INFORMATIQUE TRONC COMMUN
DEVOIR SURVEILLÉ 2 (sujet B)
Corrigé

1. On considère le graphe pondéré sous forme de listes d'adjacences
   `GB = [[(5,1)],[(2,1),(5,5)],[(0,2)],[(0,4),(5,6),(1,2)],[(2,1),(1,8)],[(1,2)]]`
   Représenter graphiquement `GA`.
   Corrigé : Je ne m'embête pas à le tracer au propre. Le premier couple, (5,1), est dans la liste d'indice 0, c'est-à-dire la liste d'adjacence de 0. Il faut donc dessiner un arc de 0 à 5, et indiquer à côté son poids de 1. On fait de même avec les autres arcs.
2. Donner sans justifier la distance et un plus court chemin dans `GB` pour aller du sommet 4 au sommet 1.
   Corrigé : Cette distance est de 6, réalisée par le chemin 4-2-0-5-1.
3. Appliquer à la main l'algorithme de Dijkstra sur le graphe `GB` et le sommet source 3. On précisera soigneusement à chaque étape quel est le sommet traité, et quels sont les valeurs actuellement calculées des distances et prédécesseurs.
   Corrigé :
   On commence par traiter 3, on a alors

   ```
   D = [4, 2, inf, 0, inf, 6]
   P = [3, 3, None, None, None, 3]
   ```

   On traite ensuite 1 (sommet non marqué de distance D minimale)

   ```
   D = [4, 2, 3, 0, inf, 6]
   P = [3, 3, 1, None, None, 3]
   ```

   On traite ensuite 2 (sommet non marqué de distance D minimale)

   ```
   D = [4, 2, 3, 0, inf, 6]
   P = [3, 3, 1, None, None, 3]
   ```

   On traite ensuite 0 (sommet non marqué de distance D minimale)

   ```
   D = [4, 2, 3, 0, inf, 5]
   P = [3, 3, 1, None, None, 0]
   ```

   On traite ensuite 5 (sommet non marqué de distance D minimale)

   ```
   D = [4, 2, 3, 0, inf, 5]
   P = [3, 3, 1, None, None, 0]
   ```

   On traite enfin 4 (on peut sauter cette étape car on a la garantie que toutes les distances sont déjà correctes)

   ```
   D = [4, 2, 3, 0, inf, 5]
   P = [3, 3, 1, None, None, 0]
   ```

4. Écrire une fonction `dijkstra` prenant en argument un graphe pondéré (sous forme de listes d'adjacence) et un sommet source et implémentant l'algorithme de Dijkstra.
   Corrigé :

   ```
   def dijkstra(G,s):
       n = len(G)
       D = [float("inf")]*n
       D[s] = 0
       P = [None]*n
       marque = [False]*n
   ```

```
    for _ in range(n):
        min = float("inf")
        for i in range(n):
            if not marque[i] and D[i] <= min:
                min = D[i]
                u = i
        marque[u] = True
        for v,w in G[u]:
            d2 = D[u] + w
            if d2 < D[v]:
                D[v] = d2
                P[v] = u
    return D,P
```

5. Déterminer, en la justifiant, la complexité de cet algorithme, en fonction du nombre de sommets n.

   Corrigé :

   Les initialisations sont en $O(n)$.

   On fait n passages dans la boucle principale. Pour chaque passage, on effectue une recherche de minimum en $O(n)$ et un parcours de `G[u]` en $O(n)$. La complexité totale est donc en $O(n^2)$.

6. Quelle condition doit-on supposer sur le graphe pour avoir la garantie que l'algorithme de Dijkstra est correct ?

   Corrigé : les poids doivent être positifs.

7. Donner un exemple de graphe pondéré, ne respectant donc pas cette condition, dans lequel l'algorithme de Dijkstra calcule une distance incorrecte entre deux sommets. On précisera clairement quelle distance est calculée par l'algorithme, et quel est le résultat correct (qui devra être bien défini).

   Corrigé : On considère `G = [[(1,2),(2,1)],[(2,-10)],[(3,0)],[]]`. On remarque alors que le plus court chemin de 0 à 3 est 0-1-2-3, de poids -8, mais l'algorithme de Dijkstra calcule une distance de 1 entre 0 et 3.

8. Donner l'invariant de la boucle principale de l'algorithme de Dijkstra justifiant que cet algorithme est correct (on ne demande pas la démonstration du fait que c'est bien un invariant de boucle).

   Corrigé : pour tout sommet u marqué, on a `D[u]` $= \delta(s,u)$, autrement dit la distance calculée pour u est correcte.

9. Expliquer à l'aide du résultat de la question 3 comment retrouver un plus court chemin du sommet source s à un sommet u à partir du tableau de prédécesseurs renvoyé par `dijkstra`.

   Corrigé :

   Le tableau de prédécesseur obtenu à la question 3 est

   ```
   P = [3, 3, 1, None, None, 0]
   ```

   Pour obtenir le plus court chemin allant du sommet source 3 au sommet 2, on construit le chemin de droite à gauche : 2 a pour prédécesseur 1, qui a pour prédécesseur 3. On obtient donc le chemin 3-1-2.

10. Écrire une fonction `chemin` prenant en argument un tableau de prédécesseurs et un sommet u, et calculant un tel plus court chemin du sommet source à u, sous forme de liste de sommets.

    Corrigé :

    Solution récursive :

    ```
    def chemin(pred,u):
        if pred[u] == None:
            return [u]
    ```

```
    L = chemin(pred,pred[u])
    L.append(u)
    return L
```

Solution itérative :

```
def chemin(pred,u):
    L = [u]
    while pred[u] != None:
        L.append(pred[u])
        u = pred[u]
    #Il faut retourner L
    for i in range(len(L)//2):
        L[i], L[len(L)-1-i] = L[len(L)-1-i], L[i]
    return L
```

11. Écrire une fonction `poids_chemin` prenant en argument un chemin (sous forme de liste de sommets) et un graphe pondéré (sous forme de listes d'adjacences) et renvoyant le poids de ce chemin dans le graphe.

    Corrigé :

```
def poids_chemin(C, G):
    p = 0
    k = len(C)
    for i in range(k-1):
        a = C[i]
        b = C[i+1]
        for c,w in G[a]:
            if c==b:
                p = p + w
                break
    return p
```

12. Représenter le graphe `GB` sous forme de matrice d'adjacence.

    Corrigé :

    On choisit d'écrire `None` dans une case de la matrice correspondant à une absence d'arc (on ne peut pas mettre 0, qui correspondrait à un arc de poids 0). On obtient :

```
[[None, None, None, None, None, 1],
[None, None, 1, None, None, 5],
[2, None, None, None, None, None],
[4, 2, None, None, None, 6],
[None, 8, 1, None, None, None],
[None, 2, None, None, None, None]]
```

13. Préciser clairement quelles modifications doivent être apportées à la fonction `dijkstra` pour lui faire prendre en argument un graphe représenté par matrice d'adjacence plutôt que par listes d'adjacences.

    Corrigé :

```
def dijkstra2(G,s):
    n = len(G)
    D = [float("inf")]*n
    D[s] = 0
    P = [None]*n
    marque = [False]*n
    for _ in range(n):
        min = float("inf")
        for i in range(n):
```

```
            if not marque[i] and D[i] <= min:
                min = D[i]
                u = i
        marque[u] = True
        #DEBUT PARTIE MODIFIEE
        for v in range(n):
            w = G[u][v]
            if w != None:
                d2 = D[u] + w
                if d2 < D[v]:
                    D[v] = d2
                    P[v] = u
        #FIN PARTIE MODIFIEE
    return D,P
```

14. Déterminer la complexité de la fonction ainsi modifiée.

    Corrigé :

    La complexité n'est pas modifiée. On a seulement changé le parcours des successeurs de u, qui reste en $O(n)$. La complexité totale est toujours en $O(n^2)$.

15. On considère une grille de dimension 10×10 dans laquelle chaque case peut être vide, ou contenir un pion blanc, ou contenir un pion noir. Une telle grille sera représenté en Python par une matrice dont les cases prennent les valeurs `None` (vide), 0 (blanc) ou 1 (noir). On cherche à déterminer le nombre minimum de pions noirs à ajouter sur les cases vides de la grille pour qu'il existe un chemin ininterrompu de pions noirs (chaque pion étant dans une case partageant une arête avec celle de son successeur) partant d'une case du bord supérieur à une case du bord inférieur de la grille.

    (a) Expliquer clairement comment formuler ce problème comme un calcul de plus court chemin dans un graphe à déterminer.

    Corrigé :

    On considère un graphe dont les sommets sont les cases de la matrice, ainsi qu'un sommet au-dessus du bord supérieur et un sommet en-dessous du bord inférieur. Ce graphe contient un arc de la case u à la case v si u et v sont adjacentes, et que v ne contient pas de pion noir. Le poids de cet arc est 1 si v est vide, et 0 si v contient un pion blanc. De la même façon, on relie par un arc le sommet au-dessus du bord supérieur à chaque case du bord supérieur ne contenant pas de pion noir. Chaque case du bord inférieur est reliée par un arc de poids 0 au somment en-dessous.

    Par construction, un chemin du sommet au-dessus du bord supérieur au sommet en-dessous du bord inférieur correspond à un chemin dans la matrice allant du bord supérieur au bord inférieur et ne passant jamais par un pion noir. De plus, le poids du chemin correspond au nombre de cases vides traversées (car le poids pour s'y déplacer est 1), donc au nombre de pions blancs qu'il faut ajouter pour compléter ce chemin. Le problème se ramène donc à une recherche de plus court chemin dans ce graphe.

    (b) Écrire une fonction `nb_min_pions_blancs` prenant en argument une telle grille, et implémentant cette méthode. La fonction renverra `None` s'il n'est plus possible de relier le bord supérieur au bord inférieur avec des pions blancs.

    Corrigé :

    On peut procéder de différentes manières, ici on va construire explicitement le graphe sous forme de listes d'adjacences, puis appeler `dijkstra` dessus. On associe à la case (i, j) le sommet numéro $10i + j$, le bord supérieur sera le sommet 100 et le bord inférieur sera le sommet 101.

    La fonction auxiliaire `ajoute_arc` ajoute un arc entre u et v dans G, selon le contenu de la case correspondant à v.

    ```
    def ajoute_arc(G, M, u, v):
        iv = v//10
    ```

```
    jv = v%10
    if M[iv][jv] == None:
        G[u].append((v,1))
    elif M[iv][jv] == 0:
        G[u].append((v,0))

def nb_min_pions_blancs(M):
    G = [[] for _ in range(102)]
    for i in range(10):
        for j in range(10):
            u = 10*i+j
            if i>0:
                v = 10*(i-1) + j
                ajoute_arc(G,M,u,v)
            if j>0:
                v = 10*(i) + j-1
                ajoute_arc(G,M,u,v)
            if i < 9:
                v = 10*(i+1) + j
                ajoute_arc(G,M,u,v)
            if j < 9:
                v = 10*(i) + j+1
                ajoute_arc(G,M,u,v)
    for j in range(10):
        ajoute_arc(G, M, 100, j)
        G[90+j].append((101,0))
    D, P = dijkstra(G, 100)
    if D[101] != float("inf"):
        return D[101]
    else:
        return None
```