TP : Pyramide de nombres

Mise en situation

On a une pyramide de nombres à n étages. Par exemple la pyramide à 6 étages ci-contre. On parcourt cette pyramide de son sommet (étage 5) à sa base (étage 0) en sommant les nombres par lesquels on passe. Chaque déplacement est une descente d'un étage vers un nombre soit juste à gauche soit juste à droite.

L'objectif du problème est de trouver le chemin du sommet à la base de la pyramide maximisant la somme des n nombres de ce chemin et de calculer cette somme maximale.

Etages	Pyramide
5	3
4	7 4
3	2 4 6
2	5 8 9 3
1	1 5 2 6 1
0	4 2 8 3 2 9

Programmation naïve et limite de l'algorithme

Ouvrir avec Pizo (ou Spider) le code intitulé « TP – Pyramide des nombres.py »

Question 1

La ligne 8 du code de ce fichier Python permet de définir la pyramide ci-dessus

a- Sous quelle forme est codée la pyramide ? (Donner le type Python de la variable **`p_exemple`**)

b- Si vous exécutez ce code, quel nombre va retourner la commande : **`p_exemple[1][3]`** ?

c- Avec quelles commandes peut-on retourner les deux nombres situés (à gauche et à droite) sous le nombre p_exemple[i][j] ? Tester cela avec le nombre : **`p_exemple[1][3]`** .

Question 2

Que renvoie la fonction : **`p_a(n)`** ?

La fonction **`affiche_p(p)`** prend en argument une pyramide de nombres et l'affiche en positionnant les nombres en pyramide. Tester avec : **`affiche_p(p_exemple)`** et **`affiche_p(p_a(15))`**

Question 3

Le principe de la résolution du problème (Equation de Bellman de l'algorithme dynamique) est que la somme maximale obtenu en partant d'un nombre $n_{i,j}$ (nombre d'indice j de l'étage d'indice i) de la pyramide jusqu'à la base de la pyramide est l'addition de ce nombre $n_{i,j}$ et du maximum des sommes obtenues $\begin{cases} \text{en partant du nombre } n_{i-1,j} \text{ (en dessous à gauche de } n_{i,j}) \text{ jusqu'à la base de la pyramide} \\ \text{en partant du nombre } n_{i-1,j+1} \text{ (en dessous à droite de } n_{i,j}) \text{ jusqu'à la base de la pyramide.} \end{cases}$

Implémenter une fonction récursive **`somme_et_chemin_naif(p)`** utilisant cette équation de Bellman qui prend en argument une pyramide de nombre **`p`** (Liste de listes) et retourne un couple (somme maximale , chemin donnant cette somme maximale). L'algorithme de cette fonction sera le suivant :

Si la pyramide n'a qu'un seul nombre, on retourne ce nombre et une liste vide **`(p[0][0],[])`**

Sinon : On construit **`pg`** *la pyramide inférieure de gauche : celle dont le sommet est* **`p[len(p)-2][0]`**

On construit **`pd`** *la pyramide inférieure de droite : celle dont le sommet est* **`p[len(p)-2][1]`**

On fait 2 appels récursifs de **`somme_et_chemin_naif(p)`** *pour avoir les sommes et les chemins des pyramides* **`pg`** *inférieure de gauche :* **`(SG,CG)`** *et* **`pd`** *inférieure de droite :* **`(SD,CD)`**

Si **`SG>SD`** *(ou* **`SG≥SD`***) alors on retourne le couple* **`(p[len(p)-1][0]+SG,['Gauche']+CG)`**

Sinon on retourne le couple **`(p[len(p)-1][0]+SD,['Droite']+CD)`**

Tester votre fonction avec la commande : **`somme_et_chemin_naif(p_exemple)`** qui doit renvoyer : **`(35,['Gauche', 'Droite', 'Gauche', 'Gauche', 'Droite'])`**

Question 4

La fonction **`test_naif(n)`** prend en argument un nombre entier n crée aléatoirement une pyramide de n étages, affiche dans le Shell cette pyramide, détermine et affiche la somme maximale obtenue par un parcours du sommet à la base de cette pyramide ainsi que le chemin nécessaire à cela. Cette fonction affiche également le temps nécessaire pour faire cette résolution avec la fonction **`somme_et_chemin_naif(p)`** .

Tester cette fonction **`test_naif(n)`** avec : n=16 , 18 , 19 , 20 et 21. Quel est la complexité de la fonction **`somme_et_chemin_naif(p)`** ? Conclure sur cet algorithme.

Programmation dynamique avec mémoïsation

Pour cette programmation dynamique nous avons vu en cours qu'il fallait réaliser une pyramide de nombre Σ_{max} de même dimension que la pyramide P. Et en notant pour ces deux pyramides P et Σ_{max} :

☞ P_i^j le $j^{ième}$ nombre (en partant de la gauche) du $i^{ième}$ étage de la pyramide P.

☞ Σ_i^j le $j^{ième}$ nombre (en partant de la gauche) du $i^{ième}$ étage de la pyramide Σ_{max}.

Les nombres Σ_i^j de cette nouvelle pyramide correspondent aux sommes maximales pouvant être obtenues en parcourant la pyramide P à partir du nombre P_i^j de la pyramide P jusqu'à la base de la pyramide P.

Question 5

Ecrire une fonction **`pyramide_des_sommes(p)`** qui prend en argument une pyramide de nombres P et retourne la pyramide de nombres Σ_{max} correspondante à P. Comme vu en cours, on construira cette pyramide en partant de sa base et en allant jusqu'à son sommet.

Tester votre fonction avec la pyramide **`p_exemple`** , la fonction doit vous retourner la pyramide construite manuellement en cours.

Question 6

Ecrire une fonction **`somme_et_chemin(p)`** qui prend en argument une pyramide de nombres P et retourne le même résultat que la fonction **`somme_et_chemin_naif(p)`**. Cette fonction n'utilisera plus un algorithme récursif tel que celui vu à la question 3 mais utilisera la fonction **`pyramide_des_sommes(p)`** de la question 5.

Question 7

Tester votre fonction **`somme_et_chemin(p)`** avec la pyramide p_exemple cela doit renvoyer : **`(35,['Gauche', 'Droite', 'Gauche', 'Gauche', 'Droite'])`** (Comme vu en cours). Puis tester avec une pyramide de nombre aléatoires de 50 étages.

Quelle la complexité de cette fonction ? Conclure

Question 8 (Pour les plus rapides)

Il est possible de résoudre le problème en reprenant l'algorithme récursif de la question 3. Mais pour éviter le double appel récursif de cette fonction qui induit la complexité exponentielle ($O(2^n)$) on mémoïse le résultat dans un dictionnaire dont les clefs sont les couples des indices de la pyramide et les valeurs correspondantes le couple (valeur de la pyramide des sommes, chemin pour arriver à cette somme maximale). Pour la première ligne : **`D={(0,i):(p[0][i],[]) for i in range(len(p))}`**

Reprendre l'algorithme de la question 3 pour parvenir à implémenter cette fonction **`somme_et_chemin_memoisation(p)`** qui utilisera un dictionnaire et une sous fonction récursive : **`sous_fonction_rec(p,j)`** . Il suffit alors de retourner **`sous_fonction_rec(p,0)`**.