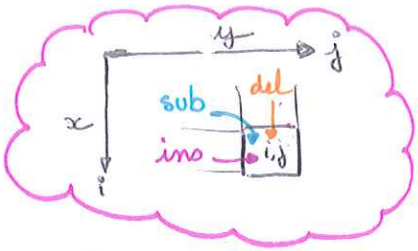
CALCUL DE LA DISTANCE DE LEVENSHTEIN ET D'UN ALIGNEMENT

Considérons les coûts del et ins fixés.
Considérons aussi sub une distance sur Σ fixée.

Soient $x = x_1 \cdots x_n$ et $y = y_1 \cdots y_m$ deux mots sur Σ.
On introduit $\underline{\ell_{i,j}} = Lev(x_1 \cdots x_i, y_1 \cdots y_j)$ pour $(i,j) \in [0..n] \times [0..m]$.

Ptés

- $\underline{\ell_{n,m}} = Lev(x,y)$



- $\begin{cases} \forall i \in [0..n] & \ell_{i,0} = i \times del \\ \forall j \in [0..m] & \ell_{0,j} = j \times ins \end{cases}$
- $\forall (i,j) \in [1..n] \times [1..m] \quad \underline{\ell_{i,j}} = \min \begin{cases} \ell_{i-1,j-1} + sub(x_i, y_j) \\ \ell_{i-1,j} + del \\ \ell_{i,j-1} + ins \end{cases}$

Preuve

a) Définition suffit

b) passer de $x_1 \cdots x_i$ à ε requiert au minimum de supprimer i lettres, et cela suffit, d'où $Lev(x_1 \cdots x_i, \varepsilon) = i \times del$

c) à l'inverse passer de ε à $y_1 \cdots y_j$ requiert au minimum d'insérer j lettres, et comme cela suffit si on insère les bonnes lettres dans l'ordre càd y_1 en posi 1, y_2 en posi 2 ... jusqu'à y_j en position j, on a $Lev(\varepsilon, y_1 \cdots y_j) = j \times ins$.

d) Considérons $(\tilde{x}, \tilde{y})$ un alignem^t de $x_1 \cdots x_i$ et $y_1 \cdots y_j$ de longueur r.
$(\hat{x}, \hat{y}) = (\tilde{x}_1 \cdots \tilde{x}_{r-1}, \tilde{y}_1 \cdots \tilde{y}_{r-1})$ reste un alignement, mais pas de x et y.

- Si $\tilde{y}_r = -$ alors $(\hat{x}, \hat{y})$ est un alignem^t de $(x_1 \cdots x_{i-1}, y_1 \cdots y_j)$ qui au minimum sera de coup $\ell_{n,m}$, auquel il faudra ajouter le coût de la dernière opération : $c(\tilde{x}_r, \tilde{y}_r) = c(\tilde{x}_r, -) = del$ donc au min $\ell_{i,j} = \underline{\ell_{i-1,j} + del}$
- Si $\tilde{x}_r = -$ alors $(\hat{x}, \hat{y})$ est un alignem^t de $(x_1 \cdots x_i, y_1 \cdots y_{j-1})$ dont le coût est au minimum $\ell_{n,m-1}$, donc au min $\ell_{i,j} = \underline{\ell_{i,j-1} + ins}$
- Si $\tilde{x}_r \neq -$ et $\tilde{y}_r \neq -$ $(\hat{x}, \hat{y})$ est un alignem^t de $(x_1 \cdots x_{i-1}, y_1 \cdots y_{j-1})$, et la dernière opéra^t est une substitution de cout $sub(x_i, y_j)$, donc $\ell_{i,j}$ vaut au minimum $\ell_{i,j} = \underline{\ell_{i-1,j-1} + sub(x_i, y_j)}$

d) (suite). On conclut en se rappelant que $l_{i,j}$ est le cout d'un alignement minimal, il faut donc considérer le min entre ces trois options.

Algo

La propriété précédente indique et prouve l'algorithme de programmation dynamique suivant pour calculer la distance de Levenshtein entre deux mots.

```
LEV(x, y)
    n ← longueur de x
    m ← ——————— y.
    L ← tableau indexé par [0..n] x [0..m]
    Pour i allant de 0 à n
        L[i, 0] ← i × (del)
    Pour j allant de 0 à m
        L[0, j] ← j × (ins)
    Pour i allant de 1 à n
        Pour j allant de 1 à m
            L[i, j] ← min(L[i-1, j] + (del) ; L[i, j-1] + (ins) ; L[i-1, j-1] + (sub(x_i, y_j)))
    Retourner L[n, m]
```

Complexité

* Cet algorithme a une complexité temporelle en $O(n \times m)$.

* tel qu'il est écrit sa complexité temporelle est en $O(n \times m)$, mais on peut facilement l'améliorer, en ne gardant en mémoire que la ligne courante et la précédente

Donc la complexité spatiale est plutôt en $O(m)$

Amélioration

Quitte à calculer à chaque étape le minimum entre les 3 opérations possibles, autant retenir laquelle (une de celle en fait) on a choisi, afin de pouvoir finalement retourner un alignement réalisant la distance annoncée

Algo bis

```
LEV_bis (x, y)
n ← |x|
m ← |y|.
L ← tableau indexé par [0..n] × [0..m]
M ← ———————————— [1..n] × [1..m]

Pour i allant de 0 à n
    L_{i,0} ← i × del
Pour j allant de 0 à m
    L_{0,j} ← j × ins

Pour i allant de 1 à n
    Pour j allant de 1 à m
        D ← L[i-1, j] + del
        I ← L[i, j-1] + ins
        S ← L[i-1, j-1] + sub(x_i, y_j)
        Si min(I, D, S) = I
        alors { L[i, j] ← I
                M[i, j] ← (i, j-1)
        Si min(I, D, S) = D
        alors { L[i, j] ← D
                M[i, j] ← (i-1, j)
        Si min(I, D, S) = S
        alors { L[i, j] ← S
                M[i, j] ← (i-1, j-1)

x̃ ← ε ; i ← n
ỹ ← ε ; j ← m
tant que i > 1 ou j > 1
    match M[i,j] with
    | (i, j-1) → { x̃ ← -x̃
                   ỹ ← y_j ỹ
    | (i-1, j) → { x̃ ← x_i x̃
                   ỹ ← -ỹ
    | (i-1, j-1) → { x̃ ← x_i x̃
                     ỹ ← y_j ỹ
    (i,j) ← M[i,j]
retourner (x̃, ỹ, L[n,m])
```

Cplx complexité temporelle $O(n \times m)$
spatiale $O(n \times m)$

exc Σ l'alphabet romain.

ins = del = 1

$$sub(\alpha,\beta) = \begin{cases} 0 & \text{si } \alpha=\beta \\ 1 & \text{si } \alpha \text{ et } \beta \text{ sont 2 voyelles} \\ 1 & \text{——————— consonnes} \\ 2 & \text{sinon} \end{cases}$$

NB c'est une distance !

x = RAPE m = 4
y = LAPIN m = 5

L :

i \ j →	0	1	2	3	4	5
	ε	L	A	P	I	N
0 ε	0	1	2	3	4	5
1 R	1	1	2	3	4	5
2 A	2	2	1	2	3	4
3 P	3	3	2	1	2	3
4 E	4	4	3	2	2	3

ex $1 = \min(1+1; 1+1; 0+1)$ s(L,R)

→ LEV(x,y) = 3

M

	1	2	3	4	5
1	(0,0)	(1,1)	(1,2)	(1,3)	(1,4)
2	(1,1)	(1,1)	(2,2)	(2,3)	(2,4)
3	(2,1)	(2,2)	(2,2)	(3,3)	(3,4)
4	(3,1)	(3,2)	(3,3)	(3,3)	(44)

$\tilde{x} = \varepsilon$, $\tilde{y} = \varepsilon$, $i = 4$, $j = 5$

comme M[4,5] (4,4) = (4,5-1) $\begin{cases} \tilde{x} = \_ \\ \tilde{y} = y_5 = N \end{cases}$

comme M[4,4] (3,3) = (4-1, 4-1) $\begin{cases} \tilde{x} = x_4 \_ = E\_ \\ \tilde{y} = y_4 N = IN \end{cases}$

puis de $\hat{m}$ $\begin{cases} \hat{x} = PE\_ \\ \hat{y} = PIN \end{cases}$

puis $\begin{cases} \tilde{x} = APE\_ \\ \tilde{y} = APIN \end{cases}$

puis $\begin{cases} \tilde{x} = RAPE\_ \\ \tilde{y} = LAPIN \end{cases}$

D'où

RAPE_ , 3
LAPIN
1 0 0 1 1