TD 2 - Algorithmes gloutons, programmation dynamique

Exercice 1 – Ensemble indépendant maximum

On rappelle qu'un ensemble indépendant dans un graphe $G = (V, E)$ est un sous-ensemble I de sommets deux à deux non reliés par des arêtes.

Pour déterminer un ensemble indépendant le plus grand possible, on considère l'algorithme glouton suivant. On initialise $I = \varnothing$, et pour chaque $v \in V$, on l'ajoute à I s'il n'a aucun voisin dans I.

1. Montrer que l'ensemble I obtenu est un ensemble indépendant maximal (par inclusion).
2. Est-ce toujours un ensemble indépendant maximum ?

Exercice 2 – Coloration gloutonne

On considère l'algorithme glouton de coloration suivant : on choisit un ordre $v_1, \ldots, v_n$ des sommets de G, puis on colore dans l'ordre croissant en donnant à v_i la plus petite couleur disponible (c'est-à-dire qui n'apparaît sur aucun voisin de v_i).

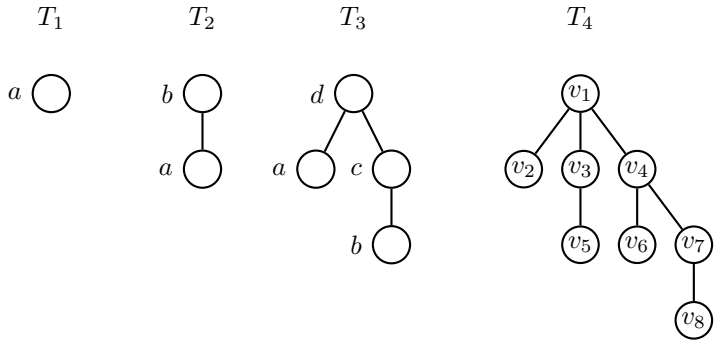

FIGURE 1 – Les graphes $T_1, \ldots, T_4$.

1. Colorer les graphes T_1, T_2, T_3 de la Figure 1 dans l'ordre alphabétique des sommets (directement sur la figure). La coloration gloutonne est-elle toujours optimale ? Autrement dit, minimise-t-elle toujours le nombre de couleurs utilisées ?
2. Soit $v_1, \ldots, v_n$ l'ordre des sommets choisi par l'algorithme. Montrer par récurrence que la plus grande couleur utilisée par l'algorithme glouton est au plus $\Delta + 1$, où Δ est le degré maximum du graphe (i.e. le maximum sur tous les sommets v du nombre de sommets adjacents à v).
3. Soit T_4 le dernier graphe de la Figure 1. Trouver un ordre sur les sommets de T_4 de façon à ce que le sommet v_1 reçoive la couleur 4.
4. En vous inspirant de la construction de T_4, proposer une construction d'un arbre T_i et d'un ordre sur T_i tel qu'il existe un sommet de T_i soit coloré i avec l'algorithme glouton.
   Quel est le degré maximum de votre graphe T_i ?

Exercice 3 – Couverture par segments

Soient $X = \{x_1, \ldots, x_n\}$ un ensemble de n points de $\mathbb{R}$ ordonnés de manière croissante.

1. Proposer un algorithme glouton qui détermine un nombre minimum d'intervalles de longueur 1 qui contient tous les points.
2. Prouver l'optimalité de l'algorithme et sa complexité.

Exercice 4 – Suite récurrente

Étant donnés deux entiers $0 \leqslant k \leqslant n$, on définit la suite

$$u_{k,n} = \begin{cases} 1 & \text{si } k \in \{0, n\} \\ u_{k,n-1} + u_{k-1,n-1} & \text{sinon.} \end{cases}$$

1. Écrire un algorithme permettant de calculer $u_{k,n}$ en temps $O(n^2)$.
2. Adapter votre algorithme pour qu'il n'utilise que $O(n)$ cases mémoires à tout moment.

Exercice 5 – Chemins dans une grille

On s'intéresse au problème suivant. Étant donné un tableau bidimensionnel d'entiers, on veut déterminer quel est le poids maximum d'un chemin allant de la case $(0, 0)$ à la case $(n-1, n-1)$ en ne se déplaçant que vers la droite ou vers le bas. Par exemple, sur la grille suivante, un des chemins solution (de poids 40) est en gras :

2	8	10	5
7	3	1	7
5	**7**	**9**	3
8	2	**6**	**4**

On définit $C_{i,j}$ le poids maximum d'un tel chemin entre la case $(0, 0)$ et (i, j).

1. Donner une relation de récurrence vérifiée par $C_{i,j}$, ainsi que les cas de base.
2. En déduire un algorithme de programmation dynamique permettant de résoudre le problème.
3. Quelles sont ses complexités spatiales et temporelles ?

Exercice 6 – Distance d'édition

Quand un correcteur orthographique trouve une erreur dans un texte, il cherche dans son dictionnaire le mot le plus proche. Pour mesurer la distance entre deux mots, on cherche à les *aligner*. Par exemple, voici deux alignements :

S	-	N	O	W	Y
S	U	N	N	-	Y

-	S	N	O	W	-	Y
S	U	N	-	-	N	Y

Le symbole - indique que l'on n'avance pas dans le mot. Le coût d'un alignement est défini comme le nombre de colonnes avec une différence. La *distance d'édition* entre deux mots est le coût minimum d'un alignement. Elle correspond au nombre minimum d'insertions, suppression ou modification de caractère pour passer d'un mot à l'autre. Par exemple, l'alignement de gauche correspond à insérer un U, modifier un O en N et supprimer un W. On a donc fait 3 éditions. Dans l'exemple de droite, on en a fait 4.

Étant donnés deux mots x, y de taille n et m, on cherche un algorithme de programmation dynamique permettant de calculer la distance entre eux.

1. Identifier les bons sous-problèmes et expliciter la relation de récurrence permettant de résoudre le problème.
2. Exécuter votre algorithme sur les mots SNOWY et SUNNY.
3. Quelle est la complexité de votre algorithme ?
4. Comment retrouver un alignement permettant d'atteindre la distance d'édition ?

Exercice 7 – Plus long sous-mot commun

Un sous-mot d'un mot X est un ensemble de lettres de X consécutives. Par exemple, les mots $A = abbaba$ et $B = bbbaab$ contiennent tous les deux le sous-mot bba. Par contre aab est un sous-mot de B mais n'est pas un sous-mot de A. Le but est de trouver le plus long sous-mot commun qui apparaît à la fois dans le mot $A = a_1 \cdots a_n$ et dans le mot $B = b_1 \cdots b_m$.

On note $M[i, j]$ la longueur du plus long sous mot commun qui se termine en a_i dans A et b_j dans B. Par convention, on note $M[i, 0] = 0$ et $M[0, i] = 0$.

1. Donner la valeur de $M[i, j]$ si a_i et b_j sont différents.
2. Donner une équation de récurrence satisfaite par $M[i, j]$ si $a_i = b_j$.
3. Comment déterminer la longueur du plus long sous-mot commun entre A et B étant donné l'ensemble des valeurs $M[i, j]$?

Exercice 8 – Péages

Un automobiliste se trouve sur une autoroute à n sorties. À chaque sortie se trouve un péage. Pour chaque paire (i, j), l'automobiliste a accès au prix $c_{i,j}$ à payer si on entre sur l'autoroute au i-ème péage et qu'on en sort au j-ème, sans sortir entre temps. Par exemple, la Figure 1 représente les prix de l'autoroute A7 entre Auberives et Valence nord.

FIGURE 2 – Prix des péages sur une portion de l'A7. Un automobiliste entrant à Auberives et sortant à Tain (sans quitter l'autoroute à Chanas) payera 3.2€.

L'automobiliste doit traverser l'autoroute d'un bout à l'autre. Pour optimiser ses dépenses, il s'autorise à sortir puis re-rentrer immédiatement à une ou plusieurs sorties sur son trajet.

1. Quel est le coût optimal d'un trajet entre Auberives et Valence nord ?
2. On définit M_i comme le coût optimal d'un trajet entre le début de l'autoroute et la sortie i. Donner une relation de récurrence sur M_i.
3. En déduire un algorithme de programmation dynamique permettant de résoudre le problème. Quelle est sa complexité ?
4. Comment peut-on retrouver l'itinéraire de coût minimal ?
5. De manière alternative, ce problème peut-être modélisé par un graphe et résolu grâce à un des algorithmes vus en cours. De quel algorithme s'agit-il, et comment effectuer cette modélisation ?

Exercice 9 – Problème du sac à dos

Pendant un cambriolage, un voleur trouve plus de butin que ce à quoi il s'attendait, et doit décider quoi emporter. Son sac peut supporter une charge de P kilogrammes. Le voleur peut choisir son butin parmi n objets, de poids $p_1, \ldots, p_n$ et de valeurs $v_1, \ldots, v_n$.

Le but de cet exercice est de déterminer (avant l'arrivée de la police) quelle est le butin le plus cher qu'il pourra transporter dans son sac.

1. On considère l'exemple suivant :

Objet	Poids (kg)	Valeur (€)
1	6	30
2	3	14
3	4	16
4	2	9

   Résoudre cette instance quand $P = 10$kg.
2. On considère l'algorithme glouton suivant : tant qu'il y a de la place dans le sac pour au moins un objet, on ajoute l'objet le plus cher qui rentre dans le sac. Donner une instance pour laquelle cet algorithme ne renvoie pas une solution optimale.
3. On note $f(P, j)$ la valeur optimale transportable dans un sac de capacité P quand on se restreint aux j premiers objets. Donner une relation de récurrence sur $f(P, j)$.
4. En déduire un algorithme de programmation dynamique permettant de résoudre le problème. Quelle est sa complexité ?

Exercice 10 – Rendu de monnaie

On dispose d'un ensemble de pièces de monnaie de valeurs entières dans $P = \{p_1, \ldots, p_n\}$. Chaque pièce est disponible en quantité suffisante. On veut déterminer pour une somme fixée S le nombre minimum de pièces nécessaires pour pouvoir faire exactement la somme S.

1. Écrire un algorithme glouton pour résoudre ce problème.
2. Cet algorithme est-il optimal pour tous les systèmes de pièces ?
3. On note $M(s, i)$ le nombre de pièces minimum utilisées pour faire la somme s en utilisant les pièces p_1 à p_i. Quelle relation de récurrence est satisfaite par $M(s, i)$?
4. En déduire un algorithme de programmation dynamique permettant de résoudre le problème.
5. Comparer la complexité de cet algorithme avec celle de l'algorithme glouton.