

TP 7 - Synthèse Graphes

I Calcul d'hyperstatisme (Parcours de graphe)

Objectif

L'objectif de ce TP est de déterminer à l'aide d'un fichier texte `.txt` le degré d'hyperstatisme d'un mécanisme. Il va être nécessaire notamment de déterminer le nombre cycle présent dans le graphe de liaisons.

Vous disposez de deux fichiers « `graphe_cinematique_test.txt` » et « `graphe_cinematique_exo.txt` ». Le premier est celui qu'on utilisera durant la quasi totalité du TP pour mettre en place l'algorithme et le tester. Le dernier est celui représentant le mécanisme dont on cherche à déterminer le degré d'hyperstatisme.

I.1 Problème étudié¹

On s'intéresse au système de direction du système « Effibot » qui est un robot suiveur d'un utilisateur. Ce type de robot peut ainsi porter des charges tout en suivant la personne marchant devant lui. Ce type de robot est développé dans la logistique, le BTP ou même la Poste. (française et allemande).



FIGURE 1 – Robot Effibot

Le système de direction de l'Effibot est un système avec 4 roues directrices. Voici le schéma cinématique du système avec son graphe de liaisons.

1. Issu du concours ICNA 2018

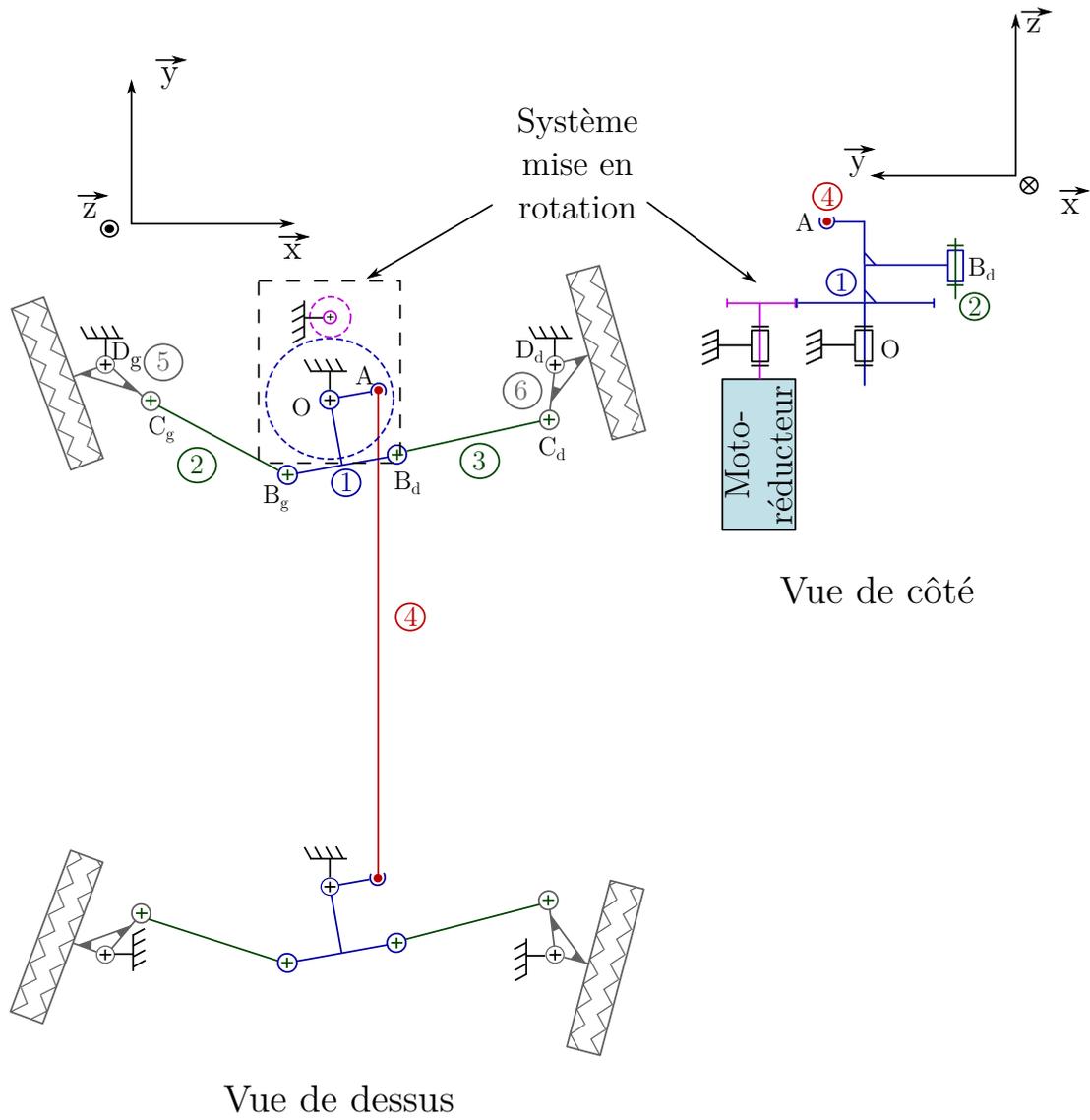


FIGURE 2 – Schéma cinématique du système de direction du robot Effibot.

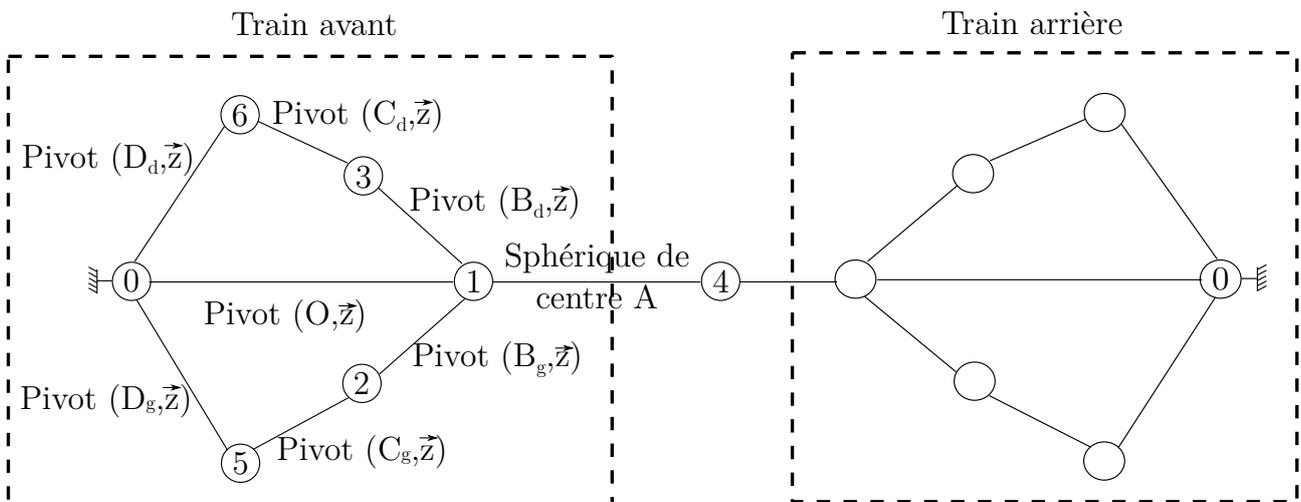


FIGURE 3 – Graphe de liaisons du système de direction du robot Effibot.

I.2 Format du fichier décrivant le graphe de liaisons d'un mécanisme

Les fichiers `og` `graphe_cinematique_test.txt` et `graphe_cinematique_exo.txt` contiennent les liaisons mécaniques entre les différentes classes d'équivalence du mécanisme. Le format de fichier est toujours le même.

- Une ligne d'introduction
- La forme d'une ligne est la suivante `piece1-piece2 : Liaison`
- La dernière ligne se termine en indiquant la mobilité du mécanisme `mecanisme =`

I.3 Récupération des données du fichier .txt.

Q1. Créer une fonction `recup_pieces(ligne)` permettant de récupérer dans une liste de taille 2, les pièces du mécanisme qui sont en liaison. Par exemple, pour la ligne `ligne="10-2 : Liaison pivot glissant\n"`, la fonction `recup_pieces(ligne)` doit renvoyer `[10,2]`.

Réponse

Le principe est de chercher des séparateurs entre les chaînes de caractères renseignant sur le numéro des pièces. Le premier séparateur est le `-` qui sépare les deux pièces en liaison. Le second est `:` qui sépare la deuxième pièce du nom de la liaison (on pourrait également choisir l'espace comme caractère séparateur). Une fois que la position de ces séparateurs est connue, il n'est plus très compliqué de récupérer les pièces qui sont en liaison.

```

1 def recup_piece(ligne):
2     position_courante=0 # Position du caractère étudié
3     carac=ligne[position_courante] # Caractère courant
4     while carac != "-": # Tant que l'on a pas atteint le "-"
5         position_courante+=1 # Position du caractère étudié
6         carac=ligne[position_courante] # Caractère courant
7     piece_une=int(ligne[0:position_courante]) # Récupération de la première pièce
8     position_courante+=1 # Evolution de la position courante
9     depart_piece_deux=position_courante # Position de départ pour récupérer la 2eme pièce
10    while carac != ":": # Tant que l'on a pas atteint :
11        position_courante+=1
12        carac=ligne[position_courante]
13    piece_deux=int(ligne[depart_piece_deux:position_courante-1]) # Pour récupérer la 2eme
    pièce, bien penser à écrire position_courante-1 car avant les :, il y a un espace.
14    return [piece_une, piece_deux]
```

On peut également envisager une fonction en utilisant des méthodes disponibles pour les chaînes de caractères.

```

1 def recup_piece2(ligne):
2     position1=ligne.index("-")
3     position2=ligne.index(":")
4     piece_une=int(ligne[0:position1]) # Récupération de la première pièce
5     piece_deux=int(ligne[position1+1:position2-1]) # Pour récupérer la 2eme pièce, bien pen-
    ser à écrire position_courante-1 car avant les :, il y a un espace.
6     return [piece_une, piece_deux]
```

Q2. Créer une fonction `recup_liaison(ligne)` permettant de récupérer sous forme de chaîne de caractère, la liaison entre les deux pièces du mécanisme. Par exemple, pour la ligne `ligne="10-2 : Liaison pivot glissant\n"`, la fonction `recup_liaison(ligne)` doit renvoyer `pivot glissant`.

Réponse

On utilise ici un algorithme consistant à trouver la position du mot "Liaison ". En effet, connaissant

la position de ce mot, on peut en déduire que le nom de la liaison correspond au reste de la ligne ôtée de `\n`. On peut reprendre une structure semblable au code de la question précédente, c'est-à-dire rechercher la position du caractère « : » et ainsi en déduire la position du mot "Liaison " et donc par la suite du nom de la liaison. La correction proposée ici est différente. En effet, on commence par la fin de la ligne pour chercher le mot "Liaison " (c'est juste une autre façon de faire).

```
1 def recup_liaison(ligne):
2     limite=len("Liaison ")
3     position_courante=len(ligne)-limite
4     while ligne[position_courante:position_courante+limite]!="Liaison ":
5         position_courante-=1
6     return ligne[position_courante+limite:-1]
```

Q3. Créer une fonction `recup_mobilite(ligne)` permettant de récupérer sous forme d'entier le nombre de mobilité du mécanisme. La ligne est du type `ligne="mobilité = 2"`.

Réponse

Ici, il est plus facile de récupérer cette information, en effet, on connaît la longueur de la chaîne de caractère "mobilité = " = 11. Ainsi l'information peut-être récupérer à l'aide de la commande `ligne[11:]`

```
1 def recup_mobilite(ligne):
2     longueur=len("mobilite = ")
3     return int(ligne[longueur:])
```

I.4 Mise en forme des données

Q4. Initialiser un dictionnaire `liaisons`, avec pour clés : pivot, pivot glissant, glissière, hélicoïdale, cylindre-plan, sphère-plan, sphère-cylindre, appui plan, sphérique, sphérique à doigt, avec pour valeur 0 (qui représente le nombre de liaisons détectées).

Réponse

```
1 liaisons={"pivot":0,"glissiere":0,"helicoïdale":0,"pivot glissant":0,"spherique":0,
2 "appui plan":0,"sphere-cylindre":0,"sphere-plan":0,
3 "spherique a doigt":0,"cylindre-plan":0}
```

Q5. Initialiser un dictionnaire `mobilités`, avec pour clés : pivot, pivot glissant, glissière, hélicoïdale, cylindre-plan, sphère-plan, sphère-cylindre, appui plan, sphérique, sphérique à doigt, avec pour valeur le degré de mobilité de chacune de ces liaisons.

Réponse

```
1 mobilites={"pivot":1,"glissiere":1,"helicoïdale":1,"pivot glissant":2,"spherique":3,
2 "appui plan":3,"sphere-cylindre":4,"sphere-plan":5,
3 "spherique a doigt":2,"cylindre-plan":4}
```

Q6. Créer une fonction `lecture_fichier(fichier, liaisons)` avec comme argument d'entrée une chaîne de caractères `fichier` correspondant au nom du fichier à traiter et le dictionnaire `liaisons`. Cette fonction renverra la liste des listes (`listes_relations_pieces`) des pièces en relation, la mise à jour du dictionnaire `liaisons` avec le nombre de liaison pivot, pivot glissant, etc. et le nombre de mobilité du mécanisme. Tester votre fonction sur le fichier « `graphe_cinematique_test.txt` ». Vous devez obtenir pour `listes_relations_pieces`, la liste `[[0,1],[1,2],[0,2]]` et la mise à jour des liaisons hélicoïdale, glissière et pivot. **Réponse**

```

1 def lecture_fichier(fichier, liaisons):
2     fichier=open(fichier, "r", encoding='utf8') # Ouverture du fichier en mode lecture dans un
    format adapté
3     ligne=fichier.readline() # Lecture de la première ligne inutile dans le traitement de nos
    données
4     listes_relations_pieces=[] # Initialisation de la liste de stockage des différentes pièces
    en relation
5     for ligne in fichier: # Pour toutes les lignes ...
6         if ligne[0:len("mobilite")]!="mobilite": # ... jusqu'à atteindre la dernière ligne
7             listes_relations_pieces.append(recup_piece(ligne)) # On récupère les diffé-
    rentes pièces en liaison grâce à la fonction précédemment mise en place
8             liaisons[recup_liaison(ligne)]+=1 # mise à jour du dictionnaire liaisons.
9     return listes_relations_pieces
10
11 listes_relations_pieces=lecture_fichier("graphe_cinematique_test.txt", liaisons)

```

Q7. À la suite de la commande `lecture_fichier(fichier, liaisons)` à partir de la liste `listes_relations_pieces` créer une fonction `nombre_pieces(listes_relations_pieces)` retournant `nombre_pieces` le nombre de pièces constituant le mécanisme.

Réponse

Ce code n'est pas nécessairement difficile à mettre en place. Il faut initialiser une liste des pièces constituant le système. Au départ, cette liste est vide et on la remplit au fur et à mesure que l'on parcourt la liste `listes_relations_pieces`. Quand une nouvelle pièce est découverte, on l'ajoute alors à cette liste et on retourne la taille de celle-ci.

```

1 def nombre_pieces(listes_relations_pieces):
2     pieces_trouvees=[]
3     for i in range(len(listes_relations_pieces)):
4         if listes_relations_pieces[i][0] not in pieces_trouvees:
5             pieces_trouvees.append(listes_relations_pieces[i][0])
6         if listes_relations_pieces[i][1] not in pieces_trouvees:
7             pieces_trouvees.append(listes_relations_pieces[i][1])
8     return len(pieces_trouvees)
9
10 nombre_pieces=nombre_pieces(listes_relations_pieces)

```

I.5 Étude du graphe de liaisons

Q8. Initialiser alors 0 la matrice d'adjacence du graphe de liaisons du mécanisme de taille `nombre_pieces` × `nombre_pieces`

Réponse

```

1 mat_adjacence=[[0]*nombre_pieces for j in range(nombre_pieces)]

```

Q9. Créer alors la matrice d'adjacence du graphe de liaisons de ce système.

Réponse

```

1 for i in range(len(listes_relations_pieces)):
2     mat_adjacence[listes_relations_pieces[i][0]][listes_relations_pieces[i][1]]=1
3     mat_adjacence[listes_relations_pieces[i][1]][listes_relations_pieces[i][0]]=1

```

Q10. À l'aide d'un parcours de graphe, déterminer le nombre cyclomatique (nombre de cycles élémentaires) du graphe de liaisons.

Réponse

Dans le code suivant, il faut adapter un algorithme de parcours de graphe vu dans le TP précédent. On choisit la méthode de parcours en largeur du graphe pour compter ce nombre de cycle.

```

1 def parcours_largeur_graphe(mat_adj,depart):
2     file=deque([depart]) # Initialisation de la file avec le premier noeud à visiter
3     noeuds_visites=[depart] # Initialisation de la variable qui contiendra les numeros de noeuds
    visités, on commence par le noeud départ
4     cycle=0
5     pere_liste=[-1]*len(mat_adj)
6     while len(file) != 0:
7         noeud_courant=file.popleft() # Recupération du numéro du noeud courant
8         voisins=recherche_voisins(mat_adj,noeud_courant) # Recherche des voisins
9         pere=noeud_courant
10        for v in voisins: # Pour tous les noeuds voisin du noeud courant non visités...
11            if v not in noeuds_visites:
12                noeuds_visites.append(v) # ... on l'ajoute aux noeuds visités ...
13                file.append(v) # ... et à la file d'étude.
14                pere_liste[v]=pere
15            else:
16                if pere_liste[pere]!=v:
17                    cycle+=1
18        return noeuds_visites,cycle//2 # PENSEZ A DIVISER PAR 2, CAR EN EFFET LORS D'UN PARCOURS,
    ON DECOUVRE 2 FOIS UNE MEME ARETE

```

I.6 Résolution du problème

Sachant que l'hyperstatisme se calcule par la formule :

$$h = 6 \times \mu - I_c + m$$

avec :

- μ le nombre cyclomatique ;
- I_c nombre d'inconnues cinématiques ;
- m nombre de mobilité du mécanisme.

Q11. À l'aide d'un programme, déterminer alors le degré d'hyperstatisme du mécanisme. Vérifier que le degré d'hyperstatisme du système du fichier « graphe_cinematique_text.txt » est bien de 4.

Réponse

```

1 noeuds_visites,nombre_cyclo=parcours_largeur_graphe(mat_adjacence,0)
2 inconnues_cinematiques=0
3 for i in mobilites:
4     inconnues_cinematiques+=mobilites[i]*liaisons[i]
5 hyperstatisme=6*nombre_cyclo-inconnues_cinematiques+mobilite

```

Q12. Quel est alors le degré du système de direction du robot Effibot.

Réponse

On trouve un degré d'hyperstatisme de 12.

II Sherlock Holmes et les graphes²

Un jour, Sherlock Holmes reçoit la visite de son ami Watson que l'on avait chargé d'enquêter sur un assassinat mystérieux datant de plus de dix ans. A l'époque, le Duc de Densmore avait été tué par l'explosion d'une bombe artisanale, qui avait également détruit le château de Densmore où il s'était retiré. Les journaux d'alors relataient que le testament, détruit lui aussi par l'explosion, avait tout pour déplaire à l'une de ses huit ex-femmes. Or, avant sa mort, le Duc les avait toutes invitées à passer quelques jours dans sa retraite écossaise.

Holmes : Je me souviens de l'affaire ; ce qui est étrange, c'est que la bombe avait été fabriquée spécialement pour être cachée dans l'armure de la chambre à coucher, ce qui suppose que l'assassin a nécessairement effectué plusieurs visites au château ! L'une pour prendre les mesures de l'armure, une autre pour y placer la bombe faite sur-mesure.

Watson : Certes, et pour cette raison, j'ai interrogé chacune des ex-épouses : Anne, Betty, Cynthia, Diane, Émilie, Félicie, Georgia et Hélène. Elles ont toutes juré qu'elles n'avaient été au château de Densmore qu'une seule fois dans leur vie.

Holmes : Hum ! leur avez-vous demandé à quelle période ont eu lieu leurs séjours respectifs ?

Watson : Hélas, aucune ne se rappelait les dates exactes, après dix ans ! Néanmoins, je leur ai demandé qui elles y avaient rencontré :

- Anne a déclaré y avoir rencontré Betty, Cynthia, Félicie et Georgia ;
- Betty a déclaré y avoir rencontré Anne, Cynthia et Hélène ;
- Cynthia a déclaré y avoir rencontré Anne, Betty, Diane, Émilie et Hélène ;
- Diane a déclaré y avoir rencontré Cynthia et Émilie ;
- Émilie a déclaré y avoir rencontré Cynthia, Diane et Félicie ;
- Félicie a déclaré y avoir rencontré Anne et Émilie ;
- Georgia a déclaré y avoir rencontré Anne et Hélène ;
- Hélène a déclaré y avoir rencontré Betty, Cynthia et Georgia.

Vous voyez mon cher Holmes, ces réponses sont concordantes ! C'est alors que Holmes prit un crayon et dessina un étrange petit dessin (un graphe). Puis, après moins de 30 secondes, Holmes s'écria : "Tiens, tiens ! Ce que vous venez de dire détermine de façon unique qui est l'assassin !"

Q1. Dessiner le graphe fait par Holmes.

Réponse

A faire

Q2. Comment de manière simpliste peut-on résoudre le problème ? Est-ce facile ?

Réponse

À partir de ce graphe, on peut dessiner des intervalles sur une ligne de temps. Ces intervalles de temps représente les moments où les personnes peuvent s'être rencontrées. Le problème est que l'on connaît pas l'ordre d'arrivée des personnes ni leur moment de départ, il faut donc un peu tâtonner pour trouver une solution adéquate. Ici, le problème n'est composée que de 8 personnes, on peut donc encore s'en sortir mais cela peut prendre un certain temps (certainement plus que 30 s en tout cas ;-)).

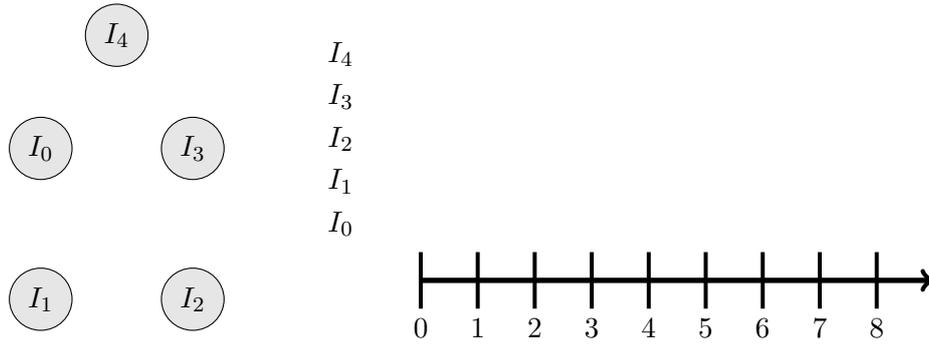
Sherlock Holmes pour résoudre ce problème utilise, une propriété des graphes dit « d'intervalles ». Nous allons dans un premier temps déterminer et expliquer ce qu'est un graphe d'intervalles.

On considère $\mathcal{F} = \{I_0, I_1, I_2, \dots, I_n\}$ une famille d'intervalles appartenant à l'ensemble des réels. Le graphe associé à \mathcal{F} est un graphe non orienté dont les sommets sont bijection avec les intervalles et tel que deux sommets i et j sont reliés par une arête si et seulement si $I_i \cap I_j \neq \emptyset$.

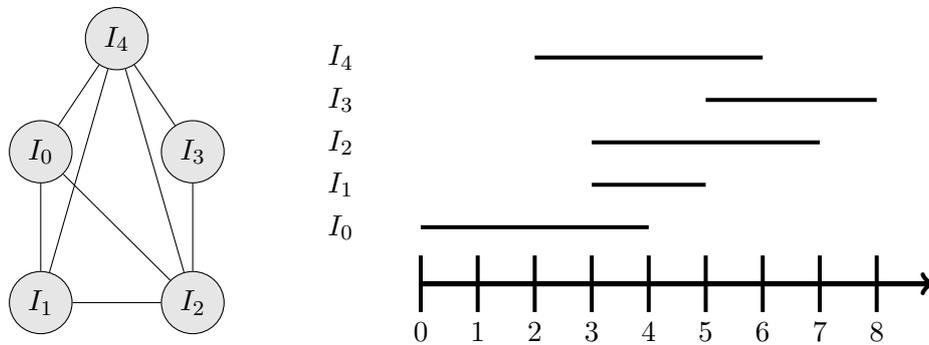
2. Enigme proposée par Claude Berge « Regards sur la théorie des graphes », merci à Olivier Guindet pour la retranscription de l'énoncé. Exemples inspirés de <https://perso.esiee.fr/~coupriem/IT3004/sherlock.pdf>

Q3. Tracer les intervalles et le graphe associé à \mathcal{F} pour :

$$\mathcal{F} = \{I_0 = [0, 4], I_1 = [3, 5], I_2 = [3, 7], I_3 = [5, 8], I_4 = [2, 6]\}$$

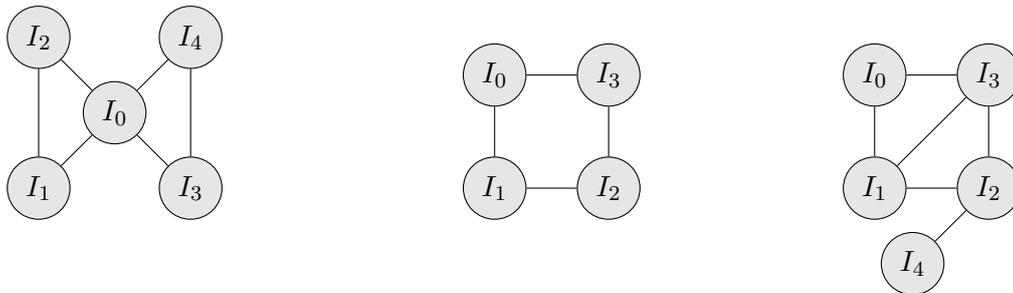


Réponse



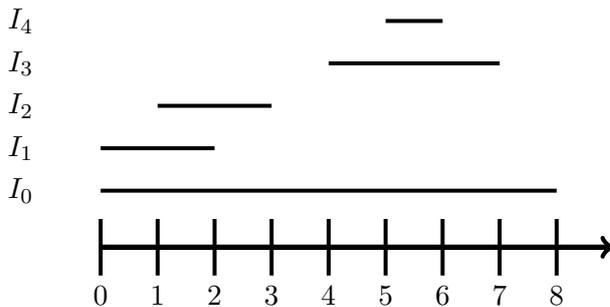
On dit qu'un graphe $G = (S, A)$ est un graphe d'intervalles s'il existe une famille \mathcal{F} d'intervalles sur une droite telle que G soit le graphe représentatif de \mathcal{F} .

Q4. Les graphes suivants sont-ils des graphes d'intervalles :

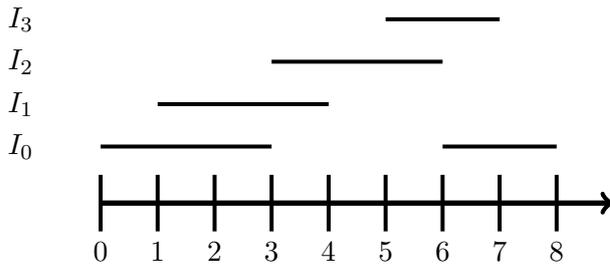


Réponse

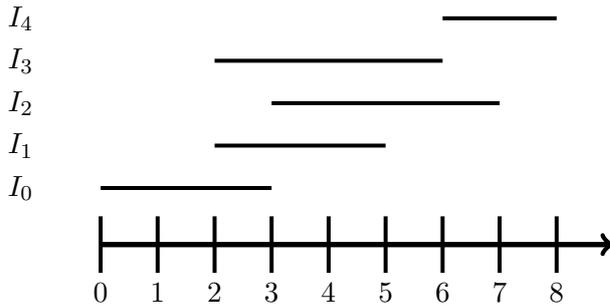
Pour le premier graphe, une famille d'intervalle possible est :



Pour le deuxième graphe, il n'est pas possible de déterminer des intervalles représentatif de ce graphe à moins qu'un des intervalles I_0, I_1, I_2 ou I_3 apparaissent deux fois.



Le dernier graphe est bien un graphe d'intervalle, voici des intervalles possibles représentatifs de ce graphe.



Q5. D'après, les exemples traités, proposer une condition nécessaire pour qu'un graphe soit d'intervalles.

Réponse

Théorème 1 *Un graphe est d'intervalles est un graphe cordal. C'est-à-dire qu'il n'est pas possible de trouver un ensemble d'intervalles qui réalise un cycle de longueur supérieur à 3 sans corde (ou raccourci).* C'est

ainsi que le deuxième graphe proposé ci-dessous ne peut pas être un graphe d'intervalle car le cycle est de longueur 4 sans corde, c'est à-dire sans lien entre I_0 et I_2 ou entre I_1 et I_3 . ATTENTION, la réciproque est fautive. Ce n'est pas parce qu'un graphe est cordal qu'il est d'intervalles.

Q6. Selon le graphe de Sherlock Holmes, puisqu'il n'y a qu'une seule meurtrière qui est venue deux fois au manoir, justifier qu'en éliminant la meurtrière du graphe, celui-ci est d'intervalles.

Réponse

Actuellement, le graphe d'Holmes ne peut pas être d'intervalles puisque l'on sait qu'une personne est venue deux fois au château. En éliminant cette personne du graphe, on obtiendra alors un graphe avec des personnes venues qu'une seule fois et qui n'ont pu alors se rencontrer qu'une seule fois. Le graphe obtenu est donc représentatif d'une famille d'intervalles (intervalles de rencontre entre les personnes en l'occurrence), c'est donc un graphe d'intervalles.

Définition 1 *Un sommet est « simplicial » si ses voisins (c'est-à-dire les sommets auxquels il est relié par une arête) sont tous reliés entre eux par une arête. Un « schéma d'élimination parfait » dans un graphe à n sommets est un ordre v_1, \dots, v_n des sommets tel que v_i est simplicial dans le graphe qui ne contient que les sommets v_i, \dots, v_n .*

Théorème 2 *Un graphe est d'intervalles si, et seulement si, il possède un schéma d'élimination parfait.*

Q7. Créer un algorithme permettant de vérifier si un graphe possède un schéma d'élimination parfait. Le graphe sera renseigné à l'aide d'une matrice d'adjacence. Celui-ci fonctionnera sur le principe suivant :

- Choisir un sommet simplicial (c'est-à-dire dont tous les voisins sont reliés entre eux.
- Supprimer ce sommet du graphe ;
- Répéter les actions précédentes jusqu'à ce que le graphe soit vide ou que l'on ne puisse plus

choisir de sommet simplicial.

Réponse

Voir fichier python

Q8. Déterminer alors la meurtrière du Duc de Densmore.

Réponse

Voir fichier python : Anna est la meurtrière

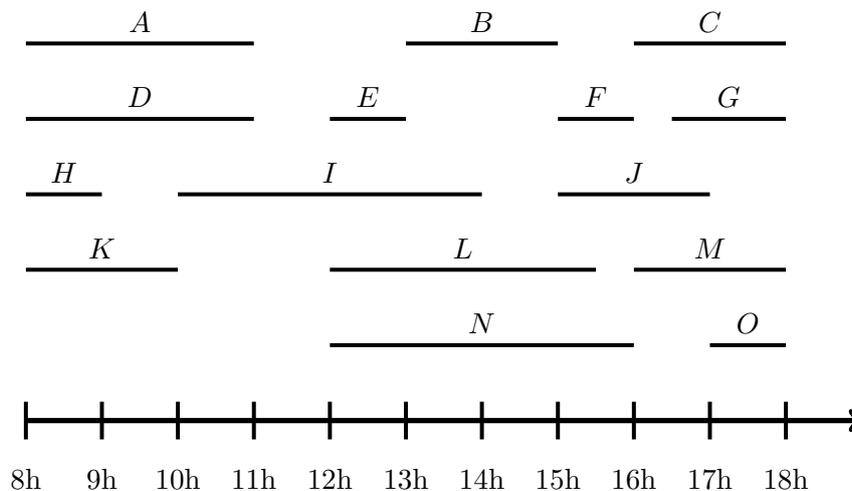
III Coloration de graphes et répartition de tâches ³

Un chef d'entreprise a un chantier qui risque de prendre du retard. Il reste 15 tâches à effectuer sur celui-ci. Ayant une certaine expérience et connaissant les connaissances de ses employés, il est capable d'estimer le temps nécessaire par tâche. Toutes ces tâches ne sont pas indépendantes entre elles. Afin de ne pas pénaliser d'autres chantiers, il veut limiter le nombre d'intervenant.

Objectif

Proposer une solution optimale à l'aide d'un graphe d'intervalle.

III.1 Présentation du problème



Q1. Tracer le graphe d'intervalle associé correspondant aux tâches à assigner.

Réponse

A faire

La matrice d'adjacence de ce graphe est donnée dans le fichier « exo_coloration_eleve.py ».

III.2 Coloration du graphe

Afin de résoudre le problème du nombre minimal d'employés à astreindre à ces différentes tâches, on va utiliser le principe de coloration. Chaque employé est associé à une couleur et ainsi chaque sommet du graphe sera coloré par la couleur de l'employé devant réaliser cette tâche. Afin d'assurer la possible réalisation des différentes tâches, il ne faut pas que deux sommets voisins soient colorés par la même couleur car cela signifierait qu'un même employé serait assigné à deux tâches en même temps, ce qui n'est pas possible.

L'algorithme de coloration des sommets d'un graphe d'intervalles avec un nombre minimum de couleurs est le suivant :

- **Initialisation :** On dispose d'un ensemble de couleurs (15 au maximum dans ce problème) ;
- Déterminer un schéma d'élimination parfait (voir TD précédent) du graphe d'intervalle et mettant dans une liste les sommets éliminés au fur et à mesure. Le sommet éliminé au tour i est noté v_i .

3. inspiré de <https://www.gerad.ca/~alainh/Chapitre3.pdf>

- Colorer les sommets les uns après les autres, selon l'ordre inverse v_n, \dots, v_1 en choisissant toujours la première couleur disponible.

Q2. En vous inspirant de ce qui a été codé dans le TD précédent, réaliser le schéma d'élimination parfait du graphe d'intervalle et mémoriser l'ordre dans lequel les sommets ont été éliminés.

Réponse

Voir fichier python

Q3. Réaliser l'algorithme de coloration du graphe, vous pouvez pour cela créer des listes de stockage intermédiaires. Pour rappel, cette coloration s'appuie sur le fait que des sommets voisins ne doivent pas avoir la même couleur. **Réponse**

Voir fichier python

Q4. Testez votre algorithme et vérifiez qu'il ne faut que 4 employés pour réaliser l'ensemble des 15 tâches. **Réponse**

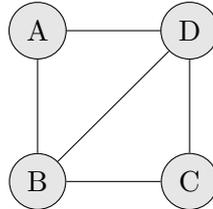
Voir fichier python

IV Graphe de comparabilité (graphe orienté)⁴

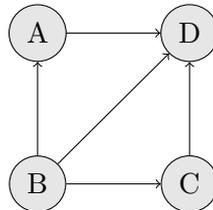
Un graphe est dit de comparabilité si on peut orienter ses arêtes de façon transitive, c'est-à-dire de telle sorte que s'il existe un arc du sommet i vers le sommet j puis du sommet j vers le sommet k , alors il existe également un arc de i vers k .

Exemple

Le graphe suivant est dit de comparabilité



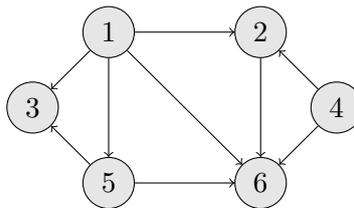
En effet, il est possible de transformer ce graphe en graphe orienté, de façon à obtenir des orientations transitives des arcs.



Objectif

Dans ce problème, l'objectif est de vérifier si un graphe orienté l'est de façon transitive.

Prenons en exemple le graphe suivant :



On peut représenter ce graphe à l'aide de la matrice d'adjacence suivante :

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ -1 & 0 & 0 & -1 & 0 & 1 \\ -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 1 \\ -1 & -1 & 0 & -1 & -1 & 0 \end{pmatrix}$$

Dans cette matrice d'adjacence, 1 représente le lien entre un sommet et son successeur. -1 représente le lien entre un sommet et son prédécesseur.

On va également représenter un graphe orienté à l'aide de listes d'adjacence.

4. « Introduction à la théorie des graphes » Didier Müller

$$L = [[2, 3, 5, 6], [-1, -4, 6], [-1, -5], [2, 6], [-1, 3, 6], [-1, -2, -4, -5]]$$

De nouveau, un signe positif, précise que le sommet est un successeur du sommet courant et un signe négatif, un prédécesseur.

Q1. Le graphe orienté donné dans l'exemple est-il orienté de façon transitive ?

Réponse

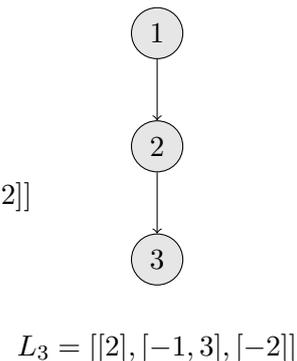
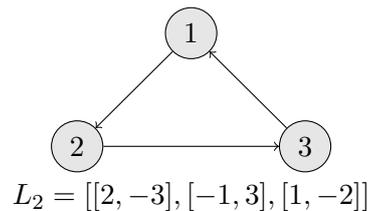
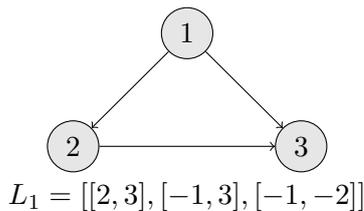
Oui, le graphe est ici orienté de façon transitive. Quand il existe un arc du sommet i vers le sommet j puis vers le sommet k , alors il existe également un arc de i vers k .

Dans un premier temps, on va créer un fonction permettant de vérifier qu'un graphe (ou sous graphe) composé de 3 sommets reliés par des arcs l'est de façon transitive ou non (voir question précédente).

Q2. Est-ce que les graphes ou sous-graphes de 3 sommets représentés par les différentes listes d'adjacence suivantes vérifient t-ils une relation transitive :

- $L_1 = [[2, 3], [-1, 3], [-1, -2]]$
- $L_2 = [[2, -3], [-1, 3], [1, -2]]$
- $L_3 = [[2], [-1, 3], [-2]]$

Réponse



Le graphe 1 vérifie bien une relation de transitivité contrairement aux deux autres.

Q3. D'après votre réponse à la question précédente, expliquez comment à l'aide des listes d'adjacence précédente, il est possible de déterminer si un graphe orienté vérifie une relation de transitivité.

Réponse

Pour vérifier si un graphe de 3 sommets est de transitivité, il faut tout d'abord que la liste des listes proposées pour étudier un graphe de 3 sommets soit **une liste de longueur 3**. Les sous-listes d'adjacence de chacun des sommets sont **des listes de taille 2**. Lorsque ces deux conditions sont réunies, le graphe obtenu forme alors un cycle ou une relation transitive. Le sous-graphe forme un cycle si chacun des sommets possèdent un sommet successeur et un prédécesseur. Le sous-graphe vérifie une relation de transitivité lorsqu'**un sommet possède 2 successeurs, un autre possède 2 prédécesseurs et le dernier possède un successeur et un prédécesseur**.

L'algorithme consistant à vérifier si le graphe orienté est bien de comparabilité consiste à :

- Choisir un sommet i que l'on a pas encore testé
- Choisir un voisin j (successeur ou prédécesseur) de ce sommet i
- Choisir un voisin k de j différent de i .
- Déterminer la relation entre ces 3 sommets : 2 possibilités existent

- Les 3 sommets sont reliés entre eux ;
- Les 3 sommets ne sont pas reliés entre eux (c'est-à-dire que i et k ne sont pas voisins).
- Si le sous-graphe extrait doit vérifier une relation de transitivité et que ce n'est pas le cas ($i \rightarrow j \rightarrow k$ mais pas $i \rightarrow k$) alors l'algorithme s'arrête sinon on réitère les opérations précédentes. en choisissant un autre sommet k . S'il n'y a plus de sommets voisins de j disponibles, on choisit un autre voisin j de i . Si ce n'est pas possible, on choisit un nouveau sommet i jusqu'à avoir choisi l'ensemble des sommets disponibles.

On s'intéresse au cas où les 3 sommets ne sont pas reliés entre eux.

Q4. Donnez les 4 sous-graphes possibles lorsque 3 sommets ne sont pas reliés entre eux.

Réponse

On s'intéresse au type de sous-graphe suivant :



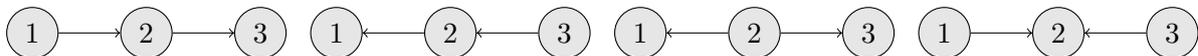
Q5. Parmi les sous-graphes proposés à la question précédente, lesquels permettent d'affirmer que le graphe étudié n'est pas transitif.

Réponse

Les deux premiers sous-graphes permettent d'affirmer que si de tels sous-graphes existent dans le graphe étudié alors celui-ci n'est pas de comparabilité. En effet, un graphe vérifie une relation de transitivité, s'il existe une arc entre 1 et 2 et un arc entre 2 et 3 alors il existe un arc entre 1 et 3, ce qui n'est pas le cas pour les deux premiers graphes. Pour les deux autres, il n'y a pas de relation de transitivité possible.

Q6. Donnez les listes d'adjacence des sous-graphes proposés précédemment.

Réponse



$[[2], [-1, 3], [-2]]$

$[[-2], [1, -3], [2]]$

$[[-2], [1, 3], [-2]]$

$[[2], [-1, -3], [2]]$

Q7. À l'aide de ces listes d'adjacence, proposez un moyen d'affirmer que le graphe étudié n'est pas de comparabilité si les sous-graphes ne vérifie pas la relation de transitivité.

Réponse

Un moyen efficace de repérer que le graphe n'est pas de comparabilité est que le sommet central (auxquels les deux autres sommets sont reliés) possède un successeur et un prédécesseur.

Q8. Programmez une fonction `test_transitivite_simple(graphe)` où `graphe` est une liste de listes d'adjacence des sommets composant un graphe (ou le sous-graphe) orienté de 3 sommets **qui ne sont pas tous reliés entre eux**, qui renvoie `False` si le graphe ne vérifie pas la relation de transitivité ($i \rightarrow j \rightarrow k$ existe mais $i \rightarrow k$ n'existe pas) et `True` sinon.

Réponse

```

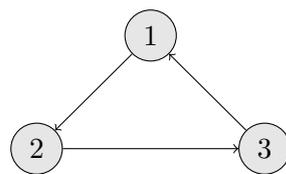
1 def test_transitivite_simple(graphe):
2     '''graphe est une liste contenant les listes d'adjacence dont les 3 sommets ne
3     sont pas tous reliés entre eux.
4     Dans les sous-listes, un nombre positif represente un sommet successeur
5     et un nombre negatif un sommet predecesseur du sommet courant.
6     Sortie : Boolean : le graphe oriente de 3 noeuds
7     n'est pas transitif -> False'''
8     n=len(graphe)
9     assert n==3 # Vérification de la taille limite du graphe
10    for i in range(n): # Pour tous les sommets du sous-graphe (ou graphe)
11        if len(graphe[i])==2: # Lorsque l'on trouve le sommet central
12            if graphe[i][0]*graphe[i][1]<0: # S'il y a un successeur et un prédécesseur alors
13                le graphe n'est pas transitif
14                    return False
15            else:
16                return True

```

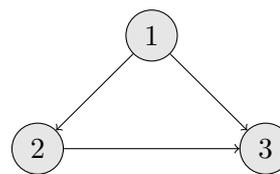
On étudie maintenant le cas où les sommets du sous-graphe à 3 sommets **sont tous reliés entre eux**.

Q9. Quels sont les deux cas possibles ?

Réponse



Circuit



Transitivité

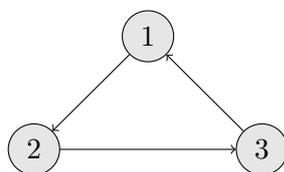
Q10. Parmi les sous-graphes proposés à la question précédente, lequel permet d'affirmer que le graphe étudié n'est pas transitif.

Réponse

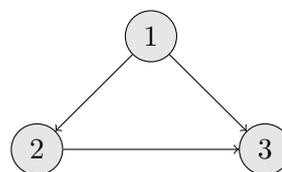
Le premier sous-graphe forme un circuit, il n'est donc pas transitif. Le second vérifie par contre cette relation de transitivité.

Q11. Donnez les listes d'adjacence des sous-graphes proposés précédemment.

Réponse



Circuit : $[[2, -3], [-1, 3], [1, -2]]$



Transitivité : $[[2, 3], [-1, 3], [-1, -2]]$

Q12. À l'aide de ces listes d'adjacence, proposez un moyen d'affirmer que le graphe étudié n'est pas de comparabilité si les sous-graphes ne vérifie pas la relation de transitivité.

Réponse

Dans le cas d'un circuit, on constate que tous les sommets possèdent un successeur et un prédécesseur.

Q13. Programmez alors la fonction `test_transitivite_boucle(graphe)` où `graphe` est une liste de listes d'adjacence des sommets composant un graphe (ou le sous-graphe) orienté de 3 sommets **tous**

reliés entre eux. Cette fonction doit renvoyer le booléen `True` si le graphe vérifie une relation de transitivité ou `False` s'il s'agit d'un cycle.

Réponse

```

1 def test_transitivite_boucle(graphe):
2     '''graphe est une liste contenant les listes d'adjacence dont les 3 sommets ne
3     sont pas tous relies entre eux.
4     Dans les sous-listes, un nombre positif represente un sommet successeur
5     et un nombre negatif un sommet predecesseur du sommet courant.
6     Sortie : Booleen si le graphe oriente de 3 noeuds
7     est transitif ou non'''
8     n=len(graphe)
9     assert n==3 # Vérification de la taille limite du graphe
10    compteur=0 # Compteur pour savoir combien de sommet courant possède un successeur et un pré-
    decesseur
11    for i in range(n): # Pour tous les sommets du sous-graphe (ou graphe)
12        if graphe[i][0]*graphe[i][1]<0:
13            compteur=compteur+1
14    return compteur==1 # Si le compteur vaut 1 alors la transitivité est vérifiée.

```

Q14. Créez une fonction `test_lien_sous_graphe(graphe)` permettant de tester si tous les sommets du graphe à 3 sommets sont reliés entre eux ou non. Si les sommets sont tous reliés entre eux, la fonction renverra `True`, `False` sinon. `graphe` est une liste de listes d'adjacence des sommets composant un graphe (ou le sous-graphe) orienté de 3 sommets

Réponse

```

1 def test_lien_sous_graphe(graphe):
2     '''graphe est une liste contenant les listes d'adjacence
3     Si les sommets sont tous relies alors la fonction renverra True, False sinon
4     '''
5     compteur_taille_2=0 # Compteur du nombre de sous-listes de taille 2
6     for i in range(len(graphe)): # Pour toutes les sous-listes de graphe
7         if len(graphe[i])==2: # Si la sous-liste est de taille 2
8             compteur_taille_2+=1
9     return compteur_taille_2==3 # Si toutes les sous-listes sont de taille 2 alors tous les som-
    mets sont reliés entre eux.

```

Q15. En considérant une liste `voisins` de trois sommets $[i,j,k]$, dont nous savons que j est voisin de i et k , on cherche à déduire à l'aide de la matrice d'adjacence complète du graphe, les listes d'adjacence du sous-graphe composé des sommets i,j et k . Créez une fonction `creation_liste(matrice,liste_voisins)` renvoyant la liste d'adjacence du sous-graphe.

Réponse

Voir fichier Python

Un problème se pose avec le sommet 0. En effet, en python `+0` et `-0` représente le même nombre. Donc il est nécessaire de renseigner le numéro réel des sommets et non pas leur numéro sous python (le premier sommet ne sera donc pas 0 mais bien 1).

Q16. Créez alors le programme permettant de vérifier si le graphe orienté renseigné par sa matrice d'adjacence est bien un graphe vérifiant les relations de transitivité. ATTENTION, pensez à gérer éventuellement le problème du sommet 1 (en effet dans python, ce sommet correspond à la ligne 0 de la matrice d'adjacence). Lors de la création des listes d'adjacence, il n'y a pas de différence en Python entre `+0` et `-0`. Il faut donc renseigner les listes d'adjacences à l'aide de la fonction `creation_liste` par les numéros réels des sommets.

Réponse

Voir fichier Python