Explications du Code : TP7 Graphes

1. Le décorateur @property

Le décorateur @property en Python permet de transformer une méthode de classe en un attribut (variable) accessible sans utiliser de parenthèses ().

Il sert principalement à trois choses dans ce TP :

A. Lisibilité (Syntaxe)

Il permet d'utiliser la syntaxe 'g.order' au lieu de 'g.order()'. Cela donne l'impression qu'on accède à une variable stockée, alors qu'en réalité, on exécute une fonction.

```
class Graph:
  @property
  def order(self) -> int:
  return len(self.adj)

# Utilisation :
print(g.order)  # Pas de parenthèses !
```

B. Attributs calculés (Dynamisme)

C'est le point crucial. L'ordre du graphe change quand on ajoute un sommet. Au lieu de maintenir manuellement une variable 'self.nb_sommets' (risque d'erreur), @property calcule la valeur à la volée (len(self.adj)) au moment où vous la demandez. La donnée est toujours à jour.

C. Protection (Lecture seule)

Par défaut, on ne peut pas modifier une property. Ecrire 'g.order = 10' provoquera une erreur. Cela protège l'intégrité de vos données.

2. Principes clés du code fourni

Le code peut sembler long car il gère tous les cas (orienté/non-orienté, pondéré/non-pondéré). Voici les 3 blocs logiques pour comprendre l'essentiel.

Bloc 1 : La structure de données (Dictionnaire de Dictionnaires)

Choix technique ligne 158 : self.adj: Dict[str, Dict[str, int]] = {}
Au lieu d'une liste de listes, on utilise des dictionnaires imbriqués.
- self.adj['A'] donne les voisins de A.
- self.adj['A']['B'] donne directement le POIDS de l'arête A->B.
Avantage : Accès instantané en O(1) pour vérifier une connexion.

Bloc 2 : Gestion Automatique (Orienté vs Non-Orienté)

L'utilisateur n'a pas besoin de gérer le miroir des arêtes. La méthode add_edge vérifie si le graphe est non-orienté et crée le retour automatiquement.

```
def add_edge(self, u, v, weight=1):
  self.add_vertex(u) # Sécurité
  self.add_vertex(v)
  self.adj[u][v] = weight # Ajout u->v

  if not self.is_directed:
  self.adj[v][u] = weight # Ajout automatique v->u
```

Explications du Code : TP7 Graphes

Bloc 3 : La Contraction (Passage complexe)

La fonction contract_edge(x, y) fusionne Y dans X.

1. Redirection des entrants : Tout ce qui pointait vers Y pointe vers X.
2. Transfert des sortants : Tout ce que Y visait, X le vise aussi.
3. Gestion des conflits (Min) : Si X et Y avaient tous les deux une route vers Z, on conserve la route la plus courte (min).

```
# Logique simplifiée de fusion des poids
if z in self.adj[x]:
  # On garde le meilleur poids entre celui de X et celui venant de Y
  self.adj[x][z] = min(self.adj[x][z], poids_venant_de_y)
else:
  self.adj[x][z] = poids_venant_de_y
```

3. Outils Python utilisés

* Type Hinting (: List[str], -> bool) : Ce sont des indications visuelles pour aider à la lecture et au débogage. Elles n'affectent pas l'exécution.
* @classmethod (from_edge_list) : Une 'usine' qui permet de créer un graphe directement depuis une liste brute, sans faire 50 appels à add_edge.
* del : Commande pour supprimer proprement une entrée dans un dictionnaire.