(Jo6)

NOM : AUFORT Prénom : William Jury :

~~Algèbre~~ ← Entourez l'épreuve → ~~Analyse~~ Informatique

Sujet choisi : 906 Programmation dynamique : Exemples et applications

Autre sujet :

I) Principe général de la programmation dynamique

Méthode générale

Remarque 1 : On raisonne sur des sous-problèmes, comme diviser-pour-régner, mais ces sous-problèmes ne sont plus indépendants.

Remarque 2 : Souvent utilisé pour résoudre des problèmes d'optimisation.

Méthode 3 : Méthode générale en prog. dynamique

1) On identifie les sous-problèmes intéressants
2) On définit récursivement la valeur d'une solution optimale pour chaque sous-problème
3) On définit les cas initiaux
4) On calcule la valeur correspond à nos problèmes à partir des sous-problèmes.

Remarque 4 : En général, on essaye de reconstruire une solution optimale à partir de la valeur optimale.

II) Exemple détaillé : produit parenthésé de matrices

Problème 5 : Etant données $A_1, A_2, \dots, A_n$ n matrices, calculer le produit $A_1 A_2 \dots A_n$ en un nombre minimum d'opérations.

a) Sous problèmes et réunions :

Exemple 6 :

Définition 7 : des sous-problèmes considérés sont la meilleure manière de calculer :

$A_{i,j} = A_i A_{i+1} \dots A_j$

on note $m[i,j]$ ce nombre minimum d'opérations.

Proposition 8 : $m[i,j] = \begin{cases} 0 \text{ si } i=j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \end{cases}$

où A_i est de dimension $p_{i-1} \times p_i$.

Remarque 9 : On adapte la formule précédente pour le retrouver le parenthésage optimal.

Remarque 10 : Les preuves d'algo. de prog. dyn. consistent généralement à prouver la relation de récurrence (ici prop. 8).

Il y a 2 manières d'écrire l'algorithme,

b) Version ascendante

Remarque 11 : Idée générale : on suit l'ordre des calculs.

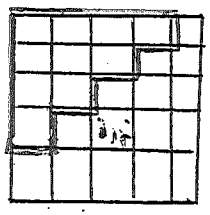
Algorithme 12 Ordre_produit_matrices

Entrée : $p = (p_0, p_1, \dots, p_n)$ dimensions des matrices
Sortie : la table m, et une table s telle que $s[i,j]$ contient l'argmin dans la formule.

```
Pour i=1 à n faire                    // Initialisation
    m[i,i] ← 0
Pour l=2 à n faire                    // Longueur de la chaîne.
    Pour i=1 à n-l+1 faire
        j ← i+l-1
        m[i,j] ← +∞
        Pour k=i à j-1 faire
            tmp ← m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
            si (tmp < m[i,j])
                m[i,j] ← tmp
                s[i,j] ← k
retourner (m,s)
```

Remarque 13 Pour visualiser l'algorithme, on peut tracer le graphe des sous problèmes (par exemple sous forme de tableau)

Exple 14



Dans le calcul de $m[i,j]$, on fait appel à des $m[k,l]$ / $k+l < i+j$
→ partie diagonale supérieure.

Remarque 15 L'algorithme s'exécute en $O(n^3)$.

Remarque 16 : La force des algorithmes et leur analyse sont assez génériques (on regarde les boucles et opérations dans les boucles)

c) Mémoïsation

Remarque 17 : Idée générale : éviter l'approche récursive, mais en ne résolvant un sous problème que s'il n'a pas déjà été résolu.
On stocke les sous-problèmes résolus dans un tableau, ou une table de hachage.

Algorithme 18 : Ordre_produit_matrice_memoïsé :

Entrée : p
Sortie : la table m.

```
Pour i=1 à n faire
    Pour j=1 à n faire
        m[i,j] = +∞
retourner AUX(m,p,1,n)

AUX(m,p,i,j) =
  si (m[i,j] < +∞)
      retourner m[i,j]
  si (i=j)
      retourner 0
  Sinon, pour k=i à j-1 faire
      tmp ← AUX(m,p,i,k) + AUX(m,p,k+1,j)
                + p_{i-1} p_k p_j
      si (tmp < m[i,j])
          m[i,j] ← tmp
  retourner m[i,j]
```

Remarque 19 La complexité en temps et en espace reste la même ici.

III] Applications de la programmation dynamique en algorithmique de mots

a) Plus longue sous-séquence commune (PLSC)

Exemple 20 Si $u = abcbdab$ et $v = bdcaba$, alors $bcba$ et $bdab$ sont des PLSC de u et v.

Exercice 21 : Concevoir un algo de prog. dyn qui calcule la longueur de la PLSC en $O(n^2)$, et qui, après ce précalcul, renvoie une PLSC en $O(n)$ ($n = \max(|u|,|v|)$).

DEV 1

b) Distance d'édition

Problème 22 On veut transformer un mot x en un mot y en utilisant 3 opérations ayant comme coût 1 : substitution, insertion et suppression. La distance d'édition est le plus petit nombre d'opérations à effectuer.

Proposition 23 :
$$d(\varepsilon, y) = |y|$$
$$d(x, \varepsilon) = |x|$$
$$d(ua, va) = d(u,v)$$
$$d(ua, vb) = \min\big(d(u,v) + d(ua,v) + d(u,vb)\big) + 1$$

Remarque 24 On obtient un algorithme semblable à celui pour la PLSC.

c) Appartenance à un langage algébrique

Problème 25 Etant donné G une grammaire algébrique mise sous forme normale de Chomsky (ie chaque règle est de type : $X \to aa$ ou $X \to YZ$) et un mot w, a-t-on $w \in \mathcal{L}(G)$?

Définition 26 On s'intéresse aux sous-problèmes $T[i,k,X]$, où $T[i,k,X]$ est vrai si et seulement si $w_i w_{i+1} \dots w_k$ peut être dérivé à partir de la variable X.

Proposition 27 :
1) Si $i=k$, $T[i,i,X]$ = vrai $\Leftrightarrow$ il existe une règle $X \to w_i$
2) Si $i<k$, $T[i,k,X]$ = vrai $\Leftrightarrow$ il existe $X \to YZ$ telle que $\exists j \in [i, \dots, k-1]$ tel que $T[i,j,Y]$ et $T[j+1,k,Z]$.

Théorème 28 : Il existe un algorithme qui teste l'appartenance à un langage algébrique en $O(|w|^3)$ (G est supposé fixée). Il s'agit de l'algorithme CYK (Cocke-Younger-Kasami)

III) Autres applications

a) Algorithme de Floyd-Warshall

Problème 29 Etant donné $G=(V,E,w)$ un graphe orienté pondéré, sans cycle de poids négatif, trouver, pour chaque $(i,j) \in V^2$, le poids des plus court chemin allant de i à j.

Théorème 30 L'algorithme de Floyd-Warshall, après un prétraitement en $O(n^3)$ ($n=|V|$), permet de trouver le poids des plus court chemin en $O(1)$, et le chemin en lui-même en $O(n)$.

DEV 2

b) En combinatoire : échange de pièces et sac à dos

Problème 31 (Echange de pièces) Etant donné un ens. de pièces $\{v_1, \dots, v_m\}$ et une somme S, quel est le nombre minimal de pièces à rendre pour arriver à la somme S ?

Définition 32 $f(T,i)$ est le nombre minimal de pièces parmi $\{v_1, \dots, v_i\}$ à rendre pour atteindre la somme T.

Proposition 33 :
$$f(T,i) = \min\left(f(T,i-1), f(T-v_i, i)\right).$$
$$f(0,-) = 0$$
$$f(-,0) = +\infty$$

Exemple 34 :
avec $\{1,2,5,10\}$ et $S=13$.

i \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	0	1	1	2	2	1	2	2	3	3	1	2	2	3
3	0	1	1	2	2	1	2	2	3	3	2	3	3	4
2	0	1	1	2	2	3	3	4	4	5	5	6	6	7
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

En gardant trace de l'algorithme (cf flèches) on obtient $10+2+1$.

Remarque 35 Complexité en espace : $O(S \times i)$, alors que la taille du problème est de $O(\sum_i \log(v_i) + \log S)$

Problème 36 (Sac à dos) Etant donné $\{(w_i, c_i)\}_{i \in [1,n]}$, trouver $I \subset \{1, \dots, n\}$ maximisant $\sum_{i \in I} c_i$, mais tout en respectant $\sum_{i \in I} w_i < W$.

Définition 37 $C(w,i)$ est le coût maximal en prenant $I \subset \{1, \dots, i\}$ et $\sum_{i \in I} w_i < w$

Proposition 38
$$C(0,-) = 0$$
$$C(-,0) = 0$$
$$C(w<0,-) = -\infty$$
$$C(w,i) = \max\left(C(v,i-1), C(v-w_i, i-1)+c_i\right).$$

Remarque 39 (Analogue à la remarque 35) on a un algo en $O(nW)$, alors que ce problème est NP-Complet.

c) Calcul des polynômes d'interpolation de Lagrange

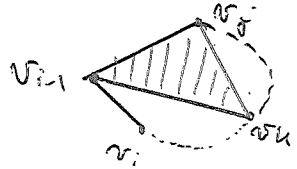
Exercice 40 Concevoir un algo de prog dyn qui, après un prétraitement en $O(n^2)$, permet à partir de $f, x_0, \dots, x_n, x$, de calculer $P_n(x)$ où P_n est le poly. d'interpolation de Lagrange en $x_0, \dots, x_n$ de f, en temps $O(n)$.

DEV 3

d) Triangulation optimale de polygônes

Problème 41 On veut trianguler un polygône $\mathcal{C}$, convexe en minimisant une fonction de coût w sur les triangles ($w : (v_i, v_j, v_k) \to w_{ijk}$. $P = (v_0, v_1, \dots, v_n)$.

Proposition 42 : En notant $c(i,j)$ le coût optimal pour le polygône $(v_{i-1}, v_i, \dots, v_j)$, on a :
$$c(i,i+1) = w_{i-1,i,i+1}$$
$$c(i,j) = \min_{k \in \{i, \dots, j-1\}} \left(w_{i-1,k,j} + c(i,k) + c(k+1,j)\right)$$



Références :
- Carmen, Introduction à l'algorithmique
- Berrut, Rebet, Vivien : A guide to algorithm design (pour quelques preuves)
- Floyd, Le langage des machines... (CYK)
- Demailly, Analyse numérique et équations diff. (pour les polynômes d'interpolation de Lagrange)