Université de Batna
Faculté des sciences de l'ingénieur
Département d'informatique
Module : Algorithmique avancée et preuve de programme
Durée : 1h30 Le 26/09/2010

Contrôle de rattrapage
Master I

Questions de cours (4 pts)

1. Donnez les différents types de récursivité qu'on a vue dans le cours. [2 pts]
2. Quel est le problème majeur de la récursivité ? [1 pt]
3. Quels sont les deux grands éléments de la stratégie gloutonne ? [1 pts]

Problème (16 pts)

Soit un ensemble de pistes audio stockées sur un disque dur. Nous cherchons à enregistrer ces pistes sur une casette de **T** minutes tel que la face A peut contenir $\mathbf{t_1}$ minutes et la face B peut contenir $\mathbf{t_2}$ minutes ($T = t_1+t_2$).

1. Proposez un algorithme **récursif** sous forme d'une fonction **DUREE(i, D, t_1, t_2)** qui calcule la durée de musique qu'on peut enregistrer sur la casette sachant que l'on n'enregistre que les '**i**' premières pistes. Le tableau D contient les durées des pistes du disque dur (exp. D[5]=la durée de la 5ème piste du disque dur).
   Une piste audio doit être enregistrée soit sur la face A ou la face B mais pas les deux, ou n'est pas enregistrée. [6 pts]
2. Calculez la complexité de votre algorithme en nombre de comparaisons avec les éléments du tableau **D**. [2 pts]
3. Nous voulons maintenant remplir la casette d'une manière à ce qu'on y laisse le minimum de vide possible. Pour cela, nous allons réfléchir d'une manière gloutonne. Cette méthode ne produit pas toujours une solution optimale. Nous allons tout d'abord trier le tableau **D** d'une manière croissante (fonction **TRI(D)** tri par tas ou autre) puis choisir les premières pistes pour laisser le minimum de vide possible.
   Supposant que la fonction **TRI(D)** existe, modifiez l'algorithme précédent pour réaliser la fonction **PISTES(i, D, t_1, t_2, C, dc)** qui retourne la durée de musique qu'on peut enregistrer sur la casette ainsi que la liste des pistes choisies (dans le tableau '**C**') pour laisser le minimum de vide possible sur la casette. **dc** est la somme des durées des pistes qui sont dans **C**. [6 pts]
4. Calculez la complexité de votre algorithme **PISTES()** en nombre de comparaisons avec les éléments du tableau **D** sachant que la complexité de l'algorithme de tri est **O(x)**. [2 pts]

Bonne chance…

Correction du contrôle de rattrapage
Master I

Questions de cours (4 pts)

1. Donnez les différents types de récursivité qu'on a vue dans le cours.
   - **La récursivité simple** [0.5 pt]
   - **La récursivité multiple** [0.5 pt]
   - **La récursivité mutuelle** [0.5 pt]
   - **La récursivité imbriquée** [0.5 pt]

2. Quel est le problème majeur de la récursivité ?
   - **Le problème majeur de la récursivité est la terminaison.** [1 pt]

3. Quels sont les deux grands éléments de la stratégie gloutonne ?
   - **Choix glouton** [0.5 pt]
   - **Sous-structure optimale** [0.5 pt]

Problème (16 pts)

Soit un ensemble de pistes audio stockées sur un disque dur. Nous cherchons à enregistrer ces pistes sur une casette de **T** minutes tel que la face A peut contenir $\mathbf{t_1}$ minutes et la face B peut contenir $\mathbf{t_2}$ minutes ($T = t_1+t_2$).

1. Proposez un algorithme **récursif** sous forme d'une fonction **DUREE(i, D, t_1, t_2)** qui calcule la durée de musique qu'on peut enregistrer sur la casette sachant que l'on n'enregistre que les '**i**' premières pistes. Le tableau D contient les durées des pistes du disque dur (exp. D[5]=la durée de la 5ème piste du disque dur).
   Une piste audio doit être enregistrée soit sur la face A ou la face B mais pas les deux, ou n'est pas enregistrée. [6 pts]

```
DUREE(i,D,t1,t2)
    di ← 0
    si (D[i] ≤ t1) alors [1.5 pt]
        t1 ← t1 - D[i]
        di ← D[i]
    sinon si (D[i] ≤ t2) alors [1.5 pt]
        t2 ← t2 - D[i]
        di ← D[i]
    si ( i < taille(D) AND (t1 + t2)>0 ) alors [1 pt]
        ret ← DUREE(i+1,D,t1,t2) [1 pt]
    retourner di + ret [1 pt]
```

2. Calculez la complexité de votre algorithme en nombre de comparaisons avec les éléments du tableau **D**. [2 pts]

La complexité : Au maximum on fait 2 comparaisons par appel à la fonction DUREE(), et pour chaque **i** nous faisons un appel à cette fonction. La valeur maximale de **i** est nombre de pistes dans le disque dur.
Donc la complexité est : **O(n) = 2*n**. Tel que n = nombre de pistes dans le disque dur. [2 pts]

3. Nous voulons maintenant remplir la casette d'une manière à ce qu'on y laisse le minimum de vide possible. Pour cela, nous allons réfléchir d'une manière gloutonne. Cette méthode ne produit pas toujours une solution optimale. Nous allons tout d'abord trier le tableau **D** d'une manière croissante (fonction **TRI(D)** tri par tas ou autre) puis choisir les premières pistes pour laisser le minimum de vide possible.
   Supposant que la fonction **TRI(D)** existe, modifiez l'algorithme précédent pour réaliser la fonction **PISTES(i, D, t_1, t_2, C, dc)** qui retourne la durée de musique qu'on peut enregistrer sur la casette ainsi que la liste des pistes choisies (dans le tableau '**C**') pour laisser le minimum de vide possible sur la casette. **dc** est la somme des durées des pistes qui sont dans **C**. [6 pts]

```
PISTES(i,D,t1,t2,C, dc)
    TRI(D) // tri croissant [0.5 pt]
    si (D[i] ≤ t1) alors [0.5 pt]
        dc ← dc + D[i] [0.5 pt]
        t1 ← t1 - D[i] [0.5 pt]
        taille(C) ← taille(C)+1 [0.25 pt]
        C[taille(C)] ← i [0.25 pt]
    sinon si (D[i] ≤ t2) alors [0.5 pt]
        dc ← dc + D[i] [0.5 pt]
        t2 ← t2 - D[i] [0.5 pt]
        taille(C) ← taille(C)+1 [0.25 pt]
        C[taille(C)] ← i [0.25 pt]
    si ( i < taille(D) AND (t1 + t2 > 0) ) alors [0.5 pt]
        PISTES(i+1,D,t1,t2,C,dc) [1 pt]
```

4. Calculez la complexité de votre algorithme **PISTES()** en nombre de comparaisons avec les éléments du tableau **D** sachant que la complexité de l'algorithme de tri est **O(x)**. [2 pts]

La complexité : La complexité de l'algorithme de tri est O(x).
En plus de la complexité du tri, on fait 2 comparaisons par appel à la fonction PISTES(), et pour chaque **i** nous faisons un appel à cette fonction. La valeur maximale de **i** est **taille(D)**.
Donc la complexité de la fonction PISTES() est : **O(n + x) = 2*n + O(x)**. Tel que n = nombre de pistes dans le disque dur. [2 pts]