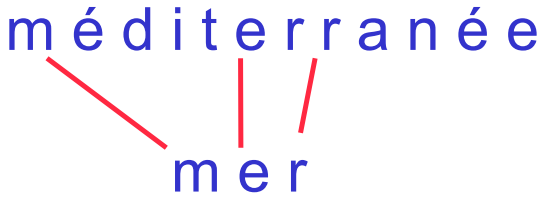
Sous-mots

mot = suite sur l'alphabet A
sous-mot = sous-suite



$u = u_0u_1\ldots u_{k-1}$ sous-mot de $x = x_0x_1\ldots x_{m-1}$
si $x = v_0u_0v_1u_1\ldots\ v_ku_{k-1}v_k$ avec $v_0,\ v_1,\ \ldots,\ v_k \in A^*$

« sous-mot de » : relation d'ordre sur A^*
SC(x, y) = {u sous-mot de x et de y}
Problème 1 :
Calculer PLSC$(x, y) = \max \{ |u| / u \in SC(x, y) \}$
Problème 2
Déterminer un mot $u \in SC(x, y)$ tel que $|u| = PLSC(x, y)$

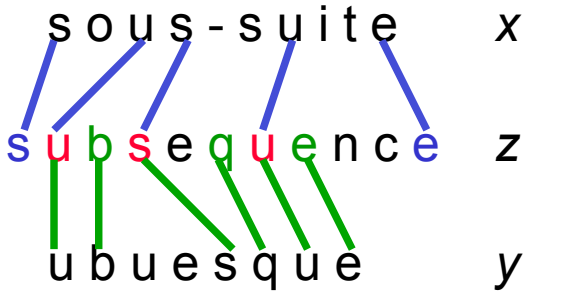
Distance

Comparaison de mots par distance

$d(x, y) = |x| + |y| - 2.\mathrm{PLCS}(x, y)$

d est une distance

- $d(x, y) \geq 0$
- $d(x, y) = 0$ ssi $x = y$
- $d(x, y) = d(y, x)$
- inégalité triangulaire ?



u = un plcs de x et z
v = un plcs de z et y
k = nombre d'occurrences de lettres de z communes à u et v

$$\begin{aligned} d(x, z) + d(z, y) &= (|x| + |z| - 2.|u|) + (|z| + |y| - 2.|v|) \\ &= |x| + |y| + (|z| - |u|) + (|z| - |v|) - |u| - |v| \\ &\geq |x| + |y| + (|v| - k) + (|u| - k) - |u| - |v| \\ &= |x| + |y| - 2k \\ &\geq d(x, y) \end{aligned}$$

diff

Comparaison de fichiers lettre = ligne

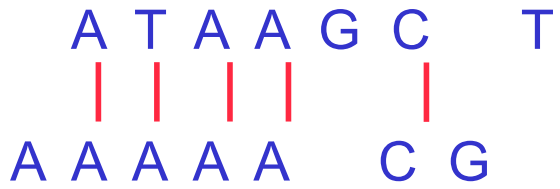
```
> cat A
Belle Marquise,
vos beaux yeux
me font mourir d'amour
> cat B
D'amour mourir me font,
Belle Marquise,
vos beaux yeux
> diff A  B
0 a 1
> D'amour mourir me font,
3 d 3
< me font mourir d'amour
```

Applications

- gestion de versions
- compression par stockage de différences
- restauration avec "ed" , "sed", ...

Alignement

Comparaisons de séquences moléculaires lettre = nucléotide



Transformation de *x* en *y* par :

insertions, suppressions, remplacements

Chaque opération possède un coût

Problème :

calculer le coût minimal de la transformation (distance d'alignement)

un alignement correspondant

Distance d'édition

Calcul de PLSC

Par énumération de sous-mots

temps exponentiel

Par "programmation dynamique"

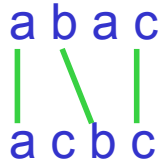
temps $O(|x| \times |y|)$

Par automate

même temps

Propriété

$x = x_0x_1\ldots x_{m-1}$ $\quad y = y_0y_1\ldots y_{n-1}$
$u = u_0u_1\ldots u_{k-1}$ un plcs de x et y

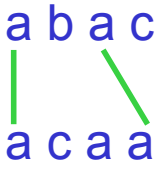


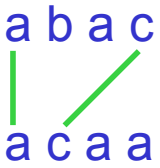
Proposition

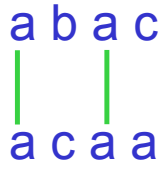
$x_{m-1} = y_{n-1} \Rightarrow u_{k-1} = x_{m-1} = y_{n-1}$
et $u_0u_1\ldots u_{k-2}$ plsc de $x_0x_1\ldots x_{m-2}$ et $y_0y_1\ldots y_{n-2}$

$x_{m-1} \neq y_{n-1}$ et $u_{k-1} \neq x_{m-1} \Rightarrow$ u plsc de $x_0x_1\ldots x_{m-2}$ et y

$x_{m-1} \neq y_{n-1}$ et $u_{k-1} \neq y_{n-1} \Rightarrow$ u plsc de x et $y_0y_1\ldots y_{n-2}$







Algorithme

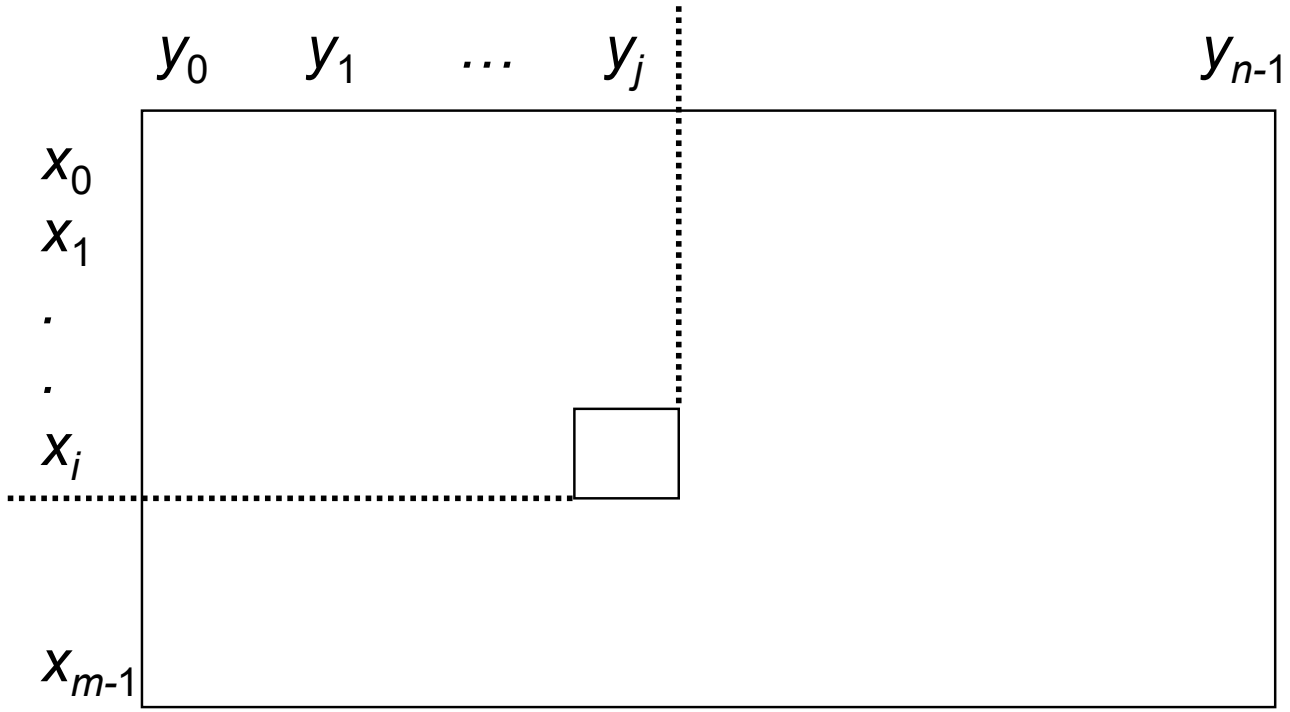
```
entier PLSC(mot x, longueur m, mot y, longueur n){
      si (m = 0 ou n = 0) alors
            retour 0 ;
      sinon si (x[m-1] = y[n-1]) alors
            retour PLSC(x,m-1,y,n-1)+1
      sinon
            retour max{PLSC(x,m,y,n-1),PLSC(x,m-1,y,n)}
}
```

Stupidement exponentiel !

Programmation dynamique

Problème décomposable en nombre fini de sous-problèmes mais chevauchement des sous-problèmes

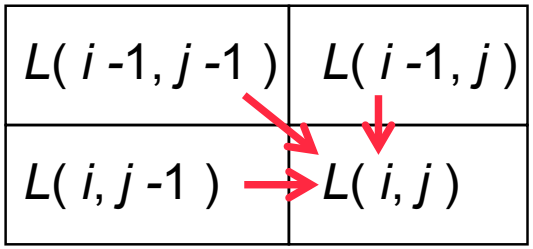
Méthode : mémoriser les résultats intermédiaires



Récurrence

$L(i, j) = \text{PLSC}(x_0 x_1 \ldots x_i, y_0 y_1 \ldots y_j)$

$$L(i,j) = \begin{cases} 0 & \text{si } i = -1 \text{ ou } j = -1 \\ L(i-1, j-1) + 1 & \text{sinon si } x_i = y_j \\ \max(L(i, j-1), L(i-1, j)) & \text{sinon} \end{cases}$$



Exemple

L		-1	0	1	2	3	4	5	6	7	8
			c	b	a	c	b	a	a	b	a
-1		0	0	0	0	0	0	0	0	0	0
0	a	0	0	0	1	1	1	1	1	1	1
1	b	0	0	1	1	1	2	2	2	2	2
2	c	0	1	1	1	2	2	2	2	2	2
3	d	0	1	1	1	2	2	2	2	2	2
4	b	0	1	2	2	2	3	3	3	3	3
5	b	0	1	2	2	2	3	3	3	4	4

PLCS(a b c d b b , c b a c b a a b a) = 4

Algorithme

```
entier PLSC(mot x, longueur m, mot y, longueur n){
     pour i ← -1 à m-1 faire L[i,0] ← 0 ;
     pour j ← -1 à n-1 faire L[0,j] ← 0 ;
     pour i ← 0 à m-1 faire
          pour j ← 0 à n-1 faire
               si x[i] = y[j] alors{
                    L[i,j] ← L[i-1,j-1]+1 ;
                    P[i,j] ← '\' ;
               }sinon si L[i-1,j] ≥ L[i,j-1] alors{
                    L[i,j] ← L[i-1,j] ;
                    P[i,j] ← '↑' ;
               }sinon{
                    L[i,j] ← L[i,j-1] ;
                    P[i,j] ← '←' ;
               }
     retour L[m-1,n-1] ;
}
```

Exemple (suite)

L		-1	0	1	2	3	4	5	6	7	8
			c	b	a	c	b	a	a	b	a
-1		0	0	0	0	0	0	0	0	0	0
0	a	0	0	0	1	1	1	1	1	1	1
1	b	0	0	1	1	1	2	2	2	2	2
2	c	0	1	1	1	2	2	2	2	2	2
3	d	0	1	1	1	2	2	2	2	2	2
4	b	0	1	2	2	2	3	3	3	3	3
5	b	0	1	2	2	2	3	3	3	4	4

PLCS(abcdbb , cbacbaaba) = 4

P		-1	0	1	2	3	4	5	6	7	8
			c	b	a	c	b	a	a	b	a
-1		0	0	0	0	0	0	0	0	0	0
0	a	0	↑	↑	\	←	←	\	\	←	\
1	b	0	↑	\	↑	↑	\	←	←	\	←
2	c	0	\	↑	↑	\	↑	↑	↑	↑	↑
3	d	0	↑	↑	↑	↑	↑	↑	↑	↑	↑
4	b	0	↑	\	←	↑	\	←	←	\	←
5	b	0	↑	\	↑	↑	\	↑	↑	\	←

acbb est un plcs

Variantes

Problème ouvert

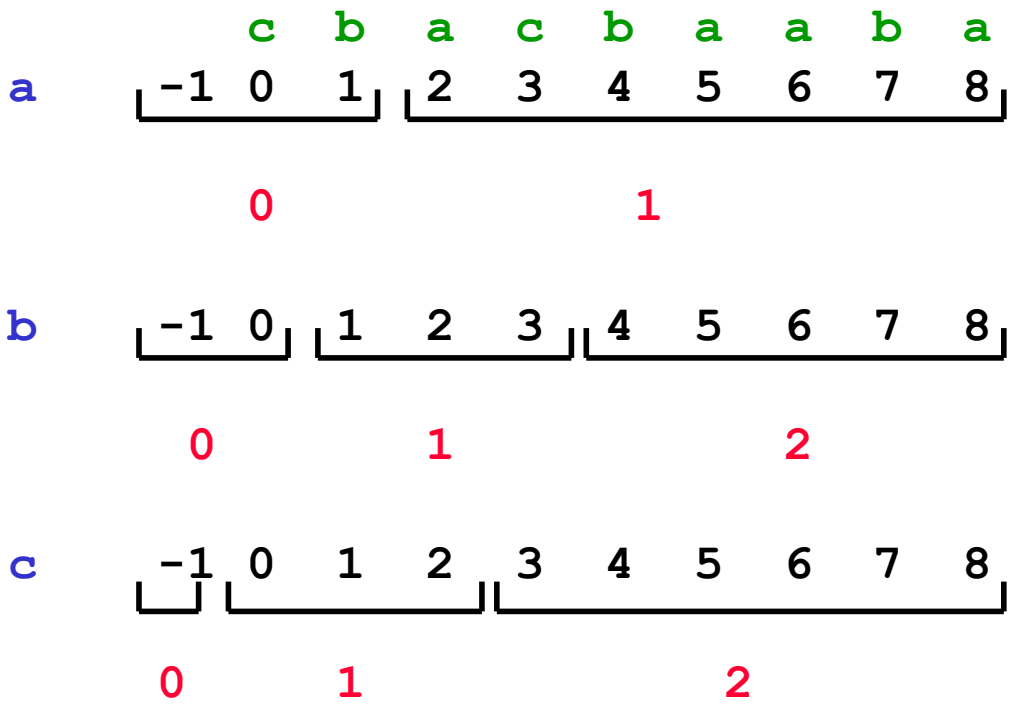
calcul de PLSC en temps < O(n^2 / log n)

Variantes

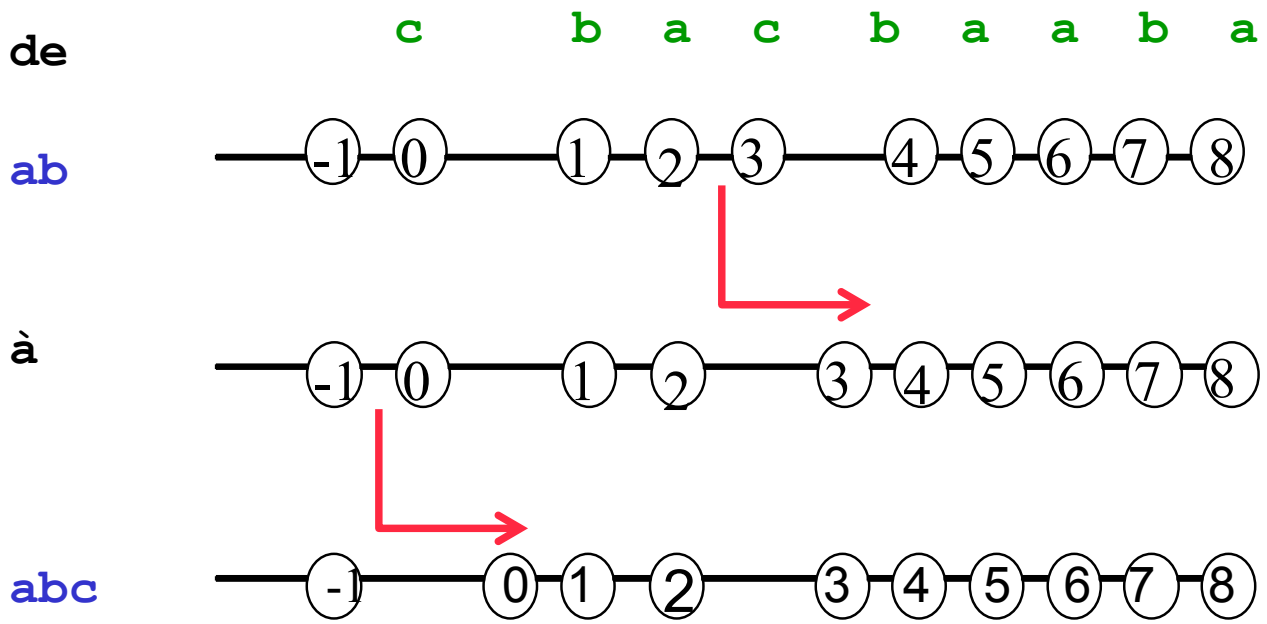
- calcul de PLSC en espace linéaire
  [une ligne de L suffit]
- calcul d'un plsc en espace linéaire
  [algorithme de Hirschberg]
- utilisation des « dominants »
- algorithme rapide en pratique
  bien que temps maximal = O(n^2 log n)
  dans certaines implantations de **`diff`**
  [algorithme de Hunt et Szymanski]

Algorithme de Hunt & Szymanski

Calcul de la matrice *L* par ligne



Comme un boulier



Algorithme

$x = x_0 x_1 \ldots x_{m-1}$ $\quad y = y_0 y_1 \ldots y_{n-1}$

Pos[a] = { j | $y_j = a$ } pour $a \in A$

Pour i fixé : J [k] = { j | PLSC($x_0 x_1 \ldots x_i$, $y_0 y_1 \ldots y_j$) = k }

```
entier PLSC(mot x, longueur m, mot y, longueur n){
      J[0] ← {-1,0,1,…,n-1};
      pour k ← 1 à n faire J[k] ← ∅ ;
      pour i ← 0 à m-1 faire
            pour chaque p ∈ Pos[x[i]] en décroissant faire{
                  k ← CLASSE(p);
                  si k = CLASSE(p-1) alors{
                        (J[k],X) ← PARTAGE(J[k],p);
                        J[k+1] ← UNION(X,J[k+1]);
                  }
            }
      retour  CLASSE(n-1);
}
```

Type abstrait "PARTAGE/FUSION"

Ensemble de base

suites (E_0, E_1, ... , E_n) d'ensembles disjoints
telles que $E_0 \cup E_1 \cup ... \cup E_n = \{ 0, 1, ... , n \}$

Opérations

Initialisation
CLASSE : élément → ensemble
CLASSE(p) = k tel que $p \in E_k$
PARTAGE : élément x ensemble → ensemble x ensemble
PARTAGE(E, p) = (S, T)
$S = \{ q \in E \mid q < p \}$, $T = \{ q \in E \mid q \geq p \}$
UNION : ensemble x ensemble → ensemble

Implémentations possibles

listes, arbres, ...
Si les E_k sont des intervalles : B-arbres, 2-3-arbres, ...

Temps de calcul de PLSC

2-3-arbres pour les J_k

initialisation	O (*n*)
CLASSE(*p*)	O (log *n*)
PARTAGE(*J, p*)	O (log *n*)
UNION (*X, J*)	O (log *n*)

Temps : si Pos listes pré-calculées sur *y*

$O(\log n \cdot \sum (| Pos (x_i) | i = 0, 1, \ldots, n-1)$
$= O(\log n \cdot \text{card} \{ (i, j) / x_i = y_j \} |)$
$= O(n \cdot m \cdot \log n)$

En pratique, sur fichiers : O (n. log n)

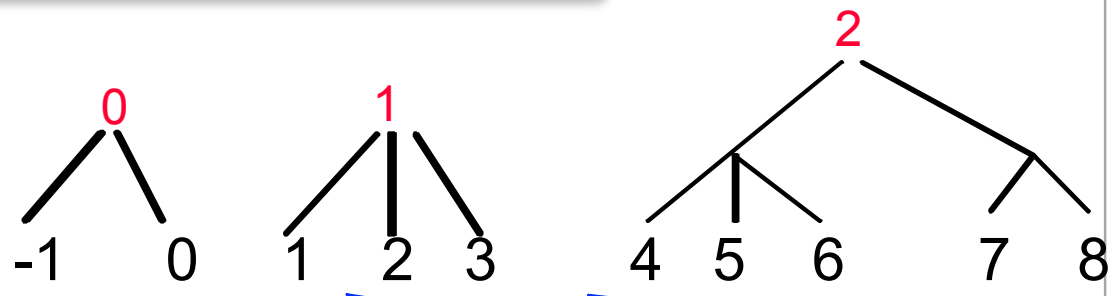
Pré-calcul de Pos

$| y | = n \rightarrow O(n)$

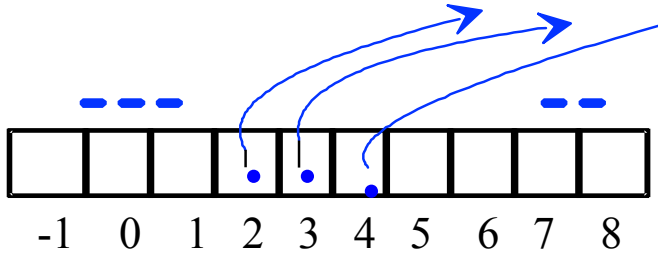
si fichiers de *n* lignes, par hachage O(*n*) en moyenne

Utilisation des 2-3 arbres

Numéro de classe à la racine



Pointeurs sur feuilles



CLASSE : par liens « père »
PARTAGE : en descendant, duplication, et restructuration
UNION : rattachement au bon niveau, et restructuration