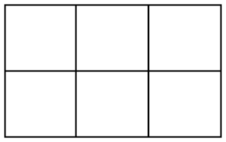
Programmation dynamique

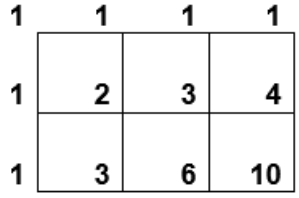
Exercice 2 : Nombres de chemins dans une grille

Pour une grille de taille `n×m` donnée, combien de chemins mènent du coin supérieur gauche au coin inférieur droit, en se déplaçant uniquement le long des traits horizontaux vers la droite et le long des traits verticaux vers le bas ?



1 Relation de récurrence

On a vu dans l'introduction du cours que l'on peut calculer ce nombre en utilisant le nombre de chemins pour des grilles plus petites. Dans l'exemple au-dessus on a indiqué le nombre de chemin pour arriver à chaque intersection (en partant du coin supérieur gauche).



On rappelle que chaque nombre est la somme du nombre situé à gauche et du nombre situé au-dessus. Autrement dit, on a la relation de récurrence suivante :

$$\text{nb\_chemin}(n,m)=\begin{cases}1 \text{ si } n=0 \text{ ou } m=0\\ \text{nb\_chemin}(n-1,m) + \text{nb\_chemin}(n,m-1) \text{ sinon}\end{cases}$$

2 Version récursive naïve

La relation de récurrence permet d'écrire la fonction récursive naïve suivante :

```
def nb_chemins_recursif(n, m):
    if n == 0 or m == 0:
        return 1
    else:
        return nb_chemins_recursif(n-1, m) + nb_chemins_recursif(n, m-1)

print(nb_chemins_recursif(3, 2))
print(nb_chemins_recursif(10, 10))
```

Vous pouvez constater que pour des valeurs un peu plus élevée, le calcul prend plusieurs secondes et si on augmente encore les valeurs de n et m, le calcul ne termine pas en un temps raisonnable voire jamais.

```
import time
t1 = time.time()
nb_chemins = nb_chemins_recursif(13, 13)
t2 = time.time()
print(nb_chemins)
print("temps (en s.) : ", t2 - t1)
```

3 Objectif

Utiliser la programmation dynamique pour améliorer l'efficacité de l'algorithme. Vous écrirez une version récursive avec mémoïsation (approche descendante) puis une version itérative avec mémoïsation (approche ascendante).

4 Version récursive avec mémoïsation (approche descendante)

On va utiliser un tableau `memo` pour mémoriser les résultats des calculs afin de ne pas les effectuer deux fois.
La liste `memo` est de taille `(n+1)×(m+1)` et l'élément `memo[i][j]` représente le nombre de chemins sur une grille `i×j`.
On peut créer la liste `memo` avec uniquement des 1 au départ, puisque c'est la valeur minimale des `memo[i][j]`.

Question 1 : Écrivez

- la fonction récursive `nb_chemins_recursif_memo(n, m, memo)` qui renvoie le nombre de chemins sur une grille de taille `n×m` en mettant à jour la liste `memo` pour stocker les résultats intermédiaires calculés (afin de ne pas les calculer plusieurs fois lors des appels récursifs)
- la fonction d'interface `nb_chemins_recursif_memoise(n, m)` qui renvoie le nombre de chemins d'une grille `n×m` (il suffit de lancer le premier appel de la fonction `nb_chemins_recursif_memo` sur une liste ne contenant que des 1).

5 Version itérative (approche ascendante)

On peut procéder de manière classique en trois étapes (comme vu dans les autres exemples).
On va utiliser un tableau grille de taille `(n+1)×(m+1)` dans lequel `grille[i][j]` est le nombre de chemins sur une grille de taille `i×j`.
On peut commencer par créer la liste `grille` avec uniquement des 1, ce qui permet d'initialiser correctement la première ligne et la première colonne. Ensuite, avec deux boucles for imbriquées on peut calculer les valeurs `grille[i][j]` en progressant dans le sens des indices croissants et en utilisant la relation de récurrence.

Question 2 : Écrivez une fonction `nb_chemins_iteratif_ascendant(n, m)` qui renvoie le nombre de chemins d'une grille de taille `n×m` en construisant et complétant la liste `grille` au fur et à mesure.