Complexité sur les graphes

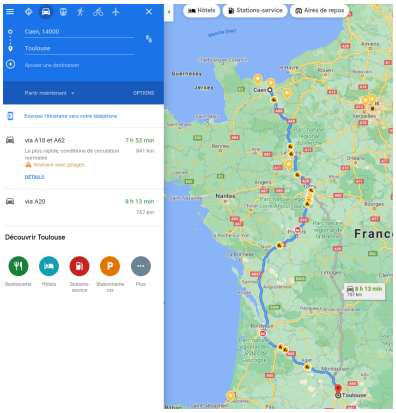
EXERCICE 1 – QUEL ALGORITHME DE PPC CHOISIR ?

Pour chacun des graphes suivants, indiquez s'il vaut mieux choisir :

- l'algorithme de Dijkstra où la file de priorité est implémentée avec des tableaux
- ou l'algorithme de Dijkstra où la file de priorité est implémentée avec un tas binaire min
- ou l'algorithme de Bellman-Ford.

Bien sûr, la complexité est primordiale !

Q1. le graphe constitué des 10000 plus grandes communes françaises, où tout couple de sommets est reliée par une arête dont le poids est égal au temps de trajet en voiture entre les deux communes (avec comme référence Google Maps – par exemple)

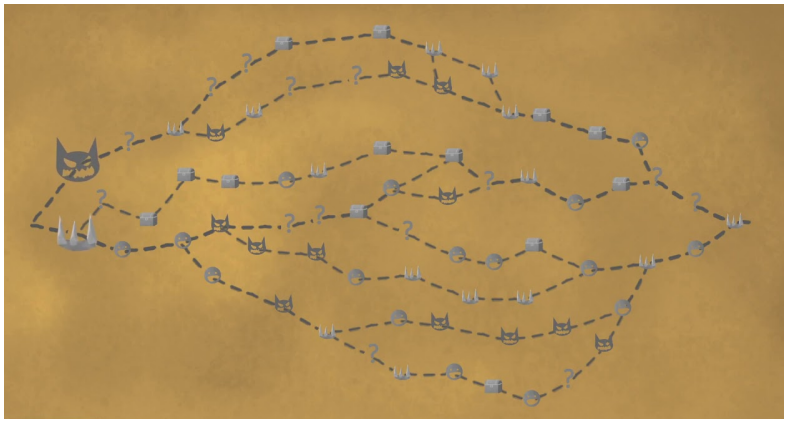


Correction. Le graphe en question est un graphe complet aux poids positifs. Donc $|A| = \frac{|S|(|S|-1)}{2} = O(|S|^2)$.
L'algorithme de Dijkstra avec implémentation tableau est le meilleur car la complexité est en $O(|S|^2)$ (alors que les autres sont respectivement $O(|S|^2 \log(|S|))$ et $O(|S|^3)$.

Q2. le graphe de la ville de Caen, où chaque sommet est une intersection et où les arêtes représentent les rues reliant les intersections

Correction. Très certainement le degré moyen est très petit : le nombre de rues incidentes à une intersection dépasse rarement 8 !. Donc $|A| = O(|S|)$.
L'algorithme de Dijkstra avec implémentation tas binaire min est le meilleur car la complexité est en $O(|S| \log(|S|))$ (alors que les autres sont en $O(|S|^2)$).

Q3. un graphe provenant d'un donjon dans un jeu vidéo. Certaines salles nous font perdre des points de vie, d'autres nous en fait gagner. On veut accéder à la salle au trésor depuis l'entrée du donjon en minimisant la perte totale de points de vie.



Correction. Le graphe peut avoir des poids négatifs, donc on est obligés d'utiliser Bellman-Ford !

Q4. un graphe non orienté avec des arêtes au poids positifs où le degré moyen serait $\sqrt{|S|}$ où $|S|$ est le nombre de sommets.

Correction. On a donc $|A| = O(|S|\sqrt{|S|})$.
L'algorithme de Dijkstra avec implémentation tas binaire min est le meilleur car la complexité est en $O(|S|\sqrt{|S|}\log(|S|))$ (alors que les autres sont en $O(|S|^2)$ et $O(|S|^2\sqrt{|S|})$).

Problèmes algorithmique sur les graphes

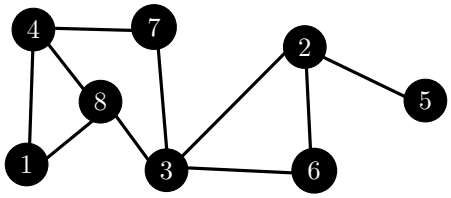
EXERCICE 2 – NOMBRE D'ARÊTES INTERNES

Soit G un graphe non orienté.

Écrire en pseudo-code une fonction qui étant donnés un graphe non orienté sans boucle G et une liste de sommets X donne le nombre d'arêtes qui ont leurs deux extrémités dans X.

Par exemple, pour le graphe ci-dessous, et pour `X = [1,2,3,6]`, la fonction doit renvoyer `T`.

Quelle est la complexité de votre algorithme ?



Correction.

```
Fonction nombre_aretes_internes(G : Graphe, X : Liste de Sommets)
        cpt = 0
        E = ensemble obtenu à partir de la liste X
        Pour chaque sommet s de G
        Faire   Si s appartient à E
                Alors   Pour chaque voisin t de s
                        Faire   Si (t appartient à E)
                                Alors cpt = cpt + 1
        // Ne pas oublier de diviser par 2 car on compte deux fois les arêtes
        Renvoyer cpt/2
```

Changer une liste en ensemble nous permet de tester l'appartenance en $O(1)$. La transformation de la liste en ensemble se fait en $O(|X|)$.

La boucle "Pour chaque voisin de s" a un nombre de passages égal au degré de s.

Cette boucle n'est exécutée que si s appartient à E. Le nombre de fois que la boucle "Pour chaque voisin de s" est répétée pour la totalité des s est alors égale au nombre d'arêtes qui sortent d'un sommet de X. Ce nombre est plus généralement borné par un $O(|A|)$.

Le nombre de tests d'appartenance "s appartient à E" est égal à $O(|S|)$.

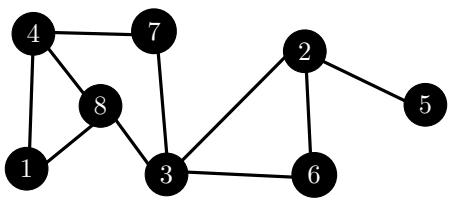
Au final, la complexité de l'algorithme est en $O(|X| + |S| + |A|) = O(|S| + |A|)$.

EXERCICE 3 – ENSEMBLE DOMINANT

Soit G un graphe non orienté.

Un ensemble D de sommets est dit dominant pour G si tout sommet du graphe G est soit dans D, soit voisin d'un sommet de D.

Q1. Existe-t-il un ensemble dominant à deux sommets pour le graphe ci-dessous?



Correction. Oui, l'ensemble $\{2, 4\}$.

Q2. Ecrire en pseudo-code une fonction qui prend un graphe G et une liste de sommets D, et qui renvoie vrai si D est dominant ; faux sinon.

Votre algorithme doit être efficace !

Correction. Voici la version naïve que beaucoup risquent de faire :

```
Fonction est_dominant(G : Graphe, D : Liste de Sommets)
        Pour chaque sommet s de G
        Faire   b = Faux
                Pour chaque sommet d de D
                Faire Si (d == s)
                        Alors b = vrai
                        Pour chaque voisin t de d
                        Faire   Si (t == s)
                                Alors b = Vrai
                        FinFaire
                FinFaire
                Si non(b)
                Alors Renvoyer Faux
        FinFaire
        Renvoyer Vrai
```

Non seulement c'est délicat, mais ce n'est pas efficace !
La complexité est en $O(|S| \times |A|)$. En effet, en se référant au cours on voit que la complexité est en

$$\sum_{s \in G} \sum_{d \in D} \sum_{t \text{ voisin de } d} O(1) = |S| \times \sum_{\text{sommet } d \in D} \text{degré(s)} \leq |S| \times \sum_{\text{sommet } s \in G} \text{degré(s)} = O(|S||A|)$$

où la dernière égalité provient de la formule des poignées de main.
Voici le bon algo :

```
Fonction est_dominant(G : Graphe, D : Ensemble de Sommets)
        est_marque = dictionnaire initialisé à Faux pour tous les sommets
        Pour chaque sommet d de D
        Faire   est_marque[d] = Vrai
                Pour chaque voisin t de d
                Faire   est_marque[t] = Vrai
                FinFaire
        FinFaire
        Pour chaque sommet s de G
        Faire Si non(est_marque[s])
                Alors Renvoyer Faux
        FinFaire
        Renvoyer Vrai
```

Q3. Montrez la complexité de votre algorithme. Il devrait être en $O(|S| + |A|)$.

Correction. La seconde boucle est clairement en $O(|S|)$.
Le coeur de la première double boucle est en $O(1)$ (accession à un élément dans un dictionnaire).
Donc la complexité de cette double boucle (Revoir cours complexité) est :

$$\sum_{\text{sommet } d \in D} \sum_{t \text{ voisin de } d} O(1) = \sum_{\text{sommet } d \in D} \text{degré(d)} \leq \sum_{\text{sommet } s \in G} \text{degré(s)} = O(|A|)$$

où la dernière égalité provient de la formule des poignées de main.
La complexité de cette fonction est donc $O(|S| + |A|)$.

Partie TP

Aujourd'hui, notre but est de voir l'influence de la structure de données (matrice d'adjacence ou listes d'adjacence) sur l'efficacité des algorithmes de graphe. Comme d'habitude, le sujet est sur `ecampus`.

Dans le sujet, on teste les fonctions sur un graphe que voici :