TD/P : *Dynamic Programming and Friends*

TAPS : Techniques and Algorithms for Problem Solving

Ce TD/TP a pour objectif la mise en œuvre des principes de la programmation dynamique et tout particulièrement :

1. la *memoization* ;
2. la modélisation d'un problème sous une forme adaptée à la programmation dynamique et sa résolution ;

Le TD/TP se termine par un sympathique algorithme glouton.

★ Exercice 1: The wolf eating the memory

Soit $f(1) = f(2) = 1$ et :

$$f(n) = f(n - f(n-1)) + f(n - f(n-2) - 1)$$

▷ **Question 1:** Calculez $f(4)$ en dessinant l'arbre des appels et donnez le nombre d'appels à la fonction f dans cette instance.

▷ **Question 2:** Afin d'éviter les recalculs systématiques et multiples, nous pouvons nous appuyer sur l'un des volets de la programmation dynamique : la *memoization*. Connaissant la valeur de n, quelle(s) structure(s) de données est/sont nécessaires pour réaliser cela ?

▷ **Question 3:** Implémenter la version avec *memoization* et comparez ses performances avec la version initiale.

★ Exercice 2: My Trader is rich

Un trader est bon lorsque il engrange un maximum de bénéfices pour son employeur. Son activité consiste en partie à acheter et vendre divers produits financiers et ce sont ces transactions qui déterminent les pertes et gains sur les sommes engagées. Depuis quelques mois, la direction de la banque HSTN (Highly Sunny Telecom Nancy) souhaite évaluer la performance de ses traders en comparant les opérations faites par ses agents à la séquence idéale (i.e. un ensemble d'achats et de ventes qui aurait généré le bénéfice maximum atteignable). Elle souhaite disposer d'un outil numérique pour cela.

Il vous est demandé d'élaborer l'algorithme qui calcule ce maximum théorique sur une valeur boursière. Votre algorithme dispose, pour une action en bourse particulière, d'une liste de cours de l'action toutes les heures sur un intervalle de quotation donné. Par exemple $[13, 3, 20, 12, 2, 9, 16, 6, 15]$ représente la valeur de l'action TNCY Corp. sur les 12 quotations consécutives données pour le calcul. Le nombre de quotations disponible est n. Le trader :

- ne peut acheter q'une action à la fois ;
- ne peut vendre qu'une action à la fois ;
- ne peut vendre une action qu'il ne possède pas (ne peut vendre ce qu'il n'a pas acheté avant ;
- ne peut acheter et vendre en même temps ;
- démarre avec un bénéfice nul ;
- peut effectuer jusqu'à t transactions (1 transaction = un achat suivi d' une vente à une date ultérieure dans l'intervalle de quotation).

▷ **Question 1:** En supposant que le trader ne puisse faire qu'une seule transaction, écrivez un algorithme qui calcule le bénéfice maximum réalisable sur l'intervalle de quotation donné en suivant une approche *"Brute force"*. Calculez la complexité de cet algorithme ?

▷ **Question 2:** Il est possible de trouver le bénéfice maximum en ne parcourant qu'une seule fois le tableau. Trouvez l'algorithme qui réalise cela. Quelle devient alors sa complexité ?

Passons-maintenant au cas général dans lequel le trader peut réaliser jusqu'à t transactions sur une quotation. Nous allons pour cela utiliser la programmation dynamique.

▷ **Question 3:** Donnez les dimensions du problème.

▷ **Question 4:** Calculez à la main les premiers gains maximaux à l'aide de la structure précédente. Pouvez vous en déduire une formulation générale du problème sous forme de problème d'optimization ?

▷ **Question 5:** A l'aide des principes de la programmation dynamique, écrivez l'algorithme qui résout le problème formulé dans la question précédente. Quel est la complexité de cet algorithme ?

▷ **Question 6:** Compte-tenu de votre implémentation, pouvez-vous améliorer la complexité de l'algorithme ?

★ **Exercice 3:** Gloutons !

▷ **Question 1:** Concevez un algorithme qui, étant donnés une somme à payer et un montant donné par le client, calcule le nombre de pièces minimum nécessaires pour rendre la monnaie. Vous pouvez écrire une fonction spécifique pour l'euro ou mieux, une fonction générique (qui nécessitera peut-être un ou plusieurs paramètres supplémentaires). Dans une première itération de votre problème, le stock de pièces est infini. Dans une seconde, il sera limité pour chaque pièce. *Have fun !*