# Algorithmique
# TD n° 4 : Programmation dynamique

### Exercice 1 : le triangle de Pascal

On rappelle la récurrence définissant le triangle de Pascal :

$$C_n^p = \binom{n}{p} = \begin{cases} 1 & \text{si } p = 0 \text{ ou } p = n \\ \binom{n-1}{p-1} + \binom{n-1}{p} & \text{sinon.} \end{cases}$$

**1.** Quel est le temps d'exécution du programme ci-contre ? Trouver d'abord une borne supérieure en ignorant le deuxième paramètre $p$. Donner ensuite un encadrement du nombre d'appels récursifs, à $n$ fixé, pour la plus mauvaise valeur de $p$.

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

On considère maintenant les deux programmes ci-dessous, utilisant tous deux un tableau `C` de taille `(N+1)`×`(P+1)` préalablement rempli de 0.

```
def C1(N, P) :
  for n in range(N+1) :
    for p in range(P+1) :
      if p == 0 or n <= p :
        C[n][p] = 1
      else :
      C[n][p] = C[n-1][p-1] + C[n-1][p]
  return C[N][P]
```

```
def C2(n, p) :
  if C[n][p] == 0 :
    if p == 0 or p == n :
      C[n][p] = 1
    else :
      C[n][p] = C2(n-1, p-1) + C2(n-1, p)
  return C[n][p]
```

**2.** Quelle est la complexité du programme `C1` ?

**3.** Effectuer le calcul de `C2(4,2)`.

**4.** Montrer que le programme `C2` calcule bien $\binom{n}{p}$. Avec quelle complexité ?

### Exercice 2 : Road trip

Pour l'été prochain, vous et vos amis planifiez de faire un petit tour en voiture électrique de location dans le pays imaginaire de Simnation. Le trajet est déjà planifié et vous cherchez maintenant à planifier vos arrêts aux bornes de recharge. Vous avez deux objectifs : Ne jamais tomber en panne à cause d'une charge nulle de vos batteries et minimiser le coût de rechargement total. Initialement, la voiture n'est pas chargée et vous pouvez la rendre avec n'importe quel niveau de charge. La première station se trouve là où vous prenez la voiture et la dernière, là où vous la rendez. Votre voiture consomme 0.1 kWh par km et possède une batterie de 10kWh qu'on peut charger avec des paliers de 0.1 kWh.
Vous connaissez par avance la liste des n stations où vous pouvez vous arrêter, la distance qui les sépares ainsi que le prix du kWh en simflouzs :
prix[i] : le prix du kWh pour la station i.
dist[i] : distance en km entre la ville i et la ville i+1.

**1.** Quel est le coût minimal pour prix=[12,14,21,14] et dist=[31,42,31] ?

2. Si on suppose qu'on connaît le coût minimum pour rejoindre la station $i-1$ avec une certaine charge restante, quel est le coût minimum pour joindre la $i$-ème avec une charge restante de $c$ kWh ?
   Plus formellement, soit $cout(i, c)$ le cout minimum pour aller à la station $i$ et avoir une charge de $c$ (après recharge). Donnez une relation de récurrence pour $cout(i, c)$. On suppose que $c$ est donnée en multiple de 0.1 kWh. Quel est le cas de base ?
3. Écrire un algorithme récursif basé sur la relation de récurrence qui calcule le coût total du voyage.
4. Décrire comment réduire la complexité de cet algorithme en utilisant un algorithme de programmation dynamique.
5. Programmez votre algorithme pour les valeurs suivantes :
   Prix (simflouzs/kWh) [12,14,21,14,17,22,11,16,17,12,30,25,27,24,22,15,24,23,15,21]
   Distances (kms) [31,42,31,33,12,34,55,25,34,64,24,13,52,33,23,64,43,25,15]
   Quel sera le coût du road trip ?
6. Comment peut-on récupérer les charges qu'on doit effectuer à chaque station ?

**Exercice 3 : Chocolat empoisoné**

Imaginons un jeu (pas très marrant) à deux joueurs impliquant une plaque de chocolat. La plaque de chocolat a une taille de $H \times L$ carrés et contient un carré empoisonné aux coordonnées $(h, l)$ ($h \in [1, H]$ et $l \in [1, L]$)
Chaque joueur joue alternativement et doit casser la tablette en 2 soit verticalement soit horizontalement puis manger la moitié qui ne contient pas le carré piégé. C'est ensuite à l'autre joueur jusqu'à ce qu'un des deux tombe sur une tablette de taille 1x1 contenant l'unique carré piégé.

1. Prenons un exemple de tablette de taille 3x2 avec le carré piégé en coordonnée (3,1). Qui des deux joueurs va remporter le jeu ?
2. Définissez une récursion qui permet de calculer qui gagne le jeu en supposant que chaque joueur joue au mieux. Pensez à utiliser les opérateurs logiques "ou", "et" et "non".
3. Pouvez-vous écrire un algorithme récursif exponentiel qui calcule si le premier joueur va gagner ?
4. Comment réduire la complexité pour que l'algorithme soit polynomial ? Quelle est sa complexité asymptotique ?
5. Pouvez-vous réécrire l'algorithme de manière non récursive ?
6. Peut-on réduire d'avantage la complexité en tirant partie des symétries ?