

Corrigé du TP Informatique 04

Exercice 1

1. On saisit :

```
def dijkstra(S,A,s0):
    cout,predec,djvu={},{},{}
    for s in S:
        cout[s]=np.inf
        djvu[s]=False
    cout[s0]=0
    for _ in S:
        smin,cmin=s0,np.inf
        for s in S:
            if not djvu[s] and cout[s]<cmin:
                smin,cmin=s,cout[s]
        djvu[smin]=True
        for s,nu in A[smin]:
            if not djvu[s] and cout[s]>cmin+nu:
                cout[s]=cmin+nu
                predec[s]=smin
    return cout,predec
```

2. On obtient :

```
>>> dijkstra(S,A,0)
{0: 0, 1: 7, 2: 5, 3: 10, 4: 14, 5: 12},
{1: 2, 2: 0, 3: 1, 4: 5, 5: 3}
```

Exercice 2

1. On saisit :

```
def d2(A,B):
    xA,yA=A
    xB,yB=B
    return np.sqrt((xA-xB)**2+(yA-yB)**2)
```

2. On saisit :

```
def Astar_deb_fin(S,A,deb,fin):
    nb=0
    def h(s):
        return d2(s,fin)
    cout,dist,predec,djvu={},{},{},{}
    for s in S:
```

```

    cout[s]=np.inf
    dist[s]=np.inf
    djvu[s]=False
cout[deb]=h(deb)
dist[deb]=0
while not djvu[fin]:
    nb+=1
    # recherche elt à cout minimal
    smin,cmin=deb,np.inf
    for s in S:
        if not djvu[s] and cout[s]<cmin:
            smin,cmin=s,cout[s]
    djvu[smin]=True
    # relachement chez les successeurs
    for s,nu in A[smin]:
        if not djvu[s] and dist[s]>dist[smin]+nu:
            dist[s]=dist[smin]+nu
            cout[s]=dist[s]+h(s)
            predec[s]=smin
    return cout,predec,nb

```

3. On saisit :

```

def dijkstra_deb_fin(S,A,deb,fin):
    nb=0
    cout,predec,djvu={},{},{}
    for s in S:
        cout[s]=np.inf
        djvu[s]=False
    cout[deb]=0
    while not djvu[fin]:
        nb+=1
        smin,cmin=deb,np.inf
        for s in S:
            if not djvu[s] and cout[s]<cmin:
                smin,cmin=s,cout[s]
        djvu[smin]=True
        for s,nu in A[smin]:
            if not djvu[s] and cout[s]>cout[smin]+nu:
                cout[s]=cout[smin]+nu
                predec[s]=smin
    return cout,predec,nb

```

4 Avec un départ en (2,0) par exemple, on obtient

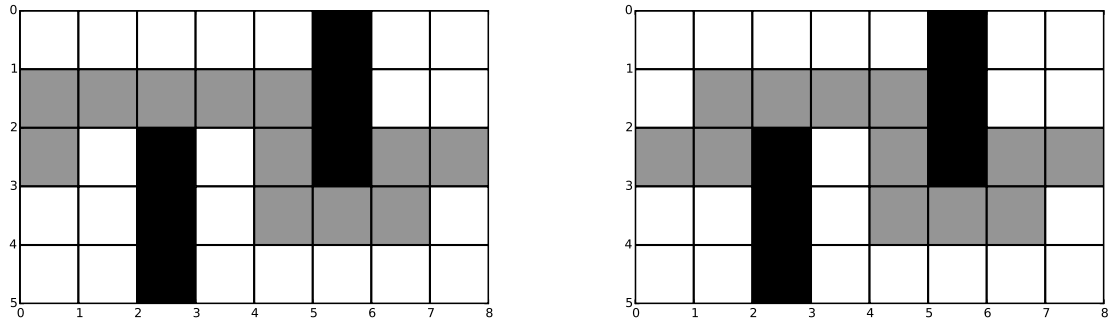


FIGURE 1 – Plus court chemin avec Dijkstra et A★

avec

Dijkstra - nb= 30
A* - nb= 24

Avec un départ en (0,4) par exemple, on obtient

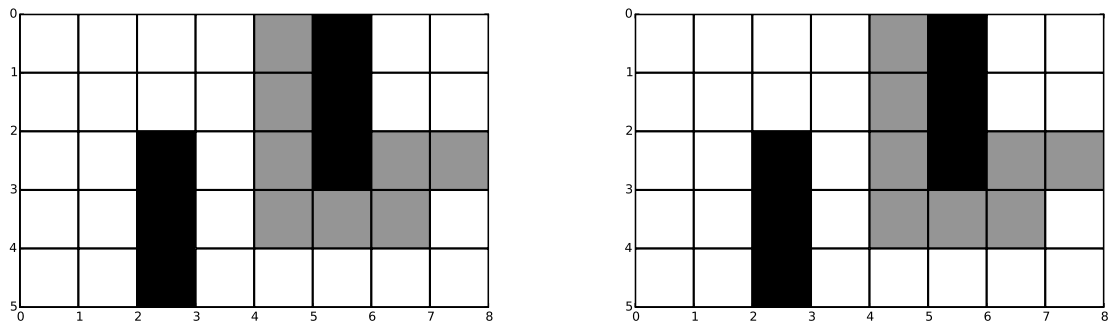


FIGURE 2 – Plus court chemin avec Dijkstra et A★

avec

Dijkstra - nb= 27
A* - nb= 11

Avec ce choix de départ en (0,4), on constate la très nette amélioration de l'algorithme A★ vis-à-vis de l'algorithme de Dijkstra.

Exercice 3

1. On saisit :

```
def BF(S,A,s0):
    cout,predec={},{}
    for s in S:
        cout[s]=np.inf
    cout[s0]=0
    for _ in range(1,len(S)):
        # pour chaque arc
        for x in A:
            for y,nu in A[x]:
                # on relâche
                if cout[y]>cout[x]+nu:
                    cout[y]=cout[x]+nu
                    predec[y]=x
    return cout,predec
```

2. Sur le graphe 1, on obtient :

```
>>> dijkstra(S,A,0)
({0: 0, 1: 3, 2: 4}, {1: 0, 2: 0})
>>> BF(S,A,0)
({0: 0, 1: 2, 2: 4}, {1: 2, 2: 0})
```

On observe que `dijkstra` se trompe tandis que `BF` trouve bien le plus court chemin. C'est conforme à ce qu'on attend : `dijkstra` est conçu pour les graphes valués positivement alors que `BF` fonctionne avec des valuations négatives. Sur le graphe 2 qui est valué positivement, on obtient les mêmes résultats :

```
>>> dijkstra(S,A,0)
({0: 0, 1: 7, 2: 5, 3: 10, 4: 14, 5: 12}, {1: 2, 2: 0, 3: 1, 4: 5, 5: 3})
>>> BF(S,A,0)
({0: 0, 1: 7, 2: 5, 3: 10, 4: 14, 5: 12}, {1: 2, 2: 0, 3: 1, 4: 5, 5: 3})
```

3. On propose :

```
def BF(S,A,s0):
    cout,predec={},{}
    for s in S:
        cout[s]=np.inf
    cout[s0]=0
    for _ in range(1,len(S)):
        # pour chaque arête
        Norelax=True
        for x in A:
            for y,nu in A[x]:
                # on relâche
                if cout[y]>cout[x]+nu:
                    Norelax=False
                    cout[y]=cout[x]+nu
                    predec[y]=x
```

```

        if Norelax:
            print("Sortie accélérée")
            break
    return cout,predec

```

On obtient :

```

>>> BF(S,A,0)
Sortie accélérée
({0: 0, 1: 7, 2: 5, 3: 10, 4: 14, 5: 12}, {1: 2, 2: 0, 3: 1, 4: 5, 5: 3})

```

4. Si après $n - 1$ répétitions du relâchement de tous les arcs on continue à relâcher encore au moins un arc, c'est qu'il n'y a pas de plus court chemin ce qui s'explique par la présence d'un circuit absorbant. On propose :

```

def BF(S,A,s0):
    cout,predec={},{}
    for s in S:
        cout[s]=np.inf
    cout[s0]=0
    for _ in range(1,len(S)):
        # pour chaque arête
        Norelax=True
        for x in A:
            for y,nu in A[x]:
                # on relâche
                if cout[y]>cout[x]+nu:
                    Norelax=False
                    cout[y]=cout[x]+nu
                    predec[y]=x
        if Norelax:
            print("Sortie accélérée")
            break
        Absorb=False
        for x in A:
            for y,nu in A[x]:
                if cout[y]>cout[x]+nu:
                    Absorb=True
                    print("Circuit absorbant")
                    break
        if Absorb:
            break
    return cout,predec

```

On obtient sur le graphe modifié :

```

>>> BF(S,A,0)
Circuit absorbant
({0: 0, 1: -3, 2: 3, 3: -4, 4: 0, 5: -2}, {1: 3, 2: 4, 3: 2, 4: 5, 5: 3})

```