

La programmation dynamique est fréquemment employée pour résoudre des problèmes d'optimisation : elle s'applique lorsque la solution optimale peut être déduite des solutions optimales des sous-problèmes (principe d'optimalité de Bellman). Cette méthode garantit d'obtenir la meilleure solution au problème étudié. La mémorisation permet de ne pas traiter des sous problèmes inutilement.

### Exercice 1 (Coefficients binomiaux)

On s'intéresse au calcul du coefficient binomial  $\binom{n}{p}$ , avec  $0 \leq p \leq n$ . Ce coefficient donne le nombre de parties à  $p$  éléments d'un ensemble à  $n$  éléments, ou le nombre de combinaisons sans répétition de  $p$  éléments parmi  $n$ .

1. Écrire une fonction `binom1` qui prend en argument les entiers  $n$  et  $p$  et qui renvoie la valeur du coefficient binomial  $\binom{n}{p}$  grâce à un double appel récursif en utilisant la formule de Pascal.
2. Soit  $C(n, p)$  le nombre d'additions réalisées par la fonction précédente. Montrer que

$$C(n, 0) = C(n, n) = 0 \quad \text{et} \quad \forall p \in \llbracket 1, n \rrbracket, C(n, p) = C(n-1, p-1) + C(n-1, p) + 1$$

Montrer par récurrence que  $C(n, p) = \binom{n}{p} - 1$  pour tout  $p \in \llbracket 0, n \rrbracket$ .

3. À l'aide de la formule de Stirling, montrer que  $\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi n}}$ . En déduire la complexité du calcul de  $\binom{2n}{n}$ . Vérifier en essayant de calculer  $\binom{32}{16}$ .
4. Écrire une fonction `binom2` qui calcule  $\binom{n}{p}$  à l'aide de la programmation dynamique bottom-up, en utilisant un tableau de taille  $(n+1) \times (p+1)$ .
5. Quelle est la complexité temporelle de la fonction `binom2` ?
6. Comment simplifier la fonction précédente pour n'utiliser qu'un tableau à une dimension ?
7. La fonction précédente remplit des cases du tableau qui ne sont jamais utilisées pour calculer  $\binom{n}{p}$ . La mémorisation et la programmation dynamique top-down permet de supprimer ce défaut. Écrire une fonction `binom3` qui résout le problème par mémorisation en stockant les coefficients intermédiaires dans un dictionnaire.

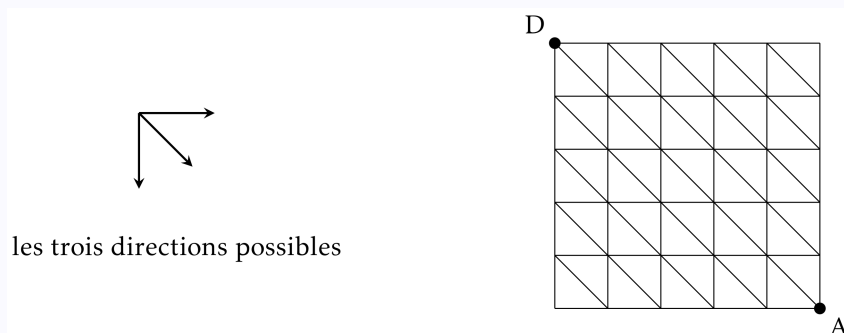
La librairie standard `functools` de Python permet l'utilisation automatique de la mémorisation : il suffit par exemple d'ajouter les lignes

```
from functools import lru_cache
@lru_cache
def binom1(n,p):
    if p == 0 or n == p:
        return ...
    return ...
```

La fonction `binom1` se voit automatiquement associée un dictionnaire pour sauvegarder des valeurs calculées (128 valeurs par défaut). Voir <https://docs.python.org/fr/3/library/functools.html>. Depuis Python 3.9, il existe aussi une fonction `functools.cache`.

**Exercice 2** (Nombre de chemins dans une grille)

Soit une grille de  $N \times N$  cases. On désire compter le nombre de chemins menant du coin en haut à gauche au coin en bas à droite de la grille. Les déplacements ne sont possibles que dans trois directions : vers le bas, vers la droite ou en suivant la diagonale d'une case en descendant vers la droite.



1. En notant  $f(n, p)$  le nombre de chemins possibles dans une grille  $n \times p$ , donner les relations de récurrence vérifiées par  $f$  pour l'utilisation d'une méthode de programmation dynamique.
2. Écrire une fonction `f1` qui calcule  $f(n, p)$  par une récursivité naïve.
3. Écrire une fonction `f2` qui calcule  $f(n, p)$  par la programmation dynamique bottom-up. On doit trouver  $f(20, 20) = 260543813797441$ .
4. En remarquant que seule la dernière ligne du tableau est utilisée, simplifier la fonction précédente pour n'utiliser qu'un tableau à une dimension.
5. Écrire une fonction `f3` qui résout le problème par mémoïsation en stockant les résultats partiels dans un dictionnaire.