Algorithmique
TD n° 4 : Programmation dynamique

Exercice 1 : Sous-suite croissante par programmation dynamique

Etant donné une suite d'entiers dans un tableau de taille n, par exemple $T = [3, 10, 4, 5, 6, 9, 7, 8]$, on veut déterminer la longueur maximale d'une sous-suite croissante ; ainsi dans l'exemple, $(3, 4, 5, 6, 7, 8)$ est une sous-suite de longueur maximum et sa longueur est 6. Dans cet exercice, nous cherchons une solution par programmation dynamique.

1. Y a-t-il une récurrence pour calculer x_i, la longueur maximum d'une sous-suite croissante du tableau consistant en les i premières cases du tableau T ?
2. Donner une récurrence pour calculer y_i, la longueur maximum d'une sous-suite croissante qui se termine par l'élément $T[i]$.
3. Proposer une algorithme par programmation dynamique.
4. Quelle est sa complexité ?
5. Comment récupérer une sous-suite croissante de longueur maximale ?

Exercice 2 : 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 3 :

Super Mario doit traverser un champ de taille $n \times m$, allant de la cellule $[0,0]$ dans le coin supérieur gauche du champ à la cellule $[n-1, m-1]$ dans le coin inférieur droit. Chaque cellule (i,j) contient un bonus $B[i,j]$ gagné si Super Mario passe par cette cellule. La matrice $B[i,j]$, de taille $n \times m$, est une donnée du problème. Les déplacements ne sont autorisés qu'à droite (est) et vers le bas (sud).

Nous notons $V(i,j)$ le bonus total maximum possible qui peut être accumulé pour aller de la cellule $[0,0]$ à la cellule $[i,j]$, c'est-à-dire la somme des bonus des cellules le long du meilleur chemin possible de la cellule $[0,0]$ à la cellule $[i,j]$. Nous visons à calculer la valeur $V(n-1, m-1)$.

1. Donnez une expression récursive pour $V(i,j)$ en fonction de $V(-,-)$ pour deux paramètres ayant des valeurs plus petites que i et j, respectivement.
   Astuce : Considérez les cellules *à partir desquelles* on peut arriver à la cellule $[i-1, j-1]$.
2. Donnez le pseudocode de la fonction récursive qui calcule $V(i,j)$ en utilisant cette formule (la fonction pourra ensuite être appelée pour $i = n-1$ et $j = m-1$).
3. Montrez avec un petit exemple que cette fonction résout plusieurs fois le même sous-problème (*subproblem overlapping*).
4. Donnez le pseudocode de la version optimisée par mémoïsation de la fonction et évaluez sa complexité en temps et en espace.

Exercice 4 :

Nous avons une boîte de Lego avec des pièces ayant toutes la même largeur et la même profondeur, mais ayant des hauteurs variables de toutes les valeurs entières possibles, en théorie une quantité illimitée de pièces de chaque type. Nous voulons calculer le nombre $T(n,k)$ de façons possibles de construire une tour de hauteur n en utilisant k pièces [1]. Par exemple :
Entrée : `n=5; k=3;`
Sortie : `6` (car il existe 6 tours possibles, comme listé dans le tableau ci-dessous) :

[3, 1, 1]	[1, 3, 1]	[1, 1, 3]
[1, 2, 2]	[2, 1, 2]	[2, 2, 1]

1. Donnez une expression récursive pour $T(n,k)$ en fonction de $T(-,-)$ pour deux paramètres ayant des valeurs plus petites que n et/ou k.
   Astuce : Considérez la dernière pièce ajoutée aux tours qui sont une solution.
2. Donnez le pseudocode d'une fonction récursive dérivée de cette formule qui calcule $T(n,k)$.
3. Montrez avec un petit exemple que cette fonction résout plusieurs fois le même sous-problème.
4. Donnez la version optimisée par mémoïsation de la fonction.

Exercice 5 : 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] ?

1. Il s'agit du nombre de compositions de l'entier n ayant k parties.

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 ?