Université des Sciences et de la Technologie Houari Boumediène
Faculté d'Electronique et d'Informatique
Département d'Informatique
LMD Master 1ère Année RSD 2009/10
Module ''Algorithmique Avancée et Complexité''

Corrigé de l'interrogation

Exercice 1 (complexité du tri) :

On considère l'algorithme suivant de tri par insertion d'un tableau A de n entiers :

Entrée : Un tableau A de n entiers
Sortie : Le tableau A trié par ordre croissant

```
TRI-INSERTION(A){
    Pour j=2 à n{
        clé= A[j]
        i=j-1
        Tant que i>0 et A[i]>clé{
                A[i+1]=A[i]
                i=i-1
                }
        A[i+1]=clé
        }
}
```

1. Calculez en fonction de n le nombre T(n) d'opérations dans le pire des cas de l'algorithme. Expliquez.

Réponse :

1. TRI-INSERTION(A){
2. **Pour** j=2 à n{
    a. clé= A[j]
    b. i=j-1
    c. **Tant que** i>0 **et** A[i]>clé{
        i. A[i+1]=A[i]
        ii. i=i-1
        iii. }
    d. A[i+1]=clé
    e. }
3. }

Instruction	Nombre d'opérations	Nombre de fois
1	1	1
2	1	n-1
2a	1	n-1
2b	2	n-1
2c	2	$\sum_{j=2}^{n}(j-1)$
2c(i)	1	$\sum_{j=2}^{n}(j-1)$
2c(ii)	2	$\sum_{j=2}^{n}(j-1)$
2d	1	n-1

Le nombre total d'opérations de l'algorithme, dans le pire des cas, est donc :
$T(n)=1+4(n-1)+5\sum_{j=2}^{n}(j-1)=4n-3+5\sum_{j=1}^{n-1}j=4n-3+5\frac{(n-1)n}{2}=\frac{8n-6+5n^2-5n}{2}=\frac{5n^2+3n-6}{2}$

2. Trouvez une fonction $f(n)=n^k$ (k constante) vérifiant $T(n)=O(f(n))$ et $f(n)=O(T(n))$. Expliquez.

Réponse :
La fonction f demandée est $f(n)= n^2$

$T(n)=O(f(n))$?
Il suffit de trouver un entier $n_0\geq 0$ et une constante réelle $c\geq 0$ tels que pour tout $n\geq n_0$ on ait $T(n)\leq c*f(n)$, c'est-à-dire $\frac{5n^2+3n-6}{2} \leq c * n^2$, ou encore $\frac{5}{2}+\frac{3n-6}{2n^2}\leq c$ (prendre $n_0=2$ et $c=4$)

$f(n)=O(T(n))$?
Il suffit de trouver un entier $n_0\geq 0$ et une constante réelle $c\geq 0$ tels que pour tout $n\geq n_0$ on ait $f(n)\leq c*T(n)$, c'est-à-dire $n^2 \leq c * \frac{5n^2+3n-6}{2}$, ou encore $\frac{2n^2}{5n^2+3n-6}\leq c$ (prendre $n_0=2$ et $c=1$)

3. Que pouvez-vous déduire de la réponse à la question 2 ?

Réponse :
On déduit de la réponse à la question 2 que $T(n)= \ f=\Theta(n^2)$

Exercice 2 (NP-complétude) :

1. Illustrez à travers un exemple la notion d'instance d'un problème.

Réponse :

- Le problème SAT
  - Description : une conjonction de m clauses construites à partir de n propositions atomiques
  - Question : la conjonction est-elle satisfiable ?
- Instance du problème SAT
  - La conjonction $p\vee q\wedge p\vee\neg r\wedge\neg p\vee\neg q\vee\neg r$

2. Définissez les notions suivantes de la théorie de la NP-complétude :
    a. Certificat
    b. Algorithme de validation
    c. La classe NP
    d. Problème NP-complet

Réponse : voir cours

3. Donnez un algorithme polynomial de validation pour le problème SAT (SATisfiabilité). Utilisez la terminologie vue en TP, en TD et en cours. Expliquez.

Réponse :

L'algorithme de validation est comme suit. Il est écrit sous forme d'une fonction booléenne à quatre entrées n, m, C et inst. Le triplet (n,m,C) donne l'instance du problème (le nombre n de clauses, le nombre m de propositions atomiques, et la n*m-matrice C représentant la conjonction de clauses). L'algorithme retourne VRAI si et seulement si le certificat valide l'instance.

Si un entier est représenté sur p bits, le triplet (n,m,C) peut être vu comme un mot de $\{0,1\}^*$ de longueur p*(n*m+2) : les p premiers bits (0 ou 1) coderont le nombre n de clauses, les p suivants coderont le nombre m de propositions atomiques, les p*m suivants coderont la 1ère clause, …, les p*m derniers bits coderont la toute dernière clause.

```
Booléen validation_sat(n,m,C,inst){
        k=0 ;
        while(k<n){
                c_satisfaite=0;
                j=0;
                while(!c_satisfaite && j<m)
                        if( (inst[j]==0 && C[k][j]==0) ||
                            (inst[j]==1 && C[k][j]==1) )c_satisfaite=1
                        else j++ ;
                if(!c_satisfaite)retourner FAUX;
                        //Le certificat inst ne peut pas valider l'instance : il ne satisfait pas la clause k//
                k++ ;
                }
        Retourner VRAI;
                        // Le certificat inst valide l'instance : il satisfait chacune de ses clauses//
        }
```

Exercice 3 (Structures de données) : Une pile est une structure de données mettant en œuvre le principe « dernier entré premier sorti » (LIFO : Last In First Out). On considère ici le cas d'une pile implémentée avec un tableau.

1. Une pile doit être initialisée. Expliquez comment.

Réponse :

On initialise la longueur longueur(P) de la pile P, l'indice sommet(P) du sommet de pile , et l'élément P[sommet(P)] de sommet de pile : on suppose que la pile est implémentée avec un tableau P de taille 100 dont les éléments sont indicés de 1 à 100, que l'indice du sommet de pile est initialement 1, et que le sommet de pile lui-même est le symbole spécial $ (tester si la pile est vide reviendra à tester si le sommet de pile est ce symbole spécial $) :

```
longueur(P)=100 ;
Sommet(P)=1 ;
P[sommet(P)]=$ ;
```

2. Ecrivez les différentes fonctions et procédures permettant la gestion d'une pile.

Réponse :

Les fonctions et procédures permettat la gestion d'une pile sont comme suit :

```
PILE-VIDE(P){
        Si sommet(P)=$
                alors retourner VRAI
                sinon retourner FAUX
        }

EMPILER(P,x){
        Si sommet(P)=longueur(P)
                        alors erreur (débordement positif)
                        sinon{  sommet(P)=sommet(P)+1
                                P[sommet(P)]=x
                                }
        }

DEPILER(P){
        Si PILE-VIDE(P)
                        alors erreur (débordement négatif)
                        sinon{  sommet(P)=sommet(P)-1
                                retourner P[sommet(P)+1]
                                }
        }
```