Partie II :

Q1.

Pour montrer que les deux problèmes (P1) et (P2) sont équivalents, on peut définir le coût de l'essence au sommet s' comme étant égal au coût de l'essence au sommet s, c(s'). La distance d(s', s) est définie comme étant égale à la quantité d'essence μs contenue dans le réservoir au départ de s dans le problème (P1). Ainsi, dans le problème (P1), le trajet peut être considéré comme commençant au nœud s', avec un réservoir plein, et se terminant au nœud t. Dans le problème (P2), le trajet peut être considéré comme commençant au nœud s, avec un réservoir vide, et se terminant au nœud t. Le nombre d'arrêts dans le problème (P1) est égal au nombre d'arrêts dans le problème (P2) plus un, car dans le problème (P1), il y a un arrêt supplémentaire au nœud s' pour remplir le réservoir. En résumé, en ajoutant le nœud s' relié à s et en définissant le coût de l'essence au sommet s' comme étant égal au coût de l'essence au sommet s, on peut montrer que les deux problèmes (P1) et (P2) sont équivalents, et que le nombre d'arrêts dans le problème (P1) est égal au nombre d'arrêts dans le problème (P2) plus un.

Q2.

Pour montrer que la stratégie optimale consiste à arriver à destination t avec un réservoir vide, on peut utiliser un raisonnement par l'absurde. Supposons qu'il existe une stratégie optimale qui consiste à arriver à destination t avec un réservoir non vide. Si la quantité d'essence dans le réservoir est supérieure à 0, cela signifie qu'il est possible de continuer à parcourir une certaine distance avec cette essence même après être arrivé à destination t. Ce qui revient à dire qu'il est possible de réduire le coût total du voyage en utilisant cette essence restante. Or si cette essence pourrait servir à rendre le voyage moins chère cela veut dire que cette stratégie n’est pas optimal. La seule possibilité restante est donc que la stratégie optimale consiste à arriver à destination t avec un réservoir vide. En conclusion, la stratégie optimale consiste à arriver à destination t avec un réservoir vide.

Q3.

C[u, q, g] = Coût minimum du voyage pour partir de u à t en utilisant q arrêts pour remplir le réservoir, sachant qu’on arrive à u avec g unités d'essence (km) et u est l'un des q arrêts

Cout de la solution optimale = C[t, Q, 0]

- u doit être égal à t, car c'est la destination finale du voyage.
- Q est le nombre maximal d’arrêts autorisés.
- et g doit être égal à 0, car la stratégie optimale consiste à arriver à destination t avec un réservoir vide.

Q4.

C[u, q, g] = min { C[v, q-1, h] + c(u) * (U-h) + cost(u, v) } pour tous les sommets v atteignables depuis u avec une quantité d'essence h dans le réservoir (h ≤ U).

C[v, q-1, h] représente le coût minimum du voyage pour partir de v à t en utilisant q-1 arrêts pour remplir le réservoir, sachant qu'on arrive à v avec h unités d'essence (km). c(u) est le coût de l'essence au sommet u et cost(u, v) est le coût associé à la distance entre les sommets u et v. U est la capacité du réservoir, c'est-à-dire la distance totale que peut parcourir le véhicule avec un réservoir plein.

Q5.

Pour initialiser la fonction coût C[u, q, g], nous pouvons utiliser les valeurs suivantes : u = s (sommet de départ), alors C[s, q, g] = 0 pour tous les entiers q et g. Cette initialisation correspond au cas où nous sommes au sommet de départ et n'avons pas encore entamé notre voyage. On peut également mettre g = 0, c'est ce que nous avons déterminer à la Q1.

Q6.

La complexité dépend de plusieurs facteurs, ici cela dépend du nombre de sommets V, de son cardinal noté K, U la capacité maximum du réservoir et q le nombre d'arrêts autorisés. On a donc: O(V * K * U * q)

Q7.

La complexité de ce programme dynamique dépend toujours du nombre de sommets du graphe V et du nombre maximal d'arrêts pour remplir le réservoir q. Si V et q sont des entiers de grande taille, la complexité de notre programme dynamique peut être encore considérée comme pseudo-polynomiale.

Par exemple, si V = 10000 et q = 1000, la complexité de notre programme dynamique sera en O(V * K * U* q / X) = O(10^9), ce qui est encore considéré comme pseudo-polynomial.

Q8.

Pour montrer que la stratégie proposée est optimale pour décider de la quantité d'essence à remplir à chaque arrêt, nous allons utiliser une démonstration par l'absurde. Nous allons supposer que la stratégie proposée n'est pas optimale et montrer que cela contredit l'hypothèse selon laquelle la solution optimale utilise au plus k arrêts.

Commençons par l'arrêt ul, où nous devons remplir juste assez d'essence pour atteindre t avec un réservoir vide. Si nous remplissons plus d'essence que ce qui est nécessaire, cela signifie que nous avons acheté de l'essence à un coût supérieur à ce qui était nécessaire. Cela implique que nous aurons un coût total plus élevé que la solution optimale, ce qui contredit l'hypothèse selon laquelle la solution optimale utilise au plus k arrêts.

Passons maintenant aux arrêts précédents ul-1, ul-2, etc. Pour chaque arrêt uj (j < l), nous avons deux possibilités :

- Si c(uj) < c(uj+1), nous devons faire le plein à uj pour minimiser le coût total du voyage. Cela est logique car nous avons la possibilité d'acheter de l'essence à un coût inférieur à celui que nous devrons payer à l'arrêt suivant
- c(uj) ≥ c(uj+1). Dans ce cas, nous devons mettre juste assez d'essence pour atteindre uj+1. Si nous mettons plus d'essence que ce qui est nécessaire, cela signifie que nous avons acheté de l'essence à un coût supérieur à ce qui était nécessaire. Cela implique que nous aurons un coût total plus élevé que la solution optimale, ce qui contredit l'hypothèse selon laquelle la solution optimale utilise au plus k arrêts.

En résumé, nous avons montré que, dans tous les cas, la stratégie proposée est optimale pour décider de la quantité d'essence à remplir à chaque arrêt. Autrement, cela entraînerait un coût total plus élevé que la solution optimale, ce qui contredit l'hypothèse selon laquelle la solution optimale utilise au plus k arrêts. Par

conséquent, nous pouvons conclure que la stratégie proposée est bien optimale pour décider de la quantité d'essence à remplir à chaque arrêt.

Q9.

La formule d'initialisation de la récurrence C[u, q, g] permet de définir la valeur de la fonction coût pour le cas où q = 1, c'est-à-dire lorsqu'il n'y a qu'un seul arrêt de remplissage. Dans ce cas, si g est inférieur ou égal à la distance entre u et t et inférieur ou égal à la capacité du réservoir U, alors la valeur de C[u, q, g] est égale à la distance entre u et t moins g multipliée par le coût de l'essence au nœud u. Sinon, si g est supérieur à la distance entre u et t ou supérieur à la capacité du réservoir U, la valeur de C[u, q, g] est définie comme étant infinie, car il n'est pas possible de parcourir une distance supérieure à la capacité du réservoir ou d'atteindre une destination située à une distance supérieure à la distance restante dans le réservoir. La formule de récurrence de la fonction coût C[u, q, g] permet de définir la valeur de la fonction pour le cas général où q est supérieur à 1, c'est-à-dire lorsqu'il y a plus d'un arrêt de remplissage. Dans ce cas, la valeur de C[u, q, g] est définie comme étant le minimum des coûts pour atteindre u à partir de tous les autres nœuds v de manière à utiliser q-1 arrêts de remplissage. Pour chaque nœud v, il y a deux possibilités :

- Si le coût de l'essence au nœud v est inférieur ou égal au coût de l'essence au nœud u, alors nous devons arriver à u avec un réservoir vide en empruntant le plus court chemin entre u et v. Dans ce cas, la valeur de C[u, q, g] est définie comme étant la valeur de C[v, q-1, 0] plus la distance entre u et v moins g multipliée par le coût de l'essence au nœud u.
- Si le coût de l'essence au nœud v est supérieur au coût de l'essence au nœud u, alors nous devons arriver à u avec la capacité maximale du réservoir moins la distance entre u et v. Dans ce cas, la valeur de C[u, q, g] est définie comme étant la valeur de C[v, q-1, U-d(u,v)] plus la capacité maximale du réservoir moins g multipliée par le coût de l'essence au nœud u.

Q10.

GV(s) = [0]
GV(1) = [0, 24, 28, 1, 5, 7]
GV(2) = [0, 22, 5, 6]
GV(3) = [0, 17, 26]
GV(4) = [0, 3, 19, 7, 0, 26, 11]
GV(5) = [0, 6, 22, 21, 14, 26, 28, 24, 25]
GV(6) = [0, 10, 2, 16, 14, 12, 28]
GV(7) = [0, 28]
GV(8) = [0]
GV(t) = [0, 11, 9, 16, 25, 28]

Q11.

(Question non traitée).

Q12.

On considère que la fonction d(w, u) qui renvoit la distance du plus court chemin entre w et u. On considère un graphe à n sommets, et la fonction getPrix(sommet) avec sommet ∈ 0..n, qui renvoit le prix de l'essence d'un sommet du graphe (d'une station service). Et U la capacité du réservoir. Ici on définit donx l'algorithme

d'une fonction C(u, q, g) qui renvoit le coût minimum d'un voyage de u à t, en utilisant q arrêts pour remplir le réservoir, en sachant que l'on arrive avec g unité d'essence.

```
Si q = 1 alors
    distance ← d(u, t)
    Si g ⩽ distance ⩽ U alors
        retourner ((distance - g) * getPrix(u))
    Sinon
        retourner +∞
    Finsi
Sinon
    min ← +∞
    c ← +∞
    Pour sommet allant de 0 à n-1 faire
        distance ← d(u, v)
        Si sommet ≠ u et d ⩽ U alors
            Si getPrix(sommet) ⩽ getPrix(u) et g ⩽ distance alors
                c ← C(v, q-1, 0) + (distance - g) * getPrix(u)
            Sinon
Si prix[v] > prix[u] alors
                    c ← C(v, q-1, U - distance) + (U - g)* getPrix(u)
                FSi
            FSi
            Si c < min alors
                min ← c
            FSi
    FSi
    FPour
    Retourner min
FSi
```

Ensuite on lance donc C(0, X, 0), avec X le nombre d'arrêts autorisés, puis X-1 etc jusqu'à 0. On prends donc la valeur optimal à la fin des différentes éxécutions.

Q13.

Voici l'implémentation en Python :

```python
import math
from collections import deque

# Constantes de mon programme
U = 40  # Réservoir
NB_ARRETS_AUTORISE = 3  # Nombre d'arrêts autorisés
# Prix de chacunes des stations, de s à t, on mets t à +inf car il est impossible
d'y faire le plein.
PRICES = [10, 26, 19, 22, 29, 39, 37, 11, 10, math.inf]

# Matrice d'adjacence représentant le graphe, avec 0 pour dire qu'il n'y a pas
d'arête.
```

```
GRAPH = [
    [0, 16, 18, 23, 0, 0, 0, 0, 0, 0],
    [16, 0, 12, 0, 21, 18, 0, 0, 0, 0],
    [18, 12, 0, 14, 0, 19, 0, 0, 0, 0],
    [23, 0, 14, 0, 0, 26, 24, 0, 0, 0],
    [0, 21, 0, 0, 0, 14, 0, 14, 0, 0],
    [0, 18, 19, 26, 14, 0, 12, 16, 15, 0],
    [0, 0, 0, 24, 0, 12, 0, 0, 12, 0],
    [0, 0, 0, 0, 14, 16, 0, 0, 12, 15],
    [0, 0, 0, 0, 0, 15, 12, 12, 0, 12],
    [0, 0, 0, 0, 0, 0, 0, 15, 12, 0],
]

# Implémentation de Djikstra
def d(w, u):
    process, result = deque(), deque()
    parent, distance, visit = [-1] * \
        len(GRAPH), [math.inf] * len(GRAPH), [False] * len(GRAPH)
    distance[w], visit[w] = 0, True
    process.append(w)
    while process:
        node = process.popleft()
        shortest = distance[process[0]] if process else 0
        if node == u:
            result.append(u)
            back = parent[u]
            while back != -1:
                result.append(back)
                back = parent[back]
            return distance[u]

        for n, d in enumerate(GRAPH[node]):
            if d and not visit[n]:
                distance[n], visit[n], parent[n] = distance[node] + \
                    d, True, node
                if distance[n] <= shortest:
                    process.appendleft(n)
                else:
                    process.append(n)
    return 0


# Implémentation de C
def C(u, q, g):
    # Implémentation de l'initialisation
    if q == 1:
        dist = d(u, 9)
        if g <= dist <= U:
            return (dist - g) * PRICES[u], [u]
        else:
            return math.inf, []
    # Implémentation de la récurrence
    else:
        min = math.inf
```

```
            c = math.inf
            result = []
            for vertice in range(len(GRAPH)):
                dist = d(u, vertice)
                if vertice != u and dist <= U:
                    c, path = C(vertice, q-1, 0)
                    c += (dist - g) * PRICES[u]
                    path = [u] + path
                    if PRICES[vertice] > PRICES[u]:
                        c, path = C(vertice, q-1, U-dist)
                        c += (U - g) * PRICES[u]
                        path = [u] + path
                    if c < min:
                        min = c
                        result = path
            return min, result


    def solve():
        res, path = C(0, NB_ARRETS_AUTORISE, 0)
        print("La solution optimale de s à t avec au plus 3 arrêts est de " +
              str(res/100) + " €")
        print("Liste des arrêts :", path)


    solve()
```

Q14.

On obtient la sortie suivante de notre programme avec NB_ARRETS_AUTORISE = 1 :
La solution optimale de s à t avec au plus 1 arrêt est de inf €
Liste des arrêts : []
On obtient la sortie suivante de notre programme avec NB_ARRETS_AUTORISE = 2 :
La solution optimale de s à t avec au plus 2 arrêts est de 11.54 €
Liste des arrêts : [0, 4]
On obtient la sortie suivante de notre programme avec NB_ARRETS_AUTORISE = 3 :
La solution optimale de s à t avec au plus 3 arrêts est de 7.48 €
Liste des arrêts : [0, 2, 8]
On obtient la sortie suivante de notre programme avec NB_ARRETS_AUTORISE = 4 :
La solution optimale de s à t avec au plus 4 arrêts est de 6.28 €
Liste des arrêts : [0, 2, 5, 8]

Voici les détails de la solution optimale en 3 arrêts au plus :
Coût de la solution : 7.48€.
Itinéraire : 0 -> 2 -> 5 -> 8.
Les arrêts aux sommets (stations-service) : 0, 2 et 8.
Quantité remplies à chaque arrêts respectivement à la liste du dessus (0, 2 et 8) : 40, 12, 12

Q15.

La fonction C est de compléxité O(n), et utilise elle même Djikstra, Djikstra étant lui aussi de compléxité polynomial, mon programme est donc de compléxité $O(n^2)$.