TP1

Very Large Graph

I. Introduction Graphe et Spark

Cours : https://miat.inrae.fr/schiex/Export/Graphe-Slides.pdf
Résumé : https://www.dil.univ-mrs.fr/~gcolas/algo-licence/slides/graphes-intro-algo2.pdf
Résumé :
https://www.imo.universite-paris-saclay.fr/~ruette/mathsdiscretes/resumecoursGraph2012-reduit.pdf
Exemples : https://www.dil.univ-mrs.fr/~gcolas/algo-licence/slides/graphes-algo1.pdf
Clustering : https://lrouviere.github.io/INP-HB/cours_graphes.pdf
Spark cheat sheet :
https://www.datasciencecentral.com/wp-content/uploads/2021/10/2808331195.jpg

II. Exercice de recherche de cliques maximales dans un Graphe

Ressources :
https://fr.wikipedia.org/wiki/Clique_(th%C3%A9orie_des_graphes)
https://fr.wikipedia.org/wiki/Algorithme_de_Bron-Kerbosch
https://fr.wikipedia.org/wiki/D%C3%A9g%C3%A9n%C3%A9rescence_(th%C3%A9orie_des_graphes)

Implémentation en Python en suivant ces pseudo-codes :

Les graphes seront représentés avec Networkx, l'affichage avec Matplotlib et les graphes peuvent être généré de manière aléatoire (par exemple Erdos Renyi https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.erdos_renyi_graph.html)

Simple/naïf

```
fonction recherche_clique(graphe G):
    n = nombre de sommets de G
    taille_max_clique = 0
    clique_max = ensemble vide

    pour chaque sous-ensemble S de sommets de G:
        si la taille de S > taille_max_clique et est_clique(G, S):
            taille_max_clique = taille de S
            clique_max = S

    retourner clique_max
```

```
fonction est_clique(graphe G, ensemble S):
    pour chaque paire de sommets v, w dans S:
        si v n'est pas adjacent à w dans G:
            retourner faux
    retourner vrai
```

Bron Kerbosch

```
algorithme BronKerbosch1(R, P, X)
    si P et X sont vides alors
        déclarer que R est une clique maximale
    pour tout sommet v dans P faire
        BronKerbosch1(R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}

BronKerbosch1(∅, V, ∅) //appel initial
```

Bron Kerbosch pivot

```
algorithme BronKerbosch2(R, P, X)
    si P et X sont vides alors
        déclarer que R est une 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}
```

Bron Kerbosch pivot et dégénérescence

```
algorithme BronKerbosch3(G)
    P = V(G)
    R = Ø
    X = Ø
    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}
```

Dégénérescence

- Initialiser la liste de sortie *L* à la liste vide.
- Calculer une valeur d_v pour chaque sommet *v* de *G*, qui est le nombre de voisins de *v* qui n'est pas déjà dans *L* (initialement, il s'agit donc du degré des sommets dans *G*).
- Initialiser un tableau *D* tel que *D*[*i*] contienne la liste des sommets *v* qui ne sont pas déjà dans *L* pour lesquels $d_v = i$.
- Initialiser la valeur *k* à 0.

- Répéter *n* fois:
    - Parcourir les cellules du tableau *D*[0], *D*[1], ... jusqu'à trouver un *i* pour lequel *D*[*i*] est non-vide.
    - Mettre *k* à max(*k*,*i*).
    - Sélectionner un sommet *v* de *D*[*i*], ajouter *v* en tête de *L* et le retirer de *D*[*i*].
    - Pour chaque voisin *w* de *v* qui n'est pas déjà dans *L*, retirer une unité de d_w et déplacer *w* de la cellule de *D* correspondant à la nouvelle valeur de d_w.