TD 6 - Programmation dynamique

Retrouvez tous les énoncés et les corrections des TP sur ma page personnelle :

`http://perso.ens-lyon.fr/hadrien.croubois/`

La programmation dynamique est une méthode permettant de résoudre efficacement des problèmes ayant une certaine structure. Plus précisément, si la solution optimale à un problème peut s'exprimer en fonction des solutions optimales à des sous-problèmes, alors la programmation dynamique est une méthode adaptée. Un cas particulier important de la programmation dynamique est celui ou les sous-problèmes sont disjoints : on peut alors "diviser pour régner".

La programmation dynamique s"apparente à la programmation récursive à une différence majeure près : on mémorise les résultats des sous-problèmes pour éviter de les recalculer.

Fibonacci

L'exemple le plus classique est celui de la suite de Fibonacci généralisé. On rappelle que cette suite est définie par la récurrence suivante :

$$\mathcal{F}_{a,b}(n) = \begin{cases} a, & \text{si } n = 0 \\ b, & \text{si } n = 1 \\ \mathcal{F}_{a,b}(n-2) + \mathcal{F}_{a,b}(n-1), & \text{sinon} \end{cases}$$

Question 1 : Quelle est la complexité d'une implémentation récursive naïve ?

La forme la plus simple de programmation dynamique consiste à obtenir une expression récursive de la solution et à faire de la mémorisation. Plus précisément, lors d'un appel récursif, la fonction vérifie si le résultat a déjà été calculé, et le cas échéant renvoie ce résultat au lieu de le recalculer.

Un cas particulier important est celui ou tous les paramètres sont des petits entiers, dans ce cas on peut utiliser un simple tableau comme moyen de mémorisation.

```
let 'a Result = None | Value of 'a;;
let results = make_vect 100 None;;
let f x = match results.(x) with
  | Some(a) -> a
  | None -> let r = compute_result x in
    results.(x) <- Some(r)
    r;;
```

Question 2 : Appliquez le schéma qui précède au calcul de la suite de Fibonacci. Quelle est la complexité de cette implémentation ? D'après un théorème de conservation bien connu, si on a gagné quelque part c'est qu'on a perdu ailleurs mais ou ?

Le sac à dos

Un mineur doit est payé au poids de minerai remonté a la surface. Il peut remonter un maximum de Nkg à la surface (conditionné par la capacité de l'ascenseur).

Il a miné k blocs de minerai de poids $k_0, \dots, k_{n-1}$

Aidez le à maximiser son revenu en lui indiquant quels blocs de minerais remonter à la surface.

Question 3 : Écrire un algorithme de programmation dynamique permettant de résoudre ce problème et l'implémenter sous la forme d'une fonction `mineur N [`$k_1, \dots, k_{n-1}$`]` qui renvoi le poids atteint et les indices des blocs à ramener.

```
mineur :  -> int -> int list -> int * int list
```

Question 4 (difficile) : A quel difficulté doit-on faire face si les blocs de minerai ne sont pas tous de même qualité et ont des valeurs indépendantes de leur poids ?

Le paquebot

Le problème du paquebot est le suivant : vous disposez de place à gauche et a droite d'un bateau ainsi que d'un nombre n de conteneurs de poids $k_0, \dots, k_{n-1}$. Tous les emplacement sont à la même distance d du plan de symétrie du paquebot.

Question 5 : Adapter l'algorithme de la question 3 afin de trouver un moyen d'équilibrer au mieux le paquebot selon l'axe de roulis.

Superman

Superman dispose initialement d'une énergie E. Il se déplace horizontalement d'un mètre par seconde et peut en même temps décider soit d'utiliser 1 d'énergie et ainsi de monter de 1 mètre soit de ne pas utiliser d'énergie et ainsi de descendre de 1 (déplacements en diagonale)

Sur son chemin sont présents des obstacles (bâtiments) de taille t et des nuages d'énergie e.

Superman doit passer par dessus les bâtiments (sinon il s'écrase sur la façade) et peut, en passant dans un nuage, récupérer toute son énergie.

Question 6 : Écrire une fonction `superman e a l` qui renvoi la distance maximal que superman peut parcourir avec une énergie initiale `e` en partant d'une altitude `a` et avec la liste `l` des événements sous la forme `(int * Event) list`

```
let Event = Cloud of int | Building of int
```

Question 7 : Discuter la complexité des fonctions écrites.

Question 8 : Demandez moi d'autres problèmes résolubles par programmation dynamique.