Programmation Dynamique: TD

Tronc Commun Scientifique
Recherche Opérationnelle
École des Mines de Nancy

1 La distance d'édition

Correction.

1. Il suffit de vérifier $d(x,y) \geqslant 0$ (évident), $d(x,y) = 0$ si et seulement si $x = y$ (évident aussi), $d(x,y) = d(y,x)$ (vrai car insertion et substitution jouent des rôles symétriques et ont le même coût), et l'inégalité triangulaire.

   Si on peut passer à l'aide de $d(x,z)$ insertions/suppressions de x à z et de $d(z,y)$ insertions/suppressions de z à y, alors on peut passer en $d(x,z) + d(z,y)$ de x à y. Donc $d(x,y) \leqslant d(x,z) + d(z,y)$.

2. Pour passer du plus court des deux mots x et y au plus long, il faut au moins $|m-n|$ insertions. Donc $|m-n| \leqslant d(x,y)$. Il y a d'ailleurs égalité si et seulement si on obtient le plus long des mots à partir du plus court en exactement $|m-n|$ insertions.

   On peut passer de x à y en m suppressions (de manière à arriver au mot vide) suivies de n insertions. D'où $d(x,y) \leqslant m+n$. On montre que le cas d'égalité correspond aux mots n'ayant pas de lettre en commun (raisonnement par l'absurde).

3. $d(x_i, y_0) = i$ et $d(x_0, y_j) = j$ (évident, ceci résulte aussi des inégalités précédentes).

4. Écrivons x_i et y_j sous la forme : $x_i = x_{i-1}\alpha$ et $y_j = y_{j-1}\beta$.

   Il y a trois manières de transformer x_i en y_j :

   a) en transformant x_{i-1} en y_{j-1}, puis,
      - si $\alpha = \beta$ on n'a plus rien à faire (coût total $d(x_{i-1}, y_{j-1})$) ;
      - si $\alpha \neq \beta$, en supprimant la i-ème lettre de x et en insérant la j-ème lettre de y (coût total de $d(x_{i-1}, y_{j-1}) + 2$) ;

   b) en transformant x_{i-1} en y_j, puis en supprimant la i-ème lettre de x_i (coût de $d(x_{i-1}, y_j) + 1$) ;

   c) en transformant x_i en y_{j-1}, puis en insérant la j-ème lettre de y_j (coût de $d(x_i, y_{j-1}) + 1$).

   Dans le premier cas, les coût des sous-cas $\alpha \neq \beta$ et $\alpha = \beta$ se confondent en $d(x_{i-1}, y_{j-1}) + 2\delta_{i,j}$.
   D'où la relation demandée.

5. Si on note $C(i,j)$ le coût pour le calcul récursif de la distance entre deux mots de longueur i et j, alors $C(i,j) \geqslant C(i-1,j-1) + C(i-1,j) + C(i,j-1)$. En "dépliant" cette récurrence sous forme arborescente (faire un dessin), on voit qu'il faut un arbre ternaire de profondeur n pour aboutir aux $C(i,0)$, $C(0,j)$ dont on connaît la valeur par la question 3. En remontant : les coûts de la profondeur n sont (au moins[1]) de 1 (et il y en a 3^n, ceux à la profondeur $(n-1)$ sont (au moins) de 3 (et il y en a 3^{n-1}), …, celui à la profondeur 0 est (au moins) de 3^n (et il y en a un).

   D'où une complexité exponentielle : $C(n,n) \geqslant 3^n$.

[1] « au moins » car il n'y a pas que des $C(i,0)$ ou $C(0,j)$ au niveau n de l'arbre ternaire…

6. Pour un algorithme de programmation dynamique, il suffit d'organiser le calcul dans un tableau de taille $(m+1) \times (n+1)$ comme en 7.
7. On applique la formule de récurrence pour remplir le tableau, on trouve $d = 3$.

	$i \downarrow$	i	n	g	e	n	i	e	u	r
$j \rightarrow$	0	1	2	3	4	5	6	7	8	9
i	1	0	1	2	3	4	5	6	7	8
g	2	1	2	1	2	3	4	5	6	7
n	3	2	1	2	3	2	3	4	5	6
e	4	3	2	3	2	3	4	3	4	5
n	5	4	3	4	3	2	3	4	5	6
e	6	5	4	5	4	3	4	3	4	5
u	7	6	5	6	5	4	5	4	3	4
r	8	7	6	7	6	5	6	5	4	3

   On voit que la complexité en espace (pour le calcul de la distance) est en fait en $\mathcal{O}(\min(m,n))$ (on n'a pas besoin de mémoriser toutes les lignes du tableau, mais essentiellement la ligne qui précède la ligne courante). Bien sûr, il faut conserver tout le tableau si on veut remonter les opérations par back-tracking.
8. Par back-tracking on peut retrouver la suite des opérations (insertion/suppression) effectuées pour arriver à la distance d'édition.

	$i \downarrow$	i	n	g	e	n	i	e	u	r
$j \rightarrow$	0	1	2	3	4	5	6	7	8	9
i	1	**0**	1	2	3	4	5	6	7	8
g	2	**1**	2	1	2	3	4	5	6	7
n	3	2	**1**	**2**	3	2	3	4	5	6
e	4	3	2	3	**2**	3	4	3	4	5
n	5	4	3	4	3	**2**	**3**	4	5	6
e	6	5	4	5	4	3	4	**3**	4	5
u	7	6	5	6	5	4	5	4	**3**	4
r	8	7	6	7	6	5	6	5	4	**3**

   En partant de la case en haut à gauche, un déplacement en diagonale correspond au cas a), un déplacement vers la droite au cas b), et vers la gauche au cas c).

   D'où la suite de transformations :

   `ingenieur` $\longrightarrow$ `igngenieur` (insertion) $\longrightarrow$ `ignenieur` (suppression) $\longrightarrow$ `igneneur` (suppression)

   Bien sûr, il n'y a pas unicité de la suite de transformations de longueur minimale.

Question subsidiaire : on autorise à présent les substitutions, avec un coût de 1 (on change le 2 en 1 dans la formule de récurrence, l'application numérique est modifiée).

2 Problème de location de skis

Correction.

1. Comme dans tout problème d'optimisation sur un support fini, le minimum existe.

   Supposons le minimum réalisé par a non croissante. Donc il existe deux skieurs i_1 et i_2 tels que $i_1 < i_2$ et $l_{a(i_1)} > l_{a(i_2)}$. On va montrer qu'en échangeant les paires de skis de i_1 et i_2, alors on

n'augmente jamais la valeur de la fonction objectif. Donc on peut transformer la fonction d'affectation a en une fonction d'affectation croissante tout en diminuant ou conservant la valeur de la fonction objectif.

Comme $i_1 < i_2$, alors $t_{i_1} \leqslant t_{i_2}$.

Soit D la différence entre la valeur de l'objectif réalisé par a et la valeur réalisée en échangeant les deux paires de skis comme précédemment. Après simplification :

$$D = |t_{i_1} - l_{a(i_1)}| + |t_{i_2} - l_{a(i_2)}| - |t_{i_1} - l_{a(i_2)}| - |t_{i_2} - l_{a(i_1)}|.$$

Alors :

a) si $t_{i_1} \leqslant t_{i_2} \leqslant l_{a(i_2)} \leqslant l_{a(i_1)} : D = 0$

b) si $t_{i_1} \leqslant l_{a(i_2)} \leqslant t_{i_2} \leqslant l_{a(i_1)} : D = 2(t_{i_2} - l_{a(iz)}) \geqslant 0$

c) si $l_{a(i_2)} \leqslant t_{i_1} \leqslant t_{i_2} \leqslant l_{a(i_1)} : D = 2(l_{a(i_1)-t_{i_1}} \geqslant 0$

d) si $l_{a(i_2)} \leqslant t_{i_1} \leqslant l_{a(i_1)} \leqslant t_{i_2} : D = 2(l_{a(i_1)} - l_{a(i_2)}) > 0$

e) si $l_{a(i_2)} \leqslant l_{a(i_1)} \leqslant t_{i_1} \leqslant t_{i_2} : D = 0$.

2. Considérons le ski j dans l'affectation optimale donnant $S(i, j)$. Ou bien ce ski n'est pas affecté, et donc $S(i, j) = S(i, j-1)$, ou bien il est affecté, et dans ce cas il sera nécessairement affecté au i-ème skieur car la fonction d'affectation est croissante.

3. Pour calculer l'affectation optimale, on commence par trier les skis et les skieurs. Puis on remplit un tableau :

	t_1	t_2	t_3	...	t_{n-1}	t_n
l_1	$\|t_1 - l_1\|$	x	x	...	x	
l_2	o	$\|t_1 - l_1\| + \|t_2 - l_2\|$	x	...	x	x
l_3	o	o	$\|t_1 - l_1\| + \|t_2 - l_2\| + \|t_3 - l_3\|$	...	x	x
$\vdots$						
l_{m-1}	x	x	x	...	o	o
l_m	x	x	x	...	x	SOL

où x désigne les cases que l'on ne calcule pas et o les cases à calculer.

En effet : 1) s'il n'y a qu'un skieur, on lui attribue le ski le plus proche de sa taille (initialisation de la colonne de t_1) ; 2) s'il y a autant de skieurs que de skis l'affectation se fait simplement par $a(i) = i$; 3) pour calculer la case courante à l'aide de la formule de récurrence, on a besoin des valeurs des cases au dessus à gauche et au dessus.

Le but est de calculer la valeur de la case SOL, donc il n'est pas nécessaire de calculer toutes les valeurs du "triangle inférieur gauche" : dans chacune des n colonnes, $m - n$ cases sont à calculer.

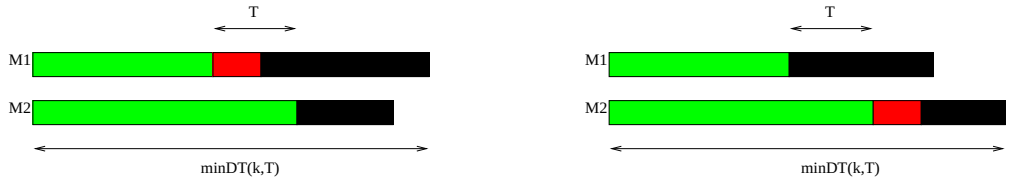
4. Complexité des tris : $\mathcal{O}(m \log(m) + n \log(n))$ (admis, cf cours 1A ou page wikipedia), et $n(m-n)$ cases à calculer. D'où une complexité globale en :

$$\mathcal{O}(m \log(m) + n \log(n) + n(m - n)).$$

3 Équilibrage de charge sur deux machines en parallèle

Correction.

1. L'expression de DT est évidente. Faites un dessin comme celui de la question suivante.
2. $\text{minDT}(n,T) = (\sum_{i=1}^{n} t_i + T)/2$. En effet, tout se passe comme si on avait une $n+1$-ème tâche de durée T, et des machines équilibrées.
   On a :
   $$\text{minDT}(k,T) = \min\big(\text{minDT}(k+1,|T-t_{k+1}|), \text{minDT}(k+1,T+t_{k+1})\big).$$
   En effet, la $k+1$-ème tâche est soit sur la machine la moins chargée avec k tâches (cas 1, deux sous-cas sont possibles selon que la durée de la $k+1$-ème tâche est inférieure ou supérieure à T, d'où la valeur absolue), soit sur la machine la plus chargée (cas 2).
   Ceci est illustré sur le dessin suivant : les k premières tâches sont en vert, la $k+1$-ème en rouge, les $n-(k+1)$ restantes placées de manière optimale en noir. Le schéma de gauche correspond au cas 1, celui de droite au cas 2.



3. On cherche la valeur de $\text{minDT}(0,0)$.
   Étape 1. T varie a priori entre 0 et 11 pour $k=4$ (somme des temps).
   On connaît les $\text{minDT}(4,T)$ et on sait que :
   $$\text{minDT}(3,T) = \min\big(\text{minDT}(4,|T-6|), \text{minDT}(4,T+6)\big),$$
   où T varie entre 0 et 5 pour $k=3$.
   On remplit le tableau ci-dessous.
   Étape 2. T varie entre 0 et 3, la récurrence s'écrit :
   $$\text{minDT}(2,T) = \min\big(\text{minDT}(3,|T-2|), \text{minDT}(3,T+2)\big).$$
   etc.

	T	0	1	2	3	4	5	6	7	8	9	10	11
étape 0	$\text{minDT}(4,T)$	5.5	**6**	6,5	7	7,5	8	8,5	9	9,5	10	10,5	11
étape 1	$\text{minDT}(3,T)$	8,5	8	7,5	7	6,5	**6**	-	-	-	-	-	-
étape 2	$\text{minDT}(2,T)$	7,5	7	6,5	**6**	-	-	-	-	-	-	-	-
étape 3	$\text{minDT}(1,T)$	6,5	**6**	-	-	-	-	-	-	-	-	-	-
étape 4	$\text{minDT}(0,T)$	**6**	-	-	-	-	-	-	-	-	-	-	-

Remarque 1 : comme dans l'exercice avec les skieurs, il n'est pas nécessaire de calculer l'intégralité du tableau pour arriver à $\text{minDT}(0,0)$ qui est la valeur cherchée. Remarquons qu'on a intérêt à ordonner les tâches par durée croissante pour minimiser le nombre d'opérations. On peut demander quelle est la complexité de l'algorithme. On voit qu'il est en :

$$\mathcal{O}\left(1+\sum_{i=1}^{n} t_i + 1 + \sum_{i=1}^{n-1} t_i + \cdots + 1 + t_1 + 1\right) = \mathcal{O}\left(n+1+\sum_{i=1}^{n}(n+1-i)\cdot t_i\right).$$

Remarque 2 : les durées de fin non-entières correspondent à des décalages T non réalisables puisque les durées des tâches sont entières. On pourrait encore simplifier les calculs en éliminant les cases correspondantes.

Conclusion : l'ordonnancement optimal a une durée de 6.

On fait une propagation arrière (en gras sur le tableau précédent). Le cas d'égalité dans l'équation de récurrence permet de déduire si la tâche est placée sur la machine la moins chargée ou sur la plus chargée. On voit qu'on arrive à un décalage $T = 1$ entre les deux machines.

Il est obtenu en plaçant : la tâche 1 sur la machine la plus chargée (machine 1 par exemple, arbitraire à cette étape), puis la tâche 2 sur la machine la plus chargée (donc la machine 1), puis la tâche 3 sur la machine la plus chargée (donc machine 1), puis la tâche 4 sur la machine la moins chargée (qui est alors la machine 2).

C'est-à-dire :
- machine 1 : tâches 1 - 2 - 3
- machine 2 : tâche 4

Sans grande surprise... (mais il fallait une application numérique « légère »).

4 Programmation dynamique et plus court chemin

Le but de cet exercice est de voir que la recherche de plus court chemin vue dans les séances précédentes peut être résolue par la programmation dynamique. C'est d'ailleurs un exemple standard d'optimisation d'une équation de Bellman. Tous les exercices de la séance GR2.2 peuvent être résolus par programmation dynamique plutôt que par point fixe.

Correction.

Soit G le graphe considéré et A et B les deux sommets.

Notons $\mathcal{C}(S_1, S_2)$ l'ensemble des chemins du sommet S_1 au sommet S_2, $l(C)$ la longueur d'un chemin (la somme des valuations des arcs parcourus), et $c(S_1, S_2)$ la valuation de l'arc (S_1, S_2).

$$\begin{aligned} d(A,B) &= \min_{C\in\mathcal{C}(A,B)} l(C) \\ &= \min_{S\in G^{-1}(B)} \min_{C'\in\mathcal{C}(A,S)} \big(l(C') + c(S,B)\big) \\ &= \min_{S\in G^{-1}(B)} \left(c(S,B) + \min_{C'\in\mathcal{C}(A,S)} l(C') \right) \\ &= \min_{S\in G^{-1}(B)} \left(c(S,B) + d(A,S) \right). \end{aligned}$$

où $G^{-1}(B)$ désigne l'ensemble des antécédents de B.

D'où la formule de récurrence satisfaisant le principe de Bellman :

$$\forall S',\ d(A,S') = \min_{S\in G^{-1}(S')} \left(c(S,S') + d(A,S) \right)$$

et l'initialisation $d(A, A) = 0$.

Remarque : cette manière de procéder pour établir l'équation de récurrence est typique de ce qu'on ferait avec une équation de Bellman générique.

Pour ce qui est de la mise en œuvre...

L'étape 1 consiste à calculer les $d(A, S')$ pour les S' successeurs de A à l'aide de la formule de récurrence.

L'étape 2 consiste à calculer les $d(A, S'')$ pour les S'' successeurs des successeurs de A **dont on connaît la distance des antécédents à** A (pour pouvoir utiliser la formule de récurrence).

etc.

Pour l'application numérique, faire dessiner le graphe et appliquer l'algorithme en partant du sommet A. On trouve $d(A, B) = 13$, et par back-tracking le chemin A, C, E, B.