Le voyageur de commerce

I. Qu'est-ce qu'une propriété NP?

Question 1 :

Certificat : Tournée possible = Une permutation des n villes (Défini par une liste des n villes)
Taille du certificat : n * log2(n) bits
Taille de l'entrée : n * n * log2(c) + log2(n) + log2(l) bits avec c la taille d'une distance entre 2 villes

La taille du certificat est bien bornée polynomialement par rapport à la taille de l'entrée.

Vérification :

- On passe une et une seule fois par ville => O(n) (i.e. vérifier que c'est une permutation)
- Longueur de la tournée inférieure ou égale à la borne => O(n)

L'algorithme est bien polynomial.

Question 2:

Question 2-1:

Pour générer aléatoirement un certificat suivant une instance du problème, il faut générer aléatoirement une permutation des n villes, pour cela, je propose de :

- Construire une liste contenant les villes de 0 à n-1
- Tant que cette liste n'est pas vide :
  - Choisir aléatoirement un élément de cette liste
  - L'ajouter à notre certificat
  - L'enlever de la liste

Question 2-2:

Un schéma d'un algorithme non-déterministe polynomial du problème serait le suivant :

- Génération aléatoire d'une permutation (algo de la Q.2.1 partie I)
- Vérification de la validité de la permutation obtenue :
  - Retourne 'Vrai' si c'est bon, c'est-à-dire que le certificat choisi aléatoirement est valide
  - Retourne "Faux' si le certificat choisi aléatoirement n'est pas valide. Cela ne signifie pas pour autant que le problème n'a pas de solutions mais seulement que celui-ci n'est pas une solution.

Question 3:

Question 3-1:

Pour une instance donnée (de n villes), il y a n! certificats possibles (=nombre de permutations imaginables de n).

Question 3-2:

Je propose l'ordre croissant pour énumérer tous les certificats.
Par exemple, si le nombre de villes est 3, j'aurais les certificats dans l'ordre ci-dessous:
[0, 1, 2], [0, 2, 1], [1, 0, 2], [1, 2, 0], [2, 0, 1], [2, 1, 0]

Question 3-3:

Afin de tester si le problème a une solution, il faut générer à chaque itération le certificat suivant possible (s'il y en a un) et regarder s'il est valide. Si on en trouve un valide, on s'arrête car on sait alors que le problème a une solution. Sinon, si on a testé tous les cetificats possibles et qu'aucun n'est valide, alors on sait que le problème n'a pas de solution.
L'algorithme dans le pire des cas a une complexité de O(n!*n). Dans le meilleur des cas, il a une complexité de O(n).

Implémentation:

Pour lancer le programme, taper : python3 voyageur_commerce.py nomFichierDeDonnees mode longueurMaximale
L'argument mode doit être:

- soit verif pour vérifier un certificat de l'utilisateur
- soit nondet qui génère aléatoirement un certificat et le teste
- soit exhaust qui génére tous les certificats jusqu'à en trouver un valide, si il en existe un. S'il n'en existe pas (après avoir testé tous les certificats possibles) retourne False

II. Réductions polynomiales

Question 1:

Question 1-1:

1) Construction d'une instance de TSP à partir d'une instance du cycle hamiltonien
Soit I_H une instance du cycle hamiltonien définie par un graphe G à n_H sommets.
On construit une instance I_T de TSP de la manière suivante:

- soit $n_T = n_H$ le nombre de villes, on associe chaque ville à un sommet de G.
- Pour chaque couple de ville (i,j): si les 2 sommets sont voisins dans G, alors la valeur dans I_T vaut 1, sinon elle vaut 2.
- l vaut le nombre de villes.

Cette construction est polynomial en $|I_H|$.

2) Montrer que l'instance de TSP est positive si et seulement si l'instance du cycle hamitonien aussi
a) S'il existe un cycle hamiltonien dans G alors il existe aussi une manière de faire la tournée
Il existe un cycle hamiltonien dans G => D'après la construction de l'instance, cela signifie qu'il existe une manière de passer par toutes les villes dans TSP en ne passant que par les arêtes de valeur 1. On a donc bien la distance de la tournée <= n (= le nombre de villes).

b) Il existe une manière de faire une tournée dans I_T => Il existe un cycle hamiltonien dans G
Supposons qu'il existe une manière de faire une tournée dans I_T, cela signifie alors qu'il existe une permutation de n ne passant que par les arêtes de valeur 1. Pour chaque paire de villes dans la tournée possible, il existe une arête entre les 2 sommets associés dans G par construction de I_C à partir de I_H.
Donc il existe un cycle hamiltonien dans G.

3) Conclusion
Le cycle hamiltonien se réduit polynomialement dans TSP.

Question 1-2:

cf. fichier hamilton_cycle.py
Pour lancer le programme, taper : python3 hamilton_cycle.py nomFichierDeDonnees mode

Question 1-3:

Hamilton Cycle se réduit polynomialement en TSP et Hamilton Cycle est NP-complet donc TSP est NP-dur. De plus, on sait que TSP est dans NP (Q.1 de la partie I). Etant donné que TSP est dans NP et qu'il est NP-dur, on sait alors qu'il est NP-complet.

Question 1-4:

TSP ne se réduit pas polynomialement dans Hamilton Cycle. En effet, TSP est plus compliqué car il y a des longueurs (distances entre les villes et distance à ne pas dépasser).

Question 2:
Question 2-1:

1) Construction d'une instance du cycle hamiltonien à partir d'une instance du chemin hamiltonien
Soit I_P une instance du chemin hamiltonien définie par un graphe G à n_P sommets.
On construit une instance I_C du cycle hamiltonien de la manière suivante:

- soit $n_C = n_P$ on associe chaque sommet de G' à un sommet de G.
- Chaque arête dans G est dans G'.
- Ajout d'un sommet que l'on relie à tous les autres sommets.

Cette construction est polynomial en $|I_P|$.

2) Montrer que l'instance du cycle hamiltonien est positive si et seulement si l'instance du chemin hamitonien aussi
a) S'il existe un chemin hamiltonien dans G alors il existe aussi un cycle hamiltonien
S'il existe un chemin hamiltonien dans G, alors il suffit de prendre ce même chemin et d'ajouter un passage du noeud qu'on a ajouté lors de la construction de l'instance afin de relier le début et la fin du chemin. On obtient alors un cycle hamiltonien.

b) Il existe un cycle hamiltonien dans I_c => Il existe un chemin hamiltonien dans G
S'il existe un cycle hamiltonien, alors il passe une unique fois par le noeud ajouté lors de la construction. Il suffit alors de couper ce cycle avant et après ce noeud afin d'obtenir le chemin hamiltonien associé d'après la constuction de l'instance de I_C d'après I_P.

3) Conclusion
Le chemin hamiltonien se réduit polynomialement dans le cycle hamiltonien.

Question 2-2:

cf. fichier hamilton_path.py
Pour lancer le programme, taper : python3 hamilton_path.py nomFichierDeDonnees mode

Question 3:

On sait que HamiltonCycle se réduit polynomialement dans TSP (Q1.1). De plus, on sait que HamiltonPath se réduit polynomialement dans HamiltonCycle (Q.2.1). Donc, d'après la transitivité de la relation, on peut en déduire que HamiltonPath se réduit polynomialement dans TSP.

Question 5:

- Si on a une solution pour TSPOpt1, il suffit pour TSP de regarder si la valeur de TSPOpt1 est inférieure ou égale à la longueur maximale "autorisée". Si, oui, alors TSP retourne "oui", sinon "non". D'après ces informations, on peut en déduire que si TSPOpt1 était P, la propriété TSP le serait aussi. Ainsi, on sait que TSP se réduit polynomialement en TSPOpt1. De plus, TSP est NP-complet donc TSPOpt1 est NP-dur.
- Si on une solution pour TSPOpt2, pour trouver la solution pour TSP, il faut trouver la distance de la tournée grâce à la matrice des distances et la distance obtenue par TSPOpt2. Puis, dans un second temps, il faut comparer cette valeur avec la longueur maximale "autorisée". Si la valeur est inférieure ou égale à la distance maximale, alors TSP retourne "oui", sinon "non". D'après ces informations, on peut en déduire que si TSPOpt2 était P, la propriété TSP le serait aussi. Ainsi, on sait que TSP se réduit polynomialement en TSPOpt2. De plus, TSP est NP-complet donc TSPOpt2 est NP-dur.

Question 6:

Si TSP était P, TSPOpt1 le serait aussi. En effet, il suffirait alors d'appeler TSP plusieurs fois en incrémentant de 1 à chaque fois la valeur de la longueur maximale autorisée. Une fois que l'algorithme retourne "oui", il suffit pour TSPOpt1 de retourner la valeur l.