

Programmation dynamique



Exemple: C_k^n

- Exemple: On veut calculer $C_k^n = \frac{n!}{k!(n-k)!}$ (aussi noté

$$\binom{n}{k})$$

- Prop:

- $C_k^n = C_{k-1}^{n-1} + C_k^{n-1}$ pour $0 < k < n$,

- 1 sinon.

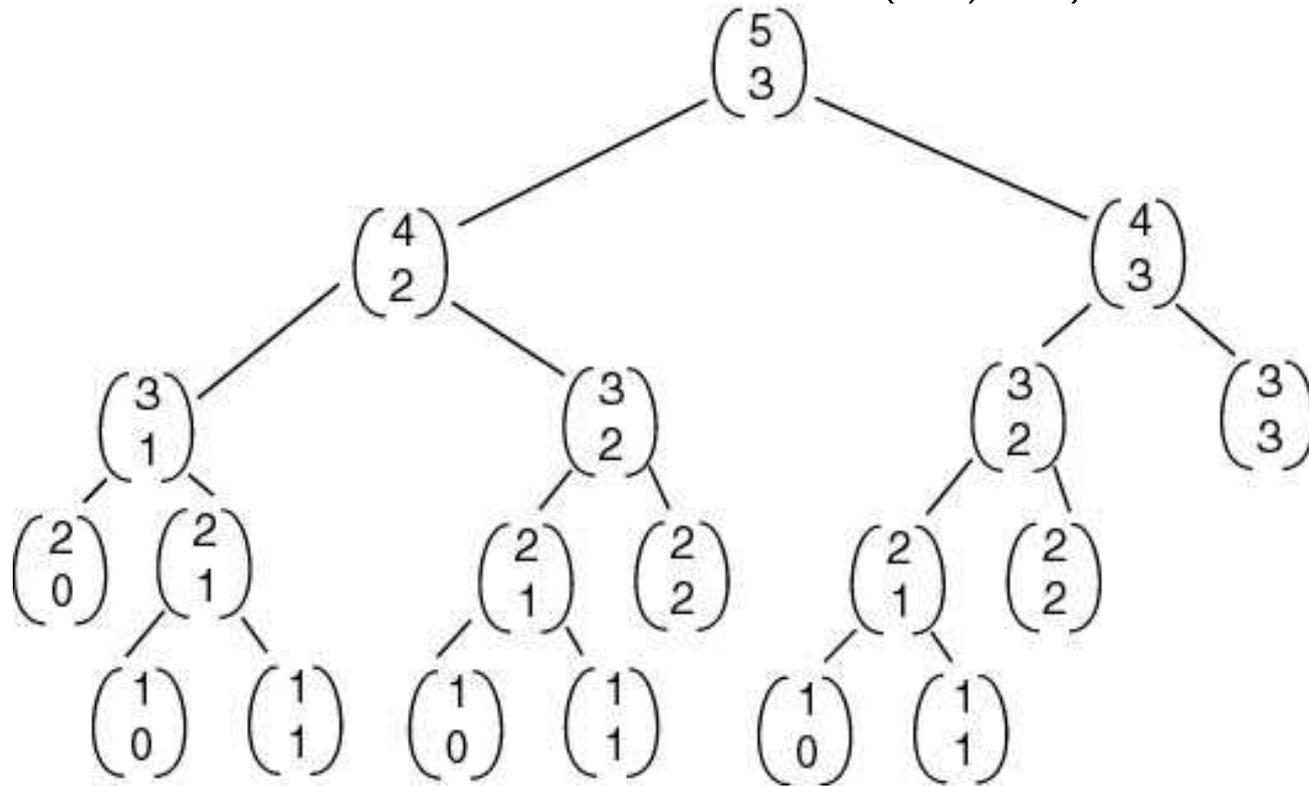
- Approche “Diviser pour régner”.

Fonction $C(n, k)$

- Si $k = 0$ ou $k = n$ alors retourner 1.

- Retourner $C(n - 1, k - 1) + C(n - 1, k)$.

Arbre de calcul de $C(5, 3)$



Fait: Le nombre total d'appels récurrents est $2\binom{n}{k} - 2$

Meilleure idée: triangle de Pascal

	0	1	2	3	...	$n-1$	n
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
\vdots	\vdots	\vdots	\vdots		\ddots		
$n-1$	1	$n-1$	$\binom{n-1}{2}$	$\binom{n-1}{3}$...	1	
n	1	n	$\binom{n}{2}$	$\binom{n}{3}$...	n	1

Triangle de Pascal.

Principe



1. On remplit les k premières cases de chaque ligne de haut en bas.
2. On arrête à la ligne n
3. Temps: $T(n, k) = O(nk)$.

Remarque: il n'est pas nécessaire de mémoriser tout le tableau.



Principes de la prog. dynamique



- Souvent utilisée lorsqu'une solution récursive se révèle inefficace.
 1. Solution récursive: Haut en bas.
 2. Programmation dynamique: Bas en haut.
- On évite de calculer plusieurs fois la même chose.



Multiplications chaînées de matrices.

- On veut calculer le produit de matrices

$$M = M_1 M_2 \cdots M_n$$

- Fait: multiplier une matrice $p \times q$, par une matrice $q \times r$ en utilisant la méthode standard nécessite pqr produits scalaires.
- Exemple: A:13x5, B:5x89, C: 89x3, D: 3x34.
- On veut calculer ABCD.



- Le nombre de produits scalaires est donné dans le tableau suivant:
 1. $((AB)C)D$: 10582
 2. $(AB)(CD)$: 54201
 3. $(A(BC))D$: 2856
 4. $A((BC)D)$: 4055
 5. $A(B(CD))$: 26418
- Question: comment trouver la meilleur parenthétisation?



Première solution

1. Essayer toutes les possibilités.

- Soit $T(N)$ le nombre de façons de parenthéser $M_1 \dots M_n$
- Écrivons:
 - $M = (M_1 \dots M_i)(M_{i+1} \dots M_n)$.
 - $T(i)$ façons 1ère parenthèse, $T(n-i)$ façons autre.
- On obtient: $T(n) = \sum_{i=1}^{n-1} T(i)T(n-i)$ (Nombres de Catalan).
- $T(2) = 1, T(3) = 2, T(4) = 5, T(5) = 14,$
 $T(10) = 4862, T(15) = 2674440.$
- Fait: $T(n)$ dans $\Omega(4^n/n^2)$.

Alternative: Prog. Dynamique



- M_i : matrice $d_{i-1} \times d_i$.
- Définissons $m_{i,j}$: nombre minimal de produits scalaires nécessaires pour évaluer $M_i \dots M_j$.
- Supposons que nous sachions que la meilleure façon de parenthésier $M_i \dots M_j$ est $(M_1 \dots M_k)(M_{k+1} \dots M_j)$.
 1. $M_i \dots M_k$ est une matrice $d_{i-1} \times d_k$ et nécessite $m_{i,k}$ produits scalaires.
 2. $M_{k+1} \dots M_j$ est une matrice $d_k \times d_j$ et nécessite $m_{k+1,j}$ produits scalaires
 3. Le nombre total de produits scalaires est $m_{i,k} + m_{k+1,j} + d_{i-1}d_k d_j$.





• On obtient

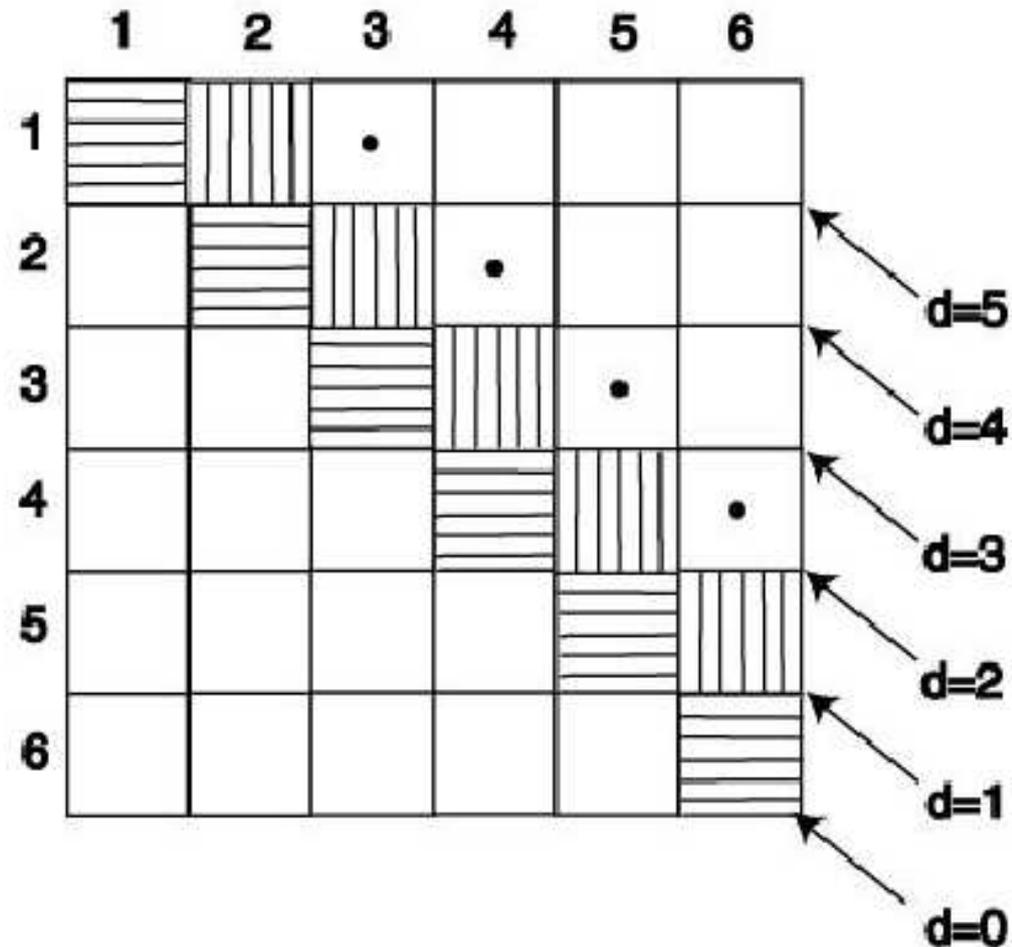
$$m_{i,j} = \begin{cases} \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1}d_kd_j) & \text{si } i < j. \\ 0 & \text{si } i = j. \end{cases}$$

Preuve:

1. le raisonnement précédent montre que le membre gauche est plus grand que le membre droit.
2. réciproquement, les réalisations des minimums des membres droits donnent une parenthétisation, et donc le membre gauche est plus petit que le membre droit.



Arbre de calcul de $m_{2,5}$



Pour calculer $m_{2,5}$ nous avons besoin de $m_{2,2}$ et $m_{3,5}$,
 $m_{2,3}$ et $m_{4,5}$, $m_{2,4}$ et $m_{5,5}$

Faits

1. Pour calculer une diagonale, on a seulement besoin de connaître les diagonales précédentes.
2. La diagonale d contient les éléments $m_{i,j}$ tels que $d = j - i$.

Algorithme

1. Initialisation: tous les éléments de la diagonale sont mis à 0
2. $i := 1$
3. Calculer toutes les entrées de la diagonale i
4. Si $i < n$ aller en 3

Exemple: Tableau $m[i, j]$

	1	2	3	4
1	0	5785	1530	2856
2		0	1335	1845
3			0	9078
4				0

Exemple

Exemple: $n = 4$

$$d_0 = 13, d_1 = 5, d_2 = 89, d_3 = 3, d_4 = 34$$

$$\underline{d = 1}$$

$$m_{1,2} = 13 \times 5 \times 89 = 5785$$

$$m_{2,3} = 5 \times 89 \times 3 = 1335$$

$$m_{3,4} = 89 \times 3 \times 34 = 9078$$

Exemple

$$\underline{d = 2}$$

$$\begin{aligned} m_{1,3} &= \min\{m_{1,1} + m_{2,3} + 13 \times 5 \times 3, \\ &\quad m_{1,2} + m_{3,3} + 13 \times 89 \times 3\} \\ &= \min\{1530, 9256\} \\ &= 1530 \end{aligned}$$

$$\begin{aligned} m_{2,4} &= \min\{m_{2,2} + m_{3,4} + 5 \times 89 \times 34, \\ &\quad m_{2,3} + m_{4,4} + 5 \times 3 \times 34\} \\ &= \min\{24208, 1845\} \\ &= 1845 \end{aligned}$$

Exemple

$$\underline{s = 3}$$

$$\begin{aligned} m_{1,4} &= \min\{m_{1,1} + m_{2,4} + 13 \times 5 \times 34, \\ &\quad m_{1,2} + m_{3,4} + 13 \times 89 \times 34, \\ &\quad m_{1,2} + m_{4,4} + 13 \times 3 \times 34\} \\ &= \min\{4055, 54201, 2856\} \\ &= 2856 \end{aligned}$$

Analyse



- Pour $d > 0$, il y a $n - d$ éléments sur la diagonale d .
- Pour chaque élément, on doit considérer d possibilités.
- Le temps d'exécution est

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

- Soit $T(n)$ dans $\theta(n^3)$.



Algorithme ProduitMatrice



- Pour $i := 1$ à n , $m[i, i] := 0$.
- Pour $l := 2$ à n //longueur de la chaîne
 - Pour $i := 1$ à $n - l + 1$
 - $j := i + l - 1$
 - $m[i, j] = +\infty$
 - Pour $k := i$ à $j - 1$
 - $q := m[i, k] + m[k + 1, j] + d_{i-1}d_kd_j$
 - **Si** $q < m[i, j]$ **alors**
 - $m[i, j] := q$
 - $s[i, j] := k$



Exemple: Tableau $s[i, j]$

	1	2	3	4
1	0	1	1	3
2		0	2	3
3			0	3
4				0

Meilleur résultat: $(A(BC))D$ avec 2856 multiplications.

Algorithme AfficheResultat(s,i,j)

AfficheResultat(s,i,j)

- Si $i = j$ alors afficher M_i
- Sinon
 - Afficher “(“
 - *AfficheResultat*(s, i, s[i, j])
 - *AfficheResultat*(s, s[i, j] + 1, j)
 - Afficher “)”

Plus court chemin dans un graphe

- Entrée : Un graphe orienté $G = (V, E)$ où chaque arc possède une longueur non-négative. Un des noeuds du graphe est appelé source.
- Problème: Trouver la longueur du plus court chemin entre toutes les paires de noeuds.
- On suppose que $V = \{1, \dots, n\}$.
- On suppose aussi que G est donné sous forme de matrice $L[1..n, 1..n]$.
- Algorithme de Floyd: cet algorithme construit une matrice D qui donne la longueur du plus court chemin entre chaque pair de noeuds.

Principe

1. On initialise D à L
2. Après l'itération k , D donne la longueur du plus court chemin lorsque l'on utilise que des noeuds dans $\{1, \dots, k\}$ comme noeuds intermédiaires (ou éventuellement aucun noeud intermédiaire).

Définition: D_k est la matrice D après l'itération k .

Récurrance

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

Exercice: expliquer pourquoi.

Floyd(L[1..n,1..n])

Procédure Floyd(L[1..n,1..n])

- D:=L
- Pour $k := 1$ à n
 - Pour $i := 1$ à n
 - Pour $j := 1$ à n
$$D[i, j] = \min(D[i, j], D[i, k] + D[k, j])$$
- Retourner D.

Temps dans $O(n^3)$.

Principe d'optimalité



- Dans une séquence optimale de décisions, ou de choix, chaque sous séquence doit aussi être optimale.
- Exemple: Si $(1,4), (4,2)$ est le chemin le plus court entre 1 et 2, alors $(1,4)$ est le chemin le plus court entre 1 et 4 et $(4,2)$ entre 4 et 2.
- Exemple: cela ne marche pas pour les chemins les plus longs.



Sous-structure optimale

- Un problème possède une sous structure optimale si toute solution optimale contient la solution optimale aux sous-problèmes.
- Exemple: chemin le plus court, multiplication chaînée de matrices, etc.
- Pour obtenir une solution efficace, l'espace des sous-structures d'un problème ne doit pas être trop grand.
- Exemple: dans la multiplication chaînée de matrices, l'espace des sous-structures consiste en l'évaluation de toutes les séquences de matrices $M_i..M_j, 1 \leq i \leq j \leq n$.

Fonctions à Mémoire



- Le fait que l'espace des sous-structures est petit implique qu'une solution récursive inefficace calcule plusieurs fois les mêmes problèmes plutôt que de toujours générer de nouveaux sous-problèmes.
- Exemple: Calculer $fib(n)$ avec $fib(0) = fib(1) = 1$,
 $fib(n + 2) = fib(n) + fib(n + 1)$.



Solution récursive

Fonction $fib1(n)$

- si $n < 2$ alors retourner n
- retourner $fib1(n-1) + fib1(n-2)$.

Temps dans $O(\phi^n)$ avec $\phi = (1 + \sqrt{5})/2$.

Solution prog. dynamique

Fonction $fib2(n)$

- $i := 1$
- $j := 0$
- Pour $k := 1$ à n
 - $j := i + j$
 - $i := j - i$
- Retourner j .

Temps dans $O(n)$.

Compromis



On veut obtenir

1. la clarté de l'approche récursive
2. l'efficacité de la programmation dynamique





- Idée: Ajouter une table à *fib1*, initialisée à “vide”, afin de ne pas faire d’appels inutiles.
- Fonction *fib(n)*
 - Si $n < 2$ alors
 - $T[n] := n$
 - Retourner n
 - Si $T[n - 1] = \text{“vide”}$ alors $T[n - 1] := fib(n - 1)$
 - $T[n] := T[n - 1] + T[n - 2]$
 - Retourner $T[n]$

