Readme

Comment éxécuter le programme ?

> L'éxécution du code est différente en fonction du fichier duquel l'éxécution est lancé.

- `graphes.py`: Il suffit de lancer le code avec un compilateur python. Un code spécial va être lancé pour l'explication du code. Pour creer soi même un graphe il suffit de lancer l'éxécution suivante et le voir:

  A = Graphe(nb_sommets, type_graphe, type_sommets_graphe)

  print(A)

  Si l'on veut afficher le résultat des fonctions utilisées, on lance l'éxécution suivante:

  print(A.fonction())

- `graphes_decidabilite.py`: Il suffit de lancer le code avec un compilateur python. Un code spécial va être lancé pour l'explication du code. Pour creer soi même un graphe il suffit de lancer l'éxécution suivante et le voir:

  B = Decide(Graphe(nb_sommets, type_graphe, type_sommets_graphe))

  print(B)

  Si l'on veut afficher le résultat des fonctions utilisées, on lance l'éxécution suivante:

  print(B.fonction())

  Si l'on veut afficher le résultat de la résolution du programme, on affiche la commande suivante:

  print(B.est_clique(k))

- `graphes_calculabilite.py`: Il suffit de lancer le code avec un compilateur python. Un code spécial va être lancé pour l'explication du code. Pour creer soi même un graphe il suffit de lancer l'éxécution suivante et le voir:

  C = Calcul(Decide(Graphe(nb_sommets, type_graphe, type_sommets_graphe)))

  print(C)

  Si l'on veut afficher le résultat des fonctions utilisées, on lance l'éxécution suivante:

  print(C.fonction())

  Si l'on veut afficher le résultat de la résolution du programme, on affiche la commande suivante:

  print(C.clique())

Fonctions

Création d'un graphe

- `recup_sommets()`: Cette fonction permet de récupérer les sommets du graphe sous une forme de liste en fonction du type de sommets choisi.

- `__str__()`: Cette fonction est appellée lorsque l'on cherche à afficher le graphe, elle affiche le graphe en fonction de son type de sommets et son type.

- `est_voisin()`: Cette fonction est différente en fonction du type de graphe. Elle ne fait qu'appeller d'autres fonctions. Elle transforme les sommets en index s'ils étaient alphabétiques.
    - `est_voisin_liste_adj()`: Cette fonction prend en paramètre deux sommets de liste adjacente et retourne vrai ou faux s'ils sont voisins.
    - `est_voisin_matrice`: Cette fonction prend en paramètre deux sommets de matrice et retourne vrai ou faux s'ils sont voisins.

- `generer_liste_adj()`: Cette fonction génère un graphe sous forme de liste adjacente avec comme type de sommets, des nombres.
- `generer_liste_adj_alph()`: Cette fonction génère un graphe sous forme de liste adjacente avec comme type de sommets, des lettres.
- `generer_matrice()`: Cette fonction génère un graphe sous forme de matrice.

- `verif_type_liste_adj`: Cette fonction vérifie si le type de graphe choisi est une liste adjacente.
- `verif_type_sommets_nombres()`: Cette fonction vérifie si le type de sommets du graphe choisi est alphabétique.

- `transformer()`: Cette fonction transforme le type de sommets alphabétique en son index correspondant: *A->0, B->1, C->2...*
- `transformer_autre_sens()`: Cette fonction transforme le type de sommets de nombres en sa lettre correspondante: *0->A, 1->B, 2->C...*

- `get_lettres()`: Cette fonction retourne les lettres de l'alphabet sous forme de liste

- `convertisseur()`:
    - `convertisseur_liste_adj_alph_vers_nombres()`: Cette fonction convertie une liste adjacente avec des sommets alphabétiques en une liste adjacente avec des sommets sous forme de nombres.
    - `convertisseur_liste_adj_nombres_vers_matrice_nombres()`: Cette fonction convertie une liste adjacente avec des sommets sous forme de nombres en une matrice avec des sommets sous forme de nombres.
    - `convertisseur_matrice_nombres_vers_matrice_alph()`: Cette fonction convertie une matrice avec des sommets sous forme de nombres en une matrice avec des sommets aphabétiques.

- `test_est_voisin()`: Cette fonction renvoie deux sommets voisins. Elle sera utilisée pour l'affichage et l'explication.

Problème de décidabilité

> On utilise des fonctions du fichier `graphes.py` pour la résolution du problème.

- `brute_force()`: Cette fonction génère tous les chemins possibles existants avec k sommets: *Pour k = 2: AA, AB, AC, BA...*
- `tri_brute_force_doublons()`: Cette fonction utilise la fonction `brute_force()` mais tri les doublons et les répétitions: *AAA et BA-AB*
- `est_clique()`: Retourne vrai si une clique de taille supérieure à k existe. Il suffit simplement de vérifier si une clique de taille k existe car si elle n'existe pas, une clique de taille k+1, k+2, k+3... n'existe pas non plus.

Problème de calculabilité

> On utilise des fonctions du fichier `graphes_decidabilite.py` et du fichier `graphes.py` pour la résolution du problème.

- `clique_plus_grande()`: Cette fonction retourne la clique la plus grande d'un arbre pondéré.