_7.
En b_6 le plus étendu à droite est I_{10}.
$F_{min}=\{3,5,7,10\}$.

Remarques
• Plus le sommet est à gauche plus il a moins de chance d'être dans l'ensemble du feedback minimum.
• L'algorithme de recherche du feedback présenté ici nous donne un seul ensemble.

2.3.3 Théorème 1 [2]

F_{min} est un Feedback.

Preuve
D'après la définition du Feedback, l'ensemble de sommets F est un feedback si et seulement si le graphe $G(X - F)$ n'a pas de cycle. Soit $I(X - F)$ l'ensemble des intervalles correspondants au graphe $G(X - F)$.
A partir de la construction de l'ensemble $I(X - F)$ qui a au plus deux intervalles qui se coupent à n'importe quel point de la droite réelle, donc $G(X - F)$ est sans cycle.
Donc F est un feedback.

Remarque
La cardinalité de F_{min} commence à partir de zéro.

Exemple 2.3

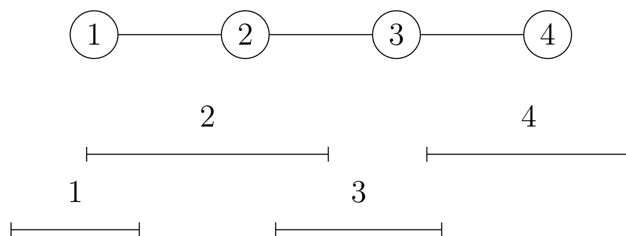


Fig 2.3. Graphe d'intervalles acyclique.

Le graphe ci-dessus est un graphe d'intervalles sans cycle, alors $F_{min}=\emptyset$.

2.4 Complexité de l'algorithme MFVS

2.4.1 Lemme 2 [2]

La complexité temporelle de l'algorithme **MFVS** est $O(n + m)$, où n représente les sommets et m les arêtes du graphe.

Preuve
Toutes les cliques maximales d'un graphe d'intervalles peuvent être calculées en temps $O(n + m)$, pour ordonner les sommets de ces cliques en ordre croissant on prend un temps $O(n)$, et le calcul des C_i prend également un temps $O(n)$. Donc la complexité globale de cet algorithme est $O(n+m)$.

2.4.2 Théorème 2 [2]

Le feedback minimum d'un graphe d'intervalles peut être calculé en temps $O(n+m)$ (complexité lineaire).

2.5 Sécurisation du feedback minimum

Dans une entreprise, un ensemble de serveurs assurent leurs tâches en chaîne. Pour que le processus ne soit pas interrompu, il faut que chacun des serveurs ait un remplaçant qui assure la même tâche que lui. Ces entreprises optent pour un minimum de remplaçants qu'on appelle sécurisants de l'ensemble de serveurs. Les remplaçants doivent appartenir au même service.

2.5.1 Définition

La sécurisation consiste à trouver des sommets X' dans $G(X-F)$ qui font le même travail que les sommets de l'ensemble F (détruire tous les cycles).

- Un sommet $x \in F$ est sécurisé s'il existe $y \in X-F$ tel que $F\backslash\{x\}\cup\{y\}$ est un feedback.
- Un feedback F est sécurisé si $\forall\ x \in F$, x est sécurisé.

2.5.2 Comment vérifier la sécurisation de l'ensemble F_{min}

On commence par déterminer tous les cycles de longueur 3.
Afin de savoir si un sommet x_1 est sécurisé ou non, on prend tous les cycles qui le contiennent puis on le remplace par l'ensemble de sommets $(X-F)$ de ces cycles dans F_{min}.
Si tous les cycles sont détruits donc x_1 est sécurisé sinon il ne l'est pas.
Si on arrive à sécuriser tous les éléments du feedback, on obtient alors un F_{min} sécurisé .
Pour dire que F_{min} n'est pas sécurisé, il suffit qu'un seul sommet du feedback soit non sécurisé.

Exemple 2.4

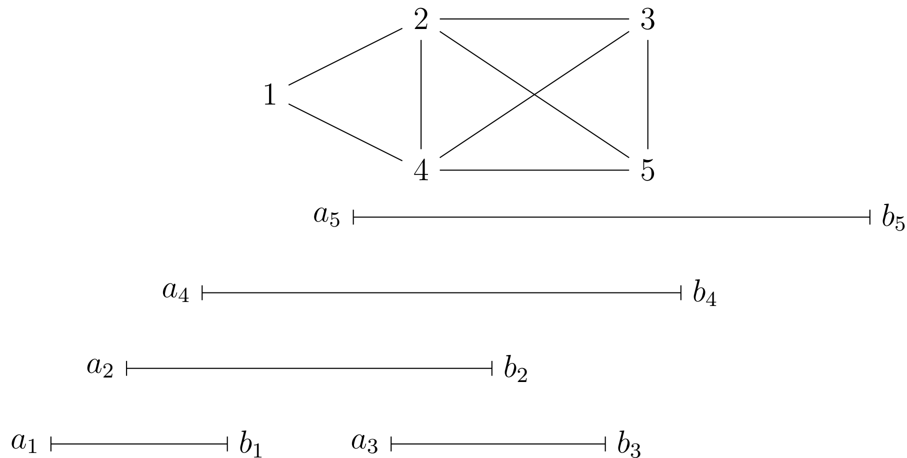


Fig 2.4. Graphe avec F_{min} sécurisé.

Les cliques maximales sont S_1={1,2,4} S_2={2,3,4,5}.
Le calcul des C_i
C_1={1,2,4}-{1,2} = {4}
C_2= {4} $\bigcup${{2,3,4,5}-{4}-{2,3}} ={4,5}
On a aussi :
I_4 et I_5 sont les plus étendus à droite en b_1 et b_2 respectivement.
F_{min} = {4,5}

La sécurisation

Les cycles de longueur 3 sont : 234, 235, 345, 124.
Pour vérifier la sécurisation de 4, on fixe 5 et on remplace 4 par les sommets contenus dans les mêmes cycles que lui
{ .,5} =F_{min} ?
25 ? 35 ? 15 ?
On constate que {2,5} détruit tous les cycles donc 4 est sécurisé par 2.
Pour 5 on a {4, .} =F_{min} ?
42 ? 43 ?
{4,2} et {4,3} détruisent tous les cycles donc 5 est sécurisé par 2 et 3.
On conclut donc que F_{min}={4,5} est sécurisé.

Remarque

• Un graphe peut avoir plusieurs feedback minimum, comme dans l'exemple précédent, on a :
F_{min1}= {2,3},
F_{min2}= {2,5},
F_{min3}= {3,4},
F_{min4}= {2,4}.

Exemple 2.5

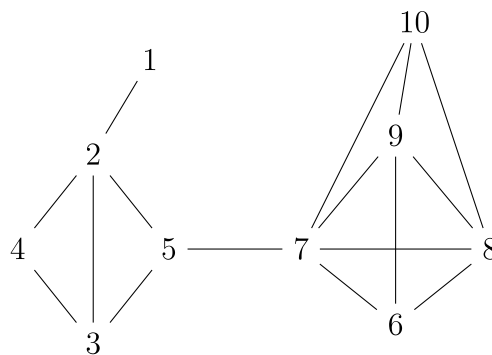


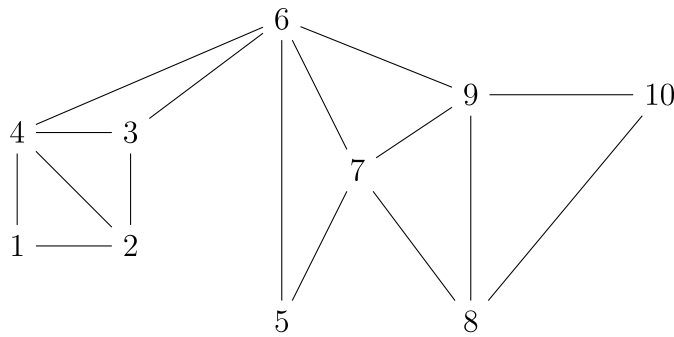
Fig 2.5. Graphe avec un F_{min} sécurisé.

S_1={2,3,4}, S_2={2,3,5}, S_3={6,7,8,9}, S_4={7,8,9,10}
C_1={4}, C_2={4,5}, C_3={4,5,9,8}, C_4={4,5,9,8}
F_{min}={4,5,8,9}
4 est sécurisé par 2
5 est sécurisé par 3
8 est sécurisé par 7
9 est sécurisé par 7
Donc F_{min} est sécurisé.

2.5.3 Remarques

• Deux sommets appartenant à un **MFVS** peuvent être sécurisés par le même sommet.
• Un sommet peut être sécurisé par un seul sommet ou plusieurs.

Exemple 2.6



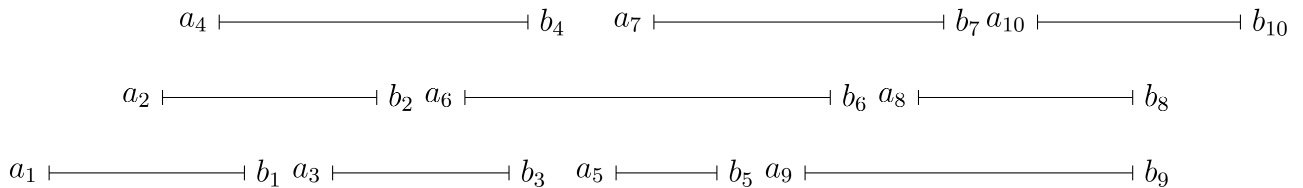


Fig 2.6. Graphe ayant un F_{min} non sécurisé.

Les cliques maximales
S_1 ={1,2,4}, S_2={2,3,4}, S_3={3,4,6} , S_4 = {5,6,7}, S_5={6,7,9}, S_6={6,7,9,8} S_7={7,8,9}et
S_8={8,9,10}
C_1=C_2=C_3={4}
C_4=C_5=C_6 ={4,7}
$C_7 = \{4, 7, 10\}$
F=C_k =C_7 = {4,7,10}

d'un autre coté on a :
En b_1 le plus étendu à droite est I_4
En b_5 le plus étendu à droite est I_7
En b_8 le plus étendu à droite est I_{10}
Donc F_{min} ={4,7,10}

La sécurisation
Les triangles sont : 142, 243, 346, 567, 679, 789, 8910.
• Pour (4) : {.,7,10} ?
1710 ? 2710 ? 3710 ? 6710 ?
Il n'y a aucun sommet qui détruit les cycles parmi 1, 2, 3 et 6, donc 4 n'est pas sécurisé.
• pour (7) : {4,.,10} ?
4510 ? 4610 ? 4810 ? 4910 ?
Il reste toujours quelques cycles non détruits donc 7 n'est pas sécurisé.
• Pour (10) : {4,7,.} ?
478 ? 479 ? Tous les cycles sont détruits donc 10 est sécurisé par 8 et 9.
on a 4 et 7 ne sont pas sécurisés, alors F_{min} n'est pas sécurisé.

2.5.4 Lemme 3 [2]

En éliminant tous les triangles, tous les cycles sont détruits (on ne risque pas d'oublier un cycle de longueur > 3).

Preuve
Les graphes d'intervalles sont des graphes triangulés donc par définition ils ne contiennent que des triangles. Ce qui prouve le lemme .

2.5.5 Lemme 4 [2]

La complexité temporelle de la procédure de sécurisation d'un feedback est $O(n^2)$.

Preuve
Pour sécuriser un feedback, on compare les sommets du feedback (cardinalité de F) aux n autres sommets restants donc la complexité est $O(n \times |F|)$.
La complexité temporelle dans le pire des cas de la procédure de sécurisation est $O(n^2)$.

2.6 Comment sécuriser un Feedback non sécurisé ?

2.6.1 Couverture minimale par sommets(transversal minimal)

Soit un graphe $G = (X, E)$. Un sous-ensemble de sommets T est un transversal de G si toute arête de G a, au moins, une extrémité dans T. Le but est de trouver un transversal de cardinal minimal. Un transversal "couvre" donc toutes les arêtes, d'où le nom synonyme de *couverture des arêtes par des sommets*.

Exemple 2.7

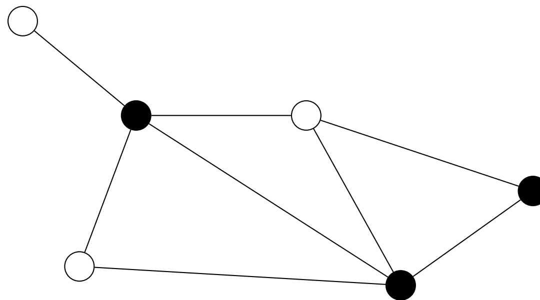


Fig 2.7. Exemple de transversal minimal.

Supposons qu'on veuille surveiller les couloirs d'un bâtiment par des caméras pivotantes. Une caméra, à une intersection, permet de surveiller tous les couloirs adjacents. Modélisons le problème par un graphe simple $G = (X, E)$. X est l'ensemble des intersections, et les arêtes corespondent aux couloirs. L'ensemble minimal de caméras à installer pour surveiller tous les couloirs est justement un transversal minimal de G.

2.6.2 Ajout de sommets

Les entreprises préfèrent toujours avoir un ensemble de serveurs sécurisés que d'avoir un ensemble minimum de serveurs.
Pour cela, une autre méthode peut s'appliquer pour sécuriser le F_{min} non sécurisé, qui est l'ajout d'un minimum de sommets assurant également la même tâche que le sommet non sécurisé
On obtient donc un nouvel ensemble noté F' qui est également un feedback, mais il n'est pas minimum.

Ce problème est un problème de transversal minimal car ici on cherche le minimum de sommets qui couvrent les arêtes des cycles d'un sommet qu'on veut sécuriser.

Exemple 2.8

Pour le graphe de la figure2.3

On a 4 et 7 qui ne sont pas sécurisés, pour les sécuriser on n'a qu'à ajouter des sommets qui les couvrent, c'est à dire on prend les sommets non sécurisés et on regarde les sommets qui détruisent les mêmes cycles que ces derniers.

Si on enlève 4 du feedback, on regarde les cycles où 4 appartient qui sont 124, 243 et 346.
Les sommets qui détruisent ces cycles (couvrent le sommet 4) sont 13, 23 ou 26.

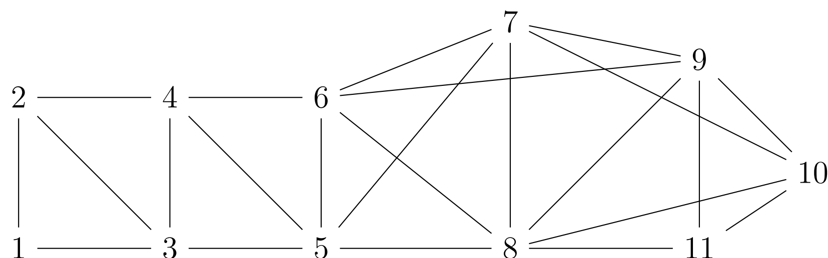
De même pour 7, les cycles qui le contiennent sont 567, 679 et 789.
les sommets qui les détruisent mis à part 7 sont 68 69 59.

Si on prend les éléments qui couvrent 4 et 7 on peut prendre 2, 6 et 9 ou 3, 6 et 9
Donc F'={2,4,6,7,9,10} qui n'est pas minimum mais sécurise F_{min}={4,7,10}

2.6.3 Exemple d'application des graphes d'intervalles dans la vie réelle

Prenons l'exemple d'un supermarché éclairé par un ensemble de lampes fluorescentes accrochées au plafond qui assurent un éclairage uniforme et important.
Le but de notre travail est d'assurer un éclairage permanent et uniforme sur l'ensemble de tous les rayons du supermarché, tout en évitant toute panne possible.
Parmi toutes les lampes, on constate que l'absence de certaines entraîne une dégradation remarquable dans la majorité des rayons.
A cet effet, pour résoudre ce problème on se ramène à modéliser ce réseau électrique avec un graphe d'intervalles, où les lampes sont considérées des sommets et les rayons sont des arêtes.

En appliquant l'algorithme proposé par Anita Saha et Madhumangal Pal, on arrive à définir un ensemble minimum de lampes fluorescentes appelé **MFVS** qui nous détermine celles qui influent le plus sur la luminosité du supermarché.
Ensuite, on le sécurise en cherchant un autre ensemble qui a la même performance que le **MFVS** et c'est ce qui nous permet d'avoir une luminosité permanente et uniforme.
Soit le graphe suivant représentant le système d'éclairage du supermarché, ils ont installé 11 lampes fluorescentes pour éclairer 23 rayons .



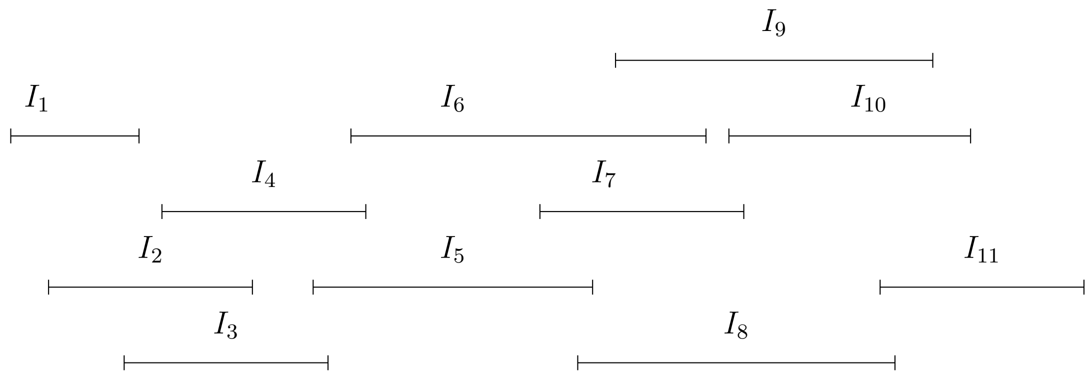


Fig 2.8. Graphe représentatif du système d'éclairage dans le supermarché.

A partir de ce graphe, on constate que dans ce supermarché il y a des lampes qui n'éclairent que 2 rayons et il y a d'autres qui éclairent jusqu'à 5 rayons, par exemple les lampes 1 et 9 .
Pour trouver l'ensemble minimum de lampes qui dégradent l'éclairage de l'espace, on calcule F_{min}.

Les cliques maximales
S_1={1,2,3} S_2={2,3,4} S_3={3,4,5} S_4={4,5,6} S_5={5,6,7,8} S_6={6,7,8,9} S_7={7,8,9,10} S_8={8,9,10,11}

Le calcul des C_i
C_1=C_2=C_3={3}
C_4={3,6}
C_5=C_6={3,6,8}
C_7=C_8={3,6,8,10}
F= C_8 = {3,6,8,10}

verification du minimum
En b_1 le plus étendu à droite est I_3 .
En b_4 le plus étendu à droite est I_6 .
En b_5 le plus étendu à droite est I_8 .
En b_7 le plus étendu à droite est I_{10} .
D'où F=F_{min} ={3,6,8,10}
Donc l'ensemble minimum de lampes est {3,6,8,10}

La sécurisation
Dans cette partie, on s'interesse à sécuriser le F_{min} pour éviter toute sorte de dégradation de lumière en cas de panne.
Vérifions la sécurisation de ces lampes,
Les triangles : 123, 234, 345, 456, 568, 567, 578, 679, 689, 7810, 8910, 81011, 9811, 879, 7910.

On vérifie pour 3, F_{min}={.,6,8,10}
123, 234, 345.
16810 ? 26810 ? 46810 ? 56810 ?
IL nous reste toujours des cycles non détruits donc 3 n'est pas sécurisé .

Pour 6, F_{min}={3,.,8,10}
456, 568, 567, 679, 689.
34810 ? 35810 ? 37810 ? 39810 ?
6 n'est pas sécurisé

Pour 8, F_{min}={3,6,.,10}
568, 578, 689, 7810, 8910, 81011, 9811, 879.

36510 ? 36710 ? 36910 ? 361110 ?
8 n'est pas sécurisé

Pour 10, F_{min}={3,6,8,.}
7810, 8910, 81011, 7910.
3687 ? 3689 ? 36811 ?
10 est sécurisé par 7 et 9.

F_{min} n'est pas sécurisé.

En ayant un **MFVS** non sécurisé, le gérant du supermarché n'est pas rassuré car en cas de panne d'un élément de cet ensemble, un nombre de rayons reste sans lumière. Alors il envisage d'ajouter des lampes au feedback minimum.
Pour s'assurer d'avoir de bonnes conditions d'éclairage on choisit d'ajouter à notre F_{min} toutes les lampes qui couvrent celles qui ne sont pas sécurisées dans F_{min}.

Donc ici on va essayer d'ajouter un minimum de sommets pour sécuriser notre premier ensemble, on a 3, 6, 8 ne sont pas sécurisés tandis que 10 est sécurisé donc on prend le sommet à ajouter dans les cycles où il n'y a pas 10.

$\underbrace{123, 234, 345}$ $\underbrace{456, 568, 567, 679, 689}$ $\underbrace{568, 689, 9811, 578, 879}$
sécurisation de 3 sécurisation de 6 sécurisation de 8,
24 25 14 59 97 711

Donc les éléments qui couvrent tous les sommets non sécurisés sont 2, 5,9 et 7 (on prend le minimum de sommets).
D'où $F^{'}$={2,3,5,6,7,8,9,10} qui est l'ensemble de lampes qui assure un éclairage uniforme et permanent du supermarché.

Chapitre 3

Algorithme et implémentation

3.1 Introduction

La réalisation d'un programme informatique de façon traditionnelle passe nécessairement par l'écriture de son code source. Tout travail consiste à analyser un problème et à décrire un moyen d'obtenir une solution. Si la programmation se sert de l'algorithmique pour être efficace, elle doit aussi être en mesure de fournir en plus des programmes à la fois lisibles, facilement utilisables et modifiables par d'autres utilisateurs. Donc un programme est obtenu après la traduction d'un algorithme dans un langage compréhensible par l'ordinateur (langage de programmation). Les langages de programmation ne cessent d'évoluer vers le plus haut niveau, à savoir deviennent toujours plus accessibles. L'un des langages de programmation les plus performants est le langage Python.

Dans ce chapitre, nous allons passer de la théorie à la pratique en programmant l'algorithme *MARK* qui calcule le feedback minimum d'un graphe d'intervalles. L'exemple pratique choisi est le système d'éclairage présenté au deuxième chapitre.

3.2 Le langage python

3.2.1 Définition

Le langage de programmation Python a été créé par Guido Van Rossum, aux Pays-bas. Son nom a été choisi en hommage à la série télévisée *Monty Python's Flying Circus* dont G. Van Rossum est fan. Le langage python est un langage informatique de haut niveau dont la syntaxe encourage à écrire un code clair et de qualité, il favorise la programmation impérative structurée et la programmation orientée objet ; dans une moindre mesure il permet de programmer dans un style fonctionnel. Il a été choisi par des acteurs majeurs comme Google, You Tube, la NASA,etc. Le langage python est un très bon choix aussi bien pour l'initiation à la programmation que pour la programmation elle même. Il est doté d'une gestion automatique de la mémoire de ramasse-miettes et d'un système de gestion d'exceptions.

3.2.2 Histoire de Python

Python est un langage de programmation objet interprété. Son origine est le langage d'exploitation AMOEBA, sa première version est sortie en 1991. Il a été développé par le néerlandais Guido Van Rossum au CWI, à l'université d'Amsterdam.

En 1995, Van Rossum continua son travail sur Python au CNRI à Reston, aux États-Unis, où il sortit plusieurs versions du logiciel. À partir d'août 1995, l'équipe Python travaille au CNRI

sur Grail un navigateur web utilisant Tk. Il est l'équivalent pour Python du navigateur HotJava, permettant d'exécuter des applets dans un environnement sécurisé. La première version publique, disponible en novembre, est la 0.2. Il a entraîné le développement de modules pour la bibliothèque standard comme rexec, htmllib ou urllib.

La version 0.6 sera la dernière de Grail ; elle est publiée en avril 1999. En 1999, le projet Computer Programming for Everybody (CP4E) est lancé avec collaboration entre le CNRI et la DARPA. Il s'agit d'utiliser Python comme langage d'enseignement de la programmation. Cette initiative conduira à la création de l'environnement de développement IDLE. Cependant, du fait du manque de financement du projet par la DARPA, et du départ de nombreux développeurs Python du CNRI (dont Guido van Rossum), le projet s'éteint en 2000. Python 1.6 fut la dernière version sortie au CNRI.

En 2000, l'équipe principale de développement de Python déménagea à BeOpen.com pour former l'équipe PythonLabs de BeOpen. Python 2.0 fut la seule version sortie à BeOpen.com. Après cette version, Guido Van Rossum et les autres développeurs de PythonLabs rejoignirent Digital Creations (à présent connue sous le nom de Zope Corporation). Andrew M. Kuchling a publié en décembre 1999 un texte nommé python warts qui synthétise les griefs les plus fréquents exprimés à l'encontre du langage.

Python 2.1 fut une version dérivée de Python 1.6.1, ainsi que de Python 2.0. Sa licence fut renommée Python Software Foundation License. Tout code, documentation et spécification ajouté, depuis la sortie de Python 2.1 alpha, est détenu par la Python Software Foundation (PSF), une association sans but lucratif fondée en 2001, modelée d'après l'Apache Software Foundation.

Afin de réparer certains défauts du langage (ex : orientation objet avec deux types de classes), et pour nettoyer la bibliothèque standard de ses éléments obsolètes et redondants, Python a choisi de casser la compatibilité ascendante dans la nouvelle version majeure : Python 3.0, publié en décembre 2008. Cette version a été suivie rapidement par une version 3.1 qui corrige les erreurs de jeunesse de la version 3.0.

La version 3.7 été publiée en juin 2018. La version 2 de Python est désormais obsolète et cessera d'être maintenue après le premier janvier 2020.

La dernière version de Python est la version 3.10, cette version est sortie le 04 octobre 2021. Voyons quelles évolutions majeures nous réserve cette nouvelle version !

Aujourd'hui, Python est un langage open source supporté, développé et utilisé par une large communauté : 300000 utilisateurs et 500000 téléchargements par an. Python offre un environnement complet de développement comprenant un interpréteur performant et de nombreux modules. Un atout indéniable est sa disponibilité sur la grande majorité des plates-formes courantes (BeOS, Mac OS X, Unix, Windows).

3.2.3 Avantages de Python

- Bibliothèques étendues

Téléchargements Python avec une bibliothèque complète et contiennent du code à des fins diverses telles que les expressions régulières, la génération de documentation, les tests unitaires, les navigateurs Web, le filetage, les bases de données, le CGI, le courrier électronique, la manipulation d'images, etc. Nous n'avons donc pas à écrire manuellement le code complet pour cela.

- Extensible

Comme nous l'avons vu précédemment, Python peut être étendu à d'autres langues. Vous pouvez

écrire une partie de votre code dans des langages comme C ++ ou C. Cela est pratique, en particulier dans les projets.

- Intégrable

Complémentaire à l'extensibilité, Python est également intégrable. Vous pouvez mettre votre code Python dans votre code source d'un langage différent, comme C ++. Cela nous permet d'ajouter capacités de script à notre code dans l'autre langue.

- Amélioration de la productivité

La simplicité du langage et les bibliothèques étendues rendent les programmeurs plus productif que les langages comme Java et C ++. En outre, le fait que vous ayez besoin d'écrire moins et de faire plus de choses.

- Opportunités IOT

Étant donné que Python constitue la base de nouvelles plates-formes comme Raspberry Pi, il trouve l'avenir brillant pour l'Internet des objets. C'est un moyen de connecter la langue avec le monde réel.

- Lisible

Parce que ce n'est pas un langage aussi verbeux, la lecture de Python ressemble beaucoup à la lecture de l'anglais. C'est la raison pour laquelle il est si facile à apprendre, à comprendre et à coder. Il n'a pas non plus besoin d'accolades pour définir les blocs, et l'indentation est obligatoire. Cela facilite davantage la lisibilité du code.

- Orienté objet

Cette langue prend en charge à la fois procédural et orienté objet paradigmes de programmation. Alors que les fonctions nous aident à réutiliser le code, les classes et les objets nous permettent de modéliser le monde réel. Une classe permet encapsulation de données et fonctionne en un seul.

- Libre et open-source

Comme nous l'avons dit plus tôt, Python est disponible gratuitement. Mais vous pouvez non seulement télécharger Python gratuitement, mais vous pouvez également télécharger son code source, y apporter des modifications et même le distribuer. Il se télécharge avec une vaste collection de bibliothèques pour vous aider dans vos tâches.

- Portable

Lorsque vous codez votre projet dans un langage comme C ++, vous devrez peut-être y apporter des modifications si vous souhaitez l'exécuter sur une autre plate-forme. Mais ce n'est pas la même chose avec Python. Ici, vous devez coder une seule fois et vous pouvez l'exécuter n'importe où. C'est appelé Écrire une fois n'importe où (WORA). Cependant, vous devez être assez prudent pour ne pas inclure de fonctionnalités dépendant du système.

- Interprété

Enfin, nous dirons qu'il s'agit d'un langage interprété. Étant donné que les instructions sont exécutées une par une, le débogage est plus facile que dans les langues compilées.

3.2.4 Inconvénients de Python

- Limitations de vitesse

Nous avons vu que le code Python est exécuté ligne par ligne. Mais comme Python est interprété, il en résulte souvent, une exécution lente. Cependant, ce n'est pas un problème à moins que la vitesse ne soit un point central du projet. En d'autres termes, à moins qu'une vitesse élevée ne soit requise, les avantages offerts par Python sont suffisants pour nous distraire de ses limitations de vitesse.

•Faible dans l'informatique mobile et les navigateurs
Bien qu'il soit un excellent langage côté serveur, Python est rarement vu sur le côté client. En plus de cela, il est rarement utilisé pour implémenter des applications basées sur smartphone. Une telle application est appelée Carbonnelle. La raison pour laquelle il n'est pas si célèbre malgré l'existence de Brython est qu'il n'est pas aussi sûr.

• Restrictions de conception
Comme vous le savez, Python est typé dynamiquement. Cela signifie que vous n'avez pas besoin de déclarer le type de variable lors de l'écriture du code. Il utilise typage de canard. Mais attendez, qu'est-ce que c'est ? Eh bien, cela signifie simplement que s'il ressemble à un canard, ce doit être un canard. Bien que cela soit facile pour les programmeurs pendant le codage, cela peut augmenter les erreurs d'exécution.

• Couches d'accès aux bases de données sous-développées
Par rapport aux technologies plus largement utilisées comme JDBC (Connectivité Java DataBase) et ODBC (Connectivité Open DataBase), les couches d'accès aux bases de données de Python sont un peu sous-développées. Par conséquent, il est moins souvent appliqué dans les grandes entreprises.

• Simple
Non, on ne plaisante pas. La simplicité de Python peut en effet être un problème. Prenez mon exemple. Je ne fais pas Java, je suis plus une personne Python. Pour moi, sa syntaxe est si simple que la verbosité du code Java semble inutile.

3.3 Algorithme et sa complexité

Pour un graphe d'intervalles et sa representation intervallaire, on calcule toutes les cliques maximales en arrangeant les sommets de chaque clique en ordre croissant puis on calcule la cardinalité de chacune. Afin de calculer les ensembles C_1, C_2,..., C_k, on fait appel au vecteur booléen *mark* utilisant l'algorithme **MARK**.
En dépendant du théorème présenté au 2eme chapitre, les importantes étapes de l'algorithme **SMFVS** pour former l'ensemble minimum de sommets du feedback d'un graphe d'intervalles sont décrites ci-dessous.

3.3.1 Algorithme SMFVS

Entrée : Les extrémités a_i, b_i de chaque intervalle $I_i = 1, 2, ..., n$.

Sortie : L'ensemble minimum de sommets du feedback F. Initialement $F = \emptyset$.

Étape 1 : Calculer toutes les cliques maximales S_j. Soit k le nombre total de cliques maximales.

Étape 2 : En chaque S_j, mettre les sommets dans un ordre croissant.

Étape 3 : Calculer N_j pour chaque $j = 1, 2, ..., n$ où N_j est la cardinalité de S_j.

Étape 4 : Calculer le vecteur $mark(i)$, pour $i = 1, 2, ..., n$, initialement $mark(i) = 0$.

Étape 5 : Mettre le sommet i dans l'ensemble F si $mark(i) = 1$, $i = 1, 2, ..., n$.

end SMFVS

Le vecteur $mark(.)$ est calculé à partir de l'algorithme suivant.

3.3.2 Algorithme MARK

```
For i = 1 to k do
        count=0;
        For j = 1 to N_i do
          if (count=2) then
            mark(j) = 1;
          else if (mark (S_i(j))= 0) then
            count=count+1;
          else
            mark(S_i(j)) = 1;
          endif;
        endfor;
endfor;
end MARK
```

3.3.3 Lemme 1 [2]

Le temps pris pour le calcul du vecteur $mark(.)$ est en $O(n+m)$.

Preuve

Le temps de calcul du vecteur $mark$ est $\sum_{i=1}^{k} N_i$; c'est la somme des cardinalités de toutes les cliques maximales. Cette complexité est en $O(n+m)$.

3.3.4 Lemme 2 [2]

L'algorithme **SMFVS** est de complexité temporelle $O(n+m)$.

Preuve

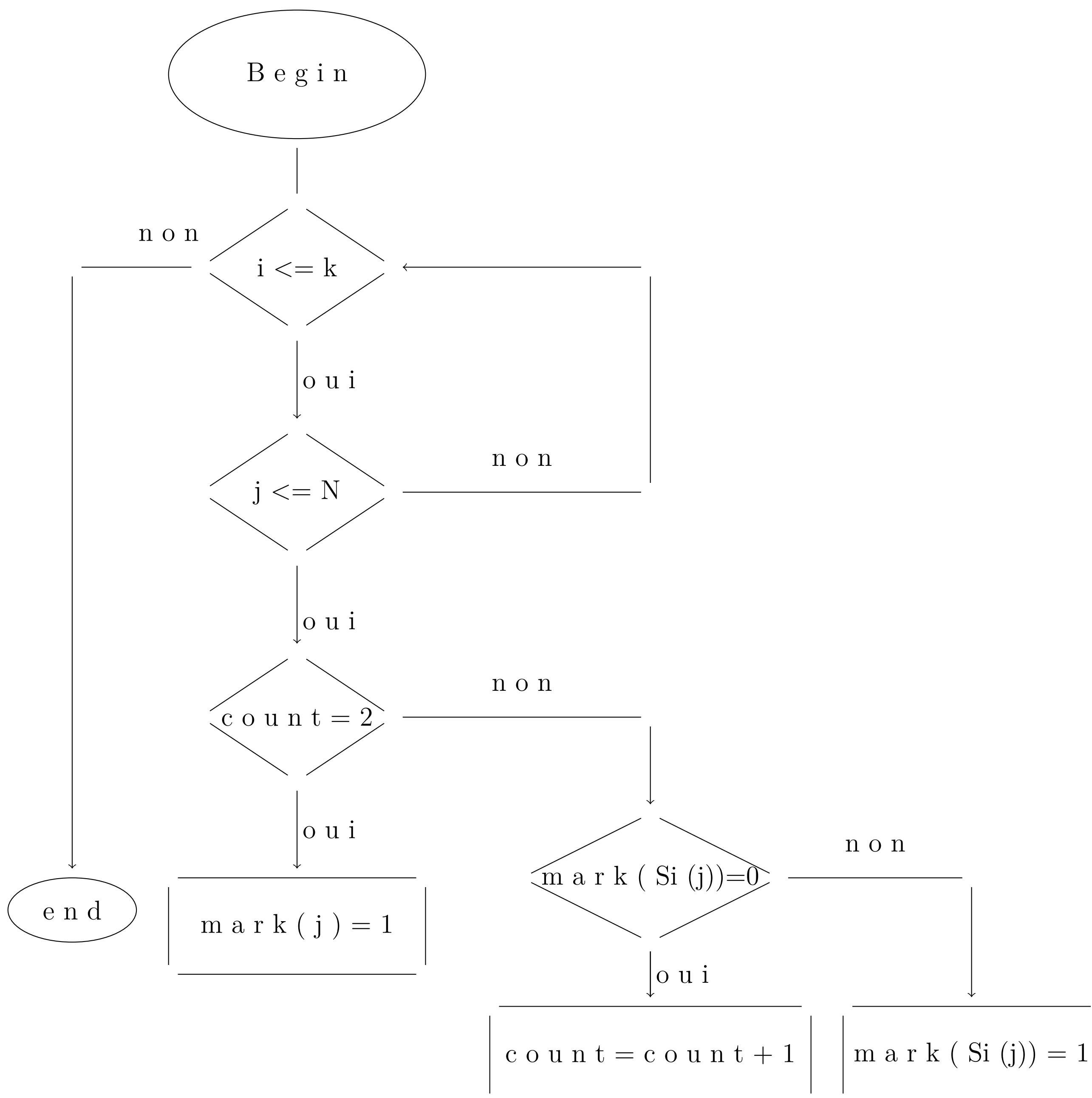
Pal et Saha ont donné un algorithme de temps $O(n+m)$ pour calculer toutes les cliques maximales d'un graphe d'intervalles avec n sommets et m arêtes. Alors l'étape 1 peut être calculée en temps $O(n+m)$. Chacune des étape 2 et étape 3 prend $O(n+m)$. L'étape 4 prend aussi un temps $O(n+m)$ pour calculer le vecteur $mark(.)$ et l'étape 5 prend également un temps $O(n)$. En général, l'algorithme **SMFVS** prend un temps de complexité $O(n+m)$.

A partir du lemme précedent on obtient le résultat suivant.

3.3.5 Théorème [2]

La cardinalité minimum de l'ensemble de sommets du feedback d'un graphe d'intervalles avec n sommets et m arêtes peut être calculée en temps $O(n+m)$.

3.4 Organigramme de l'algorithme MARK



3.5 Implémentation

Dans le deuxième chapitre,nous avons proposé un exemple d'application qui consiste à déterminer le feedback minimum dans un réseau électrique de lampes qui maintienne un éclairage uniforme et régulier dans un supermarché.

On dispose de 11 sommets où i=11, et de 8 cliques maximales où k=8, qui sont :
S_1={1,2,3} S_2={2,3,4} S_3={3,4,5} S_4={4,5,6} S_5={5,6,7,8} S_6={6,7,8,9} S_7={7,8,9,10} S_8={8,9,10,11}
$S_i(j)$ est le j^{eme} élément de la i^{eme} clique, c'est-à-dire, j est la position de l'élément dans la clique.

Les cardinalités de ces cliques maximales sont :
$N_1 = N_2 = N_3 = N_4 = 3$ et $N_5 = N_6 = N_7 = N_8 = 4$.

3.6 Déroulement de l'algorithme

Allons de i=1 jusqu'à 8, faire

Première itération
Pour i=1
count=0 ; N_1=3 ; j=1,2,3
Pour j=1 ; $mark(S_1(1) = 0)$ donc count=count+1=1
Pour j=2 ; count=1 ; $mark(S_1(2) = 0)$ donc count=count+1=2
Pour j=3 ; count=2 ; $mark$(3)=1 donc $3 \in F$

Deuxième itération
Pour i=2
count=0 ; N_2=3 ; j=1,2,3
Pour j=1 ; $mark(S_2(1) = 0)$ donc count=count+1
Pour j=2 ; $mark(S_2(2) = 1)$
Pour j=3 ; count=1 ; $mark(S_2(3) = 0)$ donc count=count+1=2

Troisième itération
Pour i=3
count=0 ; N_3=3 ; j=1,2,3
Pour j=1 ; $mark(S_3(1) = 1)$;
Pour j=2 ; count=0 ; $mark(S_3(2) = 0)$ donc count=count+1
Pour j=3 ; count=1 ; $mark(S_3(3) = 0)$ donc count=count+1

Quatrième itération
Pour $i = 4$
count=0 ; N_4=3 ; j=1,2,3
Pour j=1 ; $mark(S_4(1) = 0)$ donc count=count+1
Pour j=2 ; count=1 ; $mark(S_4(2) = 0)$ donc count=count+1
Pour j=3 ; count=2 ; $mark(6) = 1$ donc 6 ∈ F

Cinquième itération
Pour $i = 5$
count=0 ; N_5=4 ; j=1,2,3,4
Pour j=1 ; $mark(S_5(1) = 0)$ donc count=count+1
Pour j=2 ; $mark(S_5(2) = 1)$
Pour j=3 ; count=1 ; $mark(S_5(3) = 0)$ donc count=count+1
Pour j=4 ; count=2 ; $mark(8) = 1$ donc $8 \in F$

Sixième itération
Pour $i = 6$
count=0 ; N_6=4 ; j=1,2,3,4
Pour j=1 ; $mark(S_6(1) = 1)$
Pour j=2 ; $mark(S_6(2) = 0)$ donc count=count+1
Pour j=3 ; $mark(S_6(3) = 1)$
Pour j=4 ; count=1 ; $mark(S_6(4) = 0)$ donc count=count+1

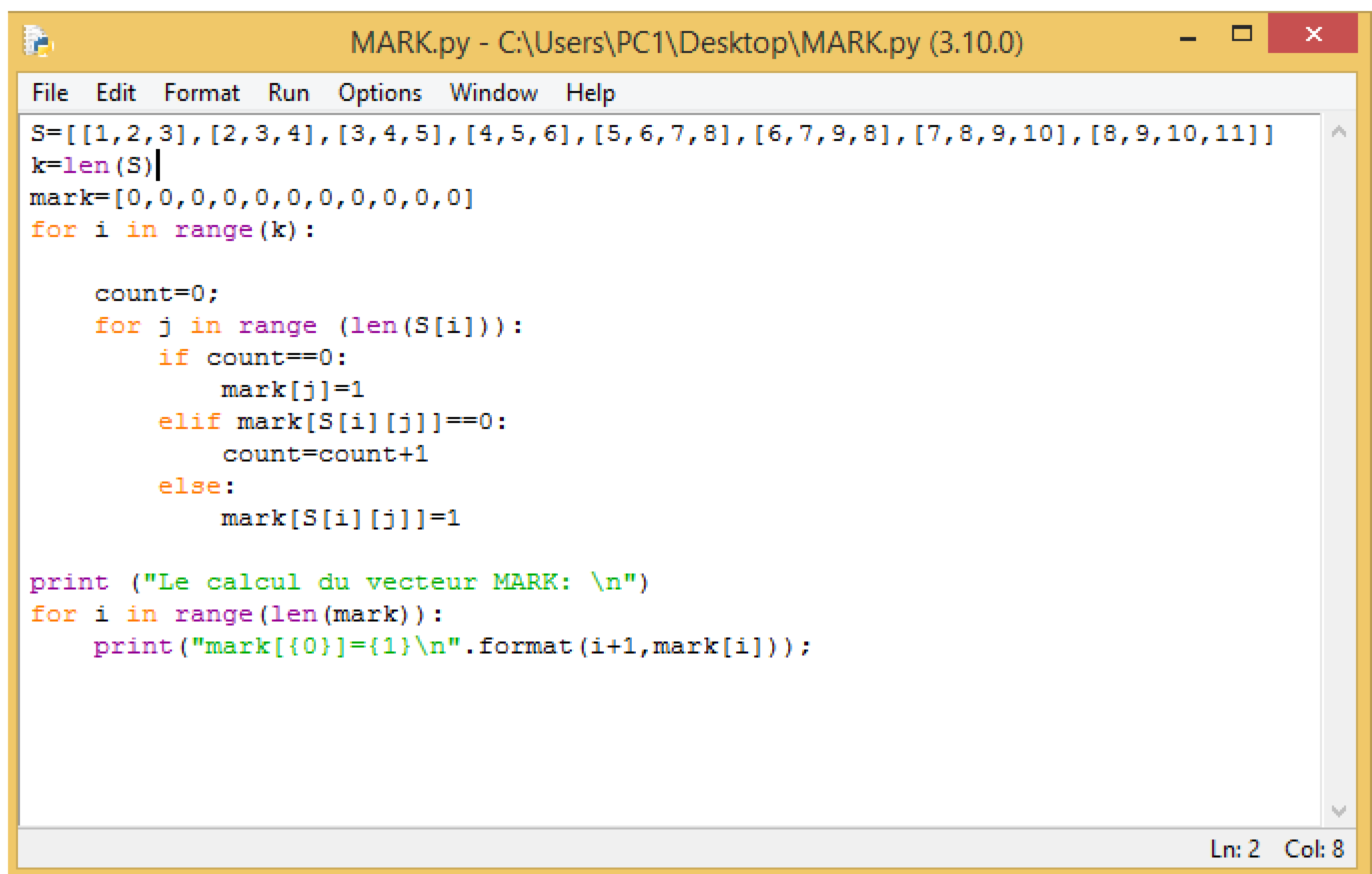
Septième itération
Pour $i = 7$
count=0 ; N_7=4 ; j=1,2,3,4
Pour j=1 ; $mark(S_7(1) = 0)$ donc count=count+1
Pour j=2 ; $mark(S_7(2) = 1)$
Pour j=3 ; count=1 ; $mark(S_7(3) = 0)$ donc count=count+1 ;
Pour j=4 ; count=2 ; $mark(10) = 1$ donc $10 \in F$

huitième itération
Pour $i = 8$
count=0 ; N_8=4 ; j=1,2,3
Pour j=1 ; $mark(S_8(1) = 1)$
Pour j=2 ; count=0 ; $mark(S_8(2) = 0)$ donc count=count+1 ;
Pour j=3 ; $mark(S_8(3) = 1)$
Pour j=4 ; count=1 ; $mark(S_8(4) = 0)$ donc count=count+1.

- L'algorithme **MARK** nous donne, $mark(3) = 1$; $mark(6) = 1$; $mark(8) = 1$; $mark(10) = 1$;

3.6.1 Programmation de l'algorithme MARK sur Python

Le code Python de l'algorithme MARK pour l'exemple précédent(l'éclairage d'un supermarché).



L'exécution de ce code :

```
IDLE Shell 3.10.0
File  Edit  Shell  Debug  Options  Window  Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: C:\Users\PC1\Desktop\nouveau meme\MARK.py ==============
le calcul du vecteur MARK:

mark(1)=0

mark(2)=0

mark(3)=1

mark(4)=0

mark(5)=0

mark(6)=1

mark(7)=0

mark(8)=1

mark(9)=0

mark(10)=1

mark(11)=0

>>>
                                                              Ln: 20   Col: 0
```

Conclusion générale

Nous avons étudié dans ce travail le problème du Feedback minimum dans une classe particulière de graphes qui sont les graphes d'intervalles.

Le **MFVS** est d'une importance remarquable dans l'étude de nombreux systèmes à grande échelle. La technique fréquemment utilisée consiste à modéliser la structure d'un système donné d'un graphe et le rendre libre de cycles, en supprimant le plus petit ensemble possible de sommets. Ainsi le feedback minimum est obtenu.

La sécurisation d'un feedback minimum est une étape primordiale dans notre travail, et dans la vie réelle pour éviter toute panne possible dans la modélisation et le fonctionnement d'un système. La sécurisation d'un **MFVS** est un problème NP-Complet.

En possédant un algorithme calculant le **MFVS** d'un graphe d'intervalles non orienté, non valué, nous espérons avoir donné un exemple concret et intéressant pour mettre en évidence la recherche et la sécurisation d'un **MFVS** dans un système d'éclairage. Nous avons programmé l'algorithme nommé "MARK" proposé par Saha et Pal avec le langage de programmation Python.

Résumé

Nous traitons dans ce document le problème du feedback minimum dans les graphes d'intervalles non valués non orientés.

Le feedback est un ensemble de sommets qu'à leurs suppression le graphe devient acyclique, cet ensemble est dit minimum si sa cardinalité est minimum parmi tous les autres ensembles, cet ensemble est noté F_{min}.

Dans le but de calculer F_{min}, nous avons procédé la méthode (algorithme) proposé par Pal et Saha, ensuite on a travaillé sur la sécurisation de cet ensemble. La sécurisation consiste à trouver pour chaque sommet un autre sommet qui assure la même tâche que lui.

Si F_{min} n'est pas sécurisé, on a opté pour une autre méthode qui est l'ajout des sommets et dans ce cas l'ensemble ne sera plus minimum. La sécurisation prend l'ampleur sur la notion du minimum.

On adonné plusieurs exemples qui facilitent la compréhension de ce problème, suivi d'un exemple réel qui est l'assurance d'un éclairage permanent et uniforme dans un supermarché.

A la fin de ce document on a programmé l'algorithme $MARK$ qui est un vecteur booléen qui calcule le F_{min} à l'aide du langage python qui est un langage très performant et contemporain.

Bibliographie

[1] A. Casamay-Bouceau, A. Chauvin, G. Connan, Programmation en Python pour les mathématiques, DUNOD, 2016.

[2] A. Saha, M. Pal, An algorithm to find a minimum feedback vertex set of an interval graph, Vidyasagar University, Midnapore, India, 2005.

[3] B. Sadi, Graphes, ordres et programmation linéaire, cours et exercices, ellipses, mars, 2020.

[4] B. Sadi, L.Nourine, Optimisation combintoires, Quelques problèmes, pages bleues, 2011.

[5] B. Sadi, Théorie des graphes, Complexité algorithmique, l'OPU, 1993.

[6] C. Berge, Graphes et hypergraphes, Dunod, 1973.

[7] J. C. Fournier, Graphes et applications 1, Lavoisier, 2007.

[8] M. C. Golumbic, Algorithmic graph theory and perfect graphs, Elsevier, 2004.

[9] M. R. Garey, D. S. Johnson, Computers and intractability, A Guide to the theory of NP-Completeness,Victor Klee, Mars, 1991.

[10] P. Fuchs, P. Poulin, Cours python, Introduction à la programmation Python pour la biologie, Univrsité de Paris, France, 2021.

[11] P. Lacomme, C. Prins, M. Sevaux, Algorithmes de graphes, Eyrolles, 2003.