RAPPORT NSI SPANNING TREE RENAUDIN ARIANE

Le vaste domaine des graphes et de la théorie des graphes offre une perspective captivante pour modéliser et résoudre une multitude de problèmes dans des secteurs allant de l'informatique à la logistique, en passant par les réseaux. Dans ce rapport, nous explorons deux codes Python qui se complètent, traitant de la génération aléatoire de graphes et de la résolution du problème de l'arbre couvrant minimal.

Le Spanning Tree Protocol (STP) est un protocole réseau de niveau 2 (la couche de liaison de données, utilisée pour transférer des données entre des nœuds de réseau adjacents d'un réseau étendu ou entre des nœuds du même réseau local) permettant de déterminer une topologie réseau sans boucle (arbre) dans les LAN avec des ponts. Il est défini dans la norme IEEE 802.1D (Selon cette norme, il ne peut exister qu'un seul chemin actif entre deux points d'extrémité ou stations pour qu'ils fonctionnent correctement) (IEEE, Institute of Electrical and Electronics Engineers)le STP repose sur un algorithme novateur décrit par Radia Perlman en 1985, comme exposé dans son rapport accessible via le lien : https://www.it.uu.se/edu/course/homepage/datakom/ht06/slides/sta-perlman.pdf.

Sans le protocole spanning-tree, cette redondance posera de gros problèmes comme les tempêtes de broadcast, les messages reçus deux fois ou l'inconstance des tables MAC (une panne réseau)
Le protocole Spanning Tree empêche les boucles de se former en fermant tous les chemins possibles, sauf un, pour chaque paquet de données. Les commutateurs d'un réseau utilisent le protocole Spanning Tree pour définir les chemins « root » et les ponts par lesquels les données peuvent transiter, et ferment fonctionnellement les chemins en double, les rendant inactifs et inutilisables tant qu'un chemin principal est disponible.

Il existe des variantes de STP comme par exemple le Rapid Spanning Tree Protocol comme le dit son nom ses transition sont plus rapide car il passe par 3 switch (rejeter, apprendre, expéditeur) contrairement à STP qui en passe par 5. Un autre exemple peut être le MSTP (multiple spanning tree protocol) En général, le MSTP peut offrir une convergence plus rapide en raison de sa capacité à isoler les changements de topologie (en réseautique, la "topologie réseau" décrit la disposition physique ou logique des composants) par instance VLAN (Virtual local area network sur un même switch créer plusieurs réseaux indépendants ne pouvant pas, par défaut, communiquer entre eux.)(technique de segmentation logique d'un réseau physique en plusieurs réseaux virtuels distincts).

Pour une mise en œuvre plus optimale de la résolution du problème de l'arbre couvrant minimal, des codes Python ont été développés et optimisés. Des exemples de tels codes peuvent être consultés aux liens suivants :
https://github.com/krithikvaidya/spanning-tree-protocol
https://github.com/whataboutamir/distributedstp
https://www.bogotobogo.com/python/python_Prims_Spanning_Tree_Data_Structure.php

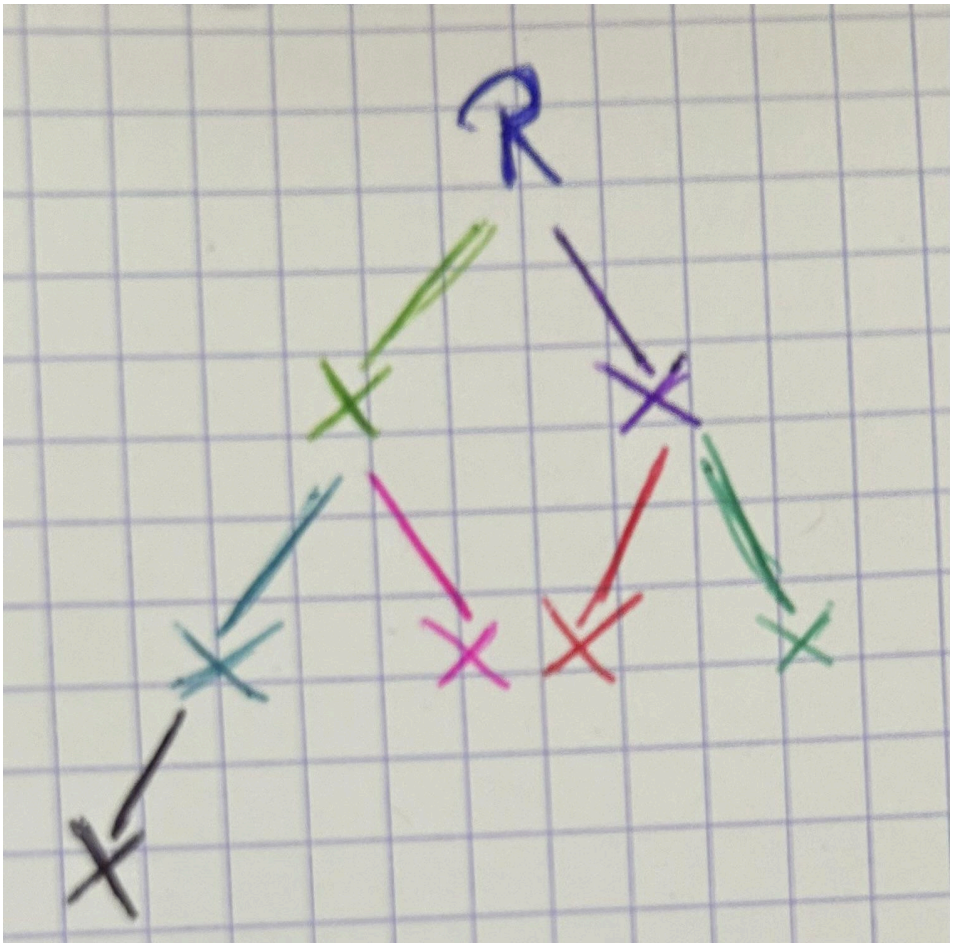
Beaucoup utilisé pour ce problème est l' algorithme de Dijkstra (L'algorithme prend en entrée un graphe orienté pondéré par des réels positifs et un sommet source. Il s'agit de construire progressivement un sous-graphe dans lequel sont classés les différents sommets par ordre croissant de leur distance minimale au sommet de départ. La distance correspond à la somme des poids des arcs empruntés.)

Le code s'attaque au problème de l'arbre couvrant minimal dans un graphe généré aléatoirement. Il utilise les arêtes générées par le premier code pour rechercher des arbres recouvrants optimaux. Plusieurs fonctions auxiliaires sont implémentées pour manipuler les ensembles d'arêtes, calculer les poids des arbres, et vérifier diverses conditions. Les fonctions principales cherchent les arbres couvrants dont le poids est inférieur à une valeur spécifiée k, et elle affiche les résultats pertinents.
Dans le code réalisé on utilise la version brute. La version brute dans le problème Spanning Tree: dans un premier temps on a une liste d'arête et on en génère tous les sous ensemble possible c'est à dire 2^n (n le nombre d'arête) possibilité. Pour réaliser cela il faut éviter les répétitions de sous ensemble, donc on utilise la fonction "set" de python (qui vérifie qu'il n ait pas de doublons). Voici un exemple de générateur d arête: generer_ensemble_arrete

```python
def genere_ensemble_arrete(graph):
    all_edges = set(graph)
    num_edges = len(all_edges)

    stocks = []
    for i in range(2 ** num_edges):
        stock = set()
        for j in range(num_edges):
            if (i // (2 ** j)) % 2 == 1:
                stock.add(tuple(list(all_edges)[j]))
        stocks.append(stock)
    return stocks
```

Une fois toutes les possibilités charger on sélectionne une première fois celle qui nous intéresse c'est à dire celle dont le nombre d'arête est égale au nombre de sommets du graphe -1. Car on sait que dans un seul arbre chaque sommet possède une arête, et la racine n' en possède pas




```
76
77
78  def nombre_arrete(stocks, num_edges):
79      return [stock for stock in stocks if len(stock) == nombre_de_points - 1]
80
```

Une fois ce premier tri effectué on en effectue un deuxième, supprimant tous les ensemble ne possédant pas tous les sommets. Si il manque un sommet comme le nombre d'arêtes est limité après le premier tri, cela signifie qu' il y a une boucle. Voici la partie du code qui vérifie que chaque point est présent.

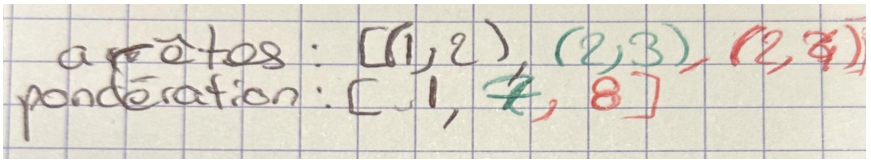
```
def arbre_couvrant(nombre_arrete, liste_point, ponderation):
    edge_weights = {edge: weight for edge, weight in zip_perso(graph, ponderation)}
    arbres_filtrés = []

    for arbre in nombre_arrete:
        sommets_arbre = set(point for arrete in arbre for point in arrete)

        # Vérifie si l'arbre couvrant contient tous les sommets
        if liste_point == sommets_arbre:
            poids_arbre = calculer_poids(arbre, ponderation, edge_weights)
            arbres_filtrés.append((arbre, poids_arbre))

    return arbres_filtrés
```

Nous avons une liste “pondération” dans le même ordre que les arêtes (l’emplacement 0 correspond l arête d'emplacement 0 dans la liste)



On utilise une fonction python zip pour associer les deux listes ou dans ce cas précis on créer une fonction annexe qui a la même utilité que le zip python.

```python
def zip_perso(iterable1, iterable2):
    min_len = min_longeur(len(iterable1), len(iterable2))
    return [(iterable1[i], iterable2[i]) for i in range(min_len)]
```

Ce qui permet d'avoir la bonne pondération associée à chaque arête.
Grâce à cela on peut calculer le poids total de chaque ensemble d'arête sélectionné par les deux tri.

```python
def calculer_poids(arbre, ponderation, edge_weights):
    poids_total = sum_perso(edge_weights[edge] for edge in arbre)
    return poids_total
```

Pour le problème de décidabilité il faut répondre par oui ou non. Spanning Tree répond à la question s'il existe un arbre couvrant du graphe et s'il a un poid total inférieur à un nombre K, si la liste est vide il n'existe pas d' arbre couvrant, l'exécution est finie .

```python
def calculer_poids(arbre, ponderation, edge_weights):
    poids_total = sum_perso(edge_weights[edge] for edge in arbre)
    return poids_total
```

Pour le problème de calculabilité on répond en renvoyant l' arbre avec le poid le plus petit. Pour cela il suffit dans la liste de poids total des arbres (calculé au préalable) relever le poid le plus petit et renvoyer l'arbre qui lui est associé.

Le graphe généré est aléatoire mais il est possible de choisir un graphe ainsi que la pondération car le code a seulement besoin de deux listes celle des arêtes et celle de la pondération.

En conclusion, ce rapport offre une plongée approfondie dans le monde fascinant du Spanning Tree Protocol et de la résolution du problème de l'arbre couvrant minimal. De la théorie à la pratique, en passant par des exemples concrets en Python, il illustre la pertinence et l'efficacité de ces concepts dans le domaine des réseaux et de la théorie des graphes.

LIENS:
https://wiki-tech.io/R%C3%A9seau/Protocoles/Spanning-Tree
https://fr.wikipedia.org/wiki/Spanning_Tree_Protocol#:~:text=Le%20Spanning%20Tree%20Protocol%20
https://fr.wikipedia.org/wiki/Radia_Perlman
https://www.techno-science.net/definition/11464.html