Université de Lille

FST- Département Informatique

ACT

2018–2019
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 : 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. Proposer 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 3 : 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 deux 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 $*bab*$, $*b$, $b*$, $*bbb*a$.
Q 1. Pour chacun des motifs suivants, dire si il filtre *abba*: $*ab*$, $*aa*$, $*a*$, $b*$, $**$
Q 2. Proposer un algorithme linéaire, i.e. en $O(|u| + |p|)$ [1], pour le cas où le motif ne contient pas d'$*$:
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:
Entrée: un mot u sur Σ et un motif p sur $\Sigma \cup \{?, *\}$
Sortie: Oui si u est filtré par p, non sinon.
Aide: Il existe plusieurs approches efficaces du problème et vous pouvez choisir celle qui vous convient. Une méthode possible et simple utilise la programmation dynamique: dans ce cas, exprimez, par exemple, $filtre(xu, yp)$ en fonction de $filtre(u, p)$, $filtre(xu, p)$, $filtre(u, yp)$, x étant une lettre de Σ, y étant une lettre de $\Sigma \cup \{?, *\}$.
L'algorithme que vous avez conçu se comporte-t-il en $O(n)$ si le motif ne comporte pas d'$*$?
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. Quelles sont les valeurs de:

1. $filtrapp(examen, ???)$
2. $filtrapp(examen, *e)$
3. $filtrapp(examen, e*)$

Q 4.2. Proposer un algorithme en $O(|u| * |p|)$ pour calculer $filtrapp(u, p)$.

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

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

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. Soient x et y, $1 \leq x \leq nj, 1 \leq y < np$: exprimer $SG(x, y)$ en fonction des $SG(x', y + 1), 0 <= x' <= x$.

Aide: pour tout x', on peut consacrer $x - x'$ jours au projet y, et donc x' aux projets de numéro supérieur à y.

Q 2.3. Compléter la table SG pour $nj = 5$.

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 : Défi: les bouteilles de vin, inspiré de *Michal Danilák*

"Given prices array of N bottles – P[N] of wine (not essentially in sorted order) arranged on shelf of a bar. Every year we've to sell one bottle of wine. Selling price of a bottle will be YEAR * P[i]. Assuming we start with YEAR=1. But constraint is that we can sell either leftmost or rightmost bottle of wine from shelf. We can't pick bottles from the middle. For any given input – print the order in which we should sell the bottles so that profit is MAXIMUM."

Q 1. Pour P=[2,3,5,1,4], quelle est la meilleure solution?

Q 2. Proposer un algorithme en $O(n^2)$ pour résoudre le problème.