Informatique — MPSI/PCSI

Correction du TP n° 13

15 Algorithme de Dijkstra

Exercice 13.1

1. On commence par faire une copie profonde du graphe `G`, copie à laquelle ont peut affecter des caractéristiques non intrinsèques au graphe de départ.

```
def init_parcours(G,r):
    """Initialise le parcours du graphe"""
    P = copy.deepcopy(G)
    for s in P:
        P[s]['e']=float('inf')
        P[s]['vu']=0
    P[r]['e']=0
    P[r]['vu']=1
    return P
```

2. On recherche le sommet d'étiquette minimum par la méthode du candidat.

```
def plus_faible(G,S):
    """Retourne de le sommet de la liste S d'étiquette la plus faible
    Donnes :
        G : graphe étiquette
        S : liste de sommets
    Retour :
        S[indmin],min : sommet d'étiquette minimum"""
    smin=S[0]
    min=G[smin]['e']
    for s in S:
        if G[s]['e']<min:
            min=G[s]['e']
            smin=s
    return smin
```

3. Pour appliquer l'algorithme de Dijkstra, on commence par initialiser le parcours, puis on définit la liste Nv des sommets non visités. Tant que Nv est non vide, on choisit alors le sujet d'étiquette la plus faible et on applique l'algorithme.

```
def dijkstra(G,r):
    """Calcule la distance des sommets de G àr par l'algorithme de Dijkstra
    Parametres :
        G : graphe
        r : sommet racine
    Retour :
        P : graphe etiquette avec les distances a r"""
    P = init_parcours(G,r)
    Nv=[s for s in P]
    while Nv:
        u=plus_faible(P,Nv)
        Nv.remove(u)
        P[u]['vu']=1
        for v in P[u]['adj']:
            if P[v]['vu']==0 and P[u]['e']+P[u]['adj'][v]<P[v]['e']:
                P[v]['e']=P[u]['e']+P[u]['adj'][v]
    return P
```

Exercice 13.2
On initialise le graphe et on applique l'algorithme de Dijkstra, en modifiant condition d'arrêt pour terminer dès que le sommet `t` est atteint. A chaque étape de l'algorithme, on attribue aussi à tout sommet dont on modifie l'étiquette son sommet parent : cela sera utile pour "remonter" les chemins les plus courts à la section suivante.

```
def distance(G,r,t):
    """Calcule la distance du sommet r a t par l'algorithme de Dijkstra
    Parametres :
        G : graphe
        r : sommet racine
        t : sommet arrive
    Retour :
        L,P[t]['e'] : chemin le plus court de r a t sous forme de liste et longueur du chemin"""
    # Initialisation
    P = init_parcours(G,r)
    Nv=[s for s in P]
    # Algorithme de Dijkstra jusqu'à atteindre t
    while t in Nv:
        u=plus_faible(P,Nv)
        Nv.remove(u)
        P[u]['vu']=1
        for v in P[u]['adj']:
            if P[v]['vu']==0 and P[u]['e']+P[u]['adj'][v]<P[v]['e']:
                P[v]['e']=P[u]['e']+P[u]['adj'][v]
                # Ne pas oublier de préciser le sommet parent
                P[v]['parent']=u
    # Remontée du chemin
    s=t
    L=[s]
    while s!=r:
        s=P[s]['parent']
        L.append(s)
    L.reverse()
    return L,P[t]['e']
```

Exercice 13.3
On reprend le code précédent, en initialisant la couleur des sommets et en modifiant celle-ci chaque fois qu'un sommet est vu (il passe en bleu) ou complètement traité (il passe en rouge). Pour la suite, il est utile d'introduire des paramètre facultatifs *Etapes* (détermine si on doit attendre à chaque étape ou non), *etiquettes* (affiche l'etiquette de chaque sommet ou non) et d (taille des sommets) pour des raisons de lisibilité.

```
def distance_col(G,r,t,etapes=True,etiquettes=True,d=20**2):
    # Initialisation
    P = init_parcours(G,r)
    for s in P:
        P[s]['coul']=JAUNE
    P[r]['coul']=ROUGE
    if etapes==True:
        trace_graphe_adj_cc(P,d,etiquettes)
        input('Attente...')
    Nv=[s for s in P]
    # Algorithme de Dijkstra jusqu'à atteindre t
    while t in Nv:
        u=plus_faible(P,Nv)
        Nv.remove(u)
        P[u]['coul']=ROUGE
        for v in P[u]['adj']:
```

```
            if P[u]['e']+P[u]['adj'][v]<P[v]['e']:
                P[v]['e']=P[u]['e']+P[u]['adj'][v]
                P[v]['parent']=u
                P[v]['coul']=BLEU
        if etapes==True:
            trace_graphe_adj_cc(P,d,etiquettes)
            input('Attente...')
    #Affichage final
    trace_graphe_adj_cc(P,d,etiquettes)
    # Remontée du chemin
    s=t
    L=[s]
    while s!=r:
        s=P[s]['parent']
        L.append(s)
    L.reverse()
    return L,P[t]['e']
```

16 Algorithme A*

Exercice 13.4

1. La fonction prend entrée un graphe tel qu'à chaque sommet est associé une clef `XY` contenant un couple de coordonnée entière et retourne la distance de Manhattan entre deux sommets.

```
def Manhattan(G,s,t):
    """Distance de Manhattan entre les sommets s et t
    Paramètres :
        G : graphe ou chaque sommet est muni des coordonnées XY
        s,t : sommets"""
    return abs(G[s]['XY'][0]-G[t]['XY'][0])+abs(G[s]['XY'][1]-G[t]['XY'][1])
```

2. On associe à chaque sommet `s` une nouvelle clef `h` dont le contenu est l'heuristique définie dans l'énoncé et on retourne le graphe ainsi complété.

```
def heuristique(G,dist,t,C):
    """Applique àchaque sommet du graphe G l'heuristique égale àC fois la distance dist de s
t"""
    for s in G:
        G[s]['h']=C*dist(G,s,t)
```

3. On utilise la fonction précédente puis modifie le graphe `P` en remplaçant la distance associée à chaque arête par une pseudo-distance.

```
def pseudo_distance(G,dist,t,C):
    """Remplace la distance associée àchaque arete par la pseudo-distance tenant compte de l
    Paramètres :
        G : graphe pondéré
        dist : distance utilisée pour l'heuristique
        t : sommet d'arrivée'
        C : coefficient multiplicateur"""
    heuristique(G,dist,t,C)
    for s in G:
        for t in G[s]['adj']:
            G[s]['adj'][t]=G[s]['adj'][t]+(G[t]['h']-G[s]['h'])
```

4. On commence par initialiser une copie du graphe avec `init_parcours`, puis utilise la fonction précédente, puis on lance l'algorithme de Dijkstra sur le graphe modifié.

   On recalcule bien la distance initiale à partir de la pseudo-distance avant de la retourner.

```
def Astar(G,dist,r,t,C):
    """Calcule le plus court chemin entre les sommets r et t àl'aide de l'algorithme A*
```