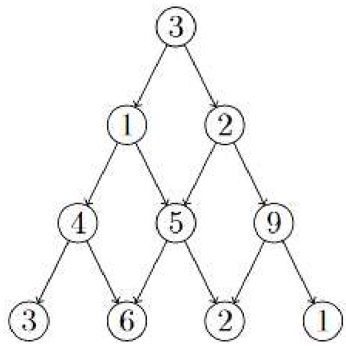
CORRECTION – 2024 Sujet Zéro B

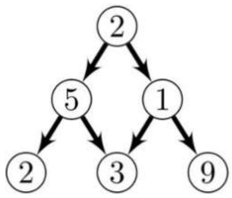
Exercice 1 : Récursivité et optimisation

1. `ex2 = [[3], [1,2], [4,5,9], [3,6,2,1]]` représente la pyramide :



2. Il y a deux conduits égaux de score maximal :
3 + 2 + 5 + 6 = 16
3 + 2 + 9 + 2 = 16

3. Conduits pour la pyramide :
2 + 5 + 2
2 + 5 + 3
2 + 1 + 3
2 + 1 + 9



4. Pour n = 2, il y a 2 = 2^1 conduits. Pour n = 3, il y a 4 = 2^2 conduits. Pour n = 4, il y a 8 = 2^3 conduits.
Ainsi pour n niveaux, il y a $\mathbf{2^{n-1}}$ conduits

5. L'algorithme testant tous les conduits a une complexité **exponentielle O(2^n)** ce qui n'est pas raisonnable.

6.
```
def score_max(i, j, p):  # indice j du niveau i de la pyramide p
    # Cas trivial : dernier niveau
    if i == len(p)-1:
        return p[i][j]
    # Cas récursif : on ajoute le score max des deux sous-arbres
    return p[i][j] + max(score_max(i+1, j, p), score_max(i+1, j+1, p))
```

7.
```
def pyramide_nulle(n):
    p = []
    for k in range(1, n+1):
        p.append([0]*k)
    return p
```

Solution plus sophistiquée :

```
def pyramide_nulle(n): # par compréhension
    return [[0]*k for k in range(1, n+1)]
```

8.
```
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-1, -1, -1):
        for j in range(i):
            s[i][j] = p[i][j] + max(s[i+1][j], s[i+1][j+1])
    # renvoie du score maximal
    return s[0][0]
```

9. On peut observer deux boucles imbriquées s'exécutant chacune au plus n fois, ce qui justifie le coût d'exécution **quadratique O(n^2)** de la fonction.

10. Le principe de la programmation dynamique est d'éviter de refaire des calculs déjà effectués. Ainsi, il est possible de stocker le résultat des calculs dans un dictionnaire et de s'en servir à chaque appel récursif pour y puiser les résultats des calculs déjà rencontrés.
Cette technique de stockage en mémoire des résultats redondants est la mémoïsation.

```python
def score_max_mem(i, j, p, mem={}):
    # Cas trivial : dernier niveau
    if i == len(p)-1:
        return p[i][j]
    # Cas déjà traité et enregistré dans la mémoire
    if (i, j) in mem:
        return mem[(i,j)]
    # Cas récursif : on ajoute le score max des deux sous-arbres
    r = p[i][j] + max(score_max(i+1, j, p), score_max(i+1, j+1, p))
    mem[(i, j)] = r # enregistrement du résultat dans la mémoire
    return r
```