Exercice 12 :

I. Première variante avec l'algorithme Djikstra

1. Modélisation :
    a. Entrées : Un graphe G= (X, U). Avec X l'ensemble de sommets qui représentent les villes, |X|= N, et U l'ensemble d'arcs (Resp. Arêtes si tu graphe n'est pas orienté). Le seuil du cout K.

    Le graphe sera implémenté sous forme d'une matrice carrée M (NxN).

    $$M = \begin{cases} P_{ij}\ le\ poids\ du\ lien\ i,j\ si\ le\ lien\ existe \\ 0 \quad Sinon \end{cases}$$

    b. Sorties : Un chemin (Resp. Une chaine) qui démarre par le sommet 1 et qui doit impérativement inclure toute les villes du graphe avec un cout totale <= K.

    Implémentation : Un vecteur de villes de taille >= N. Toutes les villes du graphe doivent appartenir au chemin (la chaine). Le cout totale doit être <= K

    c. Traitement : En utilisant l'algorithme Djiikstra construire un chemin complet qui passe par toute les villes du graphe avec un cout minimal et <= K, par portion ou tranche. D'abord un premier point d'arrivée sera définit (Il peut s'agit de la ville N ou n'importe quelle autre ville non directement reliée à la ville de départ).
2. Algorithme de résolution de cette variante :

Début

Depart = 1 ; Arrivée = N ;

S : liste d'entiers ; // chemin correspondant à la solution

Liste_ville : Liste d'entiers ; // initialiser avec l'ensemble des villes du graphe utilisée également par Djikstra pour éliminer les villes une à une à chaque fois rajouter au chemin

Supprimer (Liste_ville, Depart) ; Supprimer (Liste_ville, Arrivée) ;

// pour ne pas les reconsidérer comme extrémité des sous chemins ;

S= Djikstra (Depart, Arrivée) ;

Tant que (non vide (Liste_ville))

Faire

    Depart = Arrivée ; Arrivée = Supprimer_Tete(Liste_ville) ;

    S=S + Djikstra (Depart, Arrivée) ;

Fait ;

Retourner (S) ;

Fin.

L'algorithme Djikstra est de l'ordre de $O(N^3)$ et au pire cas le graphe est considéré complet donc toutes les villes sont accessibles entre elles, donc à chaque itération de la boucle une seule ville uniquement sera rajoutée au chemin précédemment construit. Donc nombre d'itération <= N.

Donc la complexité est de l'ordre de $O(N^4)$.

L'algorithme de résolution de cette variante est polynomiale donc nous pouvons conclure que cette variante simplifiée du problème du voyageur de commerce est polynomiale (Appartient à la classe P).

II. Variante classique du problème du voyageur de commerce

1. Modélisation

a. Entrées : IDEM que la variante précédente

b. Sortie : Chemin (chaine) élémentaire de cout total <= K (ie : chemin Hamiltonien)

Vecteur de Villes V de dimension N puisque chaque ville doit apparaitre une et une seule fois.

$$\forall i \in [1, \quad N], \mathrm{V[i]} \in [1, \quad \mathrm{N}]$$

$$\forall i, j \in [1, \quad \mathrm{N}], i \neq j, V[i] \neq V[j]$$

$$\forall i \in [1, \quad \mathrm{N}\,[, \mathrm{M}\left[\mathrm{V[i]}, \mathrm{V[i+1]}\right] \neq 0$$

$$Cout\ (V) \leq K.$$

c. Traitement :

Explorer tous les chemins élémentaires possibles à partir du graphe et les évaluer par rapport au cout. Ce qui revient à construire et à parcourir l'arbre de tous les chemins possibles.

2. Algorithme de Résolution de cette variante du PVC

Début

Pile P ;

Structure nœud = ville, pos, cout : entier ;

V : Tableau d'entiers de dimension N

Nœud e, E1;

e. Ville = 1 ; e. Pos = 1 ; e.cout = 0 ;

Booléen trouve = faux ;

Int som ;

Empiler (depart, P) ;

```
Tant que (non vide (P) et trouve = faux)
Faire e = Depiler (P) ;
V[e.pos] = e. ville ;
Si e. pos = N alors
        si e.cout <= K alors trouve = vrai ; fsi ;
    Sinon j= N ;
        Tant que (j >= 1)
        Faire Si M[e.ville, j ] <> 0 alors E1.ville = j ;
                                          E1.pos = e.pos +1 ;
                                          E1.cout = e.cout + M [e.ville, j] ;
                                          Empiler (E1) ;
            Fsi ;
            j= j-1 ;
        Fait ;
Fait ;
Retourner (V) ;
Fin.
```

Au pire cas le graphe et complet et l'arbre de tous les chemins élémentaires possible est factorielle donc la complexité de l'algorithme de résolution du PVC est O (N !).
L'ordre de complexité étant factoriel la terminaison de l'algorithme et donc la résolution n'est pas garantie en un temps raisonnable pour toutes les instances possibles et pour tous les graphes, donc il ne s'agit pas d'une solution réalisable en pratique.

3. Algorithme de validation

```
Validation_PVC ( solution S' ) : Bouléen ;
Début
Valide = vrai ;
Int i = 1, cout = 0 ;
Tant (valide = vrai et i <= N)
Faire
    si (S'[i] < 1 ou S'[i] <N) alors valide = faux ;
    sinon j= i+1 ;
        Tant que (j<=N et valide = vrai)
        Faire
```

```
                    Si S'[i] = S'[j] alors valide = faux ;
                    Sinon j = j+ 1 ;
                    Fsi ;
            Fait ;
        Fsi ;
        i= i+1 ;
Fait ;
I=1 ;
Tant que (i < N et valide = vrai)
Faire
        Si M [V[i], V[i+1]] = 0 alors valide = faux ;
                                Sinon cout = cout + [V[i], V[i+1]] ;
        Fsi ;
        i= i +1 ;
Fait ;
Si (cout > K) alors valide = faux ; fsi ;
Retourner (valide) ;
Fin.
```

Au pire cas la solution est valide. La première boucle imbriquée qui vérifie l'unicité de la ville dans le parcours est quadratique et la seconde boucle qui vérifie la validité des liens entre chaque deux villes et calcul le cout est linéaire, donc l'ordre de complexité totale est quadratique ($O(N^2)$).

L'algorithme de validation est polynomiale ce qui signifie que le problème du voyageur de commerce noté PVC appartient à la classe NP et ne peut appartenir à la classe P puisque sa résolution est factorielle.