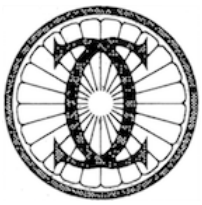

Informatique Tronc Commun

PTSI-PT*

Séquence 4

Programmation dynamique

Table des matières

TD Info - Programmation dynamique

I - Traversée du désert

Objectif

Dans cet exercice, on s'intéresse à deux fonctions coûts différentes (donc deux objectifs différents) que l'on va résoudre par deux approches différentes

1 Présentation du problème

Un voyageur souhaite traverser un désert d'une oasis A à une oasis B. Afin de réussir à traverser le désert, il dispose d'une gourde. Si malencontreusement, cette gourde vient à être vide, le voyageur meurt de soif **instantanément** (sorte de Koh-Lanta de l'extrême). Le chemin le plus court est en ligne droite. Sur celui-ci, des puits sont disposés de afin de permettre au voyageur de remplir sa gourde.

Les puits sont numérotés de 0 à $n-1$ (le puits numéro 0 (respectivement $n-1$) étant le puits de l'oasis de départ (respectivement d'arrivée)). Quand le voyageur arrive à un puits, il choisit entre deux possibilités a) poursuivre sa route, ou b) remplir sa gourde. S'il fait le second choix, il vide sa gourde dans le sable avant de la remplir entièrement au puits afin d'avoir de l'eau fraîche. À l'arrivée, il vide la gourde.

On présente dans ce tableau la distance entre le puits de l'oasis A et les autres.

N° du puits	0	1	2	3	4	5	6	7	(arrivée)
Distance (en km)	0	8	9	16	18	24	27	32	

La capacité V de la gourde est de $10\,\ell$ et le voyageur consomme $1\,\ell$ au km.

2 Résolution du problème : Parcourir la distance avec le moins d'arrêts possibles

Q1. Le voyageur veut faire le moins d'arrêts possible. Quelles sont les solutions permettant de résoudre ce problème d'optimisation.

Q2. Parmi les solutions trouvées, quelle est celle correspondant à celle de la solution gloutonne ?

On va chercher à programmer une programmation gloutonne afin de résoudre le problème de façon générale. On décompose cette résolution en deux fonctions.
Pour notre exemple, on pose `T=[8,9,16,18,24,27,32]`, la liste des distances des puits au point de départ.

Q3. Programmer une fonction `distance_max` retournant l'indice du puits pouvant être atteint de manière gloutonne. Les arguments de cette fonction seront :

- `T`, liste des distances des puits au point de départ.
- `distance_parcourue`, paramètre de la distance déjà parcourue ;
- `distance_gourde`, paramètre de la distance maximale parcourable avec la gourde remplie ;

Vérifier que `distance_max(T,9,10)` renvoie bien 3. En effet, l'indice 3 du tableau T correspond au puits numéro 4.

Q4. Créer une fonction `parcours_puits(T, distance_gourde)` qui renvoie la liste des puits de `T` que le voyageur doit traverser pour faire le moins d'arrêts possibles pour atteindre l'oasis d'arrivée en respectant l'algorithme glouton.

3 Résolution du problème : Parcourir la distance avec le moins de dépenses

Le voyageur souhaite refaire le même trajet que précédemment sauf que la législation a évolué... À chaque puits, y compris celui de l'oasis d'arrivée, un gardien lui fait payer autant d'unités de la monnaie locale que le carré du nombre de litres d'eau restant dans sa gourde à l'arrivée du tronçon qu'il a parcouru. Le problème est de choisir les puits où il doit s'arrêter pour payer le moins possible.

Q5. Quel est le résultat d'une stratégie gloutonne ? Existe-t-il une meilleure solution ? Conclure sur la validité de l'utilisation d'une stratégie gloutonne.

On cherche à trouver la solution à l'aide d'une programmation dynamique dont les éléments sont :

- `popt(i)` : somme minimale payée au total depuis le puits numéro 0 (l'oasis de départ) jusqu'au puits numéro i, étant donné que le voyageur s'arrête et paye la taxe au puits numéro i ;
- `d(i,j)` : nombre de kilomètres entre le puits numéro i et le puits numéro j ;
- `distance_gourde=D` : la distance parcourable avec la gourde.

Q6. Compléter la relation de récurrence suivante :

$$\begin{cases} \texttt{popt(0)} = \ldots\ldots\ldots \\ \texttt{popt(i)} = \min\limits_{\substack{j \in [\![0, i-1]\!] \\ \text{et } \ldots\ldots\ldots \leqslant \ldots\ldots\ldots}} \left(\ldots\ldots\ldots + (\ldots\ldots\ldots - \ldots\ldots\ldots)^2\right) & 0 \leqslant i \leqslant n-1 \end{cases}$$

Q7. Quelle est la valeur de `popt` que l'on doit calculer pour résoudre le problème ?

Q8. Calculer manuellement `popt(1)`, `popt(2)`, `popt(3)` et donner le puits prédécesseur optimal des puits 1, 2, 3.

Q9. Programmer une fonction `calcul_popt(i,liste_popt,D,T)` d'arguments :

- un entier `i` : pour lequel on effectue le calcul de `popt(i)` ;
- une liste `liste_popt` : contenant les résultat de `popt` des i premiers puits : `popt(0)`, `popt(1)`, $\cdots$, `popt(i-1)` ;
- un flottant `D` : la distance parcourable avec la gourde ;
- une liste `T` : liste des distances des puits au point de départ.

et retournant la valeur de `popt(i)` ainsi que le puits prédécesseur optimal au puits i.

Q10. Résoudre alors le problème en créant deux tableaux :

- une liste `POPT` contenant les différentes valeurs de `popt(i)` ;
- une liste `MP` contenant le meilleur prédécesseur des éléments `MP[i]`. Ainsi, `MP[i]` renvoie le meilleur prédécesseur pour atteindre le puits i. Pour le puits 0, on impose `MP[0]=None`, car il n'y a pas de prédécesseur au puits 0.

Q11. Donner alors, la somme minimale qu'il est possible d'avoir versé au total lorsque l'on arrive à l'oasis d'arrivée et donner la succession de puits où il faut s'arrêter pour que cela se produise.