TP 2 INFO — Programmation dynamique

Partie I — Les coefficients binomiaux

Dans cette partie, on souhaite calculer les coefficients binomiaux en utilisant les valeurs initiales

$$\forall (n,p) \in \mathbb{N}^2, \quad c(n,p) = \begin{cases} 1 & \text{si} \quad p = 0 \\ 0 & \text{si} \quad n = 0 \text{ et } p > 0 \end{cases}$$

et la formule de Pascal suivante

$$\forall (n,p) \in \mathbb{N}^* \times \mathbb{N}^*, \quad c(n,p) = c(n-1,p) + c(n-1,p-1).$$

Question 1 : Écrire une fonction utilisant la méthode descendante `bin_desc(n:int, p:int) -> int` qui prend en argument un couple d'entiers naturels `(n,p)` et renvoie le coefficient binomial d'indice `(n,p)`.

Remarque 1 : Vous pourrez utiliser votre fonction pour vérifier que $c(30,15) = 155\,117\,520$.

Question 2 : Justifier la terminaison de votre fonction `bin_desc`.

Dans la suite, on implémente le calcul du coefficient binomial avec la méthode ascendante.

Question 3 : Déterminer la valeur de $c(5,3)$ en utilisant le tableau ci-dessous.

	$j=0$	$j=1$	$j=2$	$j=3$
$i=0$				
$i=1$				
$i=2$				
$i=3$				
$i=4$				
$i=5$				

Question 4 : Écrire une fonction utilisant la méthode ascendante `bin_asc(n:int, p:int) -> int` qui prend en argument un couple d'entiers naturels `(n,p)` et renvoie le coefficient binomial d'indice `(n,p)`. Votre fonction devra avoir une complexité temporelle et spatiale en $O(np)$.

Question 5 : Quelles sont les cases réellement utiles dans le tableau pour le calcul de $c(n,p)$?

Question 6 : Déduire de la question précédente une amélioration de votre fonction `bin_asc` pour que sa complexité temporelle soit en $O(p(n-p))$ et sa complexité spatiale soit en $O(\min(p, n-p))$ lorsque $0 \leqslant p \leqslant n$.

Partie II Le problème du sac à dos

Étant donné une liste finie d'objets ayant chacun un poids et une valeur, le problème du sac à dos consiste à remplir un sac à dos en maximisant la valeur de son contenu tout en respectant le poids maximal qu'il peut supporter.

Plus formellement, les objets sont représentés par des couples $(p_1, v_1), \dots, (p_n, v_n) \in \mathbb{N}^* \times \mathbb{N}^*$ indiquant respectivement leur poids et leur valeur. Si on désigne par $C \in \mathbb{N}$ la capacité du sac, l'ensemble des manières de le remplir est décrit par

$$\mathscr{S} = \left\{ I \subset [\![1, n]\!] \;\middle|\; \sum_{i \in I} p_i \leqslant C \right\}.$$

Le problème du sac à dos revient ainsi à maximiser la fonction $\varphi : \mathscr{S} \to \mathbb{N}$ définie par

$$\varphi : I \mapsto \sum_{i \in I} v_i.$$

Exemple 1 : On considère un sac à dos de capacité $C = 10$ et les $n = 5$ objets

$$(p_1, v_1) = (2,2), \quad (p_2, v_2) = (3,4), \quad (p_3, v_3) = (3,7), \quad (p_4, v_4) = (5,8), \quad (p_5, v_5) = (7,11).$$

On peut remplir le sac avec les objets indiqués par $I = \{1,5\}$: on obtient une valeur totale de $v_1 + v_5 = 13$ pour un poids total de $p_1 + p_5 = 9 \leqslant C$. Cependant, on peut aisément vérifier que la valeur 9 n'est pas maximale.

En Python, on indiquera les objets considérés avec une liste comme ci-dessous.

```
lst_objets = [(2,2), (3,4), (3,7), (5,8), (7,11)]
```

II..1 - Résolution du problème avec la programmation dynamique

Pour résoudre ce problème en utilisant la programmation dynamique, on note $V(k,c)$ la valeur maximale que l'on peut obtenir en remplissant un sac à dos de capacité c avec des objets choisis parmi $(p_1, v_1), \dots, (p_k, v_k)$.

Question 7 : Déterminer les valeurs initiales $V(k,0)$ et $V(0,c)$ pour $k \in [\![0, n]\!]$ et $c \in [\![0, C]\!]$.

Question 8 : Étant donné deux entiers $k \in [\![1, n]\!]$ et $c \in [\![1, C]\!]$, justifier que l'on a la relation

$$V(k,c) = \begin{cases} \max\big(V(k-1,c), v_k + V(k-1, c-p_k)\big) & \text{si} \quad p_k \leqslant c \\ V(k-1,c) & \text{si} \quad p_k > c. \end{cases}$$

Question 9 : En remplissant le tableau ci-dessous, déterminer la valeur maximale que peut contenir le sac à dos avec les paramètres de l'exemple 1.

	$c=0$	$c=1$	$c=2$	$c=3$	$c=4$	$c=5$	$c=6$	$c=7$	$c=8$	$c=9$	$c=10$
$k=0$											
$k=1$											
$k=2$											
$k=3$											
$k=4$											
$k=5$											

Question 10 : Déduire des informations contenues dans le tableau précédent la solution au problème du sac à dos avec les paramètres de l'exemple 1.

Question 11 : Écrire une fonction utilisant la méthode ascendante `val_max(lst_objets:list, C:int)` prenant en argument la liste des objets `lst_objets` et la capacité du sac à dos `C` et renvoie un tableau contenant toutes les valeurs $V(k,c)$ pour $(k,c) \in [\![0,n]\!] \times [\![0,C]\!]$.

Question 12 : Évaluer la complexité temporelle et spatiale de votre fonction `val_max`.

Question 13 : Écrire une fonction `choix_obj(lst_objets:list, C:int) -> list` qui prend en argument la liste des objets `lst_objets` et la capacité du sac à dos `C` et renvoie la liste des objets à mettre dans le sac pour maximiser la valeur du contenu du sac.

Question 14 : Réécrire les fonctions `val_max` et `choix_obj` en utilisant cette fois-ci la méthode descendante. En particulier, la fonction `val_max` ne renverra plus un tableau, mais le dictionnaire utilisé par la méthode descendante.

II..2 - Résolution approchée du problème avec la programmation gloutonne

Lorsque les paramètres du problème sont trop grands, la résolution du problème via la programmation dynamique peut devenir inefficace. Dans ce cas, on peut opter pour l'utilisation de la stratégie gloutonne suivante : on place prioritairement dans le sac (si c'est possible) l'objet dont le quotient v_k/p_k est maximal.

Exemple 2 : En triant dans l'ordre décroissant les objets de l'exemple 1 selon le quotient v_k/p_k, on obtient

$$(p_3,v_3)=(3,7), \quad (p_4,v_4)=(5,8), \quad (p_5,v_5)=(7,11), \quad (p_2,v_2)=(3,4), \quad (p_1,v_1)=(2,2)$$

$$\frac{v_3}{p_3}\approx 2{,}33 \qquad \frac{v_4}{p_4}\approx 1{,}6 \qquad \frac{v_5}{p_5}\approx 1{,}57 \qquad \frac{v_2}{p_2}\approx 1{,}33 \qquad \frac{v_1}{p_1}\approx 1$$

On en déduit que l'application de l'algorithme glouton aux paramètres de l'exemple 1 indique de remplir le sac à dos avec les objets $[(3,7),(5,8),(2,2)]$. La valeur du sac à dos pour cette solution est $v_3+v_4+v_1=17$, ce qui est légèrement inférieur à la valeur de la solution optimale.

Pour trier la liste des objets comme dans l'exemple ci-dessus, on pourra utiliser le code Python ci-dessous.

```
def cle_de_tri(obj):
    return(obj[1]/obj[0])

lst_objets_triee = sorted(lst_objets, key=cle_de_tri, reverse=True)
```

Question 15 : Écrire une fonction `sac_glouton(lst_objets:list, C:int) -> list` qui renvoie la liste des objets à mettre dans le sac en mettant en œuvre la stratégie gloutonne décrite ci-dessus.

Question 16 : Évaluer la complexité temporelle et spatiale de votre fonction `sac_glouton`.

Partie III Plus longue sous-séquence commune

Une sous-séquence d'un mot $a = a_1 \cdots a_p$ est un mot de la forme $a_{i_1} \cdots a_{i_r}$ avec $1 \leqslant i_1 < \cdots < i_r \leqslant p$.

Exemple 3 : Les sous-séquences du mot `rare` sont $\varnothing$, `r`, `a`, `e`, `ra`, `rr`, `re`, `ar`, `ae`, `rar`, `rae`, `rre`, `are`, `rare`.

Dans cette partie, étant donné deux mots $a = a_1 \cdots a_p$ et $b = b_1 \cdots b_q$, on souhaite déterminer la plus longue sous-séquence commune à ces deux mots.

Exemple 4 : La plus longue sous-séquence commune des mots `informatique` et `ordinateur` est `inatu`.

Pour résoudre ce problème en utilisant la programmation dynamique, on note $d(i,j)$ la longueur de la plus grande sous-séquence commune entre les mots $a_1 \cdots a_i$ et $b_1 \cdots b_j$. En particulier, le nombre cherché est $d(p,q)$.

Question 17 : Déterminer les valeurs initiales $d(i,0)$ et $d(0,j)$ pour $i \in [\![0,p]\!]$ et $j \in [\![0,q]\!]$.

Question 18 : Étant donné deux entiers $i \in [\![1,p]\!]$ et $j \in [\![1,q]\!]$, justifier que l'on a la relation

$$d(i,j) = \begin{cases} \max\big(d(i,j-1), d(j-1,i)\big) & \text{si} \quad a_i \neq b_j \\ d(i-1,j-1)+1 & \text{si} \quad a_i = b_j. \end{cases}$$

Question 19 : En remplissant le tableau ci-dessous, déterminer la longueur de la plus longue sous-séquence commune entre les mots `rare` et `rapide`.

		$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
		$\varnothing$	r	a	p	i	d	e
$i=0$	$\varnothing$							
$i=1$	r							
$i=2$	a							
$i=3$	r							
$i=4$	e							

Question 20 : Déduire des informations contenues dans le tableau précédent la plus longue sous-séquence commune entre les mots `rare` et `rapide`.

Question 21 : Écrire une fonction `long_max(a:str, b:str)` qui prend en argument deux mots `a` et `b` et renvoie un tableau contenant toutes les valeurs $d(i,j)$ pour $(i,j) \in [\![0,p]\!] \times [\![0,q]\!]$.

Question 22 : Écrire une fonction `ss_seq_max(a:str, b:str) -> str` qui prend en argument deux mots `a` et `b` et renvoie une sous-séquence commune de longueur maximale de `a` et `b`.