RAPPORT PROJET OUTILS POUR LA CONCEPTION D'ALGORITHMES - DÉTECTION DES COMMUNAUTÉS DANS LES RÉSEAUX SOCIAUX

Réalisé par :

AGHLAL Nizar **ELFRANI Khadija**

Encadré par :

M. Manousakis George

Année Universitaire 2022-2023

INSTITUT DES SCIENCES ET TECHNIQUES DES YVELINES - 10-12 AV de l'Europe, 78140 Vélizy Villacoublay
Tél : 01 39 25 38 50 site web : www.isty.uvsq.fr

SOMMAIRE

INTRODUCTION

Le but du projet est de nous initier à la lecture et la compréhension d'articles scientifiques. Dans ce but, on va implémenter certains algorithmes de graphes issus de la littérature scientifique récente. La problématique principale du projet est d'énumérer certaines structures dans des graphes modélisant théoriquement certains réseaux sociaux. En particulier on va chercher dans ces graphes des communautés, c'est-à-dire des ensembles de nœuds très densément connectés entre eux.

En théorie des graphes, il existe plusieurs modélisations possibles pour la notion de communautés. On peut par exemple modéliser la notion de cliques maximales ou de bicliques maximales, dont les définitions sont données dans le projet.

PARTIE 1 : Graphes de Stanford et les graphes aléatoires

1 - Graphes générés aléatoirement

Dans la partie de la génération des graphes aléatoires , on a implémenté une fonction **generate_graph()** dans notre programme qui permet à l'utilisateur après avoir sélectionné le choix du graphe aléatoire d'entrer un nombre de sommets qui doit être supérieur ou égal à 3 et qui va être attribué au graphe, et grâce à ceci , notre fonction permettra par la suite d'avoir un graphe dont les arêtes apparaissent avec une probabilité p quelconque mais qui doit être comprise entre 0 et 1 , un critère qu'on a abordé en ajoutant des détails à notre fonction.

Cependant, on a adapté la méthode de stockage du graphe dans une liste d'adjacence pour pouvoir récupérer ce dernier et le passer à un dictionnaire d'adjacence tout en s'assurant que tous les voisins des sommets sont ajoutés dans chaque sommet de la liste , et donc à la fin , la fonction permettra d'afficher le graphe généré aléatoirement sous forme de dictionnaire d'adjacence dans notre terminale.

```
Sélectionnez un graphe : 4
Entrer un nombre de sommets (le nombre de sommets doit etre supèrieur ou ègal à 3 : 5
Le graphe générer aléatoirement :
{0: [4, 1, 2, 3], 1: [0, 4, 2], 2: [0, 1, 4], 3: [0, 4], 4: [0, 3, 1, 2]}
```

figure 1 : Affichage d'un graphe généré aléatoirement

2 - Graphes de Stanford

Dans cette partie, on a utilisé la base de données des grands graphes de Stanford et plus précisément les graphes des réseaux sociaux : egoFacebook, feather-lastfm-social et email-Eu-core.

- Dans un premier temps, on a téléchargé les fichiers contenant les arêtes de chaque graphe, pour le réseaux feather-lastfm-social , on a fait des petites modifications pour rendre la tâche plus simple, on a d'abord converti le fichier de .csv à un fichier text , et puis on a remplacé le séparateur des arêtes "," par un espace .

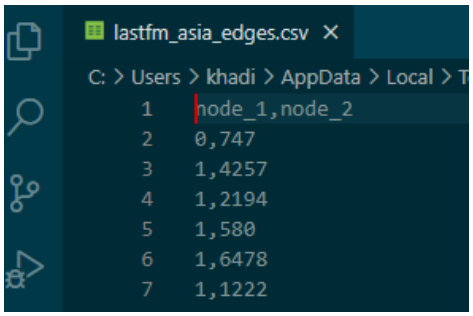


figure 2 :

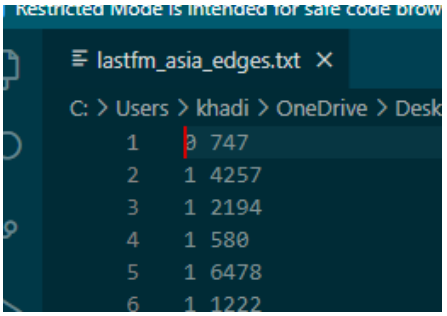


figure 3 :

- Dans un second temps, on a implémenté la fonction **get_data_from_file( )**

Elle prend le nom de fichier contenant les arêtes, et le fichier de destination puis transforme ses données en une liste d'adjacence et enfin stocke cette liste dans un fichier .json considéré comme le fichier de destination pour pouvoir manipuler le graphe lié à cette liste plus facilement.

ci-dessous, un exemple d'affichage de la liste d'adjacence à l'aide de notre menu d'accueil :

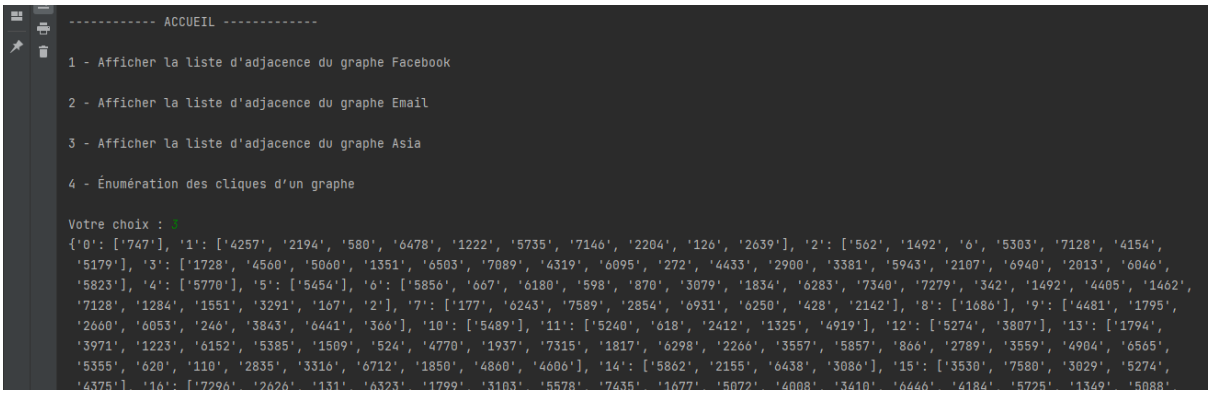


figure 4 : affichage de liste d'adjacences du graphe Asia

3 - Fonctions manipulation

a) Détermination du degré maximum du Graphe :

Pour pouvoir déterminer le degré maximum pour chacun des deux graphes (graphe aléatoire et graphe de Stanford) , on a réussi à implémenter la fonction **max_degree()** .
Cette fonction parcourt chaque sommet du graphe et calcule son degré en comptant le nombre d'adjacences . Le degré maximum est mis à jour à chaque itération si le degré du sommet courant est supérieur au degré maximum précédemment trouvé.
À la fin , la fonction affiche le degré maximum du graphe.

b) Détermination du nombre de sommets pour chaque degré du Graphe :

Pour réussir à déterminer pour chaque degré de graphe , le nombre de sommets ayant ce degré , on a implémenté la fonction **nb_sommet_degre()** , qui réussit à parcourir chaque sommet du graphe et calcule son degré en en comptant le nombre d'adjacences . Elle utilise un dictionnaire

pour enregistrer le nombre de sommets pour chaque degré . Si un degré n'a pas encore été vu , il est ajouté au dictionnaire avec une valeur initiale de 0 . À chaque itération , le compteur de degré est incrémenté de 1 et affiche en terminal le nombre de sommets pour chaque degré.
par la suite , à l'aide de la fonction **nb_sommet_degre_plot** , un graphe permettant de visualiser le nombre de sommets en fonction des degrés du graphe s'affiche.

c) Détermination du nombre de chemins induits dans le graphe de longueur 2 :

La fonction ajoute à une liste appelée chemins tous les chemins de longueur 2 qui ne sont pas des cycles. La fonction parcourt d'abord tous les noeuds du graphe (for noeud in self.graph_dic). Pour chaque noeud, elle trouve tous ses voisins (for voisin in self.graph_dic[noeud]) et regarde ensuite les voisins de ces voisins (for voisin_voisin in self.graph_dic[voisin]). Si le voisin du voisin n'est pas le noeud d'origine, et que le noeud d'origine n'est pas un voisin du voisin du voisin, et que le voisin du voisin n'est pas un voisin du noeud d'origine, alors un chemin est ajouté à la liste chemins en ajoutant la liste [noeud, voisin, voisin_voisin] à chemins. Enfin, la fonction affiche le nombre de chemins qui ont été ajoutés à la liste.

d) Test et affichage

Après avoir appelé et passé chacune de ses fonctions à un des deux graphes , ces dernières s'exécutent sur le graphe respectivement et affichent les fonctionnalités demandées .

- Test sur un graphe aléatoire.

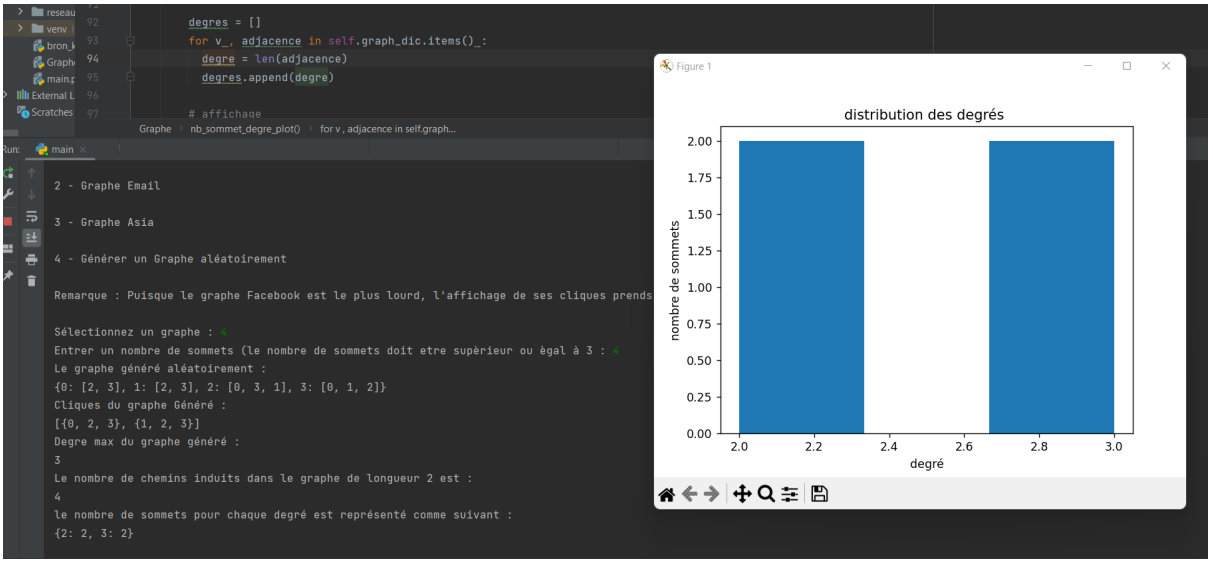


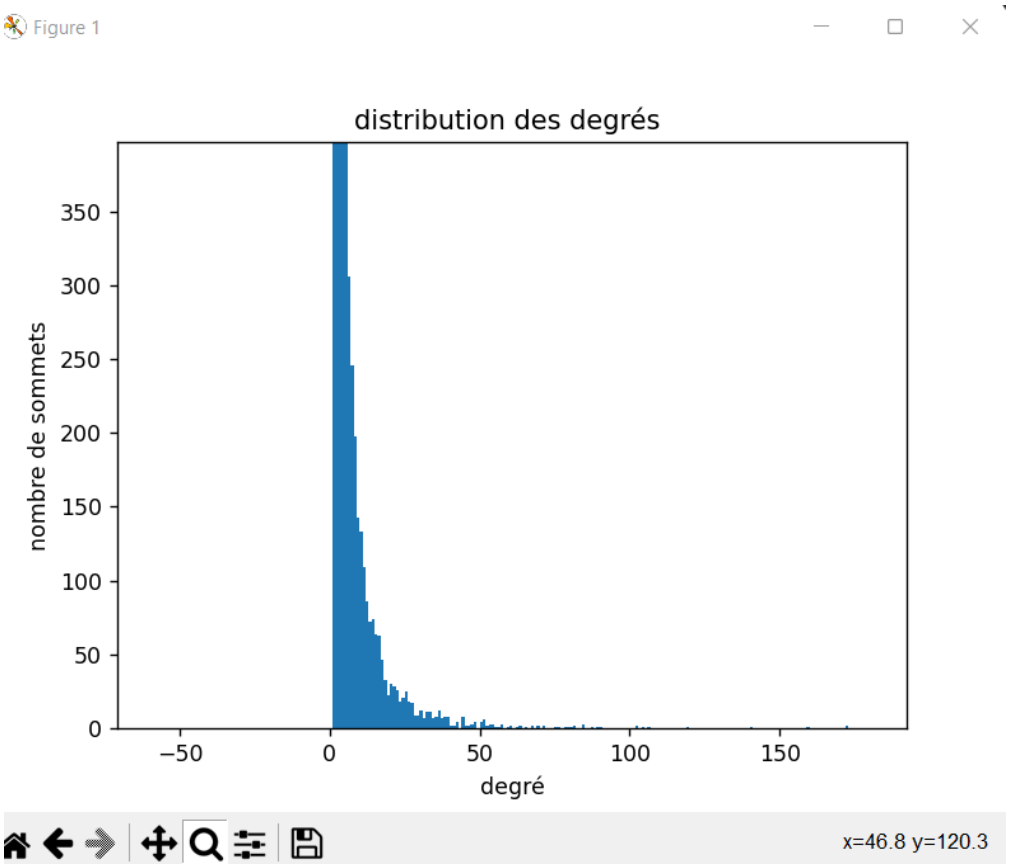
figure 5 : affichage du degré maximum , du nombre de sommets pour chaque degré et le nombre de chemins induits de longueur 2 pour un graphe aléatoire.

- Test sur un graphe de Stanford (graphe Asia par exemple)

Degré max et chemins de longueur 2 :

```
--- PARTIE 1 ---
Degre max du graphe généré :
7806
Le nombre de chemins induits dans le graphe de longueur 2
4
Nombre de sommets pour chaque degré en graphique ( veuillez fermer le graphique pour afficher les autres détails de graphe)
```

Graphique :



PARTIE 2 : Énumération des cliques d'un graphe

1 - Algorithme de Bron kerbosch

Dans sa forme élémentaire, l'algorithme de Bron–Kerbosch est un algorithme de retour sur trace récursif qui cherche toutes les cliques maximales dans un graphe G. Plus précisément, étant donné trois ensembles de nœuds disjoints R, P et X, il cherche toutes les cliques maximales contenant tous les sommets de R, certains de P, mais aucun de X. Précisément :

- L'ensemble R est un sous-ensemble des sommets de la potentielle clique.
- L'ensemble P est l'ensemble des sommets candidats pour être ajoutés à la potentielle clique.
- L'ensemble X contient des sommets déjà traités ou appartenant déjà à une clique maximale.

À chaque appel de l'algorithme, P et X sont des ensembles disjoints dont l'union forme des cliques lorsqu'on les ajoute à R. Autrement dit, $P \cup X$ est l'ensemble de nœuds qui sont connectés à tous les éléments de R. Quand P et X sont tous les deux vides, on ne peut plus ajouter

d'éléments à R, qui est donc une clique maximale que l'algorithme retourne. Au début, les ensembles R et X sont initialisés à l'ensemble vide alors que P est l'ensemble V des sommets du graphe.
La forme élémentaire de l'algorithme (sans pivot) est inefficace pour les graphes qui contiennent beaucoup de cliques non maximales (c'est le cas avec les graphes de Stanford) car on effectue un appel récursif par clique, maximale ou non. Pour économiser du temps en permettant à l'algorithme de retourner sur traces plus rapidement lorsqu'il parcourt des branches qui ne contiennent pas de cliques maximales, Bron et Kerbosch ont introduit une variante qui utilise un « nœud pivot » u, choisi dans P ou plus généralement dans P ∪ X, comme remarqué dans des travaux postérieurs. Toute clique maximale doit inclure soit u, soit un nœud qui n'est pas parmi les voisins de u, sans quoi la clique pourrait être agrandie en y ajoutant u.

```
algorithme BronKerbosch2(R, P, X)
    si P et X sont vides alors
        déclarer R clique maximale
    choisir un sommet pivot u dans P ⋃ X
    pour tout sommet v dans P \ N(u) faire
        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

figure 6 : représentation de l'algorithme de Bron Kerbosch

La version la plus efficace de l'algorithme de Bron–Kerbosch surtout lorsqu'on parle des grandes graphes est la version avec ordonnancement des nœuds.

Cette version consiste à renoncer au pivot dans le niveau externe de la récursion, et choisir à la place d'ordonner les appels récursifs de manière à minimiser la taille des ensembles P de sommets candidats dans chaque appel récursif.

La dégénérescence d'un graphe G est le plus petit entier d tel que tout sous-graphe de G a un sommet de degré d ou moins. Tout graphe de dégénérescence d admet un ordre tel que chaque sommet a d voisins ou moins situés après dans l'ordre. Un tel ordre, dit ordre de dégénérescence peut être trouvé en temps linéaire en sélectionnant itérativement le sommet de degré minimum parmi les sommets restants. Si l'algorithme de Bron–Kerbosch visite les sommets dans un ordre de dégénérescence, alors on peut garantir que l'ensemble P des sommets candidats à chaque appel (les voisins de v qui sont situés après dans l'ordre) est au plus de taille d. L'ensemble X des sommets exclus est constitué de tous les voisins de v qui sont situés avant dans l'ordre et peut être bien plus grand que d. En-dehors du niveau le plus élevé de la récursion, la version avec pivot vue précédemment peut être utilisée

```
algorithme BronKerbosch3(G)
    P = V(G)
    R = ∅
    X = ∅
    pour tout sommet v visités dans un ordre de dégénérescence de G faire
        BronKerbosch2({v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Figure 7 : L 'algorithme de Bron Kerbosch avec ordonnancement des nœuds.

- Cet algorithme à été implémenté en python (en détails) sous plusieurs fonctions dans notre programme dans le fichier **<<bron_kerbosch.py>>** .

2 - Test et affichage

a) Un graphe de Stanford (Asia par exemple)

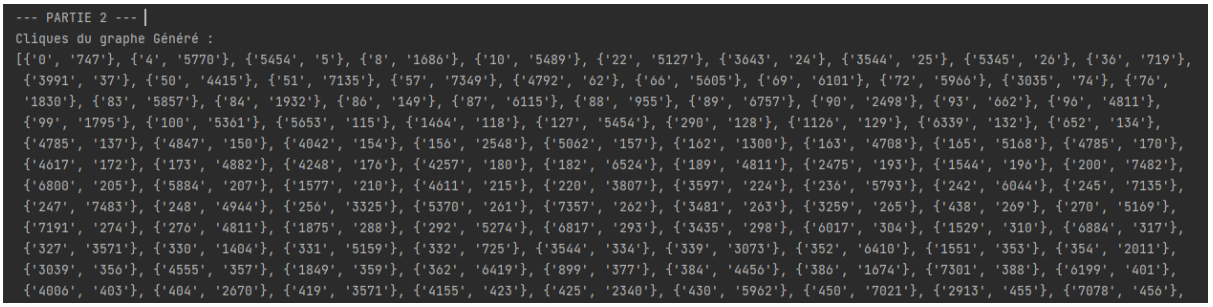


figure 7 : affichage des cliques pour un graphe de Stanford

b) Un graphe généré aléatoirement

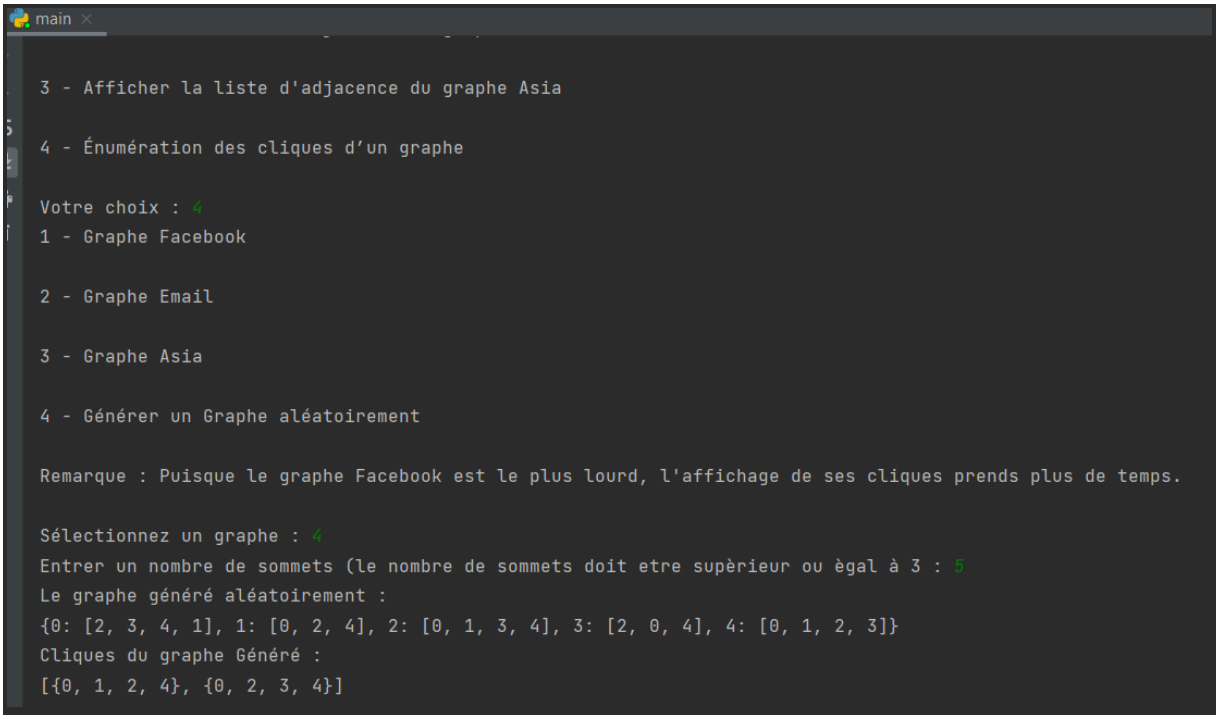


figure 8 : affichage des cliques pour un graphe généré aléatoirement

PARTIE 3 : Énumération des cliques et de bicliques maximales

1 - Algorithme de Danny Hermelin George Manoussakis

Dans cette partie, on a implémenté des algorithmes liés à l'énumération des cliques et de bicliques maximales.
L'algorithme qu'on doit implémenter est une partie d'un article scientifique qu'on doit lire et comprendre et qu'il se présente comme ci-dessous :

Algorithm 1:

Data: A graph G.
Result: All its maximal induced bicliques.

```
1  Consider any ordering σ of the vertices of G.
2  Construct the graphs G_i, i ∈ [n].
3  Initialize T an empty generalized suffix tree.
4  for i = 1 to n do
5      Compute all maximal independent sets of graph G_i.
6      for every maximal independent set I of graph G_i do
7          if I is proper then
8              Order the vertices of I following σ.
9              Search for I in T.
10             if there is a match then
11                 Reject it.
12             else
13                 Insert the proper suffixes of I in T.
14                 Output I.
```

figure 9 : Algorithme de Danny Hermelin George Manoussakis

L'implémentation de cet algorithme est présente dans notre projet sous le fichier **<<cliques_bicliques_max.py>>** . Pour l'implémentation on aura aussi besoin de l'algorithme d'énumération de la partie 2 (bron kerbosch).

2 - Tests et affichage :

- Graphe généré aléatoirement :

```
Entrer un nombre de sommets : 6
Le graphe générer aléatoirement :
{0: [1, 4, 5], 1: [0, 4], 2: [3, 5], 3: [2, 5, 4], 4: [0, 1, 3, 5], 5: [0, 2, 3, 4]}
--- PARTIE 1 ---
Degre max du graphe généré :
4
Nombre de sommets pour chaque degré en graphique ( veuillez fermer le graphique pour afficher les autres détails de graphe)
Le nombre de chemins induits dans le graphe de longueur 2
4
--- PARTIE 2 ---
Cliques du graphe Généré :
[{0, 1, 4}, {0, 4, 5}, {2, 3, 5}, {3, 4, 5}]
--- PARTIE 3 ---
Cliques et bicliques maximales du graphe Généré :
{0, 1, 4}
{0, 4, 5}
{2, 3, 5}
{3, 4, 5}
None
```

figure 10 : affichage cliques maximales graphe aléatoire.

- Graphe de Stanford (Graphe Asia par exemple)

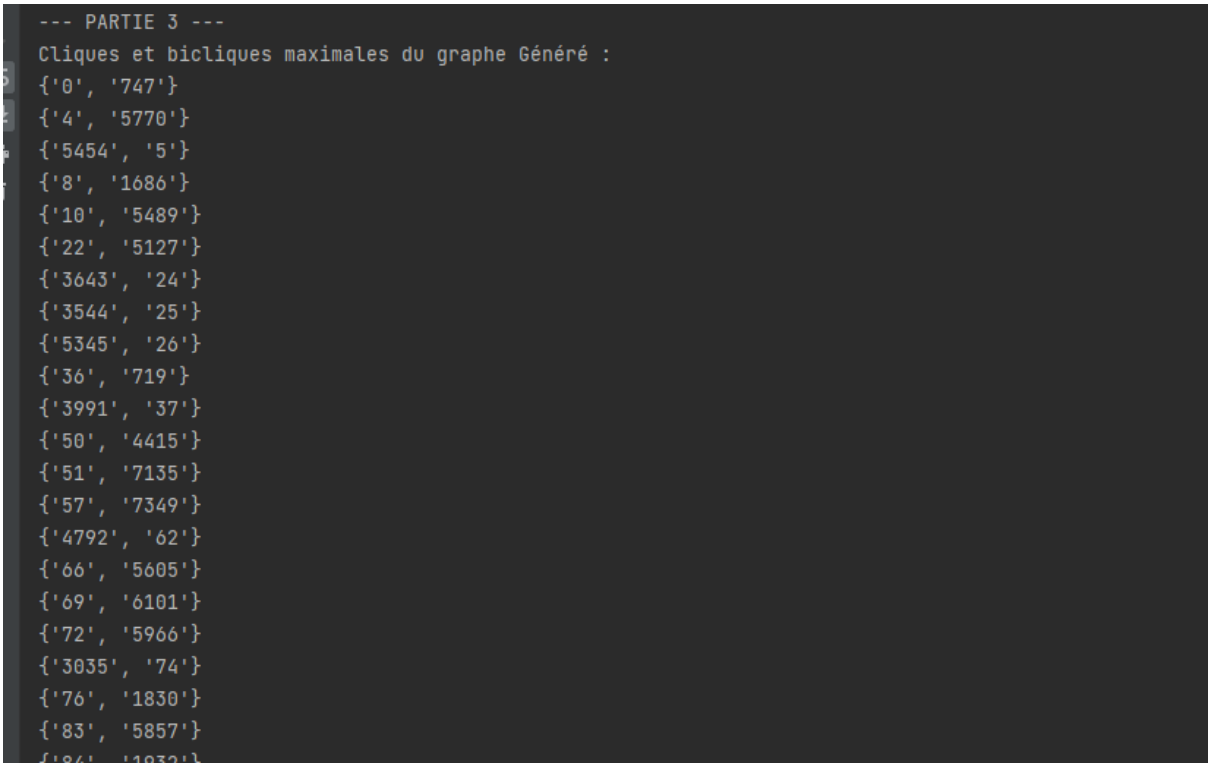


figure 11 : affichage cliques maximales du graphe asia

CONCLUSION

Pour mener à bien ce projet, on a fait face à de nombreux défis qu'on a pu surmonter par notre sérieux et notre persévérance parmi eux on cite.
Le premier défi est le temps le projet a été demandé dans une période très chargé de contrôles, de DM et en parallèle avec d'autre projets ce qui ne nous a pas laissé assez de temps d'explorer d'autres environs des notions qu'on a appris dans le cours et au durant les TD.
Pour notre part, le projet nous a permis d'apprendre comment se comporter avec des articles scientifiques et nous à aider à améliorer nos compétences en python.
Ce travail a été pour nous à la fois, un sujet de recherche et d'application, c'est vraiment une occasion parfaite pour tester nos connaissances et nos compétences dans le domaine de la programmation et d'affirmation, cette étude nous a permis également de développer nos compétences techniques sous un angle complémentaire des enseignements qui nous a été dispensés à l'ISTY.
Nous espérons que notre travail sera à la hauteur. Mais évidemment, ce travail étant une œuvre humaine, ce n'est pas un modèle parfait, c'est pourquoi nous restons ouverts à toutes les critiques et sommes prêts à recevoir toutes les suggestions et les remarques tendant à améliorer davantage cette étude, étant donné que tout travail informatique a toujours été l'œuvre d'une équipe.

WEBOGRAPHIE

Algorithme de Bron-Kerbosch — Wikipédia (wikipedia.org)
Stanford Large Network Dataset Collection
Efficient enumeration of maximal induced bicliques (sciencedirectassets.com)