Université de Lille

FST- Département Informatique

ACT
Durée 1h
Tous documents autorisés

2018–2019
Master1 Informatique
22 octobre 2018

Quelques éléments de réponse

Les algorithmes seront écrits en pseudo-langage ou dans un langage de votre choix. La clarté de vos réponses et de votre code sera prise en compte. Le barème est un barème "minimal".

Exercice 1 : Simplifier - *3 pts* -

Soit l'algorithme suivant :

```
// n entier >=0
//X un tableau  de n entiers >=0
//T un tableau nxn d'entiers
// T initialisé à 0
public void Myst () {
  for (int i=0; i<n; i++)
     for (int j=i; j <n; j++) {
        int M=0;
        for (int k=i; k<=j; k++)
             if (M<X[k]) M=X[k];
        T[i][j]=M;}}
```

Q 1. Quelle est la complexité temporelle de l'algorithme ? *Rep :* $\Theta(n^3)$

Q 2. Que contient T à la sortie de Myst ? Pouvez-vous transformer l'algorithme pour qu'il soit en $\Theta(n^2)$ et réalise la même chose ?

Rep :

A la sortie, pour i<=j, T[i][j] est le max des X[i], ...,X[j].

```
public void Myst () {
  for (int i=0; i<n; i++)
     int M=0;
     for (int j=i; j <n; j++) {
        if (M<X[j]) M=X[j];
        T[i][j]=M;}}
```

Remarque : Cela ressemblait beaucoup à un élément du TP1 pour construire l'algorithme quadratique.

Exercice 2 : Analyse d'algorithmes récursifs. - *3 pts* -

Pour les algorithmes suivants, estimer l'ordre de grandeur en fonction de n du nombre d'appels récursifs effectués :

```
int A1(int n){
  if (n>2) return A1(n-3);
  else return 1; }
```

Rep : $\Theta(n)$

```
int A2(int n){
  if (n>1) return A2(n div 3);
  else return 1;}
```

Rep : $\Theta(\log n)$

```
int A3(int n){
  if (n>1) return A3(n-1) + A3(n-1);
  else return 1;}
```

Rep : $\Theta(2^n)$

Exercice 3 : Le solitaire - *4 pts*

Pour une raison un peu étrange, dans un tableau trié T de n éléments, tous les éléments sont en double sauf un. Proposez un algorithme en $\Theta(\log n)$ pour déterminer cet élément qui n'est pas en double. Par exemple, pour $1, 1, 2, 2, 4, 4, 8, 8, 9, 11, 11$, la sortie attendue est 9.

Rep : C'est bien sûr une recherche par dichotomie qu'il fallait faire. L'invariant sera donc que la "tranche courante" contient tous ses éléments en double sauf un : le solitaire initial. "Avant le solitaire", on a T[2i]=T[2i+1], ensuite non [1]. Le plus simple, pour éviter des tests un peu pénibles selon la parité, était de raisonner sur le nombre de "paquets", un paquet étant constitué d'un élément et de son double.

```
int g=0, d=n/2;
while (g<d) {//inv: g<d,T[2*g..2*d+1] trié et contient tous ses éléments en double sauf le solitaire
   m=(g+d)/2;
   if (T[2*m]==T[2*m+1]) g=m+1; else d=m;}
return T[2*g];
```

1. On suppose que chaque élément est soit solitaire - pour un seul- soit en double et alors pas en triple ou ...

Remarque : trié sert juste ici à exprimer simplement que les valeurs égales sont côte à côte.

Exercice 4 : Somme maximale - *10 pts* -

Soit un tableau T de n entiers positif. On souhaite extraire de ce tableau des entiers tels que :

— on ne peut choisir deux entiers consécutifs dans le tableau (i.e. T[i] et T[i+1]).

— la somme des entiers choisis soit maximale.

Exemple : supposons que le tableau contienne dans cet ordre $4, 3, 7, 12, 2, 5, 9, 1$; en respectant la première contrainte, on peut choisir **par exemple** $4, 7, 2, 9$ ou $4, 12, 5, 1$ ou $3, 12, 1$ ou $4, 12, 9$. C'est $4, 12, 9$ qui maximise la somme.

Q 1. Pour $30, 50, 100, 20$, quelle est la solution optimale ? Même question pour $40, 50, 45, 50, 60, 100, 60$.

Rep : 30+100=130 ; 40 +45 + 60 + 60=205

Q 2. Pensez-vous que l'algorithme glouton naïf suivant retourne toujours la somme maximale ? Justifiez.

```
int i=0, s=0;
while (i<n) {
  if ((i==n-1) || (T[i]>T[i+1]))
    {s=s+T[i];System.out.print(T[i]); i=i+2;}
  else
    {s=s+T[i+1];System.out.print(T[i+1]); i=i+3;}}
```

Rep : Non, exemples de la question 1

Q 3. Pour i, $0 \leq i \leq n$, soit Meil(i) la somme optimale qu'on puisse produire avec la tranche de tableau T[i..n-1]. On pourra poser Meil(n)=0. Par exemple pour $4, 3, 7, 12, 2, 5, 9, 1$, Meil contient $25, 24, 21, 21, 11, 9, 9, 1, 0$.

Q 3.1. Que vaut Meil pour pour $30, 50, 100, 20$? *Rep :* 130 100 100 20

Q 3.2. Dans le cas général, que vaut Meil(n-1) ? *Rep :* T[n-1]

Q 3.3. Pour $i < n - 1$, exprimer Meil(i) en fonction de Meil(i+1), Meil (i+2) et T[i].

Rep : Meil(i)= max(Meil(i+1), T[i]+Meil(i+2))

Q 3.4. Proposer un algorithme en $\Theta(n)$ qui calcule la somme optimale. Compléter l'algorithme pour qu'il sorte également la (ou une, si il y a plusieurs possibilités) liste des entiers correspondants.

Rep :

```
//calcul de Meil
int n= T.length;
int[] M=new int[n+1];
M[n]=0;
M[n-1]=T[n-1];
for (int i=n-2; i >=0; i--)  M[i]=Math.max(M[i+1],T[i]+M[i+2]);
//remontée
int i=0;
while (i<n) {
     if (M[i]!=M[i+1]) {System.out.println(T[i]);i=i+2;} else i++;}
```

Remarques : si l'exercice a été assez bien réussi, très peu ont sorti correctement la liste des entiers correspondants.

Vous êtes très souvent tentés de sortir les résultats à la volée lors du remplissage de la table : cela ne marche en général pas, tout simplement car dans le remplissage de la table on calcule tous les sous-problèmes ; par exemple, ici trop d'entiers seront sortis. Pour ceux qui l'ont fait ainsi, déroulez votre algorithme sur un petit exemple -par exemple 5,4,3,2,1 - et vous comprendrez pourquoi :-)

Toujours pour éviter cette remontée, certains ont stocké au fur et à mesure la liste des entiers solutions ; cela marche mais attention cela peut être gourmand en mémoire ; cela pouvait, si vous ne preniez pas garde à l'implémentation, mener à un algorithme qui n'était pas en $O(n)$.

Il est vrai que dans certains cas, mémoriser également des "indices" facilite cette remontée. Comme le montre le code ci-dessus, la "remontée" dans la table était ici très simple. Il était inutile d'essayer de l'éviter ou de la faciliter en mémorisant plus que la valeur.