Schémas algorithmiques

Dans ce chapitre, on présente différents paradigmes algorithmiques qui permettent de développer des algorithmes résolvant des problèmes (souvent d'optimisation) mettant en jeu des textes, des nombres, des listes, des graphes, des expressions arithmétiques, des points du plan, des ordonnancements...

Rappel : On rappelle qu'un problème d'optimisation consiste à minimiser une fonction à valeurs réelles données sur un ensemble de solutions donné.

Définition :

Ainsi, un tel problème $\mathcal{P}$ est défini par la fonction dite fonction-objectif f et l'ensemble des solutions $\mathcal{S}$.
On note $\mathcal{P} : \max\limits_{X\in\mathcal{S}} f(X)$ (ou bien $\mathcal{P} : \min\limits_{X\in\mathcal{S}} f(X)$ s'il s'agit de minimiser f).
On appelle alors valeur du problème $\mathcal{P}$: $\mathrm{val}(\mathcal{P}) = \max\{f(X) \mid X \in \mathcal{S}\}$.
On note aussi $\operatorname{argmax}\limits_{X\in\mathcal{S}} f(X) = \{X \in \mathcal{S} \mid f(X) = \mathrm{val}(\mathcal{P})\}$ l'ensemble des solutions optimales.

Remarque : L'unicité de la valeur optimale n'entraîne pas celle des solutions optimales.

Remarque : Si $\mathcal{S} \subseteq \widetilde{\mathcal{S}} \subseteq \mathcal{D}(f)$, on dit que $\widetilde{\mathcal{P}} = \max\limits_{X\in\widetilde{\mathcal{S}}} f(X)$ est un relâché de $\mathcal{P}$.
On a alors $\mathrm{val}(\widetilde{\mathcal{P}}) \geq \mathrm{val}(\mathcal{P})$.

1 Algorithmes gloutons

1.1 L'exemple du problème du sac à dos

SAC À DOS

entrée : $P \in \mathbb{R}$
$v = (v_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$
$p = (p_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$

sortie : $\max\limits_{\substack{\delta\in\{0,1\}^n \\ \delta\cdot p\leq P}} \delta\cdot v = \max\limits_{\substack{\delta\in\{0,1\}^n \\ \sum_{i=1}^n \delta_i p_i \leq P}} \sum\limits_{i=1}^{n} \delta_i v_i$

SAC À DOS FRACTIONNAIRE

entrée : $P \in \mathbb{R}$
$v = (v_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$
$p = (p_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$

sortie : $\max\limits_{\substack{\delta\in[0,1]^n (*) \\ \delta\cdot p\leq P}} \delta\cdot v$

Remarque : *Sac à dos fractionnaire* est un relâché de *Sac à dos*.

Remarque : La contrainte $(*)$ s'écrit aussi/se traduit aussi par : $\begin{array}{l}\forall\, i \in [1..n],\ \delta\cdot\mathbb{1}_{\{i\}} \geq 0 \\ \forall\, i \in [1..n],\ \delta\cdot\mathbb{1}_{\{i\}} \leq 1.\end{array}$
En effet, $\forall\, i \in [1..n]$, $\delta\cdot\mathbb{1}_i = \sum\limits_{j=1}^{n} \delta_j \times \underbrace{(\mathbb{1}_i)_j}_{=0 \text{ sauf si } j=i} = \delta_i \times (\mathbb{1}_i)_i = \delta_i \times 1 = \delta_i$.

Exemple : Pour $P = 20$, en choisissant par v_i décroissants, on trouve $\delta^* = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$.

i	p_i	c_i
1	20	10
2	10	9
3	10	9

Alors, $\delta^* \cdot v = v_1 = 10$.
Or, $\delta = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$ vérifie $\delta \cdot p = 1 \times p_2 + 1 \times p_3 = 10 + 10 = 20 \leq P$
et $\delta \cdot v = 1 \times v_2 + 1 \times v_3 = 9 + 9 = 18 > 10$.
Ainsi, cette stratégie n'est pas optimale.

Exemple : Toujours pour $P = 20$, avec des p_i croissants, on a $\delta^* = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$, de valeur $\delta^* \cdot v = 2$.

i	p_i	c_i
1	18	10
2	10	9
3	10	9

Or, $\delta = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ est une solution, car $\delta \cdot p = 18 \leq P$, et est de valeur $\delta \cdot v = 10 > 2$.
On en déduit donc de même que cette stratégie n'est pas optimale.

Exemple : Si on sélectionne maintenant par (v_i/p_i) décroissants, on obtient $\delta^* \cdot v = v_1 = 22$.

i	p_i	c_i	v_i/p_i
1	11	22	2
2	10	15	1,5
3	10	15	1,5

Or, $\delta = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$ est une autre solution de valeur meilleure :
$\delta \cdot v = 15 + 15 = 30 > 22$
Ainsi, cette stratégie n'est pas optimale non plus.

Remarque : Les exemples précédents montrent que pour le problème *Sac à dos*, les stratégies simples consistant à prendre les objets simplement par v_i décroissants, p_i croissants ou v_i/p_i décroissants ne sont pas optimales. En fait, *Sac à dos* est un problème difficile [(?)].

Par contre, le dernier des trois exemples précédents s'avèrera utile pour le problème du *Sac à dos fractionnaire*.

Exemple : Sur le dernier exemple, $\delta^* = \begin{pmatrix} 1 \\ 9/10 \\ 0 \end{pmatrix}$ convient en tant que solution optimale, et $\delta^* \cdot v = 22 + 13{,}5 = 35{,}5$.

∗ Algorithme en pseudo-code

Rempli-sac | P un réel
$(p_i, v_i)_{i \in [1..n]} \in \left((\mathbb{R}^{+*})^2\right)^n$ une suite finie de réels strictement positifs.

Trier les objets par ratio v_i/p_i décroissant, *i.e.* trouver $\sigma \in \mathscr{B}ij([1..n], [1..n])$ tel que $\left(v_{\sigma(k)}/p_{\sigma(k)}\right)_{k \in [1..n]}$ soit décroissante.
$s \leftarrow 0$
$\delta \leftarrow$ tableau de réels indicé par $[1..n]$ initialisé à 0
$k = 1$
Tant que $k \leq n$ et $s + p_{\sigma(k)} \leq P$ // *Invariant :* $s = \sum_{i=1}^{n} \delta_i p_i$
  $s \leftarrow s + p_{\sigma(k)}$
  $\delta_{\sigma(k)} \leftarrow 1$
  $k \leftarrow k + 1$
Si $k \leq n$
  $\delta_{\sigma(k)} \leftarrow (P - s)/p_{\sigma(k)}$
  $s \leftarrow s + \delta_{\sigma(k)} p_{\sigma(k)}$
Renvoyer δ

On montre l'optimalité de cette stratégie par un argument d'échange.

Propriété :

Soit $n \in \mathbb{N}^*$.
Soit $(p_i, v_i)_{i\in[1..n]} \in (\mathbb{R}^{+*}\times\mathbb{R}^{+*})^2$ telle que $(v_i/p_i)_{i\in[1..n]}$ soit strictement décroissante.
Soit $P \in \mathbb{R}^{+*}$.
On note $\mathcal{S} = \{\delta \in [0,1]^n \,|\, \delta\cdot p \leq P\}$.

$\trianglerighteq$ Si $p\cdot\mathbb{1} > P$ (*i.e.* $\displaystyle\sum_{i=1}^{n} p_i > P$ ou encore $\mathbb{1} \notin \mathcal{S}$), alors si $\delta^* \in \operatorname*{argmax}_{\delta\in\mathcal{S}} v\cdot\delta$, il existe $m \in [1..n]$ tel que :

$$\begin{cases} \forall\, i \in [1..m[,\ \delta_i^* = 1 \\ \delta_m^* = \dfrac{P - \sum_{k=1}^{m-1} p_k}{p_m} \\ \forall\, i \in [m+1..n],\ \delta_i^* = 0 \end{cases}$$

Preuve : Soit $\delta^* \in \operatorname*{argmax}_{\delta\in\mathcal{S}} v\cdot\delta$.

▷ On sait que $\mathbb{1} \notin \mathcal{S}$, donc $\delta^* \neq \mathbb{1}$ et donc $\{i \in [1..n] \,|\, \delta_i^* < 1\} \neq \emptyset$. On peut alors définir $m = \min\{i \in [1..n] \,|\, \delta_i^* < 1\}$. Par définition de m, $\forall\, i \in [1..m[,\ \delta_i^* = 1$.

▷ Par l'absurde, on suppose qu'il existe $i_0 \in [m+1..n]$ tel que $\delta_{i_0}^* \neq 0$.
Posons $\varepsilon = \min\Big(\underbrace{p_{i_0}\delta_{i_0}^*}_{>0 \text{ par hyp.}},\ \underbrace{p_m(1-\delta^* m)}_{>0 \text{ car } \delta_m^*<1}\Big)$. On considère alors $\widehat{\delta} = \delta^* - \dfrac{\varepsilon}{p_{i_0}}\mathbb{1}_{i_0} + \dfrac{\varepsilon}{p_m}\mathbb{1}_{i_0}$.
Soit $i \in [1..n]$.

- Si $i \notin \{i_0, m\}$, $\widehat{\delta}_i = \delta_i^* \in [0,1]$.
- $\widehat{\delta}_{i_0} = \delta_{i_0}^* - \dfrac{\varepsilon}{p_{i_0}} \leq \delta_{i_0} \leq 1$.
  De plus, $\dfrac{\varepsilon}{p_{i_0}} \leq \dfrac{1}{p_{i_0}} \times p_{i_0}\delta_{i_0}^* = \delta_{i_0}^*$ donc $\delta_{i_0}^* - \dfrac{\varepsilon}{p_{i_0}} \geq 0$ soit $\widehat{\delta}_{i_0} \geq 0$, donc $\delta_{i_0}^* - \dfrac{\varepsilon}{p_{i_0}} \geq 0$.
- $\widehat{\delta}_m = \delta_m^* + \dfrac{\varepsilon}{p_m} \geq \delta_m^* \geq 0$. De plus, $\dfrac{\varepsilon}{p_m} \leq 1 - \delta_m^*$, soit $\delta_m^* + \dfrac{\varepsilon}{p_m} \leq 1$ soit $\widehat{\delta}_m \leq 1$.

Ainsi, $\widehat{\delta} \in [0,1]^n$.

$$\begin{aligned} \text{De plus, } \quad \widehat{\delta}\cdot p &= \left(\delta^* + \frac{\varepsilon}{p_m}\mathbb{1}_m - \frac{\varepsilon}{p_{i_0}}\mathbb{1}_{i_0}\right)\cdot p \\ &= (\delta^*\cdot p) + \frac{\varepsilon}{p_m}\underbrace{(\mathbb{1}_m\cdot p)}_{=p_m} - \frac{\varepsilon}{p_{i_0}}\underbrace{(\mathbb{1}_{i_0}\cdot p)}_{=p_{i_0}} \\ &= \delta^*\cdot p + \not\varepsilon - \not\varepsilon \\ &= \delta^*\cdot p \leq P \quad (\text{car } \delta^* \in \mathcal{S}) \end{aligned}$$

On a donc bien $\widehat{\delta} \in \mathcal{S}$.

$$\begin{aligned} \text{Ensuite, } \quad \widehat{\delta}\cdot v &= \delta^*\cdot v + \frac{\varepsilon}{p_m}\underbrace{(\mathbb{1}_m\cdot v)}_{=v_m} - \frac{\varepsilon}{p_{i_0}}\underbrace{(\mathbb{1}_{i_0}\cdot v)}_{=v_{i_0}} \\ &= \delta^*\cdot v + \underbrace{\varepsilon}_{>0\,(1)}\underbrace{\left(\frac{v_m}{p_m} - \frac{v_{i_0}}{p_{i_0}}\right)}_{>0\,(2)} \\ &> \delta^*\cdot v \end{aligned}$$

C'est absurde car $\delta^* \in \operatorname*{argmax}_{\delta\in\mathcal{S}} v\cdot\delta$. Donc $\forall\, i \in [1..m],\ \delta_i^* = 0$.

Corollaire :

Sous les hypothèses et notations de la propriété précédente, $\underset{\delta\in\mathcal{S}}{\operatorname{argmax}}\, v\cdot\delta$ est réduite à la solution donnée par :

$$M = \min\left\{ i\in[1..n] \,\middle|\, \sum_{k=1}^{i} p_k > P \right\}, \quad \textit{c-à-d} \quad {}^t\Big(\underbrace{1\ 1\ \ldots\ 1}_{M-1}\ w\ \underbrace{0\ \ldots\ 0}_{n-M}\Big)$$

Preuve : Soit $\delta^* \in \underset{\delta\in\mathcal{S}}{\operatorname{argmax}}\, \delta\cdot v$.

Soit $m = \min\{i\in[1..n] \,|\, \delta_i^* < 1\}$. On veut montrer que $m = M$.

Puisque $\delta^*\in\mathcal{S}$, $\delta^*\cdot p \le P$ soit $\displaystyle\sum_{k=1}^{n}\delta_k^* p_k = \sum_{k=1}^{m}\delta_k^* p_k \le P$

soit $\underbrace{\delta_m^* p_m}_{\ge 0} + \displaystyle\sum_{k=1}^{m-1}\delta_k^* p_k \le P$

donc $\displaystyle\sum_{k=1}^{m-1}\underbrace{\delta_k^*}_{=1} p_k \le P$

donc $m-1 < M$ soit $m \le M$.

De plus, on a montré que $\displaystyle\sum_{k=1}^{m} p_k > p$ (*cf.* $(*)$ dans la preuve précédente),

donc $m \in \left\{ i\in[1..n] \,\middle|\, \sum_{k=1}^{i} p_k > P \right\}$, donc $m \ge M$ qui est le minimum de cet ensemble.

1.2 Le problème du tri

TRI ‖ entrée : $\left(x_i\right)_{i\in[1..n]} \in X^n$, où $(X,\le)$ est un ensemble totalement ordonné.
‖ sortie : $\sigma\in\mathcal{S}_n$ telle que $\left(x_{\sigma(i)}\right)_{i\in[1..n]}$ soit croissante pour $\le$.

Remarque : *Tri* est un problème d'optimisation, il consiste en la recherche de :

$$\min_{\sigma\in\mathcal{S}_n}\left(\sum_{i=1}^{n-1}\max(x_{\sigma(i)} - x_{\sigma(i+1)}, 0)\right),$$

minimum qui vaut 0 et qui est atteint par n'importe quel $\sigma\in\mathcal{S}_n$ solution du problème.

$*$ Algorithme en pseudo-code pour le tri par sélection

Tri-sélection | $(x_i)_{i\in[1..n]}$ une famille d'éléments comparables avec $\le$ en $\Theta(1)$, rangés dans un tableau.

Soit T une copie du tableau (*i.e.* $\forall\, i \in [1..n], T[\,i\,] = x_i$)
Soit I un tableau identité de $[1..n]$
Soit σ un tableau d'entiers indicé par $[1..n]$ initialisé à 0

Pour k allant de 1 à n
    // *Invariant* : $\forall i \in [1..n]$, $T[\,i\,] = x_{I[\,i\,]}$
    $T[1..k-1]$ est trié
    $\forall i \in [k..n]$, $T[\,i\,] \geq \max\{T[\,j\,] \,|\, j \in [1..k-1]\}$
    $\forall\, i \in [1..k-1]$, $T[\,i\,] = x_{\sigma[\,i\,]}$
    Trouver $i_0 \in \underset{i\in[k..n]}{\operatorname{argmin}}\, T[\,i\,]$
    $\sigma[\,k\,] \leftarrow I[\,i_0\,]$
    Échanger $T[\,k\,]$ et $T[\,i_0\,]$
    Échanger $I[\,k\,]$ et $I[\,i_0\,]$
Renvoyer σ

Remarque : Cet algorithme peut être optimisé : en effet, si on avait directement initialisé σ au tableau identité, on n'aurait pas eu besoin du tableau I.

Propriété :

Soit I un ensemble fini non vide de cardinal n (typiquement $[1..n]$ ou $[0..n-1]$).
Soit $(x_i)_{i\in I} \in X^I$ où $(X, \leq)$ est un ensemble totalement ordonné.
Soit $i_1 \in \underset{i\in I}{\operatorname{argmin}}\, x_i$.
On note $\widetilde{I} = I \backslash \{i_1\}$.
Si $\widetilde{\sigma} \in \mathscr{B}ij\Big([1..n-1], \widetilde{I}\Big)$ est telle que $\Big(x_{\widetilde{\sigma}(i)}\Big)_{i\in[1..n-1]}$ est croissante, alors le prolongement σ de $\widetilde{\sigma}$ défini suivant est une bijection telle que $\Big(x_{\sigma(i)}\Big)_{i\in[1..n]}$ est croissante :

$$\sigma = \begin{pmatrix} [1..n] \to I \\ 1 \mapsto i_1 \\ i \geq 2 \mapsto \widetilde{\sigma}(i-1) \end{pmatrix}$$

Preuve : *cf.* annexe "Tri par sélection".

1.3 À retenir sur les algorithmes gloutons

Définition :

On dit qu'un algorithme qu'il est glouton lorsqu'il construit une solution à un problème (d'optimisation) en prenant des décisions localement pertinentes (*c-à-d* optimales) à chaque étape et qu'il ne revient par sur ces décisions.

Dans le cadre d'un problème d'optimisation, on dit qu'un algorithme glouton est optimal s'il renvoie toujours une solution optimale.

Exemples : L'algorithme de Huffman (*cf.* DM n°3 - Partie 2/2) et l'algorithme de résolution du problème du *Sac à dos fractionnaire* sont des algorithmes gloutons.

Remarque : Attention, selon les problèmes, un même algorithme peut être optimal ou non. En général, les algorithmes gloutons sont efficaces (faible complexité, a fortiori polynomiale) : ils ne sont donc pas exacts pour les problèmes difficiles/NP-complets (comme *Sac à dos entier*).

Néanmoins, ils peuvent être utiles pour obtenir rapidement la borne supérieure ou inférieure, *c-à-d* un majorant ou un miniorant de la valeur optimale.

Remarque : Pour savoir si un problème d'optimisation peut être résolu par un algorithme glouton, on se pose deux questions :

- La fonction-objectif est-elle décomposable comme somme de fonctions sur les sous-parties de la solution ?
- L'ensemble des solutions est-il décomposable comme produit cartésien ou au contraire défini par des contraintes liantes ?

2 Programmation dynamique

2.1 Le problème du sac à dos entier

Voir TP n°13 - Introduction à la programmation dynamique.

2.2 Plus long sous-mot commun

Dans cette section, on fixe Σ un alphabet (i.e. un ensemble fini non vide de symboles).

Rappel :

Soit $(u, v) \in (\Sigma^*)^2$. Notons $n = |u|$ et $m = |v|$.

On dit que u est un sous-mot de v et on note $u \preccurlyeq v$ ssi il existe $\varphi \in \mathscr{F}([1..n], [1..m])$ strictement croissante, telle que : $u = v_{\varphi(1)}v_{\varphi(2)}...v_{\varphi(n)}$, *c-à-d* $\forall\, i \in [1..n],\ u_i = v_{\varphi(i)}$.

PLSMC

entrée : $(u, v) \in (\Sigma^*)^2$

sortie : $\max\{|w| \,|\, w \preccurlyeq u \text{ et } w \preccurlyeq v\}$ ou $w^* \in \operatorname{argmax}\{|w| \,|\, w \preccurlyeq u \text{ et } w \preccurlyeq v\}$

Propriété :

Soit $(u, v) \in (\Sigma^*)^2$. Notons $n = |u|$ et $m = |v|$.

Pour $(i, j) \in [0..n] \times [0..m]$, on note

$$\mathcal{L}_{i,j} = \max\left\{|w| \;\middle|\; \underbrace{w \in \Sigma^*,\ \begin{matrix} w \preccurlyeq u_1...u_i \\ w \preccurlyeq v_1...v_j \end{matrix}}_{=\mathcal{S}_{i,j}}\right\} = \max_{w \in \mathcal{S}_{i,j}} |w|$$

On a alors :

i. $\forall\, i \in [0..n],\ \mathcal{L}_{i,0} = |\varepsilon| = 0$

ii. $\forall\, j \in [0..m],\ \mathcal{L}_{0,j} = 0$

iii. $\forall (i, j) \in [0..n] \times [0..m],\ \mathcal{L}_{i,j} = \begin{cases} \mathcal{L}_{i-1,j-1} + 1 \text{ si } u_i = v_j \\ \max(\mathcal{L}_{i-1,j}, \mathcal{L}_{i,j-1}) \text{ sinon} \end{cases}$

Exemple : Pour $u = \mathsf{GIRAFFE}$ et $v = \mathsf{GRAFITI}$, on a $n = 6$, $m = 7$.

$\mathcal{L}_{i,j}$			G	R	A	F	I	T	I
	i \ j	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
I	2	0	1	1	1	1	2	2	2
R	3	0	1	2	2	2	2	2	2
A	4	0	1	2	3	3	3	3	3
F	5	0	1	2	3	4	4	4	4
E	6	0	1	2	3	4	4	4	4

Les cases colorées indiquent les couples (i, j) pour lesquels on a $u_i = v_j$, et donc $\mathcal{L}_{i,j} = \mathcal{L}_{i-1,j-1}+1$.

Preuve :
i. Soit $i \in [0..n]$.
On a $\mathcal{L}_{i,0} = \max\{|w||w \in \Sigma^*,\ w \preccurlyeq u_1...u_i,\ w \preccurlyeq \varepsilon\} \leq \max\{|w||w \in \Sigma^*\} = \max \varepsilon = |\varepsilon| = 0$.
Or, $\varepsilon \preccurlyeq u_1...u_i$ et $\varepsilon \preccurlyeq \varepsilon$ donc $|\varepsilon| \leq \mathcal{L}_{i,0}$ soit $0 \leq \mathcal{L}_{i,0}$. D'où $\mathcal{L}_{i,0} = 0$ par double inégalité.

ii. La démonstration se fait sur le même principe.

iii. Soit $(i, j) \in [1..n] \times [1..m]$.
▷ Supposons que $u_i = v_j$. On a :

$$\begin{aligned}
\mathcal{L}_{i,j} &= \max\left\{|w| \,\middle|\, w \in \Sigma^*,\ \begin{array}{l} w \preccurlyeq u_1...u_{i-1}u_i \\ w \preccurlyeq v_1...v_{j-1}v_j \end{array}\right\} \\
&\geq \max\left\{|w| \,\middle|\, w \in \Sigma^*,\ w_{|w|} = u_i,\ \begin{array}{l} w \preccurlyeq u_1...u_{i-1}u_i \\ w \preccurlyeq v_1...v_{j-1}v_j \end{array}\right\} \\
&= \max\left\{|x| + 1 \,\middle|\, x \in \Sigma^*,\ \begin{array}{l} x \preccurlyeq u_1...u_{i-1} \\ x \preccurlyeq v_1...v_{j-1} \end{array}\right\} \\
&= \mathcal{L}_{i-1,j-1} + 1
\end{aligned}$$

Ensuite, montrons que $\mathcal{L}_{i,j} - 1 \leq \mathcal{L}_{i-1,j-1}$.
Il existe $w \in \mathcal{S}_{i,j}$ tel que $|w| = \mathcal{L}_{i,j}$.

- Si $w = \varepsilon$, alors $\mathcal{L}_{i,j} = |\varepsilon| = 0$, or $u_i = v_j \in \mathcal{S}_{i,j}$ donc $\mathcal{L}_{i,j} \geq |u_i| = 1$, absurde.
- Si $w \neq \varepsilon$, alors $p = |w| > 0$.

  On pose alors $x = w_1w_2...w_{p-1}$. On a $|x| = p - 1 = |w| - 1$, montrons que $x \in \mathcal{S}_{i-1,j-1}$.
  - Si $w_p = u_i = v_j$, alors $x \preccurlyeq u_1...u_{i-1}$ et $x \preccurlyeq v_1...v_{j-1}$ (par identification).
  - Si $w_p \neq u_i$ (*i.e.* $w_p \neq v_j$), alors d'après le lemme qui suit, $w \preccurlyeq u_1...u_{i-1}$ et $w \preccurlyeq v_1...v_{j-1}$. De plus, comme $x \preccurlyeq w$, par transitivité, on a $x \preccurlyeq u_1...u_{i-1}$ et $x \preccurlyeq v_1...v_{j-1}$.

Dans les deux cas, $x \in \mathcal{S}_{i,j}$ soit $\mathcal{L}_{i,j} - 1 = |w| - 1 = |x| \leq \mathcal{L}_{i-1,j-1}$.

▷ Supposons à présent que $u_i \neq v_j$.

Lemme :

Soit $(u, v) \in (\Sigma^*)^2$. On note $n = |u|$ et $m = |v|$.

Si $\begin{cases} u \preccurlyeq v \\ u_m \neq v_m \end{cases}$, alors $u \preccurlyeq v_1 ... v_{m-1}$.

Preuve : *cf.* plus tard.

2.3 À retenir sur la programmation dynamique et la mémoïsation

Lorsqu'un algorithme récursif résout un problème en faisant appel à la solution de plusieurs sous-problèmes, une implémentation naïve risque de calculer de nombreuses fois les mêmes sous-problèmes.
Dans certains cas, cela conduit à un algorithme de complexité inutilement exponentielle.

• La mémoïsation permet de pallier cette inefficacité : il s'agit de garder en mémoire à chaque appel la valeur cherchée, plus précisément, on stocke les associations entre des paramètres caractérisant une instance ou sous-instance du problème et la valeur de la solution associée.

On utilise pour cela une structure de dictionnaire, qui peut dans de nombreux cas être implémentée par un tableau, un ARN, ou encore un tas de hachage.

Remarque : Il faut prendre soin d'utiliser les valeurs déjà calculées disponibles avant de lancer les appels récursifs.

• La programmation dynamique repose sur la même idée de stockage des valeurs de sous-problèmes pour résoudre un problème mais cette approche n'est pas récursive : on calcule toutes les valeurs d'une famille de sous-problèmes des plus petits aux plus grands (selon un ordre à définir).

On effectue ci-dessous une comparaison entre ces deux paradigmes algorithmiques.

∗ Mémoïsation

Avantage : *on calcule seulement les valeurs nécessaires*
Inconvénients : – *la complexité spatiale d'une implémentation par tableau ne peut être réduite.*
– *possiblement plus difficile à programmer*

∗ Programmation dynamique

Avantages : – *permet parfois une réduction de la complexité spatiale*
– *facile à coder*
Inconvénient : *potentiellement plus d'appels*

3 Diviser pour régner (divide-and-conquer)

3.1 L'exemple du tri-fusion

∗ Algorithmes en pseudo-code

Tri-fusion | T un tableau
d, f deux indices

Soit $n = f - d + 1$ (*i.e.* Card $[d..f]$))
Si $n = 0$ ou 1 alors retourner $T[\,d..f\,]$
Sinon
  $m \leftarrow \lfloor n/2 \rfloor$
  $T_1 \leftarrow$ Tri-fusion $(T, d, m-1)$
  $T_2 \leftarrow$ Tri-fusion (T, m, f)
  Retourner Interclassement (T_1, T_2)

où Interclassement est défini comme suit :

Interclassement | T_1, T_2 deux tableaux triés

Soit $n =$ taille (T_1) et $m =$ taille (T_2)
$i = 0$
$j = 0$
Créer T un tableau de taille $n + m$
Tant que $i < n$ et $j < m$
  // Invariant : $T[\,0..i+j\,[$ est trié
  $\left\{\!\left\{ T_1[\,k\,] \,\middle|\, k \in [0..i[\right\}\!\right\} \cup \left\{\!\left\{ T_2[\,k\,] \,\middle|\, k \in [0..j[\right\}\!\right\} = \left\{\!\left\{ T[\,k\,] \,\middle|\, k \in [0..i+j[\right\}\!\right\}$
  Si $T_1[\,i\,] \leq T_2[\,j\,]$ alors
    $T[\,i+j\,] \leftarrow T_1[\,i\,]$
    $i \leftarrow i + 1$
  Sinon
    $T[\,i+j\,] \leftarrow T_2[\,j\,]$
    $j \leftarrow j + 1$
Tant que $i < n$
  $T[\,i+j\,] \leftarrow T_1[\,i\,]$
  $i \leftarrow i + 1$
Tant que $j < m$
  $T[\,i+j\,] \leftarrow T_2[\,j\,]$
  $j \leftarrow j + 1$

Remarque : Interclassement fait au plus $|T_1| + |T_2| - 1$ comparaisons.

En effet, il y a moins de comparaisons que de tours de la première boucle "Tant que" ; or, ce nombre de tours de boucle est inférieur à $n + m$ car initialement, $i + j = 0$, à chaque tour $i + j$ augmente de 1, et par la condition de boucle, $i + j \leq n + m - 2$).

Exercice : Exhiber une famille de pire cas qui atteint cette borne.

Propriété :

Pour $n \in \mathbb{N}$, on note C_n le nombre maximal de comparaisons effectuées pour trier un sous-tableau de taille n.

On a $\begin{cases} C_0 = 0 \\ C_1 = 1 \\ \forall\, n \in \mathbb{N},\, n \geq 1,\, C_n = \overbrace{0}^{\text{diviser}} + \overbrace{C_{\lfloor n/2 \rfloor} + C_{\lceil n/2 \rceil}}^{\text{régner}} + \underbrace{\overbrace{\lfloor n/2 \rfloor + \lceil n/2 \rceil - 1}^{\text{combiner}}}_{=n-1} \end{cases}$.

La complexité de tri-fusion est alors en $\Theta(n \log n)$.

Preuve : Montrons par récurrence forte sur $n \in \mathbb{N}^*$ la propriété $\mathcal{P}_n$: "$C_n \leq n \log_2 n$".

• Pour $n = 1$, on a $C_1 = 0$ et $1 \times \log_2(1) = 0$ donc $\mathcal{P}_1$ est vraie.

• Soit $n \in \mathbb{N}^*$. On suppose $\forall\, k \in [1..n]$, $\mathcal{P}_k$.

- Si $n+1$ est pair, disons $n+1 = 2p$,

$$\begin{aligned} C_{n+1} &= 2C_p + (n+1-1) \\ &\leq 2\,(p \log_2(p)) + n \\ &= \cancel{2}\frac{n+1}{\cancel{2}}(\log_2(n+1) - 1) + n \\ &= (n+1)\log_2(n+1) \underbrace{-(n+1) - n}_{=-1 \leq 0} \\ &\leq (n+1)\log_2(n+1) \end{aligned}$$

- Si $n+1$ est impair, alors $\lfloor \frac{n+1}{2} \rfloor = \frac{(n+1-1)}{2} = \frac{n}{2}$ et $\lceil \frac{n+1}{2} \rceil = \frac{(n+1)-1}{2} = \frac{n}{2} + 1$

donc
$$\begin{aligned} C_{n+1} &= C_{\frac{n}{2}} + C_{\frac{n}{2}+1} + ((n+1) - 1) \\ &\leq \frac{n}{2}\log_2\left(\frac{n}{2}\right) + \left(\frac{n}{2} + 1\right)(\log_2(n+2) - 1) + n \\ &= \frac{n}{2}\left(\log_2 n + \log_2(n+2)\right) - \frac{\cancel{n}}{\cancel{2}} - \frac{\cancel{n}}{\cancel{2}} + \cancel{n} + \log_2(n+2) - 1 \\ &\leq n \log_2\left(\frac{n + (n+2)}{2}\right) + \log_2(n+2) - 1 \\ &\leq n\log_2(n+1) + \log_2(n+2) - 1 \\ &\overset{(*)}{\leq} n\log_2(n+1) + \log_2(n+1) \\ &= (n+1)\log_2(n+1) \end{aligned}$$