Programmation dynamique : exemples et applications

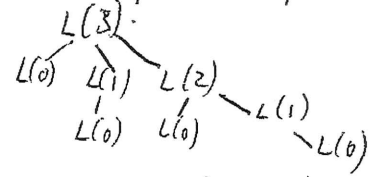
906

La programmation dynamique est un paradigme de programmation très utilisé par exemple pour des problèmes d'optimisation.

I. Présentation du paradigme

1/ Un exemple :

Soit t une suite d'entiers (donnée sous forme de tableau), on cherche dans t la plus longue sous-séquence croissante d'entiers.
Notons $L(i)$ la longueur de la plus longue sous-séquence croissante terminant à i, on a la relation $L(i) = 1 + \max\{L(j) \mid j<i \text{ et } t[i] \geq t[j]\}$
On en déduit un algorithme récursif naïf, mais voici son arbre d'appels récursifs pour l'exemple $t = [0;1;2;3]$:



On voit que cet algorithme peut être exponentiel, on va chercher à l'améliorer en se souvenant des résultats intermédiaires.

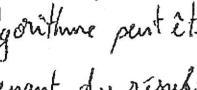
2/ Mémoïsation ou programmation dynamique ?

Une première approche consiste à garder en mémoire les $L(i)$ sous forme de tableau : à chaque appel récursif, on cherche si $L(i)$ a déjà été calculé, et on ne le calcule que si ce n'est pas le cas.
Cette approche est appelée mémoïsation, elle présente l'avantage de ne calculer que les sous-problèmes nécessaires.

Une autre approche consiste à calculer itérativement tous les sous-problèmes en commençant par les petits et à garder les résultats en mémoire.
Cette approche est appelée programmation dynamique, elle présente l'avantage de ne pas être récursive mais itérative.

3/ Quand utiliser la programmation dynamique ?

Pour que la programmation dynamique puisse être utilisée et soit efficace, il faut vérifier les deux critères suivants :
* sous-structure optimale : le problème doit pouvoir être décomposé en sous-problèmes de sorte que la solution du problème puisse être reconstituée à partir des solutions des sous-problèmes
* chevauchement des sous-problèmes : le nombre de sous-problèmes distincts doit être relativement petit par rapport au nombre d'appels récursif que ferait l'algorithme naïf.

Contre-exemple : Considérons le problème de trouver le plus long chemin simple dans un graphe, ce problème n'a pas la sous-structure optimale :
ex : 

le plus long chemin de a à c est $a \to b \to c$, alors que la concaténation des plus long chemins simples de a à b et de b à c est $a \to c \to b \to c$ (qui n'est pas simple).
On ne peut donc pas utiliser la programmation dynamique pour résoudre efficacement ce problème.

II Applications à l'algorithmique du texte

1/ Distances entre des mots (Kleinberg-Tardos)

* distance d'édition : on cherche le nombre minimal d'insertions, de suppressions et de changements de lettres pour transformer un mot en un autre. ex : SUNN-Y / S-NOWY → distance de 3

Plus précisément, si $x, y \in \Sigma^*$, notons $E(i,j)$ la distance entre $x[1..i]$ et $y[1..j]$, on a la relation :

$$E(i,j) = \min \{ 1 + E(i-1,j),\ 1 + E(i,j-1),\ \delta_{x[i] \neq y[j]} + E(i-1,j-1) \}$$

où $\delta_{x[i] \neq y[j]} = 1$ si $x[i] \neq y[j]$ et $\delta_{x[i] \neq y[j]} = 0$ si $x[i] = y[j]$.

On a donc réussi à exprimer la solution du problème en fonction des solutions de sous-problèmes plus petits, on a ainsi utilisé la sous-structure optimale.

De plus, pour tout (i,j), $E(i,j)$ devra être recalculé lors de tout appel de $E(k,l)$ avec $(k,l) > (i,j)$ si on fait un algorithme naïf, donc on a bien chevauchement des sous-problèmes.

Les conditions sont réunies pour utiliser la programmation dynamique, et en effet, un algorithme de programmation dynamique résout le problème en $O(|x||y|)$ opérations.

* plus longue sous-séquence commune (DEV)

2/ Dictionnaires

* On dispose de certains mots dont on connait la probabilité d'apparition, et on cherche à les stocker dans un arbre binaire de recherche de sorte que le temps moyen de recherche soit minimal.

Dans un arbre optimal, les fils gauche et droit de la racine sont aussi des arbre optimaux pour les probabilités conditionnelles correspondantes, ainsi le problème a une sous-structure optimale.
On peut voir que les sous-problèmes se chevauchent, ainsi la programmation dynamique est une approche adaptée pour résoudre ce problème.

* application : on dispose d'un mot x dont on sait qu'il provient d'un fichier corrompu, ie. tous les espaces et les ponctuations ont disparus. On cherche à retrouver le fichier original à l'aide d'un dictionnaire qui nous permet de savoir si un mot existe en temps constant.
Ce problème peut se résoudre par la programmation dynamique en considérant les sous-problèmes P_i : est-ce que le i^e préfixe de x est une concaténation de mots ? = en temps $O(|x|^2)$.

exemple :

x	C	E	C	I	N	E	S	T	P	A	S	U	N	E	X	E	M	P	L	E
P	F	V	F	V	F	V	V	V	F	F	V	F	V	V	V	F	F	F	F	V

on a $P(6) =$ vrai car "ce ain" est une suite de mots valides.

III Autres exemples

1/ Applications sur des graphes

Soit G un graphe sans cycles de poids négatifs, on cherche à déterminer pour tout couple de sommets (i,j) le plus court chemin de i à j.
→ algorithme de Floyd-Warshall (CDEV)

rq: On peut adapter l'algorithme pour calculer la fermeture réflexive transitive d'un graphe, le nombre de chemins élémentaires dans un graphe orienté acyclique, etc...

2/ Application aux langages algébriques

Soit G une grammaire que l'on supposera sous forme de Chomsky, et w un mot, on cherche à savoir si $w \in L_G(S)$.
Notons w_{ij} le sous-mot qui commence à l'indice i et qui est de taille j, et V_{ij} = l'ensemble des variables qui se dérivent en w_{ij}.
L'algorithme de Cocke-Younger-Kasami calcule successivement tous les V_{ij} :

```
CYK(w)
| pour i = 1 à |w|
|   [ V_{i1} = {A ∈ V | A → w[i] soit une règle de production} ]
| pour j = 2 à |w|
|   pour i = 1 à n-j+1        n = |w| ?
|     V_ij = ∅ ;
|     pour k = 1 à j-1
|       [ V_ij = V_ij ∪ {A | A → BC et B ∈ V_ik et C ∈ V_{i-k,i-k} } ]
| retourner S ∈ V_{1|w|}
```

Cet algorithme termine en $O(|w|^3)$ étapes.

3/ Problèmes NP-complets et programmation dynamique

* Le problème du sac à dos :
On dispose de n objets de poids $w_1, \dots, w_n$ et de valeurs $v_1, \dots, v_n$, ainsi que d'un sac à dos qui peut contenir un poids W. Quels sont les objets à prendre pour maximiser la valeur emportée ?
Si on note $K(W)$ la valeur maximale, on voit que

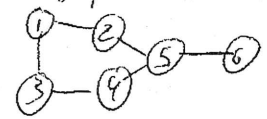
$$K(W) = \max_{1 \le i \le n} \{ K(W - w_i) + v_i \}$$

On en déduit un algorithme de programmation dynamique :

```
K(0) = 0
pour w = 1 à W
  [ K(w) = max { K(w - w_i) + v_i | w_i ≤ w } ]
retourne K(W)
```

Cet algorithme a une complexité de $O(nW)$, ce type d'algorithme est dit pseudo-polynomial, car il est polynomial en W (mais il est exponentiel en la taille de l'entrée : $\log W + n$)

* ensembles indépendants
Le problème de trouver le plus grand sous-ensemble de sommets indépendants dans un graphe est NP-complet.

ex: 

$\{1,4,6\}$ est un ensemble indépendant maximal.

Cependant, on dispose d'un algorithme de programmation dynamique pour déterminer un ensemble indépendant de taille maximale dans un arbre, en effet, si on note $I(u)$ la taille du plus grand ensemble indépendant dans le sous-arbre sous le sommet u, on a la relation

$$I(u) = \max \left\{ 1 + \sum_{v \text{ petit-fils de } u} I(v) \ , \ \sum_{v \text{ fils de } u} I(v) \right\}.$$

On obtient ainsi un algorithme en $O(|S| + |A|)$

Références:
- Cormen
- Papadimitriou
- Hopcroft-Ullman (pour CYK)
- Gondran-Minoux, Graphs, dioids and semirings, new models and Algorithms (pour la version la plus générale de Floyd-Warshall, avec un dioïde)