Proposition de corrigé du TP4 : Graphes, programmation dynamique

1 Graphes

Exercice TP4.1 *Parcours en largeur d'un graphe*

1. La commande `d = deque(t)` permet la création de la file, `d.pop()` a le même effet que sur une liste, `d.popleft()` retire le premier élément de la file, `d.append(42)` a le même effet que sur une liste, `d.appendleft(10)` ajoute `1` en tête de file.
2. On utilise l'algorithme présenté pour obtenir :

```
from collections import deque
def SommetsAccessibles_largeur_demo(G,s):
    SommetsAExplorer, SommetsVus = deque(),[]
    SommetsAExplorer.append(s)
    SommetsVus.append(s)
    while SommetsAExplorer!=deque():
        print('sommets␣à␣explorer␣:␣',SommetsAExplorer)
        print('sommets␣vus␣:␣',SommetsVus)
        s_en_cours = SommetsAExplorer.popleft()
        for v in G[s_en_cours]:
            if v not in SommetsVus:
                SommetsVus.append(v)
                SommetsAExplorer.append(v)
    return(SommetsVus)
```

   Programmer la fonction `SommetsAccessibles(G,s)` en utilisant les files de la question précédente.
3. Ce type de parcours diffère du parcours en profondeur en ce qui concerne l'ordre de parcours des sommets. Si on représente un graphe avec le sommet de départ en haut, le parcours en profondeur est un parcours « de haut en bas » alors que le parcours en largeur est un parcours de « gauche à droite ».

2 Programmation Dynamique

Exercice TP4.2 *Pyramide de nombres*

1. Le nombre $L[0,j] = T[0][j]$ pour tout entier $j \in [\![0, n-1]\!]$ puisque la sous-pyramide correspondante ne comporte qu'une valeur.
2. En complétant les explications précédentes, lorsqu'on somme avec la ligne précédente aux colonnes j et $j+1$, on cherche le maximum possible. Ceci donne la relation :

```
L[i,j]=T[i][j]+max(L[i-1, j-1], L[i-1, j])
```


3. On applique désormais les relations précédentes et on renvoie la dernière valeur calculée :

```
def max_pyramide(T):
    n = len(T)
    L = np.zeros((n,n))
    for j in range(n):
        L[0,j]=T[0][j]
    for i in range(1,n):
        for j in range(0,n-i):
            L[i,j]=T[i][j]+max(L[i-1, j], L[i-1, j+1])
    maximum = L[n-1,0]
    return(maximum,L)
```

Exercice TP4.3 *Plus longue sous-séquence commune*

1. Lorsque i ou j est nul, ceci correspond à la chaîne vide d'où le fait que $c(i,j) = 0$ dans ce cas. Lorsque $x_i = y_j$ on peut ajouter à la chaîne le caractère $x_i = y_j$ ce qui donne bien $c(i,j) = c(i-1, j-1) + 1$. Enfin, lorsque $x_i \neq y_j$, on prend le maximum entre la sous-chaîne où on a retiré x_i qui donne $c(i-1,j)$ ou y_j qui donne $c(i, j-1)$ jusqu'à retomber sur un caractère commun.
2. On calcule les $c(i,j)$ avec la matrice `matrice`. On prendra garde aux décalages d'indices entre le numéro dans la matrice et le numéro dans la chaîne : une chaîne vide correspond aux cas 0 dans la matrice alors que le premier caractère de la chaîne est numéroté 0. On renvoie ensuite le coefficient en bas à droite de la matrice.

```
def PLSC(X, Y):
    m,n = len(X),len(Y)
    matrice = np.zeros((m+1,n+1))
    for i in range(0,m+1):
        for j in range(0,n+1):
            if i == 0 or j == 0:
                matrice[i][j] = 0
            elif X[i-1] == Y[j-1]:
                matrice[i][j] = matrice[i-1][j-1] + 1
            else:
                matrice[i][j] = max(matrice[i-1][j],matrice[i][j-1])
    return matrice[m][n]
```

3. On conserve le code précédent pour obtenir la matrice. Ensuite, en de l'extrémité de chacune des chaînes, on conserve un caractère lorsqu'il est commun et on décale chaque indice vers la gauche ou alors on décale un seul indice selon la valeur dans la matrice conformément à ce qui est présenté dans le sujet.

```
def PLSC_explicite(X, Y):
    m,n = len(X),len(Y)
    matrice = np.zeros((m+1,n+1))
    for i in range(0,m+1):
        for j in range(0,n+1):
            if i == 0 or j == 0:
                matrice[i][j] = 0
            elif X[i-1] == Y[j-1]:
                matrice[i][j] = matrice[i-1][j-1] + 1
            else:
                matrice[i][j] = max(matrice[i-1][j],matrice[i][j-1])
    i,j=m,n
    chaine_explicite = ''
    while i!=0 and j!=0:
        if X[i-1] == Y[j-1]:
            chaine_explicite = X[i-1] + chaine_explicite
            i -= 1
            j -= 1
        elif matrice[i,j-1] > matrice[i-1,j]:
            j -= 1
        else:
            i -= 1
    return (matrice[m][n],chaine_explicite)
```

3 Compléments

Exercice TP4.4 *Algorithme A**

1. On a le code suivant :

```
def ajout_file(F,c):
    P,d=c
    F.append((P,d))
    n=len(F)
    i=-2
    while n+i>0 and F[i][1]<d:
        F[i],F[i+1]=F[i+1],F[i]
        i=i-1
```

2. On a le code suivant :

```
def vois_franch(Map,P):
    i,j=P
    coord=[(i-1,j)]*(i>0)+[(i+1,j)]*(i<n-1)+[(i,j-1)]*(j>0)+[(i,j+1)]*(j<m-1)
    V=[x for x in coord if Map[x]]
    return V
```

3. On initialise certaines valeurs avant de lancer le programme :

```
def dist_eucl(P,Q):
    i,j=P
    k,l=Q
    return ma.sqrt((k-i)**2+(l-j)**2)
h=dist_eucl
Map=Mapno
```


On a ensuite le code suivant :

```
def astar():
    D=(0,0)
    A=(n-1,m-1)
    pos=D
    #pos est la position courante
    F=[((None,pos),h(pos,A))]
    #dans la file on trouve des couples :
    #(position précédente,position courante),distance a l arrivee
    dist=0
    chemin=[]
    #on enregistre dans chemin tous les points visites ainsi que
    #les coordonnees de la position qui a mene a ces points'
    for i in range(n):
        for j in range(m):
            if not Map[i,j]:
                plt.scatter([i],[j], color='black',marker='s')
    #Les 4 dernieres lignes permettent de simuler
    #la carte avec les obstacles.
    #Les obstacles sont représentés par des point noirs
    while pos!=A and F!=[]:
        (prec,pos),d=F.pop()
        Map[pos]=False
        V=vois_franch(Map,pos)
        if V!=[]:
            dist+=1
            chemin+=[(prec,pos)]
            for v in V:
                c=(pos,v),h(v,A)
                if c not in F:
                    ajout_file(F,c)
        elif pos==A:
            dist+=1
            chemin+=[(prec,pos)]
    if pos==A:
        #chemin représente toutes les pistes explorées,
        #trajet représente seulement la solution retenue
        #a partir de la derniere position, on reconstruit le trajet retenu'
        prec,pos=chemin[-1]
        trajet=[prec,pos]
        i,j=prec
        k,l=pos
        plt.plot(np.array([i,k]),np.array([j,l]),color='red')
        #on trace le trajet retenu en rouge'
        while prec!=D:
            for c in chemin:
                if c[1]==prec:
                    i,j=c[0]
                    k,l=prec
                    plt.plot(np.array([i,k]),np.array([j,l]),color='red')
                    prec=c[0]
                    trajet=[prec]+trajet
                    break
        plt.show()
        return chemin,trajet,dist
    else :
        #Si la dernière position n est pas le point d arrivee, il n y a pas de solution.'
        plt.show()
        return False
```

Exercice TP4.5 *Construction d'un labyrinthe*

1. Une seule ligne ici suffit :

```
def MatrixFalse(n,p):
    return np.zeros ((n,p),dtype=bool)
```

Dans la suite, on crée dans l'éditeur de texte la variable

```
Atteint=MatrixFalse(20,20)
```

qui permettra de tester les fonctions suivantes en utilisant `Atteint` comme paramètre `M`. Le booléen de cette matrice d'indices (i, j) indiquera ensuite si un point de coordonnées (i, j) a déjà été atteint.

2. On renvoie une liste de tuple. On fait attention aux positions du bord du tableau.

```
def Adj(i,j,Atteint):
    n,p = np.shape(Atteint)
    L = []
    for (a,b) in [(i+1,j),(i-1,j),(i,j+1),(i,j-1)]:
        if (a>=0 and a<n and b>=0 and b<p) and not (Atteint[a,b]) :
            L.append((a,b))
    return L
```

3. On part de la position $(0, 0)$ que l'on ajoute à la pile positions et que l'on marque comme étant atteinte. À partir de la dernière position (i, j) de la pile position, que l'on dépile, on choisit un voisin (a, b) au hasard pour poursuivre notre chemin. Si on avait plusieurs choix pour la position voisine (a, b), on rempile la position (i, j) pour y revenir plus tard.

```
def Labyrinthe (n,p) :
    Atteint=MatrixFalse(n,p)
    PilePositions = CreerPile()
    PileChemin  = CreerPile()
    Empiler(PilePositions,(0,0))
    Atteint[0,0] = True
    while not PilePositions == CreerPile() :
        (i,j) = Depiler(PilePositions)
        L = Adj (i,j,Atteint)
        if len(L) > 0 :
            (a,b) = rd.choice (L)
            Atteint[a,b] = True
            Empiler(PileChemin,((i,j),(a,b)))
            if len (L) > 1 :
                Empiler(PilePositions,(i,j))
            Empiler(PilePositions,(a,b))
    return (PileChemin)
```

4.

```
def AffichageLab (L,n,p) :
    plt.close ()
    plt.figure ()
    plt.axis ('off')
    plt.xlim (-0.5,n-0.5)
    plt.ylim (-0.5,p-0.5)
    plt.plot ([0],[0],'ro')
    plt.plot ([n-1],[p-1],'ro')
    while not (EstVide (L)) :
        (c1,c2) = Depiler (L)
        (i1,j1) = c1
        (i2,j2) = c2
        plt.plot ([i1,i2],[j1,j2],color='b')
    plt.show ()
```