SUJET 0 – 2024 – sujet2 - Correction

Exercice 1 (6 points)

Partie 1 – Lecture et analyse de pyramide

1. Dessiner la pyramide correspondant à

```
ex2 = [[3],[1,2],[4,5,9],[3,6,2,1]]
```

Représentation :

```
      3
    1   2
  4   5   9
3   6   2   1
```

👉 **Commentaire :** *Chaque sous-liste correspond à un niveau. Le niveau i contient i+1 éléments.*

---

2. Déterminer un conduit maximal dans ex2

On teste les chemins possibles :

- 3 → 1 → 4 → 3 = 11
- 3 → 1 → 4 → 6 = 14
- 3 → 1 → 5 → 6 = 15
- 3 → 1 → 5 → 2 = 11
- 3 → 2 → 5 → 6 = 16
- 3 → 2 → 9 → 2 = 16
- 3 → 2 → 9 → 1 = 15

Score maximal = **16**

Exemple de conduit maximal :

3 → 2 → 5 → 6

👉 **Commentaire :** *Il fallait respecter les déplacements adjacents uniquement (indice j ou j+1 au niveau suivant).*

Partie 2 – Nombre de conduits

3. Énumérer les conduits (pyramide à 3 niveaux)

Avec 3 niveaux :

```
   a
  b  c
 d  e  f
```

Conduits possibles :

- a → b → d
- a → b → e
- a → c → e
- a → c → f

Il y a **4 conduits**.

👉 **Commentaire :** *Chaque niveau ajoute un choix gauche/droite.*

---

4. Nombre de conduits pour n niveaux

Un conduit correspond à une suite de (n−1) choix gauche/droite.

Donc : Nombre de conduits = 2^{n-1}

👉 **Commentaire :** *Chaque niveau (sauf le sommet) double le nombre de possibilités.*

---

5. Pourquoi la méthode exhaustive n'est pas raisonnable ?

Le nombre de conduits est exponentiel : 2^{n-1}

Donc le coût est exponentiel.

👉 **Commentaire :** *Une complexité exponentielle devient rapidement ingérable lorsque n augmente.*

Partie 3 – Récursivité

6. Écriture de la fonction récursive score_max

```
def score_max(i, j, p):
    # cas de base : dernier niveau
    if i == len(p) - 1:
        return p[i][j]

    # cas récursif
    return p[i][j] + max(
        score_max(i+1, j, p),
        score_max(i+1, j+1, p)
    )
```

Score maximal total :

```
score_max(0, 0, p)
```

👉 **Commentaire :** *La fonction suit directement la définition mathématique donnée.*

Partie 4 – Programmation dynamique

7. Fonction pyramide_nulle

```
def pyramide_nulle(n):
    return [[0 for j in range(i+1)] for i in range(n)]
```

👉 **Commentaire :** *On construit une structure identique à la pyramide initiale.*

---

8. Compléter prog_dyn

```
def prog_dyn(p):
    n = len(p)
    s = pyramide_nulle(n)

    # remplissage du dernier niveau
    for j in range(n):
        s[n-1][j] = p[n-1][j]

    # remplissage des autres niveaux
    for i in range(n-2, -1, -1):
        for j in range(i+1):
            s[i][j] = p[i][j] + max(s[i+1][j], s[i+1][j+1])

    return s[0][0]
```

👉 **Commentaire :** *On remplit la pyramide de bas en haut : c'est le principe de la programmation dynamique.*

---

9. Montrer que le coût est quadratique

Nombre total d'éléments dans une pyramide de n niveaux :

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$$

Donc le nombre d'opérations est proportionnel à n^2.

Complexité : $O(n^2)$

👉 **Commentaire :** *Chaque case est calculée une seule fois.*

---

10. Adapter score_max pour éviter la redondance (mémoïsation)

On peut utiliser un dictionnaire pour mémoriser les résultats :

```
def score_max(i, j, p, memo=None):
    if memo is None:
        memo = {}

    if (i, j) in memo:
        return memo[(i, j)]

    if i == len(p) - 1:
        memo[(i, j)] = p[i][j]
    else:
        memo[(i, j)] = p[i][j] + max(
```

```
            score_max(i+1, j, p, memo),
            score_max(i+1, j+1, p, memo)
        )

    return memo[(i, j)]
```

👉 **Commentaire :** *Chaque sous-problème est résolu une seule fois : on passe d'un coût exponentiel à un coût quadratique.*
