Programmation Effective – TD 04 : Programmation dynamique

matthieu.gallet@ens-lyon.fr

mardi 26 février 2008

1 Introduction

La programmation dynamique est une technique de programmation qui ressemble beaucoup à la méthode *diviser pour régner*. Les algorithmes *diviser pour régner* vont partitionner le gros problème d'origine en petits sous-problèmes indépendants et résoudre ceux-ci récursivement. Les sous-solutions une fois recombinées vont donner la solution du problème initial. Contrairement à la méthode *diviser pour régner*, la programmation dynamique fonctionne également quand les sous-problèmes ne sont pas indépendants. S'il y a des sous-problèmes communs, un algorithme *diviser pour régner* fera le même travail plusieurs fois, alors que le but de la programmation dynamique est d'éviter ce travail en mémorisant les réponses dans un tableau.

Le développement d'un algorithme de programmation dynamique peut être découpé en 4 étapes élémentaires :

1. caractérisation de la structure d'une solution optimale
2. définition récursive de la valeur d'une solution optimale
3. calcul d'une valeur d'une solution optimale de façon ascendante (*bottom-up*)
4. éventuellement, reconstruction de la solution optimale trouvée, en parcourant à l'envers le chemin de l'étape précédente. Cette étape peut demander de stocker des informations supplémentaires.

2 Application

Exposé du problème Considérons une usine de voitures avec 2 chaînes de montage distinctes. Un châssis de voiture arrive sur chaque chaîne et passe par un certain nombre de postes où on lui ajoute des pièces. Le châssis une fois terminé quitte l'usine à l'autre bout de la chaîne. Chaque chaîne de montage a n postes $j = 1, \ldots, n$, notés $S_{i,j}$ où i vaut 1 ou 2 suivant la chaîne. Le j-ème poste fait toujours le même travail, qu'il soit sur la première ou la seconde chaîne.

Tous les postes de montage n'ont pas été construits à la même époque, et deux postes similaires ne vont pas mettre le même temps pour faire la même opération. Ce temps de montage sera noté $a_{i,j}$, et on notera également e_i le temps d'arrivée du châssis qui entre sur la chaîne i, et x_i le temps de sortie pour la voiture complète qui sort de la chaîne i. Si jamais une voiture donnée doit être construite plus rapidement, on peut décider de la faire passer d'une chaîne à l'autre entre deux postes. La voiture passera toujours les n étapes de montage dans cet ordre. Naturellement, passer d'une chaîne à l'autre à un coût non négligeable, noté $t_{i,j}$, avec $i = 1, 2$ et $j = 1, 2, \ldots, n-1$ (il y a n postes, donc $n-1$ transitions d'un poste à l'autre). Si une voiture passe d'un poste au suivant sans changer de chaîne, on supposera que le temps pris est nul.

On chercher à déterminer quel est le chemin le plus rapide que puisse emprunter un châssis de voiture avant d'être achevé, quitte à changer plusieurs fois de chaîne de montage.

3 Éléments de la programmation dynamique

Pour pouvoir utiliser la programmation dynamique, un problème doit posséder deux propriétés importantes.

Sous-structure optimale On dit qu'un problème exhibe une sous-structure optimale si, et seulement si, une solution optimale au problème contient en elle des solutions optimales à ses sous-problèmes. Par exemple, si dans un graphe on connaît un plus court chemin $\mathcal{C}$ pour aller d'un sommet A à un sommet B, alors on a un plus court chemin pour tout couple de sommets situés sur $\mathcal{C}$. Si on arrive à exhiber une telle propriété sur un problème, c'est souvent signe qu'il faut utiliser de la programmation dynamique, ou un algorithme glouton.

La découverte de la sous-structure optimale obéit souvent au schéma général suivant :

1. Vous montrez qu'une solution du problème consiste à faire un choix, par exemple à choisir le poste précédant dans la chaîne de montage. Ayant fait ce choix, vous êtes amenés à résoudre un ou plusieurs sous-problèmes.
2. Vous supposez que, pour un problème donné, on vous donne le choix qui conduit à une solution optimale, sans chercher à connaître la façon dont on détermine ce choix.
3. À partir de ce choix, vous déterminez quels sont les sous-problèmes qui en découlent, et comment caractériser au mieux l'espace des sous-problèmes résultant.
4. Vous montrez que les solutions des sous-problèmes employées par la solution optimale du problème doivent elles-mêmes être optimales, souvent avec un raisonnement par l'absurde.

La sous-structure optimale varie en fonction de deux paramètres :

1. le nombre de sous-problèmes qui sont utilisés dans une solution optimale du problème originel,
2. le nombre de choix que l'on a pour déterminer les sous-problèmes à utiliser dans une solution optimale

Ces deux paramètres vont également conditionner le temps d'exécution de l'algorithme complet.

Chevauchement des sous-problèmes Pour que la programmation dynamique soit réellement applicable, il faut que l'algorithme récursif rencontre souvent les mêmes sous-problèmes, afin de pouvoir stocker la solution dans un tableau et de la réutiliser à chaque fois. Cela permet de réduire le temps de calcul. Un algorithme récursif naïf résolverait au contraire plusieurs fois les mêmes problèmes, perdant ainsi du temps.

Reconstruction d'une solution optimale Il ne faut pas oublier de noter à chaque étape le choix fait pour chaque sous-problème. Dans l'exemple de la chaîne de montage, il s'agit de la chaîne au j-ème poste. Si on ne mémorise pas cette information, on risque d'avoir uniquement la valeur d'une solution optimale (en l'occurrence, le temps de traversée de la chaîne complète), mais sans pouvoir la reconstruire. Cependant, c'est parfois suffisant.

4 Applications

Résolvez les problèmes suivants : `http://acm.uva.es/p/v5/590.html`,`http://acm.uva.es/p/v6/612.html`,`http://acm.uva.es/p/v2/231.html`,`http://acm.uva.es/p/v1/104.html`,`http://acm.uva.es/p/v100/10003.html`.