LINMA1691 : Théorie des graphes
Devoir 3 : Arbres couvrant

Comme vu en cours, le problème d'arbre couvrant a de nombreuses applications. Dans ce devoir, nous en explorons et implémentons deux en **Python**. Plus précisément, il vous est demandé d'implémenter efficacement les fonctions *spanning_tree_1* et *spanning_tree_2* du fichier *template.py*. Vous ne pouvez pas importer d'autres librairies que celles déjà présentes.

1 Promesses électorales

Contexte

En tant que nouveau bourgmestre de votre commune, vous désirez satisfaire votre promesse de renouveler les routes du village. Chaque route renouvelée apporte un "score de satisfaction" à la population en fonction de sa position et fréquentation mais est bloquée pendant la durée des travaux ! Pour éviter de mécontenter vos électeurs, vous voulez donc vous assurer qu'il est toujours possible à partir de chaque carrefour d'atteindre n'importe quel autre carrefour. Quel est le plus gros score de satisfaction (défini comme la somme des scores de satisfaction des routes renouvelées) que vous pouvez obtenir ?

Input

L'input vous est donné sous la forme d'arguments d'une fonction à compléter : *spanning_tree_1(N, roads)*

N est le nombre de carrefours, indexés de 0 à N-1. *roads* est une liste de dimension M décrivant le réseau. Chaque élément du tableau est un tuple (u, v, s) représentant une route à double sens entre les carrefours u et v de satisfaction s. Il est garanti que $4 \leq N, M \leq 10^3$.

Exemple d'input:

(4, [(0, 1, 3), (0, 2, 2), (1, 2, 1), (1, 3, 4), (2, 3, 5)])

Réponse : 8

Output

Il vous est demandé de compléter la fonction et de retourner le plus gros score de satisfaction possible.

Nous attendons un algorithme de complexité temporelle $\mathcal{O}(N^2)$ ou $\mathcal{O}(E \log E)$.

2 Algorithme de Karger

Contexte

Il est parfois intéressant d'utiliser des algorithmes non déterministes pour résoudre certains problèmes. Le tri rapide[1] est par exemple un algorithme de tri probabiliste très efficace. Nous allons implementer dans cet exercice un algorithme probabiliste pour estimer la coupe minimum d'un graphe. Une coupe d'un graphe est un ensemble d'arêtes déconnectant le graphe si elles sont toutes enlevées. La coupe minimum d'un graphe est donc la coupe ayant le plus petit nombre d'arêtes.

L'algorithme de Karger[2] est un algorithme probabiliste estimant la coupe minimum d'un graphe. Il consiste à prendre à chaque étape une arête uniformément aléatoirement et la contracter[3]. On répète la procédure jusqu'à ce qu'il ne reste plus que 2 noeuds dans le graphe et l'estimation de la coupe minimum est le nombre d'arêtes entre ces noeuds. Cet algorithme peut sembler très simple, mais il s'avère qu'il marche bien en pratique. On peut montrer que la probabilité de succès de l'algorithme de Karger est plus grande ou égale à $\binom{N}{2}^{-1}$. Une minute de réflexion permet de se rendre compte que cet algorithme est équivalent à l'application de l'algorithme de Kruskal au graphe dont on a pondéré les arêtes de manière aléatoire, à un détail près: Il suffit en effet d'arrêter l'algorithme de Kruskal quand il ne reste plus que deux arbres à relier et de compter les arêtes entre ceux-ci.

Input

L'input vous est donné sous la forme d'arguments d'une fonction à compléter : *spanning_tree_2(N, edges)*.

N est le nombre de noeuds du graphe, indexés entre 0 et *N-1*. *edges* est une liste de dimension *M* décrivant les arêtes du graphe. Chaque élément du tableau est un tuple (u, v) correspondant à une arête entre u et v.

Exemple d'input :

(8, [(0, 1),(0, 2),(0, 3),(1, 2),(1, 3),(2, 3),(4, 5),(4, 6),(4, 7),(5, 6),(5, 7),(6, 7),(0, 4),(1,5)])

Réponse : 2

Output

Il vous est demandé d'implémenter ainsi un algorithme donnant la coupe minimum avec une probabilité plus grande que 0.9999.

Consignes

Vous devez soumettre chaque méthode sur l'activité Inginious[4] du même nom.

Le langage de programmation est **Python 3** (version 3.5).

Deadline : 13 novembre 2019, 10h30. La deadline est stricte : il n'est plus possible de soumettre après cette date.

[1] https://en.wikipedia.org/wiki/Quicksort
[2] https://en.wikipedia.org/wiki/Karger%27s_algorithm
[3] Contracter une arête revient à la retirer et fusioner les deux noeuds qu'elle relie
[4] https://inginious.info.ucl.ac.be