Technical Assignment : EL MAHFOUD RADOUANE

Assignment 1: Puzzle

Approche 1 : Récursivité

Le nombre de façons d'atteindre la 5-ème marche d'un escalier peut être compris en considérant les déplacements que Charlie peut faire juste avant d'atteindre à cette 5ème marche.

Charlie peut atteindre la 5ème marche :

- **Soit depuis la 4ème marche, en montant d'une marche.**
- **Soit depuis la 3ème marche, en montant de deux marches.**

Voici le raisonnement :

- **Chaque façon unique d'arriver à la 4ème marche peut être prolongée d'un pas pour atteindre la 5-ème marche.**
- **De même, chaque façon unique d'arriver à la 3ème marche peut être prolongée de deux pas pour atteindre la 5ème marche.**

Il n'y a pas d'autres moyens d'atteindre la 5ème marche sans d'abord se poser sur la 3ème ou la 4ème marche. De plus, il n'y a pas d'intersection entre les façons d'atteindre la 3ème et la 4ème marche ; elles sont indépendantes. Donc , le nombre total de façons uniques d'atteindre la 5ème marche est la somme de :

Technical Assignment : EL MAHFOUD RADOUANE

- **Toutes les façons uniques d'atteindre la 4ème marche (puis de monter d'une marche),plus toutes les façons uniques d'atteindre la 3ème marche (puis de monter de deux marches).**

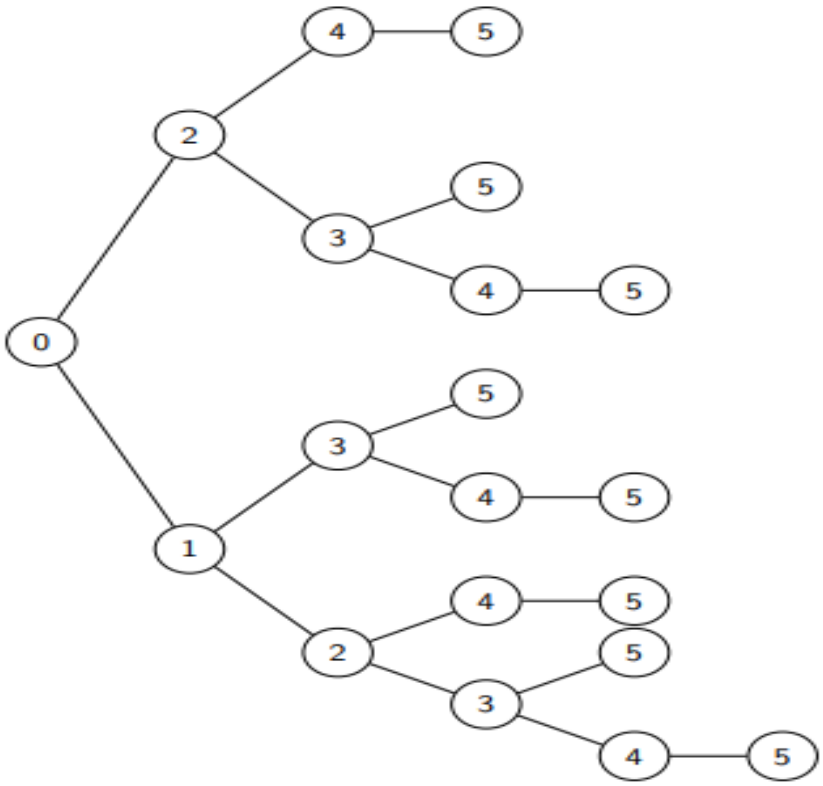
Mathématiquement :

$f(5)$ **comme étant le nombre de façons d'atteindre la** 5**-ième marche, alors pour la 5ème marche**

$f(5)=f(4)+f(3)$

Pour n>=2 : f(n)=f(n-1)+f(n-2)

qui est essentiellement la relation de récurrence qui définit la séquence de Fibonacci, où chaque terme est la somme des deux termes précédents.



Technical Assignment : EL MAHFOUD RADOUANE

Explication : La solution récursive utilise le concept des nombres de Fibonacci pour résoudre le problème. Elle calcule le nombre de façons de monter les escaliers en appelant récursivement la fonction pour (n-1) et (n-2).

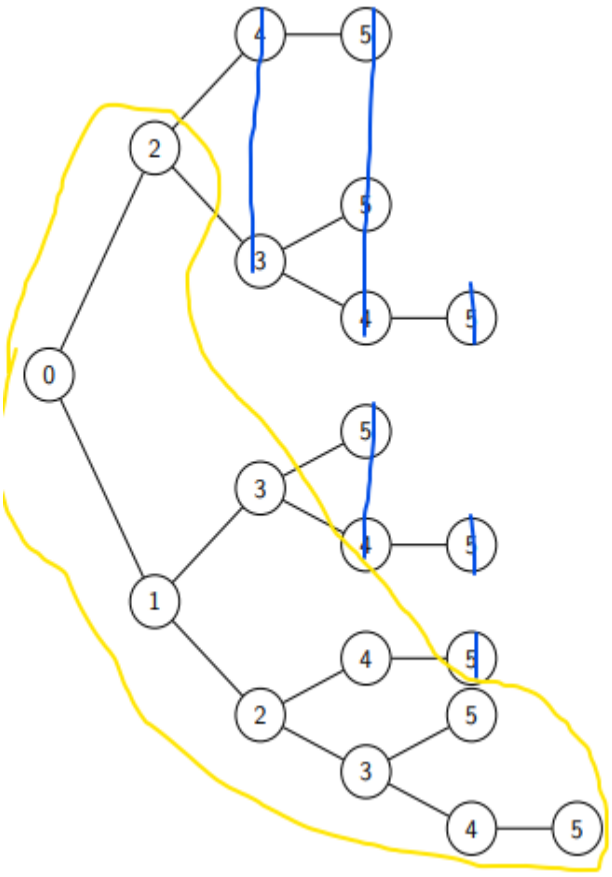
Cependant, cette solution a une complexité temporelle exponentielle (O(2^n)) en raison de calculs redondants.

```
def f(n):
    if n == 0 or n == 1:
        return 1
    return f(n-1) + f(n-2)
f(5)
```

✓ 0.0s

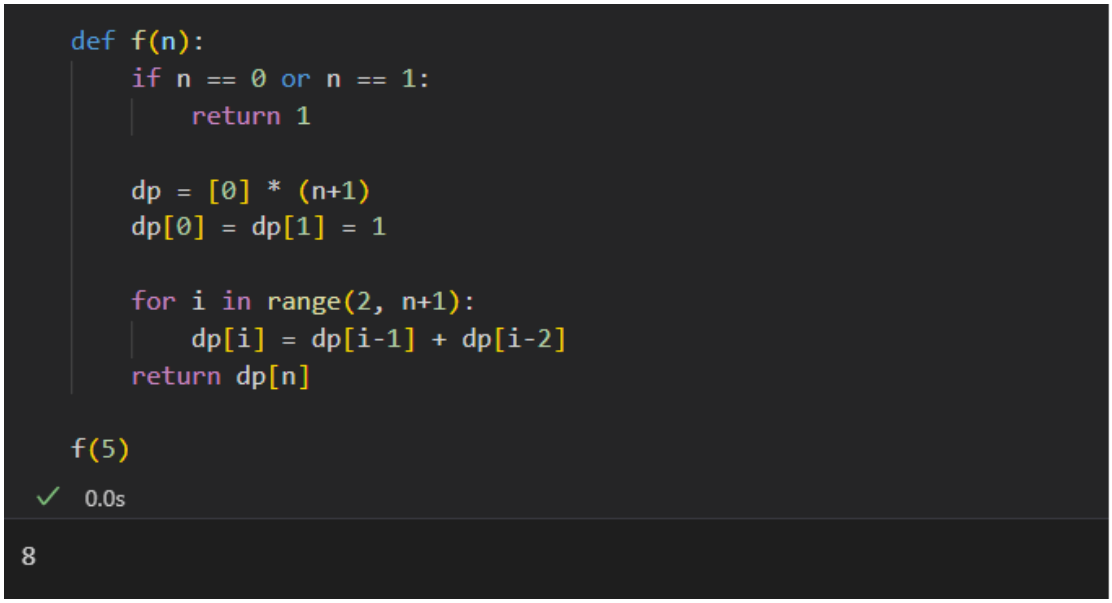
```
8
```

Approach 2: Table DP



Explication : Cette solution élimine la récursion et utilise une approche ascendante (bottom-up) pour résoudre le problème. Elle crée une table de programmation dynamique (dp) de taille n+1 pour stocker le nombre de façons d'atteindre chaque marche. Les cas de base (0 et 1 marche) sont initialisés à 1 car il n'y a qu'une seule façon de les atteindre. Ensuite, elle itère de 2 à n, en remplissant la table de programmation dynamique en additionnant les valeurs des deux marches précédentes. Enfin, elle renvoie la valeur dans la dernière cellule de la table de programmation dynamique, qui représente le nombre total de façons d'atteindre le sommet.

Technical Assignment : EL MAHFOUD RADOUANE



```python
def f(n):
    if n == 0 or n == 1:
        return 1

    dp = [0] * (n+1)
    dp[0] = dp[1] = 1

    for i in range(2, n+1):
        dp[i] = dp[i-1] + dp[i-2]
    return dp[n]

f(5)
```

✓ 0.0s

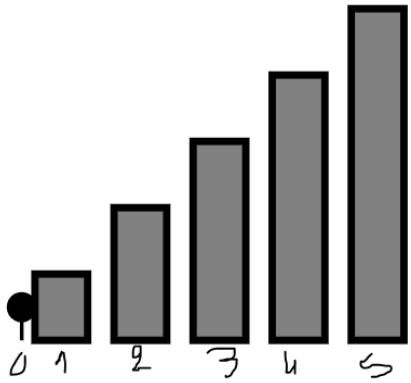
```
8
```

Approche 3: Optimisation

Explication : La solution optimisée en termes d'espace réduit davantage la complexité de l'espace en utilisant uniquement deux variables (a et b) au lieu d'une table de programmation dynamique complète.

Technical Assignment : EL MAHFOUD RADOUANE

Voilà un exemple illustratif pour n=5 ,



i=1

			2	1	1
				a	b

i=2

		3	2	1	1
			a	b	

i=3

	5	3	2	1	1
		a	b		

i=4

8	5	3	2	1	1
	a	b			

Technical Assignment : EL MAHFOUD RADOUANE

```python
def f(n):
    a,b=1,1
    for i in range(n-1):
        temp=a
        a=a+b
        b=temp
    return a
f(5)
```

✓ 0.0s

```
8
```