TP4 : Graphes, Programmation Dynamique

1 Graphes

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

On considère un graphe `G` donné par sa liste d'adjacence. On veut définir une fonction `SommetsAccessibles` qui à partir du graphe `G` et d'un sommet `s` renvoie l'ensemble des sommets atteignables. On va utiliser pour cela un parcours en largeur du graphe. On commence par lister tous les voisins de la source, pour ensuite les explorer un par un. Ce mode de fonctionnement utilise donc une file dans laquelle il prend le premier sommet et place en dernier ses voisins non encore explorés. Les sommets déjà visités sont marqués afin d'éviter qu'un même sommet soit exploré plusieurs fois. Voici les étapes de l'algorithme :

(i) mettre le sommet source dans la file ;

(ii) retirer le sommet du début de la file pour le traiter ;

(iii) mettre tous ses voisins non explorés dans la file (à la fin) ;

(iv) si la file n'est pas vide reprendre à l'étape 2.

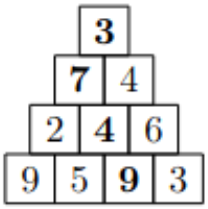
On commence par importer `from collections import deque` pour utiliser les files.

1. En considérant `t=[17, 87, 14, 18, 19]`, tester les commandes : `d = deque(t)`, `d.pop()`, `d.popleft()`, `d.append(42)`, `d.appendleft(10)`.
2. Programmer la fonction `SommetsAccessibles(G,s)` en utilisant les files de la question précédente.
3. En quoi ce type de parcours diffère-t-il du parcours en profondeur ?

2 Programmation Dynamique

Exercice TP4.2 *Pyramide de nombres*

Dans une pyramide de nombres, on cherche, en partant du sommet de la pyramide, et en se dirigeant vers le bas à chaque étape (vers la case en-dessous juste à gauche ou la case en-dessous juste à droite), à maximiser le total des nombres traversés. Dans l'exemple ci-contre, le maximum est obtenu pour le chemin en gras ($3+7+4+9=23$). On appellera ce nombre la valeur de la pyramide. On appelle n le nombre de cases à l'étage inférieur de la pyramide (sur l'exemple $n=4$) et on encode la pyramide dans un tableau T : $T[i][j]$ est la valeur du nombre situé à l'étage i en partant du bas et en position j en partant de la gauche.



Ainsi, la pyramide précédente est représentée par la liste (de listes) :

```
T = [[9,5,9,3],[2,4,6],[7,4],[3]]
```

Pour résoudre ce problème on va calculer un tableau L tel que pour tout $i \in [\![0, n-1]\!]$ et tout $j \in [\![0, i]\!]$, le nombre $L[i,j]$ est la valeur de la sous-pyramide dont le sommet est la case en position (i,j).

1. Que vaut $L[0,j]$ pour tout entier $j \in [\![0, n-1]\!]$?
2. Montrer que pour $i > 0$, $L[i,j] = T[i][j] + \max\{L[i-1,j], L[i-1,j+1]\}$ lorsque $j \in [\![0, n-i]\!]$ (`n=len(T)`).
3. Écrire une fonction `max_pyramide` qui prend en entrée une pyramide sous la forme d'un tableau T et retourne la valeur de cette pyramide.

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

La plus longue sous-séquence commune à deux chaînes de caractères, est une séquence étant sous-chaîne des deux chaînes, et étant de taille maximum.

Pour les deux séquences de caractères suivantes : `'abcde'` et `'ceij'` la plus longue sous-séquence commune est `ce` Dans ce problème, il est nécessaire que les éléments communs soient dans le même ordre dans les différentes séquences, mais pas qu'ils soient obligatoirement consécutifs : `e` n'est pas consécutif à `c` dans la première séquence. Dans la suite, on appelle simplement « plus longue sous-chaîne commune » PLSC. On considère deux chaîne $X = x_1 \dots, x_m$ et $Y = y_1 \dots y_n$ et on note X_i, Y_i les sous-chaînes de X et de Y constituées des i premiers éléments. Notons $c(i, j)$ la longueur de la PLSC entre X_i et Y_j.

1. Justifier que
$$c(i,j) = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0 \\ c(i-1, j-1) + 1 & \text{si } i, j > 0 \text{ et } x_i = y_j, \\ \max\big(c(i, j-1), c(i-1, j)\big) & \text{si } i, j > 0 \text{ et } x_i \neq y_j \end{cases}$$
2. En déduire une programmation dynamique de `LPLSC(X,Y)` qui renvoie la longueur de la PLSC à X et Y. On introduira une matrice qui comportera les $c(i, j)$.

On peut reconstituer une plus longue sous-séquence commune grâce à la matrice précédente. Pour cela on effectue un parcours depuis $c[m][n]$ suivant la règle exposée ici. Depuis une case $c[i][j]$ de valeur α :

- Si $x_i = y_j$, on passe à la case $c[i-1][j-1]$ de valeur $\alpha - 1$ et on ajoute ce caractère ($x_i = y_j$) au début de la PLSC en construction.
- Si $x_i \neq y_j$ et
  - si $c[i][j-1] = c[i-1][j] = \alpha$, on passe indifféremment à la case $c[i-1][j]$ ou $c[i][j-1]$.
  - si $c[i][j-1] = \alpha > c[i-1][j]$, on passe à la case $c[i][j-1]$.
  - si $c[i-1][j] = \alpha > c[i][j-1]$, on passe à la case $c[i-1][j]$.

Un exemple de parcours est donné par le tableau suivant, grâce auquel on déduit que MJAU est une PLSC à MZJAWXU et XMJYAUZ :

		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
		Ø	**M**	**Z**	**J**	**A**	**W**	**X**	**U**
0	**Ø**	**0**	0	0	0	0	0	0	0
1	**X**	0	0	0	0	0	0	1	1
2	**M**	0	**1**	1	1	1	1	1	1
3	**J**	0	1	1	**2**	2	2	2	2
4	**Y**	0	1	1	2	2	2	2	2
5	**A**	0	1	1	2	**3**	3	3	3
6	**U**	0	1	1	2	3	3	3	**4**
7	**Z**	0	1	2	2	3	3	3	4

3. Modifier la fonction précédente pour qu'elle renvoie la longueur de la PLSC et son expression.