

Programmation dynamique

D.E ZEGOUR

École Supérieure d'Informatique

ESI

Programmation dynamique

Sommaire

A. Principe général

B. Applications

Triangle de Pascal

Suite de Fibonacci

Série mondiale

Les plus courts chemins

Multiplication chaînée de matrices

Programmation dynamique

Principe général

Cas de problème composé de plusieurs sous problèmes identiques.

Résoudre sans tenir compte de cette duplication --> un algorithme très inefficace.

Résoudre en tenant compte la duplication --> un algorithme performant.

(Sauvegarde des résultats déjà calculés)

Idée de base :

Éviter de calculer deux fois la même chose,

Comment ?

Avec utilisation d'une table de résultats déjà calculés, remplie au fur et à mesure qu'on résout les sous problèmes.

Programmation dynamique

Caractéristiques

C'est une méthode ascendante : On commence d'habitude par les sous problèmes les plus petits et on remonte vers les sous problèmes de plus en plus difficile.

La programmation dynamique est souvent employée pour résoudre des problèmes d'optimisation satisfaisant le principe d'optimalité :

"Dans une séquence optimale (de décisions ou de choix), chaque sous-séquence doit aussi être optimale".

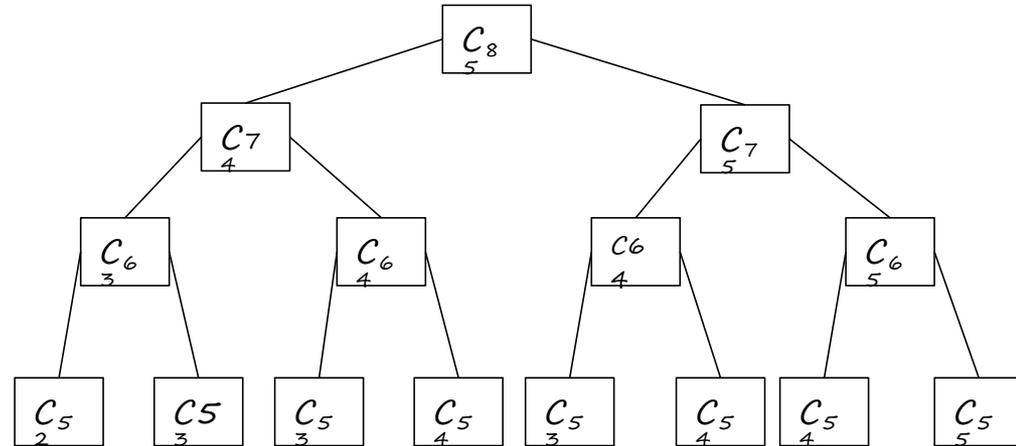
Programmation dynamique

Application 1 : Calcul de C_n^p

Formules :

$$C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$$

$$C_0^0 = C_n^0 = C_n^n = 1$$



On peut donner un algorithme récursif.

$C(n,p)$:

Si $(n=p)$ ou $(p=0)$

Retourner(1)

Sinon

Retourner($C(n-1,p-1) + C(n-1,p)$)

Fsi

Complexité :

$$k = n + p$$

$$T(k) = a \text{ si } k=1$$

$$T(k) = T(k-1) + T(k-2) + b$$

$$\rightarrow T_k - T_{k-1} - T_{k-2} = b = 1^n b$$

$$\rightarrow (x^2 - x - 1)(x - 1) = 0$$

Racines : $(1 + \sqrt{5})/2$, $(1 - \sqrt{5})/2$ et 1

$$O((1 + \sqrt{5})/2)^k \rightarrow O((1 + \sqrt{5})/2)^{2n}$$

Programmation dynamique

	Colonne p-1	Colonne p

Ligne i-1	(i-1,p-1)	(i-1, p)
Ligne i		(i, p)

Application 1 : Calcul de C_n^p

Calcul par programmation dynamique : Triangle de Pascal

Utilisation d'une table : remplir la ligne i à partir de la ligne i-1

Le p-ieme élément de la ligne i : $C(i, p) = C(i-1, p-1) + C(i-1, p)$

	0	1	2	3	4
0!	1				
1!	1	1			
2!	1	2	1		
3!	1	3	→ 3	1	
4!	1	4	6	4	1

Complexité
 $O(np) = O(n^2)$

Programmation dynamique

Application 2 : Calcul de Fib(n)

Formules :

$Fib(n) = Fib(n-1) + Fib(n-2)$ pour $n > 1$

$Fib(0) = 0, Fib(1) = 1$

Algorithme récursif

Complexité :
 $O((1 + \sqrt{5})/2)^n$

Fib(n) :

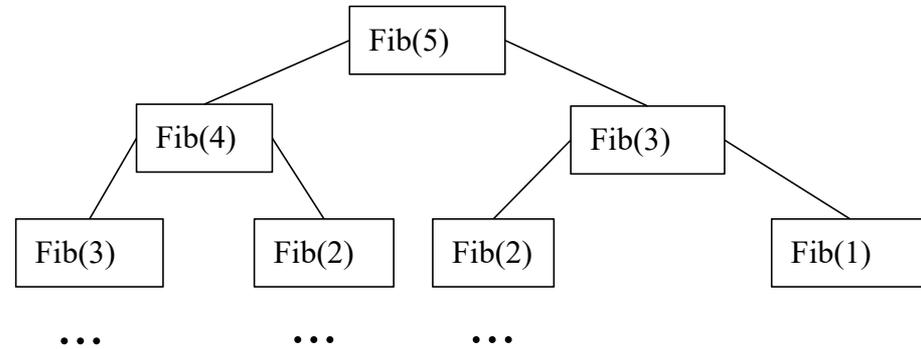
Si(n=0) ou (n=1)

Retourner(n)

Sinon

Retourner(Fib(n-1) + Fib(n-2))

Fsi



Algorithme iteratif

Complexité :
 $O(n)$

Fib(N)

SI (N = 0) OU (N = 1) Fib := N

SINON

A := 0 ; B := 1 ;

POUR I := 2 , N

C := A + B ;

A := B ;

B := C ;

FPOUR ; Fib := C FSI

Programmation dynamique

Application 3 : Série mondiale

A et B disputent une série de matchs

L'équipe gagnante est celle qui reporte n victoires.

--> (au maximum $2n-1$ matchs)

Supposition :

La probabilité que A gagne une partie = q_1

La probabilité que B gagne une partie = q_2

Programmation dynamique

Application 3 : Série mondiale

On définit $P(i, j)$ comme la probabilité que A remporte la série, sachant qu'elle doit encore gagner i victoires contre B j victoires.

On a ainsi :

$$P(0, j) = 1 \quad (\text{A a déjà gagné la série et } j > 0)$$

$$P(i, 0) = 0 \quad (\text{A a déjà perdu la série et } i > 0)$$

$$P(0, 0) \text{ indéfini}$$

$$P(i, j) = P(i-1, j) + P(i, j-1) \quad i, j > 0 \quad (\text{sans les probabilités})$$

$$P(i, j) = q_1 * P(i-1, j) + q_2 * P(i, j-1) \quad i, j > 0$$

Programmation dynamique

$$P(0, j) = 1$$

$$P(i, 0) = 0$$

$$P(i, j) = q_1 * P(i-1, j) + q_2 * P(i, j-1) \quad i, j > 0$$

Application 3 : Série mondiale

Algorithme récursif :

P(i, j) :

Si $i = 0$ et $j > 0$:

 P := 1

Sinon

 Si $i > 0$ et $j = 0$: P := 1

 Sinon

 Si $i > 0$ et $j > 0$:

 P := $q_1 * P(i-1, j) + q_2 * P(i, j-1)$

 Fsi

 Fsi

Fsi

Constater la duplication ...

T(k) = temps d'exécution de
P(i, j) avec $k = i+j$

$$\rightarrow T(k) = 2T(k-1) + b$$

$$\rightarrow T_k - 2 T_{k-1} = b = 1^n b$$

$$\rightarrow (x - 2) (x - 1) = 0$$

Solution

$$T(k) = C_1 2^k + C_2 1^k$$

$$\rightarrow O(2^k) = O(2^{i+j}) = O(2^{2n})$$

$$P(n, n) = O(4^n)$$

$$P(i, j) = q_1 * P(i-1, j) + q_2 * P(i, j-1) \quad i, j > 0$$

Programmation dynamique

Application 3 : Série mondiale

	Colonne i -1	Colonne j
Ligne i-1	 (i-1, j)
Ligne i	(i, j-1)	(i, j)

Programmation dynamique

Utilisation d'une table : remplir diagonale par diagonale

	0	1	2	3	4

0	1	1	1	1	1
1	0	1/2	3/4	7/8	15/16
2	0	1/4	1/2	11/16	13/16
3	0	1/8	5/16	1/2	21/32
4	0	1/16	3/16	11/32	1/2

CAS $p=q = 1/2$

Programmation dynamique

Application 3 : Série mondiale

Programmation dynamique : Algorithme

$P(i, j)$:

Pour $S = 1, i+j$

$P[0, S] := 1$

$P[S, 0] := 0$

Pour $k=1, S-1$

$P[k, S-k] := q_1P[k-1, S-k] + q_2P[k, S-k-1]$

Finpour

Finpour

Calcul de $P(2,4)$

$S = 1, 6$

$S=1$

--> $P[0,1]=1; P[1,0]=0$

$S=2$

--> $P[0,2]=1; P[2,0]=0$

$P[1,1]$

$S=3$

--> $P[0,3]=1; P[3,0]=0$

$P[1,2]$

$P[2,1]$

$S=4$

--> $P[0,4]=1; P[4,0]=0$

$P[1,3]$

$P[2,2]$

$P[3,1]$

Etc.

Complexité : $O(n^2)$

Programmation dynamique

Application 4 : Les plus courts chemins (Algorithme de Floyd)

Problème : trouver les plus courts chemins entre chaque couple de sommets d'un graphe.

Une solution

Il existe C_n^2 couples de sommets (X, Y)

Pour chaque (X, Y), trouver le plus court chemin de X à Y

Appliquer un parcours en profondeur (préordre) sur l'arbre de racine X en évitant les boucles infinies.

Si le nombre de noeuds et d'arcs est important, le problème devient combinatoire :
 $O(n^m)$, n nombre de noeuds et m nombre d'arcs

Programmation dynamique

Application 4 : Les plus courts chemins (Algorithme de Floyd)

Solution par programmation dynamique : Utilisation d'une matrice

C'est un problème d'optimisation et il vérifie le principe d'optimalité :

Si le plus court chemin entre A et B passe par un sommet C, alors les sous-chemins A-C et C-B doivent aussi être optimaux.

Programmation dynamique

Application 4 : Les plus courts chemins (Algorithme de Floyd)

Soit L la matrice associée à un graphe G avec

$L[i,j]$ représentant le coût de l'arc entre i et j .

$L[i,i] = 0$

$L[i,j] = \infty$ s'il n'y a pas d'arc entre les sommets i et j

Principe de l'algorithme : construire une matrice D (initialisée à L)

D contiendra après chaque itération k , les plus courts chemins entre chaque paire de sommets (i,j) , ne passant que par les sommets appartenant à l'ensemble $\{1,2,3,\dots,k\}$.

Programmation dynamique

Application 4 : Les plus courts chemins (Algorithme de Floyd)

Après l'itération n , D contiendra les plus courts chemins entre chaque paire de sommets.

A l'étape k on calcule les nouvelles valeurs de D pour chaque paire de sommets de la façon suivante :

$$D_k[i,j] = \min (D_{k-1}[i,j] , D_{k-1}[i,k] + D_{k-1}[k,j])$$

Programmation dynamique

Application 4 : Les plus courts chemins(Algorithme de Floyd)

PROCEDURE Floyd (L , D)

D := L;

Pour k := 1 , n

 Pour i := 1 , n

 Pour j := 1 , n

$D[i,j] := \min (D[i,j] , D[i,k] + D[k,j])$

 Fpour

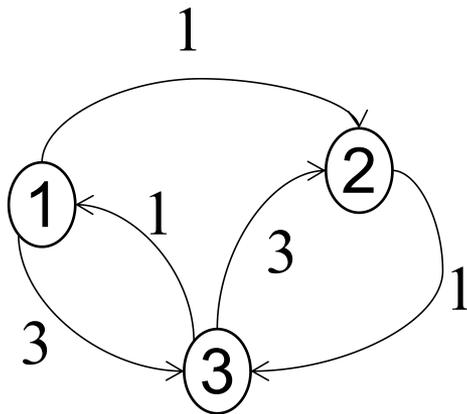
 Fpour

Fpour

Cet algorithme est en $o(n^3)$.

Programmation dynamique

Application 4 : Les plus courts chemins (Algorithme de Floyd) Exemple :



Graphe

	1	2	3
1	0	1	3
2	∞	0	1
3	1	3	0

Matrice associée

Programmation dynamique

	1	2	3
1	0	1	3
2	∞	0	1
3	1	3	0

Application 4 : Les plus courts chemins (Algorithme de Floyd)

Étape 1 : Vérifie si le sommet 1 peut améliorer les chemins existants --> D_1

Étape 2 : Vérifie si le sommet 2 peut améliorer les chemins existants --> D_2

Étape 3 : Vérifie si le sommet 3 peut améliorer les chemins existants --> D_3

	1	2	3
1	0	<u>1</u>	3
2	∞	0	1
3	<u>1</u>	<u>2</u>	0

D_1

$$(3,2) = (3,1) + (1,2)$$

	1	2	3
1	0	<u>1</u>	<u>2</u>
2	∞	0	<u>1</u>
3	1	2	0

D_2

$$(1,3) = (1,2) + (2,3)$$

	1	2	3
1	0	1	2
2	<u>2</u>	0	<u>1</u>
3	<u>1</u>	2	0

D_3

$$(2,1) = (2,3) + (3,1)$$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

On veut calculer le produit de n matrices données :

$$M = M_1 * M_2 * \dots * M_n$$

Comme le produit matriciel est *associatif*, il y a plusieurs façons de calculer ce produit :

$$M = (\dots(M_1 * M_2) * M_3) \dots M_n$$

$$M = (M_1 * M_2) * (M_3 * M_4) * \dots * M_n$$

.....

$$M = (M_1 * (M_2 * \dots (M_{n-1} * M_n) \dots))$$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

$$A(p,q) * B(q,r) = C(p,r)$$

Le produit de deux matrices $A(p,q)$ et $B(q,r)$ nécessite $p*q*r$ multiplications de scalaires

--> Le nombre d'opérations élémentaires nécessaires pour calculer M , dépend de la façon dont on insère les parenthèses.

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

Exemple : Soit à calculer $M = A * B * C * D$
avec $A(13,5)$ $B(5,89)$ $C(89,3)$ $D(3,34)$

Il existe 5 façons de calculer M

$((A B) C) D$	nécessite 10 582 mult.
$(A B) (C D)$	nécessite 54 201 mult.
$(A (B C)) D$	nécessite 2 856 mult.
$A ((B C) D)$	nécessite 4 055 mult.
$A (B (C D))$	nécessite 26 418 mult.

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

Problème : trouver la meilleure façon d'insérer les parenthèses pour calculer M
(minimiser le nombre de multiplications élémentaires)

Méthode directe : générer tous les cas possibles et choisir le meilleur d'entre eux.

Soit $T(n)$ le nombre de manières d'insérer des parenthèses dans un produit de n matrices.

$$M = (M_1 M_2 \dots M_i) (M_{i+1} M_{i+2} \dots M_n)$$


$$\underbrace{\hspace{10em}}_{T(i)} \quad \underbrace{\hspace{10em}}_{T(n-i)}$$

$$T(n) = \sum_{i=1, n-1} T(i) * T(n-i)$$

$$T(n) = \sum_{i=1, n-1} T(i) * T(n-i)$$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

n	1	2	3	4	10	15
T(n)	1	1	2	5	4862	2 674 440

Nombres de Catalan.

Croissance en $4^n/n^2$

Méthode directe : très inefficace.

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

		$m[n, n]$	j	
i	X	X	X	$m(1,n)$
		X	X	X
			X	X
				X

On remarque que ce problème vérifie le principe d'optimalité :

Si la meilleure façon d'insérer des parenthèses consiste à couper entre la matrice M_i et M_{i+1} , c'est à dire $M = \text{Prod}(M_1, M_2 \dots M_i) * \text{Prod}(M_{i+1}, \dots M_n)$; alors les deux sous produits doivent être découpés de façon optimale.

Programmation dynamique : construction d'une table $m(i,j)$ ($1 \leq i \leq j \leq n$)

où chaque élément $m[i,j]$ donnera le plus petit nombre de multiplications nécessaires pour calculer le produit $M_i * M_{i+1} * \dots * M_j$

Résultat : $m(1,n)$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

Les dimensions des matrices M_j sont données dans un vecteur $d(i)$ ($0 \leq i \leq n$) tel que M_k est de dimension $(d[k-1], d[k])$.

La construction de la matrice se fera diagonale par diagonale. La diagonale s contient les $m[i,j]$ tels que $j-i=s$.

d_0 d_1 d_2 d_3 ... d_n
 M_1 M_2 M_3 M_n

		$m[n, n]$		j	
		X	X	X	$m(1,n)$
i			X	X	X
				X	X
					X
					X

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

Pour $s = 0$	$m[i, i] = 0$
Pour $s = 1$	$m[i, i+1] = d[i-1] * d[i] * d[i+1]$
Pour $1 < s < n$	$m[i, i+s] = \text{Min} (m[i, k] + m[k+1, i+s] + d[i-1] * d[k] * d[i+s])$
$i = 1, 2, \dots, n-s$	$i \leq k \leq i+s-1$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

Exemple

$d = (13, 5, 89, 3, 34)$

Pour $s = 1$

$$m_{12} = 5785$$

$$m_{23} = 1335$$

$$m_{34} = 9078$$

Pour $s = 2$

$$\begin{aligned} m_{13} &= \text{Min}(m_{11} + m_{23} + 13 * 5 * 3, m_{12} + m_{33} + 13 * 89 * 3) \\ &= \text{Min}(1350, 9256) = 1530 \end{aligned}$$

$$\begin{aligned} m_{24} &= \text{Min}(m_{22} + m_{34} + 5 * 89 * 34, m_{23} + m_{44} + 5 * 3 * 34) \\ &= \text{Min}(24208, 1845) = 1845 \end{aligned}$$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

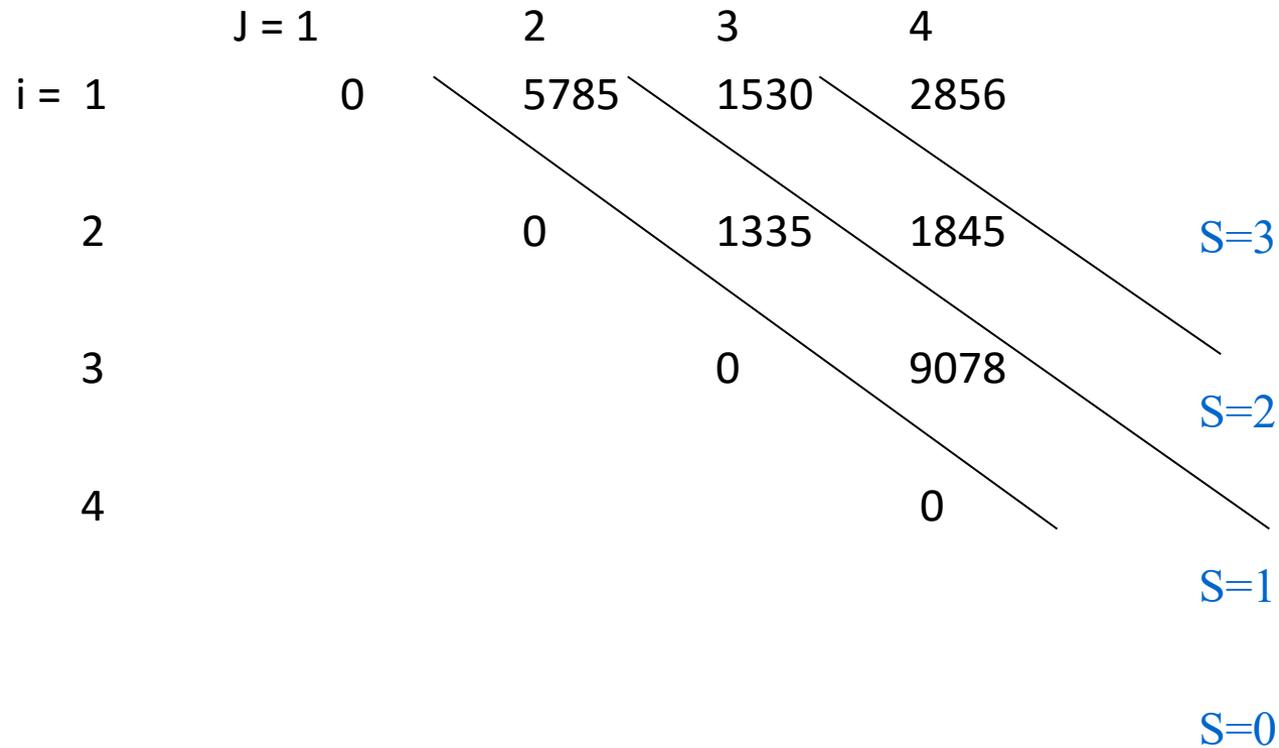
Pour $s = 3$

$$m_{14} = \text{Min} (m_{11} + m_{24} + 13 * 5 * 34, \\ m_{12} + m_{34} + 13 * 89 * 34, \\ m_{13} + m_{44} + 13 * 3 * 34)$$

$$= \text{Min} (4055, 54201, 2856) = 2856$$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices



Programmation dynamique

Pour $s = 0$ $m[i,i] = 0$
Pour $s = 1$ $m[i,i+1] = d[i-1] * d[i] * d[i+1]$
Pour $1 < s < n$:
 $m[i,i+s] = \text{Min} (m[i,k] + m[k+1,i+s] + d[i-1] * d[k] * d[i+s])$
 $i = 1, 2, \dots, n-s$ $i \leq k \leq i+s-1$

Application 5 : Multiplication chaînée de matrices (Algorithme Iteratif)

```
POUR I := 1 , n M [ I , I ] := 0 FPOUR
POUR I := 1 , n - 1 M [ I , I + 1 ] := D [ I - 1 ] * D [ I ] * D [ I + 1 ])) FPOUR
POUR s := 2, n-1
  POUR I := 1 , n - S
    Min := ∞
    POUR K := I , I + S - 1
      v := M [ I , K ] + M [ K + 1 , I + S ] + D [ I - 1 ] * D [ K ] * D [ I + S ]
      SI Mini < v Min := v FSI
    FPOUR ;
    M [ I , I + S ] := Min
  FPOUR
FPOUR
```

Programmation dynamique

Application 5 : Multiplication chaînée de matrices

Pour $s > 0$, il y a $(n-s)$ éléments à calculer dans la diagonale s

Pour chaque élément, il faut choisir entre s possibilités (les différentes valeurs de k possibles)

Temps de calcul

$$\begin{aligned} \sum_{s=1, n-1} (n-s)s &= n \sum_{s=1, n-1} s - \sum_{s=1, n-1} s^2 \\ &= n^2 (n-1) / 2 - n(n-1)(2n-1)/6 \\ &= (n^3 - n) / 6 \end{aligned}$$

$O(n^3)$

Programmation dynamique

Application 5 : Multiplication chaînée de matrices (Algorithme Iteratif)

Placement des parenthèses

```
POUR I := 1 , n M [ I , I ] := 0 ; Pos_K [ I , I ] := 0 FPOUR
POUR I := 1 , n - 1 M [ I , I + 1 ] := D [ I - 1 ] * D [ I ] * D [ I + 1 ] ; Pos_K [ I , I + 1 ] := 2 FPOUR
POUR s := 2, n-1
  POUR I := 1 , n - S
    Min := ∞
    POUR K := I , I + S - 1
      v := M [ I , K ] + M [ K + 1 , I + S ] + D [ I - 1 ] * D [ K ] * D [ I + S ]
      SI v < Min Min := v ; Sauv := K FSI
    FPOUR ;
    M [ I , I + S ] := Min ; Pos_K [ I , I + S ] := Sauv
  FPOUR
FPOUR
Afficher_par(1, n)
```

Programmation dynamique

Application 5 : Multiplication chaînée de matrices (Algorithme Iteratif)

Placement des parenthèses

Afficher_par (I , J)

SI (I = J)

 ECRIRE ('M' , I)

SINON

 ECRIRE ('(') ;

 APPEL Afficher_par (I , POS_K [I , J]) ;

 APPEL Afficher_par (POS_K [I , J] + 1 , J) ;

 ECRIRE (')') ;

FSI

FIN

13, 5, 89, 3, 34
M1*M2*M3*M4

((M 1 (M 2 M 3))M 4)