TP 25- Proposition de solutions

Solution 1 Algorithme de Dijkstra

Remarque – L'algorithme de Dijkstra ne fonctionne que sur un graphe pondéré.

```
def voisins(s,M):
    L=[]
    for i in range(len(M)):
        if M[s,i]<np.inf:
            L.append(i)
    return L

def dmin(L,D):
    p=L[0]
    for e in L[1:]:
        if D[e][0]<D[p][0]:
            p=e
    return p

def Dijkstra(s,M):
    D={s:(0,None)}
    SaV=[s]
    while len(SaV)>0:
        p=dmin(SaV,D)
        SaV.remove(p)
        V=voisins(p,M)
        for v in V:
            r=D[p][0]+M[p,v]
            if not(v in D):
                SaV.append(v)
                D[v]=(r,p)
            elif r<D[v][0]:
                D[v]=(r,p)
    return D

def chemin(s,a,M):
    D=Dijkstra(s,M)
    L=[a]
    while D[L[0]][1]!=None:
        L=[D[L[0]][1]]+L
    return L,D[a][0]
```

L'intérêt de travailler à partir du sommet `p`, le plus proche de `s` dans `SaV`, permet d'éviter de parcourir tout l'arbre pour mettre à jour les poids dans le cas où l'on modifie dans `D` la valeur d'une clé `v`.
En effet, si l'on modifie la valeur d'une clé `v` (distance au sommet d'origine, antécédent), alors il faudrait effectuer une mise à jours de tous les sommets atteignables depuis `s` en passant par `v`. Or, en travaillant à partir du sommet `p` le plus proche, il s'avère que les sommets potentiellement à mettre à jour seront traiter ultérieurement car ils sont encore dans `SaV`.