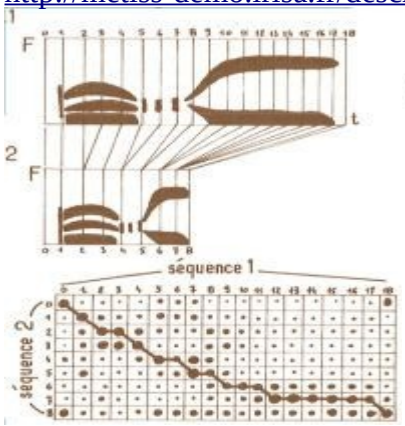
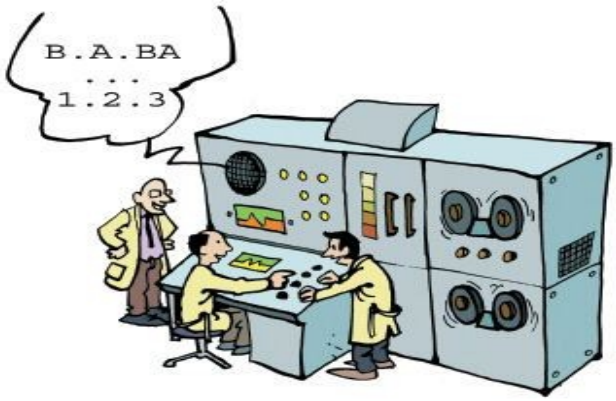
TP/TD 3 : Programmation Dynamique

Si vous avez eu le TP/TD sur Fibonacci récursif/itératif/complexité : Un algorithme qui utilise l'astuce vue précédemment pour le calcul de la suite de Fibonacci est la programmation dynamique. Une version naturelle récursive serait trop coûteuse en terme de complexité de calcul. Une version heuristique telle que la programmation dynamique a donné de très bon résultat en reconnaissance de la parole puis en bio-informatique pour l'alignement de séquence discrètes de tailles différentes.
http://metiss-demo.irisa.fr/descriptions/





Illustrons avec le problème de la plus longue sous-séquence de caractères commune à deux chaînes de caractères S1 et S2 (LCS : Longest Common Subsequence). Notez bien que le problème est différent de la Longest Common Substring (ici ce serait simplement GCG : les caractères de la solution doivent être contigus dans S1 et S2). Par ailleurs ces problèmes sont fortement liés à des analyses de complexité délicates et notamment de problèmes NP-complet (ce qui dépasse l'esprit de ce TP).
Soient deux chaînes de caractères de taille différente :
S1 = **GCC**CTA**G**CG
S2 = **GC**G**CA**AT**G**

La LCS de ces deux chaînes S1 et S2 est LCS(S1,S2)= **GCCAG**
Elle est de taille 5. C'est une façon de **mesurer la similarité entre deux séquences discrètes de taille différente.**

Modélisation récursive :
Soient :
C1 le caractère le plus à droite de S1 soit G
C2 le caractère le plus à droite de S2 soit G

S1 = S1'.C1 (. est le symbole de concaténation)
S2 = S2'.C2

Il y a alors trois problèmes récursifs à traiter :
L1=LCS(S1',S2) ou
L2=LCS(S1,S2') si C1!= C2
L3=LCS(S1',S2') si C1==C2

La solution est la plus longue des trois chaînes suivantes :
L1
L2
L3.C1 si C1=C2 et L3 sinon

On trouvera une belle explication ici https://www.guru99.com/fr/longest-common-subsequence.html

Cet algorithme est très coûteux en terme de temps de calcul (complexité exponentielle $O(k^{nm})$). D'où une méthode plus coûteuse en stockage mémoire mais plus rapide en temps de calcul (complexité quadratique en n et m les tailles de deux séquences O(nm)) : la programmation dynamique.
Le cas trivial est le cas où L1= " " ou L2 = " " (chaînes vides) auquel cas LCS(L1,L2) = " " , ce qui nous servira pour l'initialisation des bords de la matrice de coût que nous allons construire.
Matrice de coûts initial avec coûts initiaux à 0 et remplissage à mi-parcours de l'algorithme.

		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2	2
G	0	1	2	2	2	2	2	3	3	3
C	0	1	2	3						
A	0									
A	0									
T	0									
G	0									

Chaque case correspond à un score (= longueur de la LCS au point courant de résolution) pour le sous-problème considéré. Le tableau se remplit de gauche à droite et de haut en bas.
Pour remplir une case, on considère les positions de case suivantes comme figurés avec les flèches.
V1 : Case à l'Ouest
V2 : Case au Nord
V3 : Case au Nord-Ouest

Exercice : Faire un schéma générique de la case à remplir et des cases V1, V2 et V3.

Dans la cellule à remplir, on affecte le maximum de :
V1
V2
V3+1 si C1=C2 et V3 sinon où C1 et C2 sont respectivement les valeurs de caractères correspondant en colonne et en ligne de la case considérée.

Parallèlement on garde pour chaque case une trace du chemin en affectant une flèche de type V1, V2 ou V3 en fonction de l'origine de la valeur maximale retenue. Ces flèches sont essentielles et vont permettre de réaliser la dernière étape de back-tracking pour reconstruire la LCS à la fin du calcul des scores.

Cas ambigus : En cas d'égalité de valeurs de case, faire un choix arbitraire mais toujours le même par exemple ambiguïté entre case V1 et V3 choisir V3 systématiquement. A la fin de l'algorithme le score final LCS(S1,S2) sera le même quel que soit ce choix systématique. En général privilégiez la case V3.

Exercice : Remplir le tableau à la main (*forward*)

Pour le tracing back (principe de la programmation dynamique, c'est par essence un processus off-line), on part de la case en bas à droite et on remonte le chemin d'alignement selon l'algorithme suivant :

Si Cell.V3

Si Cell.V3.value = Cell.value -1, ajouter les caractères C1 (et C2 car C1=C2) correspondant à la cellule Cell courante

Sinon Si Cell.V3.value = Cell.value, on passe le caractère courant,

Si Cell.V1 ou Cell.V2

on passe le caractère C1 en colonne ou C2 en ligne respectivement.

Exercice : Faire le *tracing back* pour trouver la sous-chaîne commune maximale à partir de la case en bas à droite. Remarque : la complexité de cette étape est linéaire en n et m O(n+m)

Ecrire l'alignement obtenu avec les «sauts» via le symbole « - ».

L'alignement de séquence de type Needleman-Wunsch (NW) ou Smith-Waterman (SW) utilise le même principe de la programmation dynamique avec des fonctions de coût différentes (*mismatch*, **match** et *gap (-)*) pour remplir les scores (valeurs des cases) et obtenir un alignement plus renseigné

LCS :

GCCAG

Alignement Global (NW) :

S1 =	**G**	**C**	*C*	**C**	T	**A**	*G*	*C*	**G**
S2 =	**G**	**C**	G	**C**	-	**A**	*A*	*T*	**G**

Alignement local (SW) :

S1 = GCCCTA**GCG**

S2 = **GCG**CAATG

Perl, Java, C

Tester le programme *Perl* suivant de calcul de la plus grande sous-chaîne commune entre deux chaînes de caractères (Longuest Common Substring)

```
$perl LCS.prl "coucou" "corageco"

$perl LCS.prl "AGGGTTTTGGGAA" "AGCCCCGTTAT"
```

Compiler le code *LCS.c*

```
$gcc -c LCS.c
$gcc -o LCS LCS.o
```

et exécuter le :

```
$./LCS “coucou” “corageco”
```

Compiler le code *LCS.java*

```
$javac LCS.java
```

et exécuter le

```
$java LCS ou $java -classpath . LCS
```

Comparer les temps d'exécution :
en *perl*,

```
# start timer
$start = time();
# perform a math operation 200000 times
...
# end timer
$end = time();
# report
print "Time taken was ", ($end - $start), " seconds";
```

En C, Java ?
https://fr.wikiversity.org/wiki/Fonctions_de_base_en_langage_C/time.h
En Java ?

```
long begin = System.currentTimeMillis();
...
long end = System.currentTimeMillis();
float time = ((float) (end-begin)) / 1000f;
```

Tracer un graphique correspondant à la complexité en terme de temps de calcul de ces algorithmes (3 courbes, une par implémentation, en abscisse la quantité nm et en ordonnée le temps de calcul).