Algorithmes d'approximation et programmation dynamique

Question 1. Problème du stockage

Considérons n programmes $P_1, P_2, \ldots, P_n$ qui peuvent être stocker sur un disque dur de capacité D gigabytes.

— Chaque programme P_i a besoin s_i gigabytes pour être stocké et a une valeur v_i

— Tous les programmes ne peuvent pas être stockés sur le disque : $\sum_{i=1}^{n} s_i > D$.

Les programmes stockés dans le disque dur doivent maximiser la valeur totale, sans dépasser la capacité du disque dur. L'objectif est de concevoir un algorithme qui permet de calculer un tel ensemble.

Nous allons construire un tableau T dans lequel les lignes seront indexées par les programmes et les colonnes par les valeurs. L'élément $T[i, j]$ représentera la valeur maximale pour un disque dur de capacité j à l'aide des i premiers programmes.

Question 1.1 Donner la formule de récurrence.

Question 1.2 Donner l'algorithme utilisant la programmation dynamique.

Question 1.3 Donner la complexité de cet algorithme.

Remarque : Le problème de stockage peut se formuler sous forme du problème du Sac à Dos qui est un problème classique en informatique. Il modélise une situation analogue au remplissage d'un sac. Une personne veut remplir un sac à dos ne pouvant pas supporter plus d'un certain poids $D \in \mathbb{N}$, et elle dispose de n objets (On note l'ensemble des objects par $\mathcal{P} = \{1, \ldots, n\}$). Chaque objet i a une valeur $v_i \in \mathbb{N}\backslash\{0\}$ et un poids $s_i \in \mathbb{N}\backslash\{0\}$. Le problème est de trouver un ensemble d'objets tels que

— tous les objets de cet ensemble puissent être mis dans le sac.

— la somme des valeurs de ces objets soit maximale.

Question 2. Problème Le chemin le plus long dans un graphe

Soit $G = (V, E)$ un graphe orienté avec $V = \{v_1, \ldots, v_n\}$. On dit que G est ordonné si il vérifie les deux propriétés suivantes :

1. Chaque arc de ce graphe est de la forme $(i \to j)$ si $i < j$
2. Tous les sommets sauf le sommet v_n ont au moins un arc sortant.

Ici, par souci de simplification, nous supposerons qu'il existe un chemin allant de v_i vers v_n pour tout $i = 1, \ldots, n$.

L'objectif est de trouver le chemin le plus long entre les sommets v_1 et v_n.

Question 2.1 Montrer que l'algorithme glouton suivant ne résouds pas correctement le problème.

1. $u \leftarrow v_1$;
2. $L \leftarrow 0$;
3. Tant qu'il existe un arc sortant du sommet u

(a) choisir l'arc $(u \to v_j)$ tel que j est le plus petit possible

(b) $u \leftarrow v_j$;

(c) $L \leftarrow L + 1$;

4. retourner L

Question 2.2 Donner la formule de récurrence qui permet de calculer la longueur du chemin le plus long commençant par v_1 finissant par v_ℓ.

Question 2.3 Donner un algorithme qui retourne la longueur du chemin le plus long commençant par v_1 finissant par v_n.

Question 2.4 Modifier l'algorithme précédent afin qu'il retourne le chemin.

Nous allons considérer les graphes orientés et ordonnés possédant une fonction de poids w sur les arcs $w : E \to \mathbb{N}^+$.

L'objectif est de trouver le poids du chemin de poids maximal commençant par v_1 finissant par v_n (si il n'en existe pas, la valeur retournée doit être égale à $-\infty$). Formellement, on veut trouver $p_{max} = max\{\sum_{e\in\mathcal{P}} w(e) : \mathcal{P}$ est un chemin de v_1 à $v_n\}$

Question 2.5 Donner la formule de récurrence permettant de calculer le chemin de poids maximum commençant par v_1 finissant par v_n.

Question 2.6 En déduire l'algorithme.

Maintenant, l'objectif est de trouver le poids du chemin de poids maximal commençant par v_1 composé de k arcs. Nous utiliserons un tableau T dont la valeur de l'élément $T[i, \ell]$ correspond au poids du chemin de poids maximal commençant par v_1 finissant par v_i et composé de ℓ arcs.

Question 2.7 Donner la formule de récurrence qui permet de calculer le chemin de poids maximal de ℓ arcs commençant par v_1 finissant par v_i . En déduire l'algorithme.

Problème Sac à Dos.

Le problème du Sac à Dos est un problème classique en informatique. Il modélise une situation analogue au remplissage d'un sac. Une personne veut remplir un sac à dos ne pouvant pas supporter plus d'un certain poids $C \in \mathbb{N}$, et elle dispose de n objets (On note l'ensemble des objects par $\mathcal{O} = \{1, \ldots, n\}$). Chaque objet i a une valeur $v_i \in \mathbb{N}\backslash\{0\}$ et un poids $p_i \in \mathbb{N}\backslash\{0\}$. Le problème est de trouver un ensemble d'objets tels que

— tous les objets de cet ensemble puissent être mis dans le sac.

— la somme des valeurs de ces objets soit maximale.

Le problème d'optimisation correspond à trouver un sous-ensemble d'objets dont le poids total est inférieure à C et dont la valeur totale soit maximum. **Notons $\mathbf{v} = \mathbf{max}\{\mathbf{v_i} : \mathbf{i} \in \mathcal{O}\}$.**

Rappelons l'algorithme de la programmation dynamique. Notons $T[i, j]$ représentera la valeur maximale pour un sac à dos de capacité j à l'aide des i premiers objets.

Entrée : un ensemble d'objects $\mathcal{O} = \{1, \ldots, n\}$. L'objet i a une valeur v_i et un poids p_i.

Sortie : un entier

1. Initialiser tous les éléments du tableau T à zéro.
2. Pour tout i allant de 1 à n faire
   (a) Pour tout j allant de 1 à C faire

$$\text{Si } j < v_i, \quad \text{alors } T[i,j] = T[i-1,j] \\ \text{sinon } T[i,j] = max(T[i-1,j], T[i-1, j-p_i] + v_i)$$

3. Retourner $T[n, C]$

Question 2.8 Donner la complexité de cet algorithme. Peut-on modifier l'algorithme de telle façon que sa complexité soit exprimée en fonction de v ?

Maintenant, considérons l'algorithme suivant :

1. Etant donné $\varepsilon > 0$, $K \leftarrow \frac{\varepsilon \cdot v}{n}$
2. Pour chaque objet i, $v'_i = \lfloor \frac{v_i}{K} \rfloor$
3. Lancer l'algorithme de programmation dynamique avec ces valeurs arrondies v' et trouver un ensemble S' dont la valeur (v') totale soit maximum.
4. Renvoyer S'.

Question 2.9 Donner la complexité de cet algorithme.

Notons OPT le coût de la solution optimale.

Question 2.10 Montrer que $\sum_{i \in S'} v_i \geq (1 - \varepsilon) \cdot OPT$

Question 2.11 Qu'en concluez-vous ?

Problème Sac à Dos.

Le problème du Sac à Dos est un problème classique en informatique. Il modélise une situation analogue au remplissage d'un sac. Une personne veut remplir un sac à dos ne pouvant pas supporter plus d'un certain poids $C \in \mathbb{N}$, et elle dispose de n objets (On note l'ensemble des objects par $\mathcal{O} = \{1, \ldots, n\}$). Chaque objet i a une valeur $v_i \in \mathbb{N}\backslash\{0\}$ et un poids $p_i \in \mathbb{N}\backslash\{0\}$. Le problème est de trouver un ensemble d'objets tels que

— tous les objets de cet ensemble puissent être mis dans le sac.

— la somme des valeurs de ces objets soit maximale.

Question 3. Complexité du problème Sac à Dos

Montrer que le problème Sac à Dos est faiblement NP-complet sachant que le problème SubsetSum est faiblement NP-complet.

Question 4. Programmation dynamique

Nous allons construire un tableau T dans lequel les lignes seront indexées par les objets et les colonnes par les valeurs. L'élément $T[i, j]$ représentera la valeur maximale pour un sac à dos de capacité j à l'aide des i premiers objets.

Question 4.1 Donner la formule de récurrence.

Question 4.2 Donner l'algorithme utilisant la programmation dynamique.

Question 4.3 Donner la complexité de cet algorithme.

Plus court chemin dans un graphe orienté et ordonné

Un graphe $G = (V, A)$ est dit *ordonné* s'il vérifie la propriété suivante : il existe une numérotation L des sommets de V de 0 à $n-1$ telle que pour tout arc $(u \to v)$ de G, on a $L(u) < L(v)$.

Un chemin C de s allant à t est dit *un plus court chemin* s'il est de plus petit longueur. La *distance* de u vers v est la longueur du plus court chemin de u vers v. Remarquons, nous supposerons que la distance de u vers v est égale à $+\infty$ si il n'existe pas de chemin de u vers v.

Question 5. Considérons le graphe orienté G_2 suivant :

Question 5.1 Montrer que le graphe G_2 est ordonné en donnant une numération de sommets.

Question 5.2 Donner un plus court chemin allant du sommet u_0 au sommet v_3.

Question 5.3 Enumérer (lister) les plus courts chemins allant du sommet u_0 au sommet v_3.

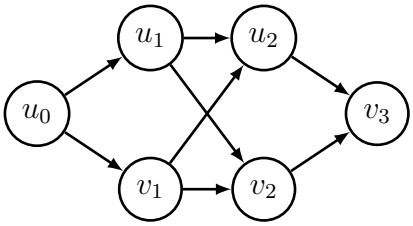

FIGURE 1 – Graphe G_2

Question 6. Nous allons considérer un graphe $\mathcal{G}_k = (V_k, A_k)$ tel que

— $V_k = \{u_0, u_1, \ldots, u_k\} \cup \{v_1, \ldots, v_k, v_{k+1}\}$

— $A_k = \{(u_i \to u_{i+1}), (u_i \to v_{i+1}), (v_i \to u_{i+1}), (v_i \to v_{i+1}) : 1 \leq i < k\}$
$\cup\{(v_k \to v_{k+1}), (u_k \to v_{k+1}), (u_0 \to v_1), (u_0 \to u_1)\}$

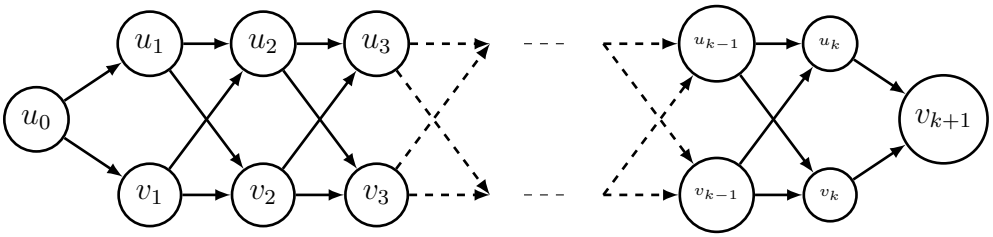

FIGURE 2 – Le graphe $\mathcal{G}_k$

Question 6.1 Montrer que le graphe $\mathcal{G}_k$ est ordonné.

Question 6.2 Compter le nombre de plus court chemin allant u_0 à v_{k+1} dans le graphe $\mathcal{G}_k$ en fonction de k. Exprimer ce nombre aussi en fonction du nombre n de sommets.

Par la suite, nous supposerons que G est un graphe orienté et ordonné. Nous supposerons par la suite que V sera numéroté de telle façon que $V = \{0, 1, \ldots, n-1\}$ et chacun des arcs de G est de la forme $(u \to v)$ avec $u < v$.

Les prochaines questions aurons pour objectif de construire les plus court chemin allant du sommet 0 vers le sommet v. Soit $d : V \to \mathbb{N}$ un tableau d'entiers. Considérons l'algorithme suivant :
Entrée : un graphe orienté et ordonné $G = (V, A)$ tel que $|V| = n$.
Sortie : un tableau d d'entiers de n éléments

1. $d[0] \leftarrow 0$;
2. pour tout v allant de 1 à $n - 1$ faire $d[v] \leftarrow +\infty$;
3. pour tout v allant de 1 à $n - 1$ faire
   3.1. pour tout sommet u $(u < v)$ tel qu'il existe un arc $(u \to v)$ dans G, faire
   si ($d[u] + 1 < d[v]$) alors $d[v] \leftarrow d[u] + 1$;
4. retourner d ;

Question 7. Evaluer la complexité de cet algorithme en considérant que le graphe est représenté par une *matrice d'adjacence*. La *matrice d'adjacence* du graphe G est une matrice M $n \times n$ telle que : $M[i, j] = \begin{cases} 1 & si \quad (i \to j) \in A \\ 0 & \quad sinon \end{cases}$.

Question 8. Montrer que l'élément $d[i]$ du tableau d retourné par l'algorithme, est la distance entre le sommet 0 et le sommet i.

Question 9. Modifier l'algorithme précédent afin qu'il retourne, pour chaque sommet v du graphe, un plus court chemin entre le sommet 0 et v.

Question 10. Dire pourquoi énumérer tous les chemins entre deux sommets 0 ne peut pas se faire en temps polynomial.

Question 11. Concevoir un algorithme polynomial qui calcule le nombre de plus courts chemins entre le sommet 0 et tous les autres sommets du graphe orienté et ordonné. Evaluer sa complexité.

Question 12. [1] Concevoir un algorithme polynomial qui retourne vrai si il existe un chemin composé d'exactement k arcs allant de 0 vers le sommet s, sinon il retourne faux. Evaluer sa complexité.

Question 13. Concevoir un algorithme qui permet de savoir si un graphe orienté est ordonné. Evaluer sa complexité.

1. difficile