Programmation dynamique. Exemples et Applications.

I. Principe

Remarque 1 [COR3 Notes 15] **Point Historique** R. Bellman commença l'étude systématique de la programmation dynamique en le formalisant en 1955. Le mot « programmation » dans ce contexte comme dans celui de la programmation linéaire fait référence à l'utilisation d'une méthode de résolution tabulaire.

A. Optimalité de Bellman

Définition 2 [BAR 6.3] **Le principe d'optimalité de Bellman** affirme que la solution optimale d'un problème est obtenue en combinant les solutions optimales de ses sous problèmes.

Algorithme 3 [TOR 8.3.3.1] **L'algorithme de Floyd Warshall** utilise cette formule pour appliquer la programmation dynamique.

Remarque 4 [COR3 15.3.c] **Contre-exemple.** Dans le graphe définit précédemment le chemin le plus long entre a et c passe par b. Cependant le chemin le plus long entre b et c passe par a et d. Le principe de Bellman ne s'applique pas.

B. Chevauchement sous problèmes

Idée 5 [NSIT 14 p.246] Une approche récursive avec appel sur les sous problèmes nous mène tout d'abord à une méthode « **diviser pour régner** ». Lorsqu'il y a chevauchement, on va vouloir se rappeler des appels.

Définition 6 [COR3 15.3.b] **Chevauchement.** Quand un algorithme récursif repasse sur le même problème constamment, on dit que le problème d'optimisation contient des **sous problèmes qui se chevauchent**.

Définition 7 [COR3 15.1.c] **Le graphe des sous-problèmes** est un graphe orienté représentant les dépendances entre les appels récursifs des sous problèmes.

Definition 8 **La suite de Fibonacci** est définie comme suit : $f_0 = f_1 = 1, \forall n \geq 2, f_n = f_{n-1} + f(n-2)$. Cette formule naturellement récursivement nous montre un bon exemple de chevauchement.

Exemple 9 **Graphe des sous problèmes de Fibonnaci** pour $n = 5$.

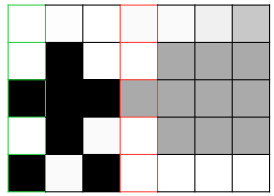
C. Programmation Dynamique

Définition 10 [COR3 15 Intro] Le développement d'un algorithme de programmation dynamique se divise en quatre 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, généralement de manière ascendante
4. construire une solution optimale à partir des informations calculées.

Remarque 11 [COR3 15.3.b] Tandis que la programmation dynamique fait un choix éclairé à chaque étape, les **algorithmes gloutons** préfèrent prendre une décision hâtive mais localement intéressante pour augmenter les performances.

Application 12 [TOR 9.4.2.1] Soit une image de taille $w \times h$ dont on a calculé le gradient d'incident lumineux. L'algorithme de **Seam Carving** pour compresser une image consiste à enlever les colonnes de sommes de luminosité minimale.



Remarque 13 Une approche gloutonne n'est pas correcte (ligne verte) mais en $\mathcal{O}(h)$. Une approche diviser pour régner pas efficace en $\mathcal{O}(2^h * w)$. On utilise de la programmation dynamique (ligne rouge) sur le tableau définit par les pixels de l'image pour déterminer le chemin le plus cours en complexité $\mathcal{O}()$. < TODO

Algorithme 14 ABR Optimaux [COR3 15.5] Si on a un ensemble de mots $m_1, ..., m_n$ avec des probabilités associés $p_1, ..., p_n$ on peut construire un arbre binaire de recherche optimal grâce à la programmation dynamique. Ici l'optimalité signifie qu'on minimise l'espérance du temps d'une requête sur l'ABR. DEV

II. Mise en Application

A. Approche Descendante

Définition 15 [NSIT 14 p.245] [TOR 9.4 p.527] La Mémoïsation est une technique de programmation venant de la programmation fonctionnelle consistant à sauvegarder les valeurs des appels d'une fonction pure pour ne calculer qu'une seule fois son résultat pour une même entrée.

Définition 16 Approche descendante [COR3 15.1.b] Une première approche pour le calcul d'un résultat de programmation dynamique est descendante avec mémoïsation. On écrit la procédure de manière récursive de façon naïve et on la modifie afin de sauvegarder le résultat de chaque sous-problème.

Exemple 17 [BAR 6.3.2.5] Distance d'édition On souhaite comparer deux mots u et v. On définit une distance, dite d'édition (ou de Levenshtein), égale au nombre d'opérations de suppression, ajout et remplacement minimum qui permettent de passer de u à v.

P_{ij} est le calcul de la distance optimale $d(i,j)$ entre $u_1...u_i$ et $v_1...v_j$. On a alors pour $i \in [\![1, |u|]\!], j \in [\![1, |v|]\!]$,

$$d(i,j) = \min\big(d(i-1,j)+1, d(i,j-1)+1, d(i-1,j-1)+\delta_{ij}\big)$$

avec $\delta_{ij} = \begin{cases} 0 \text{ si } u[i]=v[j] \\ 1 \text{ sinon} \end{cases}$ et $d(i,0) = i, d(0,j) = j$.

B. Approche Ascendante

Définition 18 [COR3 15.1.b] Approche ascendante Cette approche se base sur une notion de taille des sous-problèmes, dans le sens où résoudre un problème dépend uniquement de sous-problèmes plus petits. On peut donc calculer les sous-problèmes par taille croissante afin que lorsqu'on résout un sous-problème, tout ses sous-problèmes sont déjà résolus.

Définition 19 [BAR 6.3] Approche Itérative (ascendante) L'approche ascendante est souvent implémenté de manière itérative en stockant les résultats des sous-problèmes dans un tableau.

Exemple 20 Somme d'un sous-ensemble [BAR 6.3.2.4] Soit $\{\pi_1, ..., \pi_n\}$ un ensemble d'entiers. On note $\mathcal{P}$ le problème de recherche du sous-ensemble Q tel que : $\sum_{\pi_i \in Q} \pi_i \leq M \in \mathbb{N}$ soit maximal.

Application 21 On s'intéresse au sous-problème $\mathcal{P}_{i,p}$ de sélection d'un sous-ensemble de $\{\pi_1, ..., \pi_i\}$ de somme inférieure à $p \in [\![0, M]\!]$. Si on note $v(i,p)$ la valeur de la somme de la solution du sous-problème $\mathcal{P}_{i,p}$, on a la relation de récurrence suivante :

$$v(i,p) = \begin{cases} \max(v(i-1,p), \pi_i + v(i-1, p-\pi_i)) \text{ si } \pi_i \leq p \\ v(i-1,p) \text{ sinon} \end{cases}$$

On résout ainsi $\mathcal{P}$ en résolvant les $\mathcal{P}_{i,p}$ par ordre lexicographique.

Exemple 22 Avec $\{0,1,2,3,4,5\}$ et M = 7 on obtient de programmation dynamique suivant :

i\p	0	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1	1
3	0	1	2	3	3	3	3	3
4	0	1	2	3	4	5	6	6
5	0	1	2	3	4	5	6	7
6	0	1	2	3	4	5	6	7

Exemple 23 Rendu de Monnaie [NSIT] Une variante du problème de somme d'un sous-ensemble est le problème de rendu de monnaie. On cherche à rendre une somme de monnaie S avec les pièces d'un système monétaire $M_n = \{v_1, ..., v_n\}$ en minimisant le nombre de pièces rendues.

Remarque 24 [COR3 15.1.b] Compromis Temps Mémoire Le principe de programmation dynamique sauvegarde et réutilise les

résultats de sous-problèmes déjà calculé au lieux de les recalculer à chaque fois. On réduit donc le temps de calcul au dépend du coût mémoire.

Remarque 25 Optimisation en Espace [TOR 9.4 p.529] Certains problèmes de programmation dynamique ne nécessitent pas de connaître les solutions de tout les sous-problèmes. On peut donc économiser de l'espace en ne conservant que les solutions nécessaires au calcul des sous-problèmes de la taille supérieure.

Exemple 26 Fibonacci. Par exemple pour calculer les nombres de Fibonacci, il suffit de garder en mémoire les deux derniers nombres calculé et pas l'entièreté du tableau.

Remarque 27 [COR3 Notes] **Galil et Park** classent les algorithmes de programmation dynamique d'après la taille du tableau et le nombre d'autres éléments de tableau dont dépend chaque élément. Le problème des sommes des sous ensembles est donc en 1d/0d et Floyd Warshall en 3d/1d.

C. Reconstruction de la Solution

Méthode 28 [TOR 9.4 p.529] **Construction Solution Optimale** Jusqu'à présent, on a calculé la valeur optimale associée au problème (longueur de la plus longue sous-séquence commune, somme du sous-ensemble maximal, ...). Pour obtenir la solution (sous-séquence, sous-ensemble, ...) correspondante à cette valeur optimale, il suffit de chercher un chemin optimal dans le graphe des sous-problèmes.

Algorithme 29 [TIGER] **CYK.** Soit une grammaire donnée en forme normale de Chomsky. L'algorithme de programmation dynamique de Cook Younger Kasami peut être utilisé pour vérifier si un mot appartient au langage reconnu en $\mathcal{O}(|G|^3)$. DEV

Exemple 30 [BAR 6.3.2.3] **Sac à Dos** Le problème du sac à dos est une généralisation du problème de somme maximale d'un sous-ensemble. À chaque élément i on associe un poids w_i et un profit p_i. On cherche à maximiser le profit total que l'on peut faire rentrer dans un sac à dos de poids maximal W.

Application 31 On peut alors utiliser le même sous-problème que précédemment :

$$v(i,w) = \begin{pmatrix} \max(v(i-1,w), p_i + v(i-1, w-w_i)) \text{ si } w_i \leq w \\ v(i-1,w) \text{ sinon} \end{pmatrix}$$

On obtient la valeur optimale en calculant $v(n,W)$.

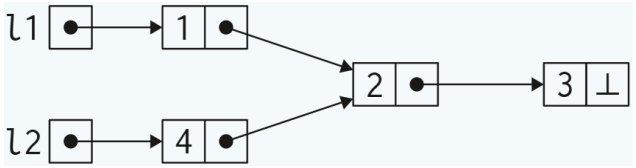
III. Application non algorithmique : Partage Maximal (Hash-Concing)

Définition 32 [OCA 11.4] **Le Hash Concing** consiste à applique le principe de mémoïsation à l'allocation mémoire. Si on a déjà alloué une donnée identique, alors on la réutilise plutôt que de l'allouer de nouveau. Si les données sont immuables, alors un tel partage est possible sans risque.

Exemple 33 [TOR 9.4 p.533] **Listes Chaînées** Sur des listes chaînées en Ocaml, on peut écrire une fonction `cons` qui se comporte comme l'opérateur `::` mais qui implémente du hash-consing :

```
let l1 = cons 1 (cons 2 (cons 3 []))
let l2 = cons 4 (cons 2 (cons 3 []))
```

donnera la situation suivante, où deux cellules sont partagées entre les deux listes :



Programmation dynamique. Exemples et Applications.

I. Principe

1 Rem [COR3 Notes 15] Point Historique

A. Optimalité de Bellman

2 Def [BAR 6.3] Le principe d'optimalité de Bellman

3 Algo [TOR 8.3.3.1] L'algorithme de Floyd Warshall

4 Rem [COR3 15.3.c] Contre-exemple.

B. Chevauchement sous problèmes

5 Idée [NSIT 14 p.246]

6 Def [COR3 15.3.b] Chevauchement.

7 Def [COR3 15.1.c] Le graphe des sous-problèmes

8 Def La suite de Fibonacci

9 Ex Graphe des sous problèmes de Fibonnaci

C. Programmation Dynamique

10 Def [COR3 15 Intro]

11 Rem [COR3 15.3.b]

12 App [TOR 9.4.2.1]

13 Rem

14 Algo ABR Optimaux [COR3 15.5]

II. Mise en Application

A. Approche Descendante

15 Def [NSIT 14 p.245] [TOR 9.4 p.527] La Mémoïsation

16 Def Approche descendante [COR3 15.1.b]

17 Ex [BAR 6.3.2.5] Distance d'édition

B. Approche Ascendante

18 Def [COR3 15.1.b] Approche ascendante

19 Def [BAR 6.3] Approche Itérative (ascendante)

20 Ex Somme d'un sous-ensemble [BAR 6.3.2.4]

21 App

22 Ex

23 Ex Rendu de Monnaie [NSIT]

24 Rem [COR3 15.1.b] Compromis Temps Mémoire

25 Rem Optimisation en Espace [TOR 9.4 p.529]

26 Ex Fibonacci.

27 Rem [COR3 Notes] Galil et Park

C. Reconstruction de la Solution

28 Métho [TOR 9.4 p.529] Construction Solution Optimale

29 Algo [TIGER] CYK.

30 Ex [BAR 6.3.2.3] Sac à Dos

31 App

III. Application non algorithmique : Partage Maximal (Hash-Concing)

32 Def [OCA 11.4] Le Hash Concing

33 Ex [TOR 9.4 p.533] Listes Chaînées

Extension possible

- [COR3 15.4] Plus Longue Sous Séquence Commune
- [NSIT 14.2] Alignement de Séquence
- Anecdote orale sur l'origine du Hash-Consing (Lisp / Hashage)
- Ordonnancement de tâches [BAR 6.3.2.2]
- Découpe de Barre [COR3]
- [BAR] est une source intéressante bien que non nécessaire.
- Heuristique sur la reconstruction de la solution
- [BAR 6.3.2.1] [COR3 15.2] Multiplication matrices

Développements possible

- tout algorithme de programmation dynamique

Bibliographie

[COR3] T. H. Cormen, *Introduction à l'algorithmique (3rd édition).*

[BAR] V. Barra, *Informatique MPI/MP2I.*

[TOR] T. Balabonski & S. Conchon & J. Filliâtre & K. Nguyen & L. Sartre, *MP2I MPI, Informatique Cours et exercices corrigés.*

[NSIT] T. Balabonski & S. Conchon & J. Filliâtre & K. Nguyen, *Numériques et Sciences Informatiques Terminale.*

[TIGER] A. Appel, *Modern compiler implementation.*

[OCA] S. Conchon & J. Filliâtre, *Apprendre à programmer avec Ocaml.*