Programmation Dynamique

I Un exemple

- On dispose d'une planche de longueur n (en dm)

par exemple, $n = 100$.

- On dispose d'un tableau p de longueur m (en dm)

Par exemple $m=6$.

- Ce tableau est tel que $p.(i)$ me donne le prix d'une planche de longueur i

Par exemple [| 0; 1; 3; 30; 38; 46; 51 |]

objectif. Découper notre planche de longueur n pour vendre les morceaux de sorte que le gain soit maximal

1. Approche exhaustive

On teste tous les découpages possibles et on prend le meilleur outil adapté: la récursivité (en enveloppe)

Pour tout i de 1 à n:

On fait un appel récursif de longueur $n-i$

On ajoute $p.(i)$

Notons $g(n)$ le gain max. pour une planche de longueur n.

La stratégie s'appuie sur la relation de récurrence

$$\begin{cases} g(0) = 0 \\ g(n) = \max\limits_{i \in [\![1,n]\!]} g(n-i) + p.(i) \end{cases}$$

avantage: ça marche

inconvénient: c'est lent.

2. Approche gloutonne

On définit la rentabilité de $i \in [\![1, n]\!]$ $r(i) = \dfrac{p(i)}{i}$

On cherche $i \leq n$ tq $r(i)$ est maximal.

On choisit un tel i et on recommence sur une planche de longueur $n-i$

avantage: c'est très rapide

inconvénient: ça ne marche pas toujours:

ex $\begin{cases} n = 8 \\ p = [\![0, 1, 3, 30, 38, 46, 51]\!] \end{cases}$

i	1	2	3	4	5	6
p(i)	1	3	30	38	46	51
r(i)	1	1,5	10	9,5	9,2	8,5

L'algorithme glouton demande de prendre

- un morceau de 3
- 3
- 2

→ total : 63

Mais en prenant 3 puis 5 on a un gain de 76 > 63.

3 Approche dynamique

On garde la relation de récurrence mais on la combine avec de la mémoization (bottom-up).

On stocke les $g(k)$ pour k de 1 à n

Puis on reconstitue le découpage permettant d'obtenir $g(n)$

II Généralité

1 Programmation dynamique

Adaptée au problèmes d'optimisation (i.e. on veut {max, min}imiser qqch et determiner comment l'obtenir)

4 étapes

1. Identifier des sous-problèmes

   ex pour l'exemple du I

   - maximiser $g(k)$ avec $k < n-i$

2. Déterminer une relation de récurrence qui lie le problème au sous problème

   ex

$$g(n) = \max_{i \in [\![1, n]\!]} \underbrace{g(n-i) + p(i)}_{\text{sous-problème}}$$

   (avec $p.(i) = 0$ si $i \geqslant$ taille de p)

3. Calculer l'optimum .. par mémoization

4. Reconstituer la solution

2 Programmation gloutonne

On procède de proche en proche en réalisant à chaque fois l'optimum local. Ça ne marche pas toujours (selon les données et/ou les problèmes)

3 Implémentation de l'exemple

[implémentation_exemple.ml]