INFORMATIQUE TRONC COMMUN
DEVOIR SURVEILLÉ 3
Chemins optimaux dans un graphe temporel
Corrigé

Question 1 L'entrée est un graphe **pondéré par des poids positifs** (orienté ou non), et un sommet source s. En sortie, on obtient la distance de s à chaque sommet u (au sens du poids total minimum d'un chemin de s à u, ou $+\infty$ si u n'est pas accessible depuis s), et une représentation de ces plus courts chemins sous forme de tableau de prédécesseurs.

Question 2

```
def chemin_valide(chemin,u,v,t):
    n = len(chemin)
    if chemin == []:
        return u == v
    if u!= chemin[0][0] or t > chemin[0][2] or v!=chemin[n-1][1]:
        return False
    for i in range(n-1):
        u1, v1, td1, ta1 = chemin[i]
        u2, v2, td2, ta2 = chemin[i+1]
        if v1 != u2 or ta1 > td2:
            return False
    return True
```

Question 3

```
def meilleur_horaire(horaires, t):
    td = ta = float("inf")
    for L in horaires:
        t1, t2 = L
        if t1 >= t and t2 < ta:
            td = t1
            ta = t2
    return td, ta
```

Question 4

```
def meilleur_horaire(horaires, t):
    td = ta = float("inf")
    for L in horaires:
        if len(L) == 2: #trajet ponctuel
            t1, t2 = L
            if t1 >= t and t2 < ta:
                td = t1
                ta = t2
        elif len(L) == 1: #marche
            t2 = t + L[0]
            if t2 < ta:
                td = t
                ta = t2
        else: #trajet periodique
            premier_depart, premiere_arrivee, periode, dernier_depart = L
```

```
            if t <= dernier_depart:
                gap = (t - premier_depart)%periode
                if t < premier_depart:
                    t1 = premier_depart
                elif gap == 0:
                    t1 = t
                else:
                    t1 = t - gap + periode
                t2 = t1 + premiere_arrivee - premier_depart
                if t2 < ta:
                    td = t1
                    ta = t2
    return td, ta
```

Question 5

Itération	u	T	P
0	0	[0, 10, 5, inf, inf]	[None, (0, 3, 10), (0, 2, 5), None, None]
1	2	[0, 8, 5, inf, 21]	[None, (2, 6, 8), (0, 2, 5), None, (2, 14, 21)]
2	1	[0, 8, 5, 30, 20]	[None, (2, 6, 8), (0, 2, 5), (1, 10, 30), (1, 8, 20)]
3	4	[0, 8, 5, 30, 20]	[None, (2, 6, 8), (0, 2, 5), (1, 10, 30), (1, 8, 20)]
4	3	[0, 8, 5, 30, 20]	[None, (2, 6, 8), (0, 2, 5), (1, 10, 30), (1, 8, 20)]

Question 6 En partant du temps $t = 20$, le chemin arrivant le plus tôt en 3 depuis 0 est : `[(0, 2, 26, 29), (2, 4, 34, 41), (4, 3, 46, 54)]`, arrivant au temps 54. Le temps perdu par l'élève est donc $54 - 30 = 24$.

Question 7

```
def dijkstra_temporel(G, s, t_dep):
    n = len(G)
    marque = [False]*n
    T = [float("inf")]*n
    T[s] = t_dep
    P = [None]*n
    for _ in range(n):
        t_min = float("inf")
        for i in range(n):
            if not marque[i] and T[i] <= t_min:
                t_min = T[i]
                u = i
        marque[u] = True
        for v, horaires in G[u]:
            td, ta = meilleur_horaire(horaires, T[u])
            if ta < T[v]:
                T[v] = ta
                P[v] = (u, td, ta)
    return T, P
```

Question 8

```
def chemin(P, u):
    if P[u] == None:
        return []
    pred_u, td, ta = P[u]
    L = chemin(P, pred_u)
```

```
    L.append((pred_u, u, td, ta))
    return L
```

Question 9 Intuitivement, on part du sommet d'arrivée, et on procède à rebours, symétriquement à l'algorithme précédent. Il faut ainsi disposer d'une fonction `meilleur_horaire_de_depart` prenant en entrée une liste d'horaires et un temps d'**arrivée**, et renvoyant le plus grand temps de départ pour lequel il existe un horaire arrivant au plus tard au temps d'arrivée. On obtient alors l'algorithme suivant :

```
Fonction Dijkstra_depart_le_plus_tard(G, a, t_arr):
    T[a] vaut t_arr
    T[u] vaut -infini pour les autres sommets u
    les sommets n'ont initialement pas de père
    les sommets sont non marqués
    tant qu'il reste un sommet non marqué:
        soit u sommet non marqué de T[u] maximal
        on marque u
        pour chaque prédécesseur (dans le graphe) v de u:
            soit (td, ta) le meilleur horaire de départ pour aller de v à u arrivant avant T[u]
            Si td > T[v]:
                T[v] = td
                P[v] = (u, td, ta)
    on renvoie T, P
```

Pour accéder efficacement aux prédécesseurs d'un sommet donné, il faudrait commencer par calculer le transposé du graphe, c'est-à-dire de passer des listes d'adjacences, qui sont des listes de successeurs, à des listes de prédécesseurs.

Question 10 La démonstration est essentiellement similaire à la correction de l'algorithme de Dijkstra classique.

On fixe le sommet de départ s et le temps de départ t. Pour un sommet u, on note δ_u le temps d'arrivée d'un chemin de s à u à partir du temps t arrivant le plus tôt (ou $+\infty$ s'il n'en existe pas).

Pour un sommet u, `T[u]` est infini ou correspond au temps d'arrivée d'un chemin de s à u à partir du temps t, donc $\delta_u \leq$ `T[u]`. De plus, `T[u]` ne peut que décroitre au fil de l'exécution.

On démontre que la proposition suivante est un invariant de la boucle `while` : pour tout sommet u, si u est marqué, `T[u]` $= \delta_u$ depuis ce marquage.

Initialement, aucun sommet n'est marqué, donc la proposition est vraie.

Au cours de la première itération, s est marqué, on a alors `T[s]` $= t = \delta_s$ (le chemin arrivant le plus tôt est le chemin vide).

On considère une autre itération, et on suppose que la proposition est vraie au début. Soit u le sommet qui va être marqué lors de cette itération.

S'il n'existe pas de chemin de s à u à partir du temps t, alors $\delta_u = +\infty \leq$ `T[u]` d'où $\delta_u =$ `T[u]`.

S'il existe un chemin de s à u à partir du temps t, alors parmi ceux qui ne contiennent pas de cycle ni d'attente avant une marche, il y en a un de temps d'arrivée minimum (car c'est un ensemble fini), et c'est bien un chemin de s à u à partir du temps t arrivant le plus tôt (car ajouter des cycles ou de l'attente ne peut qu'augmenter le temps d'arrivée).

Dans ce chemin, on note y le premier sommet non marqué (qui existe car le chemin contient au moins u non marqué) et x son prédécesseur (qui existe car le premier sommet, s, est marqué).

D'après l'hypothèse d'invariance, `T[x]` $= \delta_x$ depuis que x est marqué. De plus, quand x a été marqué, le meilleur horaire t_y pour aller de x à y à partir de δ_x a été calculé, et attribué à `T[y]` s'il était inférieur, d'où `T[y]` $\leq t_y$ (ce qui a été conservé depuis puisque `T[y]` ne peut que décroître).

Par définition du meilleur horaire, le chemin considéré ne peut pas arriver avant t_y en y, or il arrive en δ_u en u, donc $t_y \leq \delta_u$.

Finalement, lorsque u est marqué, c'est le sommet non marqué de valeur minimale dans `T`, d'où `T[u]` $\leq$ `T[y]`. On a alors la série d'inégalités :

$$\texttt{T[u]} \leq \texttt{T[y]} \leq t_y \leq \delta_u \leq \texttt{T[u]}$$

d'où `T[u]` $= \delta_u$.