TP D'INFORMATIQUE N°4
Programmation dynamique

1 Problème du sac-à-dos

Dans ce problème, on considère n objets, de poids entiers $(p_i)_{0\leq i<n}$ et de valeurs flottantes $(v_i)_{0\leq i<n}$, ainsi qu'un poids entier maximal $Pmax$ correspondant au poids que peut contenir le sac-à-dos. On cherche alors à trouver quels objets prendre pour maximiser la somme des valeurs, tout en respectant la contrainte de ne pas dépasser un poids de $Pmax$.

1.1 Relation de récurrence

On note $m_{i,pr}$ la valeur cumulée maximale en choisissant les objets parmi les i premiers (ie pour les indices de 0 à $i-1$), sous la contrainte que leur poids ne dépasse pas pr (poids restant). Justifier, pour $pr \in [\![0, Pmax]\!]$ et $i \in [\![0, n-1]\!]$:

1. $m_{0,pr} = 0$
2. $m_{i+1,pr} = m_{i,pr}$ si $p_i > pr$
3. $m_{i+1,pr} = \max(m_{i,pr}, m_{i,pr-p_i} + v_i)$ si $p_i \leq pr$

1.2 Implémentation ascendante

1. Écrire une fonction `sac_ascendant` prenant en argument un tuple de poids p, un tuple de valeurs v et un poids entier $Pmax$, et renvoyant la valeur cumulée maximale définie par le problème, en implémentant la méthode du calcul de bas en haut.
2. On remarque qu'une ligne de la matrice où l'on stocke les $m_{i,p}$ n'a besoin que de la ligne précédente pour être calculée. Il est donc possible d'optimiser le coût en espace en ne stockant que la dernière ligne. Il faut alors à chaque itération de la boucle principale remplir cette ligne de la droite vers la gauche. Écrire la fonction `sac_ascendant_opti` mettant en oeuvre cette optimisation.

1.3 Implémentation descendante

1. Écrire une fonction `sac_descendant_aux` prenant en argument $p, v, Pmax$, mais également un entier i, et renvoyant la valeur cumulée maximale **en ne considérant que les i premiers objets**. On utilisera la méthode du calcul de haut en bas, en mémoïsant les résultats des appels dans un dictionnaire.
2. En déduire une fonction `sac_descendant` prenant en argument p, v et $Pmax$ et renvoyant la valeur cumulée maximale définie par le problème.
3. On considère l'exemple suivant :
   - `p = (23, 26, 20, 18, 32, 27, 29, 26, 30, 27)`
   - `v = (505, 352, 458, 220, 354, 414, 498, 545, 473, 543)`
   - `Pmax = 67`

   Déterminer le nombre de $m_{i,p}$ calculés lors du calcul de haut en bas sur cet exemple, et le comparer à celui du calcul de bas en haut.

1.4 Reconstruction de la solution détaillée

1. Écrire une variante `sac_ascendant_detail` de `sac_ascendant` renvoyant la liste des indices des objets à prendre dans le sac, plutôt que leur valeur cumulée.
2. Écrire de même `sac_descendant_detail` (on se permettra de mémoïser toutes les listes correspondant aux résultats intermédiaires).

2 Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall permet de calculer la distance minimale entre chaque couple de sommets dans un graphe orienté pondéré. Contrairement à l'algorithme de Dijkstra, il ne considère donc pas un seul sommet source. De plus, il reste correct en présence de poids négatif (mais pas en présence d'un cycle de poids négatif, puisque la notion de distance n'est alors plus définie).

2.1 Relation de récurrence

On note M la matrice d'adjacence du graphe considéré, où une absence d'arc entre les sommets u et v se traduit par un coefficient $+\inf$ dans la case $M_{u,v}$.
On note $M_{k,u,v}$ le poids minimum d'un chemin allant du sommet u au sommet v dont les sommets intermédiaires (les sommets qui ne sont ni le sommet de départ u ni le sommet d'arrivée v) sont tous strictement inférieurs à k.

1. Pour quelle valeur de k la matrice $(M_{k,u,v})_{0\leq u,v<n}$ contient-elle les distances recherchées ?
2. Tracer au papier le graphe dont la matrice d'adjacence est :

```
inf = float("inf")
G = [[inf, inf, 0, inf], [-1, inf, inf, 2], [inf, 1, inf, 2], [inf, inf, inf, inf]]
```

3. Déterminer sur cet exemple les coefficients $M_{k,u,v}$, pour $0 \leq k \leq 4, 0 \leq u < 4, 0 \leq v < 4$.
4. Déterminer dans le cas général la valeur de $M_{0,u,v}$ pour u et v deux sommets.
5. Déterminer dans le cas général une relation de récurrence exprimant $M_{k,u,v}$ en fonction de $M_{k-1,u,v}, M_{k-1,k-1,v}$ et $M_{k-1,u,k-1}$ pour $k \geq 1$, u et v deux sommets.

2.2 Implémentation

1. Écrire une fonction `floyd_warshall` prenant en argument un graphe sous forme de matrice d'adjacence, et calculant la distance entre chaque couple de sommets en utilisant la relation de récurrence précédente.
2. Déterminer les complexités temporelle et spatiale en fonction de n le nombre de sommets du graphe.
3. Écrire une version optimisée ayant une complexité spaciale en $O(n^2)$ (on pourra remarquer que pour le calcul de $M_{u,v}$ vérifiant $\delta(u,v) \leq M_{u,v} \leq M_{k,u,v}$, on peut garder l'ordre de remplissage de la version no optimisée).
4. Écrire une variante de la fonction précédente renvoyant également une matrice P telle que $P_{u,v}$ contienne le prédécesseur de v dans le plus court chemin allant de u à v, ou -1 s'il n'existe pas de chemin de u à v.
5. Écrire une fonction `chemin` prenant une telle matrice P et deux sommets u et v, et renvoyant un plus court chemin de u à v sous forme de liste.