PROGRAMMATION DYNAMIQUE

INTRODUCTION

C'est une méthode introduite dans le cadre des algorithmes d'optimisation.

Optimisation. — Répond à la problématique de maximisation d'une fonction sous un ensemble de contraintes.

Exemples. — En autres :
- — la planification ;
- — satisfaction de contraintes ;
- — mathématiques financières.

Il y a deux grandes familles de méthodes en optimisation :
- — la programmation dynamique ;
- — la programmation linéaire.

1. CALCULER UNE SUITE DÉFINIE PAR UNE RÉCURRENCE SUR DEUX INDICES

Soit $f : \mathbf{N}^3 \to \mathbf{N}$. On suppose que l'on dispose d'un algorithme F pour calculer f. On s'intéresse à la suite définie par :
- — $(a_{0,j})_{j \in \mathbf{N}}, (a_{i,0})_{i \in \mathbf{N}}$ donnés ;
- — $a_{i+1,j+1} = f(a_{i,j+1}, a_{i+1,j}, a_{i,j})$.

On vérifie qu'il s'agit bien d'une suite :

Lemme 1.0.0.1. —
Soient $(a_{i,j})_{i,j \in \mathbf{N}}$ et $(b_{i,j})_{i,j \in \mathbf{N}}$ telles que :

$$a_{i+1,j+1} = f(a_{i,j+1}, a_{i+1,j}, a_{i,j})$$
$$b_{i+1,j+1} = f(b_{i,j+1}, b_{i+1,j}, b_{i,j})$$

et :
$$\forall i \in \mathbf{N},\ a_{i,0} = b_{i,0} \text{ et } a_{0,i} = b_{0,i}.$$
Alors $a_{i,j} = b_{i,j}$ pour tout couple $(i,j) \in \mathbf{N}^2$.

Démonstration 1.0.0.1. —
Par double récurrence.
On imbrique une récurrence (externe) sur i avec une récurrence (interne) sur j. Sur i on veut montrer :
$$P_i \equiv \forall j \in \mathbf{N},\ a_{i,j} = b_{i,j}.$$
Par définition, P_0 est vraie. On suppose P_i vérifiée pour tous $i \leq i_0$. Il s'agit de montrer P_{i_0+1} par récurrence sur j.
On définit la proposition :
$$Q_j^{i_0+1} \equiv a_{i_0+1,j} = b_{i_0+1,j}.$$
On remarque que P_{i_0+1} est vraie si, et seulement si, $Q_0^{i_0+1}, Q_1^{i_0+1}, \ldots$ sont vraies. C'est-à-dire que P_{i_0+1} est vraie si, et seulement si :
$$\bigwedge_{j \in \mathbf{N}} Q_j^{i_0+1}.$$
Par définition, $Q_0^{i_0+1}$ est vraie. On suppose $Q_j^{i_0+1}$ vraie pour tous $j \leq j_0$. On a :
$$\begin{aligned} a_{i_0+1,j_0+1} &= f(a_{i_0,j_0+1}, a_{i_0+1,j_0}, a_{i_0,j_0}) \\ a_{i_0+1,j_0+1} &= f(b_{i_0,j_0+1,b_{i_0+1,j_0},b_{i_0,j_0}} \\ a_{i_0+1,j_0+1} &= b_{i_0+1,j_0+1} \end{aligned}$$
et donc $Q_{j_0+1}^{i_0+1}$ est vraie.
On en conclut que pour tout i, P_i est vraie et donc le lemme est démontré.

Soit T un tableau tel que $T[i,j] = a_{i,j}$.

Algorithme de calcul de T. — On se donne un algorithme A qui calcule $(a_{i,0})$ et un autre algorithme B qui calcule $(a_{0,i})$.

```
function tabIter(n,m)
        for i from 0 to n do
                T[i,0] = algoA(i)
        for j from 0 to m do
                T[0,j] = algoB(j)
        for i from 1 to n do
                for j from 1 to m do
                        T[i,j]=F(T[i-1,j],T[i,j-1],T[i-1,j-1])
        return T
```

```
function tabRec(i,j)
        if i = 0 then
                return algoA(i)
        if j = 0 then
                return algoB(j)
        return F(tabRec(i-1,j),tabRec(i,j-1),tabRec(i-1,j-1))
```

2. LCS (LEAST COMMON SUBSEQUENCE) : PROBLÈME DE LA PLUS LONGUE SOUS-SUITE COMMUNE

Exemple. — Supposons que l'on dispose de deux brins d'ADN :

$$\mathrm{u} = \mathrm{t\ t\ a\ t\ a\ t\ g\ c\ g\ t}$$
$$\mathrm{v} = \mathrm{t\ a\ t\ c\ c\ c\ c\ t\ t\ a},$$

le mot « t a t t t » apparait dans les deux brins et est le plus grand.

Soit Σ un alphabet fini. Soient $u, v \in \Sigma^*$. Une sous-suite commune à u et v est un mot $w \in \Sigma^*$ tek que :

$$\exists 1 \leq i_0 < i_1 < \ldots < i_{|w|}, \exists 1 \leq j_0 < j_1 < \ldots < j_{|w|}, \forall k \leq |w|\,,\ w_k = u_{i_k} = v_{j_k}.$$

Notations. — On suppose que les mots $w \in \Sigma^*$ sont indexés de 1 à $|w|$. On note également

$$w[1\ldots i] = w[1]w[2]\ldots w[i] = w_1 w_2 \ldots w_i.$$

Dans la suite on considère le tableau $A : \{0,\ldots,n\} \times \{0,\ldots,m\}$ tel que $A[i,j]$ est la longueur de la plus longue sous-suite commune à $u[1\ldots i]$ et $v[1\ldots j]$.

PROPOSITION 2.0.0.1 (Optimalité des sous-structures). —
Si w est une sous-suite commune à u et v de longueur maximale k, alors pour tout $h \leq k$, $w[1\ldots h]$ est une sous-suite commune maximale à $u[1\ldots i_h]$ et $v[1\ldots j_h]$.

DÉMONSTRATION 2.0.0.2. —
$w[1\ldots h]$ est bien une sous-suite commune. Si b était une sous-suite, strictement plus grande, à $u[1\ldots i_h]$ et $v[1\ldots j_h]$ alors $bw[h+1,\ldots k]$ serait une sous-suite qui contredirait la maximalité de w.

Notation. — $\wedge_{x=y}$ est la fonction constante égale à 1 si $x = y$ et 0 sinon.

Proposition 2.0.0.2. —
Si $A[0,j] = A[i,0] = 0$. Alors

$$A[i+1,j+1] = \max \begin{cases} A[i+1,j] \\ A[i,j+1] \\ \wedge_{u[i+1]=v[j+1]} + A[i,j] \end{cases}.$$

Démonstration 2.0.0.3. —
Si une plus longue sous-suite commune à $u[1 \dots i+1]$ et $v[1 \dots j+1]$ à pour dernier élément $u[i+1] = v[j+1]$ alors c'est vérifié.
Si ce n'est pas le cas, le dernier élément de w est différent, soit de $u[i+1]$ soit de $v[j+1]$

- dans le premier cas, w est une plus longue sous-suite commune à $u[1 \dots i]$ et $v[i \dots j+1]$ et c'est donc vérifié ;
- dans le second cas, w est une plus longue sous-suite commune à $u[1 \dots ui+1]$ et $v[i \dots j]$ et c'est vérifié.

```
function tabDynLCS(u,v)
        n, m = len(u), len(v)
        for i from 0 to n do A[i,0] = 0
        for j from 0 to m do A[0,j] = 0
        for i from 1 to n do
                for j from 1 to m do
                        if A[i-1,j] > A[i,j-1] then
                                L[i,j] = nord
                                A[i,j] = A[i-1,j]
                        else
                                L[i,j] = ouest
                                A[i,j] = A[i,j-1]
                        if u[i] = v[j] and A[i,j] < 1 + A[i-1,j-1] then
                                L[i,j] = nord-ouest
                                A[i,j] = 1 + A[i-1,j-1]
        return A,L
function LCS(L,u,v)
        k, i, j = 0, len(n), len(m)
        while min(i,j)) > 0 do
                if L[i,j] = nord-ouest then
                        k = k +1
                        z[k] = u[i-1]
                        i = i - 1
                        j = j -1
```

```
                        else if L[i,j] = ouest then j = j - 1
                        else i = i -1
        return reverse(z)
```