Programmation dynamique

N. TSOPZE

Département d'Informatique - Université de Yaoundé I

Programmation dynamique

- appliquée aux problèmes d'optimisation;
- nombreuses solutions possibles, ayant une valeur chacune;
- trouver une solution ayant la valeur optimale (minimale ou maximale)
- possibilité d'avoir plusieurs solutions ayant la valeur optimale

étapes

1. Caractériser la structure d'une solution optimale;
2. Définir récursivement la valeur d'une solution optimale;
3. Calculer la valeur d'une solution optimale de manière ascendante (bottom-up).
4. Construire une solution optimale à partir des informations calculées.

1. Disposer d'un problème de taille n;
2. Décomposer en sous problème;
3. résoudre chaque sous problème indépendamment (DPR)

Inconvénient: Possibilité de resoudre un sous problème identique plusieurs fois.

Idée générale de PD

Décomposer un problème global en sous problèmes, puis résoudre chaque sous exemplaire différent une seule fois

Programmation dynamique

Idée

Eviter de faire le même calcul deux fois

1. Utiliser un tableau pour stocker les résultats des sous problèmes
2. Remplir le tableau au fur et à mesure

Principe d'optimalité

Dans une séquence optimale de décisions ou de choix, chaque sous séquence doit aussi être optimale.

Principe d'optimalité

La solution optimale à un problème est composée de solutions optimales à des sous-problèmes

Etude de cas

1. Calcul de C_n^p
2. Suite de Fibonacci
3. plus courts chemins (Algo. de Floyd)
4. Multiplication chaînée de matrices
5. Knapsack

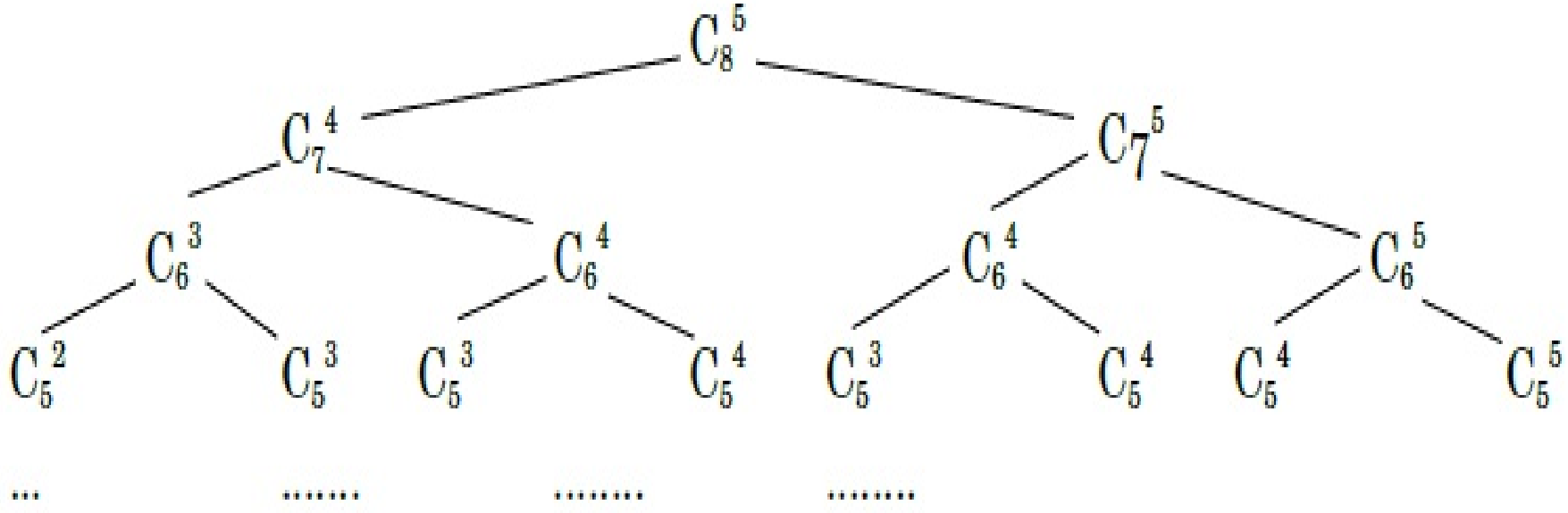
Etude de cas - Calcul de C_n^p

$$\begin{cases} C_n^n = C_n^0 = C_0^0 = 1 \\ C_n^p = C_{n-1}^{p-1} + C_{n-1}^p \end{cases}$$

Fonction Comb (n,p)

1. Si $(n = p)$ ou $(p = 0)$ alors retourner 1
2. Sinon retourner $Comb(n-1, p-1) + Comb(n-1, p))$
3. Fsi

$\Rightarrow$ Grand nombre de sous problèmes identiques


C_8^5
C_7^4
C_7^5
C_6^3
C_6^4
C_6^4
C_6^5
C_5^2
C_5^3
C_5^3
C_5^4
C_5^3
C_5^4
C_5^4
C_5^5
...
.....
.....
.....

Version PD - Triangle de Pascal

1. calcul ligne par ligne, de gauche à droite,
2. calculer pour les sous exemplaires les plus simples et sauvegarder les résultats,
3. se servir de ces résultats pour la calcul des exemplaires plus grands

Suite de Fibonacci

$$\begin{cases} f_1 = f_0 = 1 \\ f_n = f_{n-1} + f_{n-2}, n > 1 \end{cases}$$

Approche dynamique:

1. commencer par les sous problèmes les plus petits (f_0 et f_1)
2. calculer les sous problèmes plus complexes (f_2, f_3, f_4, ..., f_n) en gardant la trace des résultats intermédiaires nécessaires au calcul

$\Rightarrow$ Version itérative de l'algorithme = version PD

plus court chemin

Dans un graphe de n sommets, calculer les longueurs des plus courts chemins entre chaque pair de sommets en utilisant comme sommets intermédiaires, dans l?ordre et de façon successives les sommets 1, 2, 3 ... n

principe

Si le plus court chemin (chemin optimal) entre deux sommets A et B passe par un sommet intermédiaire C, alors les portions du chemin entre A et C et entre C et B doivent forcément être optimale.

Algorithme de Floyd

$$\begin{cases} D_0[i,j] = arc(i,j) \\ D_0[i,j] = +\infty \end{cases}$$

Par récurrence

$$\begin{cases} D_1[i,j] = Min(D_0[i,j], D_0[i,1] + D_0[1,j]) \\ D_k[i,j] = Min(D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]) \end{cases}$$

Multiplication matricielle

1. Associativité $M_1(M_2M_3) = (M_1M_2)M_3$
2. multiplier $M_1(p, q)$ par $M_2(q, r)$ requiert $p * q * r$ multiplications élémentaires

Problème

trouver le nombre minimal $m(1, n)$ de multiplications élémentaires nécessaire pour multiplier une série de n matrices : $M_1M_2M_3...M_n$.

Multiplication matricielle

1. $m(i,j)$ $(m(i,i)=1)$ nombre minimal de multiplications élémentaires nécessaires pour faire le produit de matrices suivant : $M_i * M_{i+1} * M_{i+2} ... * M_j$
2. $D[0..n]$ dimensions d'une matrice M_i soient données par $D[i-1]$ (nb de lignes) et $D[i]$ (nb de colonnes)
3. $M_i M_{i+1}$ requiert $m(i,i+1) = D[i-1] * D[i] * D[i+1]$ multiplications élémentaires

Optimalité

Si pour faire le produit de n matrices avec un nombre minimal d'opérations élémentaires on découpe le produit des n matrices en deux sous produits au niveau de la i^e matrice :

1. $(M_1 * M_2 * ...M_i) * (M_{i+1} * M_{i+2} * ...M_n)$ alors chacun des deux sous produits doit aussi être optimal
2. P_1 : $(M_1 * M_2 * ...M_i)$ calculé avec $m(1, i)$ opération élémentaires; soit une matrice X de dimensions : $D[0]xD[i]$.
3. P_2 : $(M_{i+1} * M_{i+2} * ...M_n)$ calculé avec $m(i + 1, n)$ opérations élémentaires; soit une matrice Y de dimensions : $D[i]xD[n]$;
4. Le produit final : $X * Y$ nécessitera alors $D[0] * D[i] * D[n]$ multiplications élémentaires.

$\Rightarrow$ découpe le produit initial au niveau de la i^e matrice, on pourra écrire : $m(1, n) = m(1, i) + m(i + 1, n) + D[0] * D[i] * D[n]$

Problème du sac à dos

Cas entier

1. $A[i,p]$ = profit maximal obtenu en prenant un sous-ensemble d'objets de $\{1,...,i\}$ avec un poids total est inférieur ou égal à p.
2. $A[i,p] = 0$ s'il n'existe aucun sous-ensemble de $\{1,...,i\}$ de poids au plus égal à p;
3. gain maximal $A[i+1,p]$

$$\begin{cases} A[i+1,p] = max(A[i,p], A[i,p-P_{i+1}] + V_{i+1}) \text{ si } P_{i+1} \leq p \\ A[i+1,p] = A[i,p] \text{ sinon.} \end{cases}$$