Université Lille1 Sciences et Technologies

UFR IEEA

AAC

2012–2013
Master d'Informatique-S1

TD-Programmation Dynamique

Exercice 1 : Calcul des valeurs d'une suite

Soit la suite $T(n)$ définie par $T(2) = T(1) = T(0) = 1$ et:

$$T(n) = T(n-1) + 2 * T(n-2) + T(n-3), n > 2$$

On considère le problème de calculer $T(n)$ pour l'entrée n. Soit l'algorithme naïf récursif associé:

```
int  T(int n){
  if (n<=2)
    return 1;
  else
    return T(n-1)+2*T(n-2)+T(n-3);
}
```

Q 1. Soit $A(n)$ le nombre d'appels récursifs à T effectués lors de l'exécution de la function $T(n)$. Exprimez $A(n)$ en fonction de $A(n-1), A(n-2), A(n-3)$.

Qu'en déduire sur la complexité de l'algorithme naïf ?

Q 2. Proposez un algorithme en $O(n)$ (en supposant qu'on est dans le cadre du coût uniforme, i.e. en ne prenant pas en compte la taille des opérandes).

Exercice 2 : Placement

On cherche à placer k stations-services le long d'une nouvelle portion d'autoroute. Les stations ne peuvent être placées qu'à certains emplacements identifiés qui sont au nombre de p, $p > k$.

Q 1. Combien de placements possibles y a t-il?

Q 2. On cherche un placement "optimal". Pour cela on associe à chaque placement la somme des carrés des distances entre une station et la suivante (ou entre le début de la portion et elle-même pour la première, elle-même et la fin de la portion pour la dernière).

Par exemple si on a placé sur le tronçon [0..380] 3 stations respectivement au kilomètre 75, 126, 300, la valeur associée est $75^2 + (126-75)^2 + (300-126)^2 + (380-300)^2$.

Le problème est donc:

Donnée:

k, entier, le nombre de stations à placer

$d_0, ..., d_{p-1}$ les emplacements potentiels (donnés par le nombre -entier- de kilomètres depuis le début de la portion

Sortie: un placement optimal de k stations, i.e. qui minime la somme des carrés des distances entres deux stations consécutives (ou entre le début de la portion et elle-même pour la première, elle-même et la fin de la portion pour la dernière)

Proposez un algorithme en $0(k * p^2)$ qui répond au problème.

Exercice 3 : La plus longue sous-suite croissante

On a une suite d'entiers : $x_1, \ldots, x_n$. On veut en extraire une sous-suite (i.e. une suite obtenue en supprimant certains éléments mais sans changer l'ordre des éléments) (strictement) croissante de longueur maximale.

Q 1. Donner une sous-suite strictement croissante de longueur maximale pour 9,2,7,4,6,1,8,6

Q 2. Combien existe-t-il de sous-suites d'une suite de longueur n (on supposera tous les éléments distincts)?

Q 3. Construire l'arbre de toutes les sous-suites croissantes pour la donnée ci-dessus.

Q 4. En déduire un algorithme quadratique pour calculer la longueur maximale d'une sous-suite croissante extraite de $x_1, ..., x_n$.

Q 5. Comment calculer aussi la (une) sous-suite correspondante?

Exercice 4 : Gestion de Projet

Une entreprise étudie sa stratégie pour les jours à venir. Elle a un certain nombre de jours de travail à consacrer à ses projets en cours et cherche à les répartir de façon à optimiser son gain. Pour chaque projet, elle a une estimation du gain en fonction du nombre de jours consacrés, comme par exemple:

Gain	*projetA*	*projetB*	*projetC*
$0jour$	0	0	0
$1jour$	1	1	1
$2jours$	1	2	3
$3jours$	1	4	3
$4jours$	6	4	5

On suppose que pour les trois projets, le gain pour x jours avec $x > 4$ est le même que pour $x = 4$.

Donc si deux jours sont disponibles, il vaut mieux les consacrer au projet C pour un gain de 3; si on dispose de 4 jours, il vaut mieux les consacrer à A. Pour 6 jours, on consacrera 4 jours à A, 2 jours à C. Pour 8 jours, on a plusieurs solutions pour un gain optimal de 11: par exemple 4 jours pour A, 2 à B et 2 à C.

Formellement, le problème est donc le suivant:

Donnée: np –nombre de projets

nj –nombre de jours disponibles

G –un tableau à 2 dimensions: G(n,p) est le gain pour n jours consacrés au projet p, $0 <= n <= nj, 1 <= p <= np$

Sortie: aff: $[1..np] \rightarrow [0..nj]$–à un projet, on associe le nombre de jours consacrés

.telle que $\sum_{i=1}^{np}(aff(i)) = nj$ –on a utilisé en tout nj jours

.et qui maximise $\sum_{i=1}^{np} G(aff(i), i)$ –le gain total

Q 1. Dans l'exemple, que vaut-il mieux faire pour 9 jours? pour 10 jours? pour 11 jours?

Q 2. On peut adopter la stratégie gloutonne suivante: pour chaque jour, l'affecter à un projet de façon à maximiser le gain "immédiat":

```
initialiser aff à 0;
pour chaque jour in 1..nj
  choisir un p tel que G(aff(p)+1,p)-G(aff(p),p) soit maximal;
  aff(p)=aff(p)+1;
fin pour;
```

Montrer que cette stratégie ne donne pas toujours une solution optimale.

On définit $SG(x, y)$, le gain maximum pour x jours consacrés à des tâches de numéro supérieur ou égal à y et inférieur ou égal à np $(0 \leq x \leq nj, 1 \leq y \leq np)$. On aura donc $SG(0, y) = 0$, pour tout y et le gain recherché est $SG(nj, 1)$.

Q 2.1. Que vaut $SG(x, np)$

Q 2.2. Compléter la table SG pour $nj = 5$. **Q 2.3.** Soient x et y, $1 \leq x \leq nj, 1 \leq y < np$: exprimer $SG(x, y)$ en fonction des $SG(x', y+1), 0 <= x' <= x$. **Q 2.4.** En déduire un algorithme de programmation dynamique pour le calcul du gain optimum. Quelle est sa complexité? Comment récupérer la stratégie correspondante?

Exercice 5 : Motifs

Le problème: On cherche à vérifier qu'un mot est bien filtré par un motif très simple. Plus précisément, un mot sera une suite finie de lettres d'un alphabet donné. Un motif sera un mot sur le même alphabet enrichi de trois symboles ("wildcards"), ?, + et $*$.

Le mot u filtre le motif p si il peut être obtenu à partir de p en remplaçant chaque ? par une lettre, chaque $*$ par un mot quelconque (éventuellement vide), chaque + par un mot quelconque non vide.

Par exemple, $abba$ est filtré par les motifs $*$, $a+$, $*a$, $*b$,$abb * a$, $a * b * a$, $a * b * b * a*$, mais n'est pas filtré par les motifs $+a+$, $*b + b* a * b$, $b*$, $*bbb * a$.

Dans tout l'exercice, on pourra supposer que les mots sont des objets JAVA de type STRING et qu'on dispose des fonctionnalités de bases du type. On notera Σ l'alphabet.

Q 1. *abba* est-il filtré par $*a*$? par $+a+$? par $+b+$? par $a + a$?

Q 2. Proposer un algorithme linéaire, i.e. en $O(|u| + |p|)$ [1], pour le cas où le motif ne contient ni $*$, ni $+$:

Entrée: un mot u sur Σ et un motif p sur $\Sigma \cup \{?\}$

Sortie: Oui si u est filtré par p , non sinon.

Q 3. Proposer un algorithme en $O(|u| * |p|)$ dans le cas général en utilisant la programmation dynamique:

Entrée: un mot u sur Σ et un motif p sur $\Sigma \cup \{?, *, +\}$

Sortie: Oui si u est filtré par p, non sinon.

Remarque: Quelle autre approche proposeriez-vous pour le problème?

Q 4. On cherche maintenant à faire du filtrage de motifs (pattern-matching) "approximatif", c'est à dire qu'on peut admettre des erreurs. Par exemple, `examen` est filtré par `e?men`, `x*n` ou encore `*ae*`, si on admet une erreur de 1. Si on admet une erreur de 2, `examen` est filtré par `x*e`, `amen` ou `*an`.

Formellement, soit u un mot sur Σ, p un motif sur $\Sigma \cup \{?, +, *\}$.

$filtrapp(u, p)$ est le plus petit entier k tel qu'il existe un mot v à distance k de u tel que $filtre(v, p)$.

La distance de deux mots u et v est la distance de Levensthein, i.e. le nombre minimal de suppressions, insertions, modifications de caractères que l'on doit faire pour passer de u à v.

Par exemple, $filtrapp(examen, *) = 0$, $filtrapp(examen, e?men) = 1$, $filtrapp(examen, x * e) = 2$, $filtrapp(examen, *abc*) = 2$.

Q 4.1. Que vaut $filtrapp(examen, ???)$? $filtrapp(examen, +e)$? $filtrapp(examen, *z*)$? $filtrapp(examen, *z)$? $filtrapp(examen, z)$?

Q 4.2. Proposer un algorithme efficace pour calculer $filtrapp(u, p)$.

Exercice 6 : L'algorithme de Viterbi

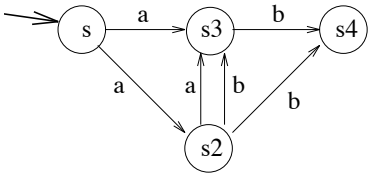
On a un graphe dirigé $G = (S, A)$, S étant l'ensemble des sommets, A l'ensemble des arcs. Les arcs ont la particularité d'être étiquetés par des lettres: un arc est donc un triplet (o, b, l) avec o, l'origine, et b, le but, deux sommets du graphe et l une étiquette (=lettre) appartenant à un alphabet. De plus on a un sommet privilégié appelé source, noté s. (On peut voir un tel graphe comme un automate non nécessairement déterministe, sans état de sortie).

Le problème est de savoir si pour un mot donné u et un sommet c, il y a des chemins partant de la source s, arrivant en c et étiquetés par u.

Par exemple, soit le graphe donné par:

$S = \{s, s_2, s_3, s_4\}$

$A = \{(s, s_2, a); (s, s_3, a); (s_2, s_3, a); (s_2, s_3, b); (s_2, s_4, b); (s_3, s_4, b); \}$



Dans ce graphe, il y a deux chemins étiquetés par ab qui mènent de s à s_4 et un qui mène de s à s_3. Il n'y en a pas étiqueté par aa qui mène de s à s_4 mais il y en a un qui mène de s à s_3.

Q 1. On propose le schéma de fonction suivant

```
--un graphe G, un sommet source s
boolean chemin (Sommet  c, String u)  is
--retourne TRUE SSI il existe dans G un chemin de s a c etiqueté par u
{  si u.length==0  return (s=c)
  --seul chemin de longueur 0 partant de s
  else
  for noeud in predecesseurs de c{
      if  chemin(noeud,u.subString(0,u.length()-2))
        return TRUE; }
    return FALSE;
    --on a essayé en vain tous les chemins possibles
end chemin;
```

Q 1.1. Montrer que dans le pire des cas la complexité de la fonction ci-dessus peut être exponentielle par rapport à la longueur du mot u.

Q 1.2. Que dire du nombre d'appels **différents**, i.e. avec des paramètres différents, de la fonction chemin lors de l'exécution de $chemin(c, u)$?

Q 1.3. Déduire de ce qui précède une version dynamique de l'algorithme ci-dessus dont la complexité soit en $O(l * n^2)$ avec l la longueur du mot u et n le nombre de sommets du graphe.

Peut-on raffiner en fonction du degré maximum entrant (ou sortant) d'un noeud?

Q 1.4. Modifier l'algorithme de la question précédente pour qu'il retourne en plus un chemin de s à c étiqueté par u, si il en existe un.

Q 2. On s'intéresse maintenant à une variante du problème ci-dessus qui apparaît fréquemment dans les problèmes de transmission avec bruit ou de reconnaissance de la parole.

A chaque arc est maintenant associée une probabilité. Un arc est donc maintenant un quadruplet (o, b, l, p) avec o et b deux sommets du graphe, l une étiquette et p une probabilité, i.e. un réel compris entre 0 et 1. La probabilité d'un chemin est le produit des probabilités des arcs qui composent le chemin.

Par exemple associons aux arcs du graphe du début les probabilités suivantes:

$A = \{(s, s_2, a, 2/3); (s, s_3, a, 1/3); (s_2, s_3, a, 1); (s_2, s_3, b, 3/4); (s_2, s_4, b, 1/4); (s_3, s_4, b, 1); \}$.

Etudions les chemins étiquetés par ab qui partent de s. L'un $(s- > s_2- > s_4)$ a la probabilité $2/3 * 1/4$ soit 1/6. L'autre $(s- > s_3- > s_4)$ a la probabilité 1/3. Le troisième chemin étiqueté par ab partant de s est $s- > s_2- > s3$ avec la probabilité $2/3 * 3/4$ soit 1/2: c'est le chemin le plus probable partant de s et étiqueté par ab.

Modifier l'algorithme de la première question pour calculer le (ou un des) chemin(s) de probabilité maximale partant de s et étiqueté par u ainsi que sa probabilité (- si il existe un chemin partant de s et étiqueté par u; sinon, un message sera affiché-).

On pourra supposer que la fonction $ARC(o, b, l)$ retourne maintenant la probabilité associée à l'arc (o, b, l) si il existe, 0 sinon.

Exercice 7 : le jeu du cochon

Le jeu du cochon est un jeu à deux joueurs qui se joue avec un dé classique; le gagnat est le premier qui a un score total d'au moins 100 points. Au départ, chacun a un score total de 0 point; Chaque jour joue à tour de rôle: après chaque lancer de dé, le joueur a deux possibilités: relancer le dé ou passer la main; Si il lance le dé: .si le joueur obtient un 1, il ajoute 1 à son score total - quelque soit le score du tour- et passe la main.

si il fait un nombre de 2 à 6, ce nombre est ajouté au score du tour.

Si le jouer décide de passer la main, il ajoute à son score le max du score du tour et de 1.

Exemple:

Au départ A et B ont tous les deux un score nul. A lance le dé; il fait un 6.

Si il décide de passer la main, on arrive à B:0; A:6

sinon il décide de continuer, le score du tour est 6

Si il obtient un 1 à son deuxième lancer de dé, on arrive à B: 0 A:1

sinon si il obtient un 2 le score du tour est 8.

Si il décide de passer la main, on arrive à B:0; A: 8

Si il décide de continuer .....

Q 1.supposons que A ait 92 alors que B a 98. C'est à A de jouer, il lance le dé et obtient 6; a-t-il intérêt à continuer ou à passer la main?

Q 2. Ecrire un algorithme qui permet à un joueur de définir la meilleure stratégie.

[1] $|u|$ (resp. $|p|$) désigne la longueur de u (resp. p).