TD/TP Algorithme et Complexité

Ces 3 exercices traitent de la complexité algorithmique. Vous choisirez le langage PYTHON pour réaliser les implémentations demandées. Vous me remettrez un document pdf (éventuellement obtenu au travers d'un ipython notebook pour ceux qui le souhaitent) précisant les réponses aux exercices, les codes demandés ainsi que les évaluations empiriques. Pour mesurer les temps, nous utiliserons les commandes suivantes :

```
import time
tmps1=time.clock()
% your code
tmps2=time.clock()
print "Temps d'execution = %d\n" %(tmps2-tmps1)
```

1 Exercice 1 : recherche d'un élément dans un tableau

On souhaite effectuer la recherche d'un élément dans un tableau. Pour cela on considère deux mods de recherche : séquentielle et dichotomique. On s'intéresse à leur complexité temporelle. Pour cela, considérez un tableau ayant dix mille éléments (version triée, et version non triée). Pour chaque algorithme, et pour chaque version du tableau, combien de comparaisons sont à effectuer pour :

- trouver un élément qui y figure ?
- trouver un élément qui n'y figure pas ?

Quels sont les cas où le tableau est parcouru complètement et les cas où un parcours partiel est suffisant ? Conclure en donnant la complexité temporelle pour chaque algorithme Donnez une implémentation pour chacune de ces recherches, ainsi que les temps de calcul associés.

2 Exercice 2 : matrices creuses

Nous avons évoqué dans un partiel récent la manière de représenter ce que l'on appelle des « matrices creuses », c'est-à-dire des matrices de valeurs contenant plusieurs éléments nuls. Dans la version simple de gestion d'une matrice, celle-ci est représentée par un tableau à deux dimensions dont les cases contiennent les éléments. Il est aussi possible de la représenter par un tableau à une dimension. On ne s'intéresse alors qu'aux éléments de la matrice qui ne sont pas nuls. Chaque case du tableau contient un tuple (i, j ,a) correspondant à l'indice de ligne, l'indice de colonne, et la valeur d'un élément non nul.

Dans un premier temps, le problème considéré consiste à calculer la somme des éléments d'une matrice. Ecrivez et implémentez un algorithme permettant de calculer cette somme, pour chacune des deux représentations, puis comparez leur complexité spatiale (espace mémoire occupé) et leur complexité temporelle (nombre d'opérations à effectuer). Que peut-on conclure de cette comparaison ? Montrez qu'il existe une valeur critique du nombre d'éléments non nuls à partir de laquelle une méthode l'emporte sur l'autre.

Discutez dans un second temps le cas où l'on fait le produit entre deux matrices, en implémentant les méthodes associées.

Mesurez les temps de calcul obtenus avec ces méthodes pour des tailles de matrices allant de 10×10 à 1000×1000, que vous initialiserez aléatoirement en fonction d'un pourcentage de valeurs nulles. Donnez les courbes associées.

3 Exercice 3 : Partition d'un tableau

On dispose d'un tableau non trié A composé de n entiers. On souhaite décider s'il existe un sous-ensemble $I \subseteq \{1, 2, \cdots, n\}$ tel que

$$\sum_{i=I} A[i] = \sum_{i=\{1,2,\cdots,n\}/I} A[i]$$

Proposez un algorithme pour résoudre ce problème. Quelle est sa complexité en temps ? Implémentez cette algorithme et testez le.

4 Exercice 4 : étude d'algorithme

Calculer la complexité du programme ci-dessous en nombre d'appels de la fonction f en fonction de n dans les deux cas suivants :

```
for i = 1 to n-1:
  for j = i+1 to n:
    for k = 1 to j:
      t[i,j,k] := f(t[i,j,k])

for i = 1 to n:
  for j = 1 to i:
    for k = j to i:
      t[i,j,k] := f(t[i,j,k])
```

Vérifiez expérimentalement vos conclusions en faisant varier n dans votre programme et en comparant les deux temps d'exécution.