Programmation dynamique

La *programmation dynamique* est un des grands paradigmes de l'algorithmique. Elle s'applique généralement à des problèmes d'optimisation pour lesquels le calcul de la solution optimale fait appel à la résolution de sous-problèmes similaires au problème initial. Ces sous-problèmes naissent souvent des différents choix que l'on peut effectuer pour construire une solution. Une approche de programmation dynamique sera efficace si un même sous-problème apparaît à plusieurs reprises lors de l'analyse. L'idée est alors de stocker les solutions à tous les sous-problèmes rencontrés, pour éviter d'avoir à les recalculer par la suite. Une telle approche permet par exemple de calculer efficacement les coefficients binomiaux C_n^p en utilisant la formule du triangle de Pascal, comme on l'a vu lors du premier TD. Dans ce TD nous étudions deux variantes d'un même problème pour lequel une approche de programmation dynamique est efficace.

1 Distance d'édition

Lorsque l'on veut comparer la proximité de deux mots sur un alphabet Σ, la distance la plus simple est la *distance de Hamming*. Si $u = u_0 \dots u_{n-1} \in \Sigma^n$ et $v = v_0 \dots v_{m-1} \in \Sigma^m$, alors la disance de Hamming entre u et v est définie par

$$d_H(u,v) = \sum_{i=0}^{\max(n,m)-1} \bar{\delta}_{u_i,v_i}$$

où $\bar{delta}_{u_i,v_i} = 0$ si et seulement si $u_i = v_i$ (en particulier $\bar{delta}_{u_i,v_i} = 1$ si u_i ou v_i n'est pas défini). Par exemple, la distance de Hamming entre "soir" et "poire" est de 2.

Dans ce TD, les mots seront représentés par des vecteurs de lettres de l'alphabet :

```
type 'a mot == 'a vect;;
```

Question 1. *Écrire un programme* `hamming : 'a mot→'a mot→int` *qui calcule la distance de Hamming entre deux mots. Pour forcer Caml à considérer les entrés comme des mots, on pourra utiliser la syntaxe* `let hamming (a : mot) (b : mot) = ...`

Lorsque l'on considère un langage comme le français, on voudrait pouvoir dire que des mots tels que "parfait" et "imparfait" sont proches (on s'occuppe de la ressemblance, pas du sens... penser par exemple à des applications du type correcteur d'orthographe). Pourtant, leur distance de Hamming 9 est maximale pour des mots de cette longueur.

Définition 1. *La* distance d'édition *entre deux mots u et v est le nombre minimal d'insertions, suppressions et substitions de lettres nécessaires pour passer d'un mot à l'autre.*

Ainsi, la distance d'édition entre "parfait" et "imparfait" est de 2 seulement (il suffit d'insérer le "i" et le "m"), ce qui correspond mieux à notre intuition.

Une manière naïve de calculer la distance d'édition entre deux mots consiste à partir de l'un des deux mots, et à construire de manière itérative l'ensemble des mots accessibles en moins de k opérations élémentaires (insertion, suppression, substitution), jusqu'à trouver le deuxième mot.

Question 2. *Soient u et v deux mots de longueur n et m respectivement.*

- *Montrer que la distance d'édition entre u et v est bornée par* $\max(n,m)$. *Donner un exemple pour lequel cette distance est atteinte.*
- *Si $|\Sigma| = p$, donner une borne supérieure sur le nombre de mots dont la distance d'édition à u est inférieure à 1 ? à k ? En déduire une borne (grossière) sur la complexité de l'algorithme de calcul de la distance d'édition proposé ci-dessus. Qu'en pensez-vous ?*

Soient u et v deux mots de longueur n et m respectivement, et $1 \leq i \leq n$ et $1 \leq j \leq m$ deux entiers. On note $u(i) = u_0 \dots u_{i-1}$ et $v(i) = v_0 \dots v_{j-1}$ les préfixes de longueur respectivement i et j de u et de v, et $E(i,j)$ la distance d'édition entre $u(i)$ et $v(j)$. Notre but est donc de calculer $E(n,m)$.

Question 3. *En remarquant que la séquence de transformations faisant passer d'un mot à un autre peut être effectuée dans un ordre quelconque, établir la formule de récurrence*

$$E(i,j) = \min\big(1 + E(i-1,j),\ 1 + E(i,j-1),\ \bar{\delta}_{u_{i-1},v_{j-1}} + E(i-1,j-1)\big)$$

Il est facile de vérifier que si on utilisait cette relation pour écrire une fonction récursive calculant la valeur de $E(i,j)$, on obtiendrait à nouveau une fonction dont le coût serait exponentiel en la taille de l'entrée : l'algorithme récursif rencontre en effet chaque sous-problème de nombreuses fois dans différentes branches de l'arbre récursif, sans réutiliser toutefois les calculs déja effectués. La programmation dynamique contourne ce problème en stockant les résultats intermédiaires dans un tableau auxiliaire à deux dimensions : la case (i,j) de cette matrice contient la distance d'édition entre $u_0 \dots u_{i-1}$ et $v_0 \dots v_{j-1}$, c'est-à-dire $E(i,j)$.

Question 4. *En déduire une procédure itérative* `edit : 'a mot→ 'a mot→ int` *qui calcule la distance d'édition entre deux mots.*

Question 5. *Évaluer la complexité du programme* `edit`*, et prouver sa correction.*

2 Plus long sous-mot commun

Si $u = u_0 \dots u_{n-1}$ est un mot, un *sous-mot* de u de longueur k est un mot u' obtenu à partir de u en supprimant $n-k$ éléments tout en conservant l'ordre : $u' = u_{\sigma(0)} \dots u_{\sigma(k-1)}$ avec $0 \leq \sigma(0) < \dots < \sigma(k-1) < n$. On dit que w est un sous-mot commun de u et v si w est un sous-mot de u et un sous-mot de v.

Nous cherchons maintenant à trouver un plus longu sous-mot commun à deux mots $u = u_0, \dots, u_{n-1}$ et $v = v_0 \dots v_{n-1}$. Une méthode naïve consisterait à construire l'ensemble des sous-mots de u et de v, puis à calculer l'intersection de ces deux ensembles.

Question 6. *Évaluez la complexité de cette méthode en fonction de la taille des mots u et v.*

Dans un premier temps, nous nous limitons au problème du calcul de la *longueur* d'un plus long sous-mot commun de deux mots u et v. On note $\ell(i,j)$ la longueur du plus long sous mot commun des mots $u_0 \dots u_{i-1}$ et $v_0 \dots v_{j-1}$. La longueur recherchée est donnée par $\ell(m,n)$ (où m et n sont les longueurs respectives de u et v).

Question 7. *Donner une relation de récurrence permettant de caluler $\ell(m,n)$.*

Question 8. *En déduire une fonction* `subseq_length : 'a mot →'a mot→ int` *qui permette de calculer la longueur du plus long sous-mot commun à deux mots.*

Question 9. *En fait, la longueur du plus long sous-mot commun de deux mots est bien sûr reliée de près à la distance d'édition entre ces deux mots : prouver la relation*

$$E(n,m) = n + m - 2\ell(n,m)$$

On souhaite maintenant obtenir un des sous-mots communs de longueur maximale de deux mots. Pour cela, remarquons que la valeur de chaque case du tableau auxiliaire est obtenue à partir de celle de l'une de ses trois voisines : celle située au dessus, celle située à gauche ou celle située en diagonale en haut à gauche. En conservant (dans un second tableau) pour chaque case une marque indiquant à partir de quelle voisine sa valeur a été calculée, il est possible à la fin de l'algorithme de reconstituer un plus long sous-mot commun en parcourant le tableau obtenu en suivant les directions indiquées par les marques comme illustré sur la figure suivante :

	j	0	1	2	3
i		v_{j-1}	a	b	c
0	u_{i-1}	0	0	0	0
1	a	0	↖ 1	← 1	← 1
2	a	0	↖ 1	↑ 1	↑ 1
3	c	0	↑ 1	↑ 1	↖ 2
4	b	0	↑ 1	↖ 2	↑ 2

Pour représenter les marques, vous pouvez définir en Caml le type suivant :
`type marque = Nord | Ouest | NordOuest ;;`

Question 10. *Écrire une fonction* `subseq : 'a mot → 'a mot → 'a mot` *qui retourne un des plus longs sous-mots communs de deux mots.*