Université de Batna
Faculté des sciences de l'ingénieur
Département d'informatique

Module : Algorithmique avancée et preuve de programme

Durée : 1h30 Le 26/06/2010

Contrôle final
Master I

Questions de cours (5 pts)

1. Donnez les 7 types de complexité qu'on a vue dans le cours sous la forme suivante :
   Algorithme constant : O(1) (3 pts)
2. Dans un graphe G=(S, A) non orienté connexe, donnez la relation entre |S| et |A|. (1 pt)
3. Dans un graphe G=(S, A) non orienté connexe et acyclique, donnez la relation entre |S| et |A|. (1 pt)

Problème (15 pts)

Soit **E** un ensemble d'objets où chaque objet '**i**' possède un prix **p[i]** et un poids **q[i]**.
Proposez un algorithme récursif qui permet de mettre dans un sac avec une capacité de **5 Kg** un sous-ensemble d'objets de l'ensemble **E** avec un prix maximal.
Pour faciliter le problème, en premier lieu on choisi les objets optimaux dont le prix est supérieur au seuil **sp** et le poids est inférieur au seuil **sq**. Après, et si le sac n'est pas rempli, on choisi les objets de prix maximal et qui peuvent entrer dans le sac.
Pour faciliter la résolution de ce problème, suivez les étapes suivantes :

1. Ecrivez l'algorithme récursif de la fonction OBJETS_OPTIMAUX(P, sp, Q, sq, idx, C, poidsC, D) qui prend en argument le tableau des prix P, le tableau des poids Q, l'indexe de parcours des deux tableaux, un tableau de sortie C qui contient les indices des objets choisis, poidsC la somme des poids des objets du tableaux C et le tableau de sortie D qui contient les éléments non choisis. (4 pts)
2. Soit l'algorithme de tri par insertion suivant vu dans le cours :

```
TRI_INSERTION(A)
    Pour j ← 2 à taille(A) faire
        clé ← A[j]
        i ← j-1
        tant que i > 0 et A[i] > clé faire
            A[i+1] ← A[i]
            i ← i-1
        A[i+1] ← clé
```

   Transformer cette algorithme en un autre algorithme TRI_INSERTION(D, P) qui fait le tri du tableau des indices des objets D d'une manière décroissante des prix des objets, c-à-d P[D[1]] ≥ P[D[2]] ≥ P[D[3]]… ≥ P[D[taille(D)]] (4 pts)
3. Ecrivez l'algorithme SAC(P, sp, Q, sq, C) qui utilise la fonction OBJETS_OPTIMAUX() et la fonction transformée TRI_INSERTION() pour remplir le tableau C par les objets optimaux, puis si le sac n'est pas rempli, ajouter les objets de prix maximal et qui peuvent entrer dans le sac. (4 pts)
4. Calculez la complexité maximale de l'algorithme SAC() en nombre de comparaisons avec les éléments des tableaux. (3 pts)

Bonne chance…

Correction du contrôle final
Master I

Questions de cours (5 pts)

1. Donnez les 7 types de complexité qu'on a vue dans le cours sous la forme suivante : Algorithme constant : O(1) (3 pts)
   Algorithme sub-linéaire : O(log n) (0.5 pts)
   Algorithme linéaire : O(n) (0.5 pts)
   Algorithme quasi-linéaire : O(n.log n) (0.5 pts)
   Algorithme polynomial : O(n^k) (0.5 pts)
   Algorithme exponentiel : O(k^n) (0.5 pts)
   Algorithme factoriel : O(n!) (0.5 pts)
2. Dans un graphe G=(S, A) non orienté connexe, donnez la relation entre |S| et |A|. (1 pt)
   |A| ≥ |S| - 1 (1 pts)
3. Dans un graphe G=(S, A) non orienté connexe et acyclique, donnez la relation entre |S| et |A|. (1 pt)
   |A| = |S| - 1 (1 pts)

Problème (15 pts)

Soit **E** un ensemble d'objets où chaque objet '**i**' possède un prix **p[i]** et un poids **q[i]**.
Proposez un algorithme récursif qui permet de mettre dans un sac avec une capacité de **5 Kg** un sous-ensemble d'objets de l'ensemble **E** avec un prix maximal.
Pour faciliter le problème, en premier lieu on choisi les objets optimaux dont le prix est supérieur au seuil **sp** et le poids est inférieur au seuil **sq**. Après, et si le sac n'est pas rempli, on choisi les objets de prix maximal et qui peuvent entrer dans le sac.
Pour faciliter la résolution de ce problème, suivez les étapes suivantes :

1. Ecrivez l'algorithme récursif de la fonction OBJETS_OPTIMAUX(P, sp, Q, sq, idx, C, poidsC, D) qui prend en argument le tableau des prix P, le tableau des poids Q, l'indexe de parcours des deux tableaux, un tableau de sortie C qui contient les indices des objets choisis, poidsC la somme des poids des objets du tableaux C et le tableau de sortie D qui contient les éléments non choisis. (4 pts)
2. Soit l'algorithme de tri par insertion suivant vu dans le cours :

```
TRI_INSERTION(A)
    Pour j ← 2 à taille(A) faire
        clé ← A[j]
        i ← j-1
        tant que i > 0 et A[i] > clé faire
            A[i+1] ← A[i]
            i ← i-1
        A[i+1] ← clé
```

   Transformer cette algorithme en un autre algorithme TRI_INSERTION(D, P) qui fait le tri du tableau des indices des objets choisis D d'une manière décroissante des prix des objets, c-à-d P[D[1]] ≥ P[D[2]] ≥ P[D[3]]… ≥ P[D[taille(D)]] (4 pts)
3. Ecrivez l'algorithme SAC(P, sp, Q, sq, C) qui utilise la fonction OBJETS_OPTIMAUX() et la fonction transformée TRI_INSERTION() pour remplir le tableau C par les objets optimaux, puis si le sac n'est pas rempli, ajouter les objets de prix maximal et qui peuvent entrer dans le sac. (4 pts)
4. Calculez la complexité maximale de l'algorithme SAC() en nombre de comparaisons avec les éléments des tableaux. (3 pts)

OPJETS_OPTIMAUX(P, sp, Q, sq, idx, C, poidsC, D) (4 pts)

```
    si (P[idx]≥sp et Q[idx]≤sq et (poidsC+Q[idx])≤5) alors
```

```
        taille(C) ← taille(C)+1
        C[taille(C)] ← idx
        poidsC ← poidsC + Q[idx]
    sinon
        Taille(D) ← taille(D)+1
        D(taille(D)] ← idx
    si (idx < taille(P) et poidsC < 5) alors
        OPJETS_OPTIMAUX(P, sp, Q, sq, idx+1, C, poidsC, D)
```

`TRI_INSERTION(D, P)` (4 pts)

```
pour j ← 2 à taille(D) faire
    clé ← D[j]
    i ← j-1
    tant que (i>0 et P[D[i]] ≤ P[clé]) faire
        D[i+1] ← D[i]
        i ← i-1
    D[i+1] ← clé
```

`SAC(P, sp, Q, sq, C)` (4 pts)

```
poidsC ← 0
OPJETS_OPTIMAUX(P, sp, Q, sq, 1, C, poidsC, D)
si (poidsC < 5) alors
    TRI_INSERTION(D, P)
    i ← 1
    tant que (i ≤ taille(D) et (poidsC + Q[D[i]]) ≤ 5) faire
        taille(C) ← taille(C)+1
        C[taille(C)] ← D[i]
        poidsC ← poidsC + Q[D[i]]
        i ← i+1
```

La complexité : (3 pts)
Soit : TP = taille(P), TQ = taille(Q), TD = taille(D)
L'algorithme SAC() fait 1 comparaison (poidsC + Q[D[i]]) ≤ 5) + la complexité de TRI_INSERTION() + la complexité de OBJETS_OPTIMAUX().
La complexité de OBJETS_OPTIMAUX() = 3.TP parce que pour chaque élément de P on fait un appel à cette fonction, et chaque appel fait 3 comparaisons.
La complexité de TRI_INSERTION() : Chaque itération de la boucle « tant que » fait (j-1) comparaisons (P[D[i]] ≤ P[clé]) et le j change de 2 jusqu'à TD, ce qui fait : pour j=2 nous avons j-1=1 comparaison, pour j=3 nous avons j-1=2 comparaisons,… pour j=TD nous avons j-1=TD-1 comparaisons.
Donc la complexité de TRI_INSERTION() = 1+2+3+…+(TD-1) c'est une suite arithmétique de base 1, sa somme est :

La complexité de TRI_INSERTION() = $\frac{TD-1}{2}\cdot(1+TD-1)=\frac{1}{2}\cdot(TD^2-TD)$

Donc la complexité de SAC() est : 1 + 3.TP + ½.(TD² - TD)

$$O(TP+TD^2-TD)=1+3\cdot TP+\frac{1}{2}\cdot(TD^2-TD)$$