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'examen

Exercice 1 (NP-complétude)(4,4,2) :

On considère le problème de décision 3-COLORIAGE de 3-coloriage d'un graphe :

- **Description :** un graphe
- **Question :** Peut-on colorier les sommets du graphe avec trois couleurs (distinctes) de telle sorte qu'il n'y ait pas de nœuds adjacents de même couleur ? Deux nœuds u et v sont adjacents si et seulement si (u,v) ou (v,u) est arc du graphe

1. Donnez un algorithme polynomial de validation pour le problème 3-COLORIAGE. Expliquez la polynomialité de l'algorithme.

Réponse :

L'algorithme de validation est comme suit. Il est écrit sous forme d'une fonction booléenne à trois entrées n, C et couleurs. La paire (n,C) donne le codage de l'instance du problème :

- L'instance est un graphe G=<V,E> (V={u_1,…,u_n} est l'ensemble des sommets de G, de cardinal n, et E l'ensemble de ses arcs)
- C est la matrice d'adjacence de G, de taille n*n, booléenne
  - C[i][j]=1 si et seulement si (u_i,u_j) est arc de G

L'entrée couleurs est un certificat consistant en un tableau de taille n, dont les éléments appartiennent à l'ensemble {Vert,Blanc,Rouge}. Le certificat couleurs associe à chacun des sommets de G une couleur (l'élément couleurs[i] est la couleur associée au sommet u_i). L'algorithme retourne VRAI si et seulement si le certificat couleurs valide l'instance, c'est-à-dire si et seulement si il n'existe pas de nœuds adjacents pour lesquels il associe la même couleurs.

Si un entier est représenté sur p bits, la paire (n,C) peut être vue comme un mot de $\{0,1\}^*$ de longueur p*(n^2+1) : les p premiers bits (0 ou 1) coderont le nombre n de sommets de l'instance, les p*n suivants coderont la 1ère ligne de la matrice C, …, les p*n derniers bits coderont la toute dernière ligne de la matrice C. Mais on peut faire mieux : voir la paire (n,C) comme un mot de $\{0,1\}^*$ de longueur p+n^2 : C étant booléenne, on peut coder chacun de ses éléments avec un unique bit, et non avec p bits.

```
Booléen validation_3c(n,C,couleurs){
        i=0 ;
        while(k<n){
                j=0;
                while(j<n){
                        if(C[i][j]==1 && couleurs[i]==couleurs[j]) retourner FAUX
                                //Le certificat couleurs ne peut pas valider l'instance://
                                //les sommets i et j sont adjacents et de même couleur//
                        j++ ;
                        }
                i++ ;
                }
        Retourner VRAI;
                // Le certificat couleurs valide l'instance : il n'existe pas de nœuds//
                //adjacents de même couleur//
        }
```

L'algorithme parcourt les éléments de la matrice C jusqu'à éventuellement satisfaire la condition C[i][j]==1 et couleurs[i]==couleurs[j] (deux sommets adjacents auxquels le certificat associe la même couleur), auquel cas il retourne FAUX. Si l'algorithme parcourt tous les éléments de la matrice C sans jamais satisfaire une telle condition, il retourne VRAI. Le nombre T(n) d'opérations effectuées par l'algorithme, dans le pire des cas, sur une instance de taille (nombre de sommets) n est T(n)= $c_1 * n^2$+c_2*n+c_3, c_1, c_2 et c_3 étant des constantes. Clairement, on a T(n)=Θ(n^2) ; en d'autres termes, T(n)=O(n^2) et n^2=O(T(n)). La complexité de la validation est donc bien en Θ(n^2).

2. Ecrivez un algorithme donnant, pour toute instance de 3-COLORIAGE, une sortie booléenne (OUI/NON) égale à OUI si et seulement si l'algorithme de validation ci-dessus valide l'instance ; c'est-à-dire si et seulement si il existe un certificat validant l'instance.

Réponse :

```
main();{
        Lecture de l'instance de 3-coloriage (le nombre n de sommets et la matrice C d'adjacence)
        cond=0 ;
        max=3^n;
                //il y a 3^n coloriages (certificats) possibles//
        i=0;
        while( !cond && i<max){
                for(j=0 ;j<n ;j++)couleurs[j]=0 ;
                        //Mise à zéro du tableau couleurs//
                dividende=i ;
                j=n-1 ;
                while(dividende !=0){
                        couleurs[j]=dividende%3 ;
                        dividende=dividende/3 ;
                        j-- ;
                        }
                        //écriture de i dans le système de numération de base 3 (3 chiffres, 0, 1 et 2)//
                        //résultat dans le certificat couleurs : 0 (Vert), 1 (Blanc), 2 (Rouge)//
                if(validation_3c(n,C,couleurs))cond=1 ;
                        //si le certificat valide l'instance, on ne parcourt pas les certificats restants//
                else i++ ;
                        //si le certificat courant ne valide pas l'instance, on passe au suivant//
                }
        if(cond)printf("oui");
        else printf("non");
        }
```

3. L'existence d'un algorithme polynomial de validation pour un problème de décision suffit-elle pour dire que le problème est NP-complet ? Expliquez.

Réponse :

L'existence d'un algorithme polynomial de validation pour un problème de décision ne permet pas de dire que le problème est NP-complet. Elle permet de dire que le problème est dans la classe NP.

Pour montrer que le problème est NP-complet, il faut, en plus de l'appartenance à la classe NP (existence d'un algorithme polynomial de validation), montrer que tout autre problème NP peut se ramener à ce problème via une réduction polynomiale ; ou, de façon équivalente, qu'il existe un problème NP-complet pouvant se ramener à ce problème via une réduction polynomiale.

Exercice 2 (Arbres binaires de recherche)(5) :

Donnez un algorithme permettant l'insertion d'un nouvel élément dans un arbre binaire de recherche, de telle sorte que l'arbre résultant soit un arbre binaire de recherche.

Réponse :

```
insérer(r,x){
        Si r≠NIL{
                Père=r ;
                Si(x->clé≤r->clé) alors insérer(r->sag,x) sinon insérer(r->sad,x) finsi
                }
        Sinon
                Si(x->clé≤père->clé) alors Père->sag=x sinon Père->sad=x finsi
        }

main(){
        …
        Père=u0 ;
                //u0 est un pointeur sur la racine de l'arbre binaire de recherche//
                //x est l'élément à insérer//
                //u0 et x sont de même type : pointeurs sur un enregistrement à trois champs (sag, clé, sad)//
                //x, après son insertion, sera feuille du nouvel arbre binaire de recherche//
        x->sag=NIL ;
        x->sad=NIL ;
                //Une feuille n'a pas de fils gauche et n'a pas de fils droit//
        insérer(u0,x) ;
        …
        }
```

Exercice 3 (Structures de données)(1,(1,1.5,1.5)) : Une file est une structure de données mettant en œuvre le principe « premier entré premier sorti » (FIFO : First In First Out). On considère ici le cas d'une file implémentée avec un tableau.

1. Une file doit être initialisée. Expliquez comment.

Réponse :

La file est implémentée avec un tableau F. On initialise la taille n de F à 100, par exemple ; la tête tête(F) de F à NIL (initialement, la file est vide) ; et la queue queue(F) de F à 1 (quand la file est vide, l'insertion d'un élément se fera à la position 1). De plus, on suppose que les éléments de la file sont indicés de 1 à 100 :

```
n=100 ;
tête(F)=NIL ;
queue(F)=1 ;
```

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

Réponse :

```
FILE-VIDE(F){
        si tête(F)=NIL
                alors retourner VRAI
                sinon retourner FAUX
```

```
	}

INSERTION(F,x){
	si [tête(F)≠NIL et queue(F)=tête(F)]
		alors erreur (débordement positif)
		sinon{
			F[queue(F)]=x
			queue(F)=[queue(F)+1](modulo n)
			si[tête(F)=NIL] alors tête(F)=1
			}
	}

SUPPRESSION(F){
	si FILE-VIDE(F)
		alors erreur (débordement négatif)
		sinon{
			temp=F(tête(F));
			tête(F)=[tête(F)+1](modulo n);
			si[tête(F)=queue(F)]{
				tête(F)=NIL ;
				queue(F)=1;
				}
			retourner temp;
			}
	}
```