Compte rendu intermédiaire projet info

Perann Nedjar, Adrien Letellier

March 2023

Question 2

L'algorithme Depth first search (DFS) a une complexité en $O(V + E)$, où V désigne le nombre de sommets du graphe et E désigne le nombre d'arêtes. En effet, cet algorithme explore chaque noeud et chaque arête exactement une fois (grâce au dicitionnaire des noeuds visités, on ne passe par un noeud que si il n'a pas été visité avant).
Etant donné que l'on effectue un l'algorithme DFS sur chaque noeud du graphe, on obtient une complexité en $O(V(V + E))$. Dans le cas où le graphe est dense (E est de l'ordre de V^2), on obtient une complexité en $O(V^3)$.

Question 3

La fonction DFS_chemin a une complexité en $O(V + E)$ (même raisonnement qu'à la question précédente). La vérification de la connexité des sommets a une complexité en $O(V)$ car on parcourt tous les sommets. La reconstruction du chemin à partir du dictionnaire parent a elle aussi une complexité en $O(V)$ parcourt le dictionnaire parent à partir du sommet d'arrivée jusqu'au sommet de départ pour reconstruire le chemin.
Finalement la complexité de l'algorithme est en $O(V + E)$.

Question 6

Lors de la première boucle for, on parcourt toutes les arêtes du graphe, ce qui implique une complexité en $O(E)$. La conversion de la liste des puissances en un ensemble avec la fonction set() a une complexité de $O(E)$. Le tri de la liste avec la méthode sort() a une complexité de $O(ElogE)$ (la fonction sort() de python réalise un tri fusion, dont on connait la complexité). La boucle while de recherche dichotomique a une complexité de $O(logE)$. En effet, à chaque itération, elle divise la taille de la plage de recherche par deux.
Ainsi, étant donné que la fonction get path with power a une complexité en $O(V + E)$, la complexité de l'algorithme est $O((V + E)log(E))$.
Dans le cas d'un graphe dense, on obtient une complexité en $O(V^2log(V^2))$.

Question 12

La fonction Kruskal_int réalise l'algorithme de kruskal avec une strucutre de type union-find. La première boucle for parcourt tous le noeuds et réalise $O(1)$ opérations élémentaires, la complexité de cette partie de l'algorithme est donc

en $O(V)$.
La seconde boucle for qui sert à la création de la liste d'arête triée a une complexité en $O(E)$ puisqu'elle parcourt un nombbre de valeur du même ordre que l'ordre de l'ensemble des arêtes, la fonction de trie utilisée juste après a une complexité en $O(Elog(E))$. La complexité de cette partie de l'algorithme est donc en $O(Elog(E))$.
Dans la boucle suivante on parcourt la liste des arêtes en appelant soit la fonction find dont la complexité est donnée par la fonction d'Ackerman inverse, (on peut considérer cette complexité comme constante, car la fonction d'Ackerman est inférieur à toute fonction logarithmique) soit la fonction union dont la complexité est en $O(1)$. La complexité de cette partie est donc en $O(Elog(V))$.
Enfin la fin de l'algorithme consiste en une application de la fonction kruskal_int et en la création d'un graphe initial en parcourant tous les sommets et les arêtes, donc en $O(V + E)$. a une complexité
Cette fonction a donc une complexité finale en $O(Elog(E))$.

Question 15

La fonction get_kruskal a une complexité en $O(Elog(E))$ et la fonction de DFS a une complexité en $O(V+E)$. Notons N le nombre d'élément dans la list _routes. Dans cette boucle on éxécute d'abord un DFS (en $O(V + E)$) puis on fusionne les chemins avec une opération en $O(V)$. L'opération de calcul de la puissance minimale requise pour chaque chemin est en $O(E)$.
Finalement, on obtient une complexité en $O(N(Elog(E) + V + E))$.