Université des Sciences et de la Technologie Houari Boumediène
Faculté d'Electronique et d'Informatique
Département d'Informatique
LMD Master 1ère Année ''Réseaux et Systèmes Distribués'' 2010/2011
Module ''Algorithmique Avancée et Complexité''
Date : 18/01/2011

Corrigé de l'examen

Exercice 1 (NP-complétude)<8 points> :

On considère le problème de décision 3-COLORIAGE suivant :

- **Description :** un graphe G
- **Question :** peut-on colorier les sommets de G avec trois couleurs de telle sorte que deux nœuds adjacents n'aient pas la même couleur ?

Le but de l'exercice est de montrer que le problème 3-COLORIAGE est NP-complet. Vous supposerez que 3-COLORIAGE appartient à la classe NP (il admet un algorithme polynômial de validation), et que le problème 3-SAT défini comme suit est NP-complet :

- **Description :** une conjonction de clauses dont chaque clause a exactement trois littéraux
- **Question :** la conjonction est-elle satisfiable ?

Il vous est demandé de trouver une réduction polynômiale f transformant toute instance I du problème 3-SAT en une instance f(I) du problème 3-COLORIAGE, de telle sorte que la réponse rép(I) à la question associée à l'instance I est oui si et seulement si la réponse rép(f(I)) à la question associée à l'instance f(I) est oui. Et pour ce faire, il vous est demandé de procéder comme suit :

1. Trouvez la réduction f.
2. Expliquez la polynômialité de f.
3. Montrez que pour toute instance I de 3-SAT :

rép(I)=oui si et seulement si rép(f(I))=oui

Solution :

1) La réduction f est définie comme suit. Soit I une instance du problème 3-SAT, donnée par une conjonction C=$c_1 \wedge \ldots \wedge c_i \wedge \ldots \wedge c_m$ de m clauses dont chaque clause a exactement trois littéraux : $c_i = l_{i1} \vee l_{i2} \vee l_{i3}$, pour tout i=1…m. Soit P=$\{p_1,\ldots,p_n\}$ l'ensemble des n propositions atomiques qui occurrent dans C. La réduction f associe à I l'instance f(I) du problème de 3-coloriage donnée par le graphe G=(V,E) dont l'ensemble V des sommets et l'ensemble E des arêtes sont définis comme suit :

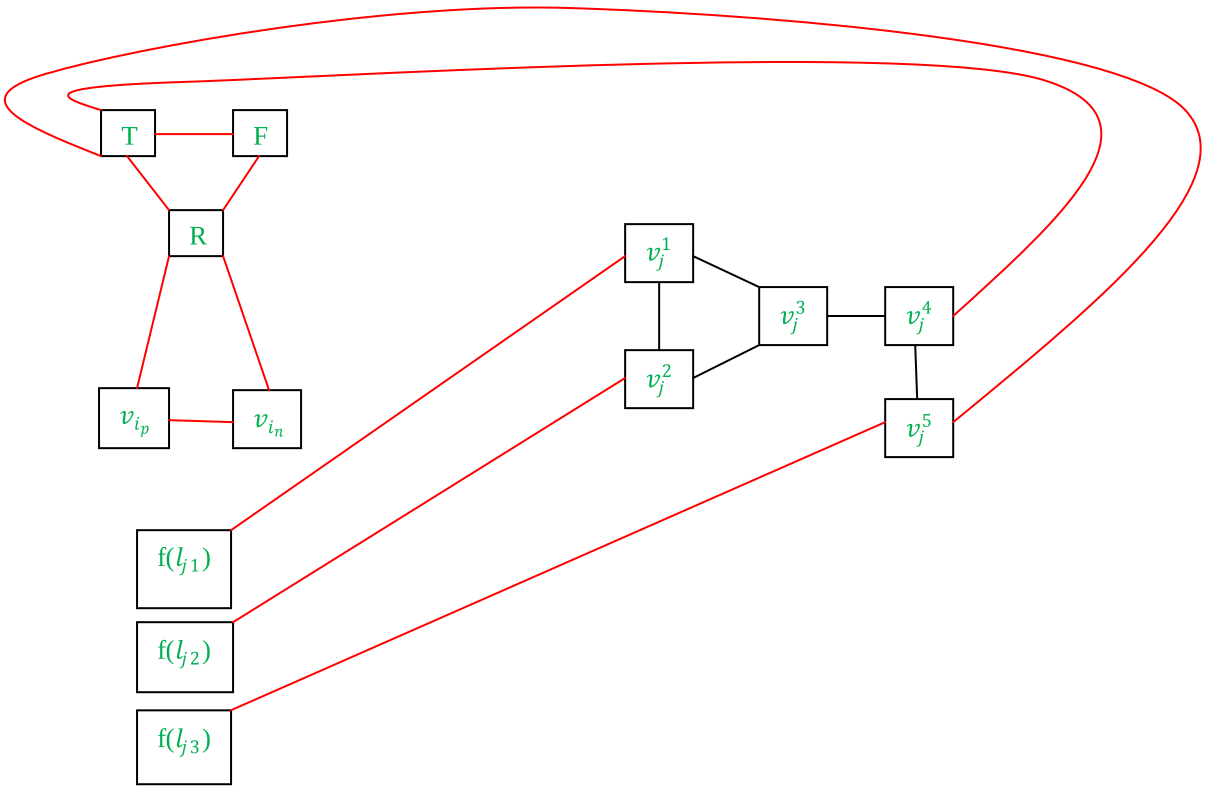
    a) Initialement V=∅ et E=∅
    b) Ajouter à V trois sommets 'distingués' T, F et R et relier ces trois sommets par un triangle :
        - V=V∪{T,F,R}
        - E=E∪{(T,F),(T,R),(F,R)}
    c) Pour toute proposition atomique p_i, ajouter à V les deux nouveaux sommets v_{i_p} et v_{i_n}, et relier ces deux sommets à R par un triangle :
        - V=V∪{v_{i_p},v_{i_n}}
        - E=E∪{(v_{i_p},v_{i_n}),(v_{i_p},R),(v_{i_n},R)}

    Dans la suite, on notera f(p_i)= v_{i_p} et f($\neg p_i$)= v_{i_n}

d) Pour toute clause $c_j = l_{j1} \vee l_{j2} \vee l_{j3}$, , ajouter à V les cinq nouveaux sommets v_j^1, v_j^2, v_j^3, v_j^4 et v_j^5, et à E les arêtes (v_j^1, v_j^2), (v_j^1, v_j^3),(v_j^2, v_j^3), (v_j^3, v_j^4),(v_j^4, v_j^5), $(v_j^1$,f$(l_{j1}))$,$(v_j^2$,f$(l_{j2}))$, $(v_j^5$,f$(l_{j3}))$,$(v_j^4$,T), $(v_j^5$,T) :

- V=V$\cup\{v_j^1, v_j^2, v_j^3, v_j^4, v_j^5\}$
- E=E$\cup\{(v_j^1, v_j^2), (v_j^1, v_j^3), (v_j^2, v_j^3), (v_j^3, v_j^4), (v_j^4, v_j^5), (v_j^1, \text{f}(l_{j1})), (v_j^2, \text{f}(l_{j2})), (v_j^5, \text{f}(l_{j5})), (v_j^4, \text{T}), (v_j^5, \text{T})\}$

La situation est résumée comme suit :



2) La polynômialité de f se déduit de la taille du graphe f(I) associé à une instance I de 3-SAT. Soit C la conjonction de clauses représentant I, et G le graphe f(I) que la réduction associe à I. Soit m le nombre de clauses de C et n le nombre de propositions atomiques occurrant dans C. Le nombre de sommets de G est clairement s(n,m)=3+2n+5m, et son nombre d'arêtes a(n,m)=3+3n+10m. La construction de G nécessite donc la création des 3+2n+5m sommets et des 3+3n+10m arêtes. On peut donc mesurer le nombre d'opérations élémentaires de f par la fonction T(n,m)=s(n,m)+a(n,m)=6+5n+15m=Θ(k), k étant le maximum entre n et m. La fonction f est linéaire, donc polynômial, en k.

3) I faut montrer ici que pour toute instance I de 3-SAT représentée par une conjonction de m clauses, l'instance f(I) de 3-COLORIAGE que f associe à I, représentée par un graphe G, vérifie ce qui suit : C est satisfiable ssi G est 3-coloriable.

    a) Supposons C satisfiable : soit t=$(t_1,\ldots,t_i,\ldots,t_n)$ une solution de C : t est une instanciation des n propositions atomiques occurrant dans C. On utilise l'ensemble couleurs={TRUE,FALSE,RED} de 3 couleurs pour 3-colorier les sommets de G sans que le 3-coloriage associe la même couleur à deux sommets adjacents. Les trois sommets distingués T, F et R sont coloriés TRUE, FALSE et RED, respectivement : tc(T)=TRUE, tc(F)=FALSE, tc(R)=RED. Pour tout i=1…n :

    - Si t_i=1 alors tc(v_{i_p})=TRUE et tc(v_{i_n})=FALSE

- Si t_i=0 alors tc(v_{i_p})=FALSE et tc(v_n)=TRUE

Pour tout j=1…m, la couleur associée à chacun des sommets provenant de la clause c_j peut être déterminée comme indiqué sur le tableau ci-dessous :

tc(f(l_{j1}))	tc(f(l_{j2}))	tc(f(l_{j3}))	tc(v_j^1)	tc(v_j^2)	tc(v_j^3)	tc(v_j^4)	tc(v_j^5)
TRUE	TRUE	TRUE	FALSE	RED	TRUE	FALSE	FALSE
TRUE	TRUE	FALSE	FALSE	RED	TRUE	FALSE	RED
TRUE	FALSE	TRUE	FALSE	TRUE	RED	FALSE	RED
TRUE	FALSE	FALSE	FALSE	TRUE	RED	FALSE	RED
FALSE	TRUE	TRUE	TRUE	FALSE	RED	FALSE	RED
FALSE	TRUE	FALSE	TRUE	FALSE	RED	FALSE	RED
FALSE	FALSE	TRUE	RED	TRUE	FALSE	RED	FALSE
FALSE	FALSE	FALSE	Im	po	ss	ib	le

b) Supposons maintenant que G est 3-coloriable, et soit tc donc un 3-coloriage acceptable de G. Sans perte de généralité, nous pouvons supposer que tc(R)=RED, tc(T)=TRUE et tc(F)=FALSE. Pour tout i=1…n, vu que les sommets provenant de la proposition p_i sont reliés au sommet R par un triangle, le 3-coloriage tc colorie les sommets v_{i_p} et v_{i_n} de l'une des deux façons suivantes : (tc(v_{i_p})=TRUE et tc(v_n)=FALSE) ou (tc(v_{i_p})=FALSE et tc(v_{i_n})=TRUE). A partir du 3-coloriage tc, on obtient trivialement une solution t=(t_1,…,t_i,…,t_n) de la conjonction C :

$$t_i=\begin{cases} 1\ si\ \mathrm{tc}\left(\mathrm{v}_{\mathrm{i}_\mathrm{p}}\right) = \mathrm{TRUE\ et\ tc}(\mathrm{v}_{\mathrm{i}_\mathrm{n}}) = \mathrm{FALSE} \\ 0\ si\ \mathrm{tc}\left(\mathrm{v}_{\mathrm{i}_\mathrm{p}}\right) = \mathrm{FALSE\ et\ tc}(\mathrm{v}_{\mathrm{i}_\mathrm{n}}) = \mathrm{TRUE} \end{cases}$$

Exercice 2 (tri d'un tableau)<3 points> :

Donnez un algorithme de complexité linéaire de fusion de deux tableaux triés T1 et T2, de telle sorte que le tableau résultant T soit trié. Justifiez la linéarité de l'algorithme.

Solution :

```
FUSIONNER(T1,n1,T2,n2){
        i=1;j=1;k=1;                                  (1)
        tant que i≤n1 et j≤n2{                        (2)
             si T1[i]<T2[j]                           (3)
                       alors{T[k]=T1[i];i=i+1;}       (4)
                       sinon{T[k]=T2[j];j=j+1;}       (5)
             fsi
             k=k+1;                                   (6)
             }
        tant que i≤n1{T[k]=T1[i];i=i+1;k=k+1;}        (7)
        tant que j≤n2{T[k]=T2[j];j=j+1;k=k+1;}        (8)
        }
```

L'algorithme est écrit sous forme d'une procédure FUSIONNER(T1,n1,T2,n2) à quatre arguments : T1 et T2 sont les tableaux triés à fusionner, n1 et n2 étant leurs tailles respectives. Le résultat de la fusion est un tableau T de taille supérieure ou égale à n1+n2. La procédure fait une passe lecture du tableau T1 et une passe lecture du tableau T2, et écrit chacun des deux tableaux dans le tableau résultat T. La complexité de la procédure est clairement linéaire, en Θ(n), n étant le maximum de n1 et n2. Les détails sont comme suit :

Instruction	Nombre d'opérations	Les opérations	Nombre de fois
(1)	3	3 affectations	1
(2)	2	2 comparaisons	n1+n2≤2n
(3)	1	1 comparaison	n1+n2≤2n
(4)	3	1 addition 2 affectations	n1≤n
(5)	3	1 addition 2 affectations	n2≤n
(6)	2	1 addition 1 affectation	n1+n2≤2n
(7)+(8)	5	2 additions 3 affectations	N ici n=max(n1,n2)

Le nombre total d'opérations élémentaires de la procédure, dans le pire cas, est T(n)=3+4n+2n+3n+3n+4n+5n=3+21n=Θ(n).

Toute fonction polynôme de degré 1 (dont la courbe est donc une droite), qui représente un nombre d'opérations élémentaires d'un algorithme (positive ou nulle pour toute taille n), est en Θ(n).

Exercice 3 (Problèmes d'optimisation)<3 points> :

On considère le problème de décision CHEMIN suivant :

- **Description :** un graphe étiqueté, deux sommets u et v du graphe, et un entier k
- **Question :** existe-t-il un chemin du graphe reliant u à v, de longueur inférieure ou égale à k ?

Expliquez comment utiliser le problème de décision CHEMIN pour résoudre le problème d'optimisation PLUS-COURT-CHEMIN suivant :

- **Description :** un graphe étiqueté et deux sommets u et v du graphe
- **Question :** trouver le plus court chemin entre les sommets u et v

Solution :

Le codage d'une instance de CHEMIN est un quintuplet (n,C,d,a,k) :

- L'instance est constituée d'un graphe orienté étiqueté G=<V,E,c> ($V=\{u_1,\dots,u_n\}$ est l'ensemble des sommets de G, de cardinal n ; E est l'ensemble de ses arcs ; et c est la fonction associant à chaque arc (u_i,u_j) de G son coût $c(u_i,u_j)$)
- C est la matrice d'adjacence et des coûts de G, de taille n*n, définie comme suit

$$C[i,j]=\begin{cases} 0 & si\ i=j \\ c(u_i,u_j) & si\ i\neq j\ et\ (u_i,u_j)\ arc\ de\ G \\ +\infty & si\ i\neq j\ et\ (u_i,u_j)\ n'est\ pas\ arc\ de\ G \end{cases}$$

- d (pour départ) et a (pour arrivée) sont des indices de deux sommets de G, les sommets u_d et u_a
- k est un entier

Soient n_1 le plus petit des coûts des arcs de G, et n_2 la somme des coûts des arcs de G : le coût de tout chemin (élémentaire) entre deux sommets u_i et u_j (en d'autres termes, la somme des coûts de ses arcs) appartient à l'intervalle $[n_1,n_2]$. Si aucun chemin n'existe entre deux sommets, il sera supposé que les deux sommets sont reliés par un chemin de coût infini !

Un certificat pour le problème de décision CHEMIN est une paire c=(T,m), T étant un tableau d'entiers entre 1 et n, tous différents, vérifiant T[1]=d et T[m]=a. Voici un algorithme de validation pour CHEMIN, trivialement polynômial :

```
Booléen validation_chemin(n,C,d,a,k,T,m)
        début
                somme=0
                pour i=1 à m-1 faire
                        si C[i,i+1]=+∞
                                alors retourner FAUX
                                sinon somme=somme + C[i,i+1]
                        fsi
                fait
                si somme≤k
                        alors retourner VRAI
                        sinon retourner FAUX

                fsi
        fin
```

Pour rappel, l'algorithme de validation ci-dessus va valider une instance (n,C,d,a,k) de CHEMIN s'il existe un certificat c=(T,m) tel que validation-chemin(n,C,d,a,k,T,m) retourne VRAI.

Soit donc I=(n,C,d,a,k) une instance du problème CHEMIN. Le quadruplet J=(n,C,d,a) peut être vu comme une instance du problème d'optimisation PLUS-COURT-CHEMIN (auquel est associé le problème de décision CHEMIN). La réponse à la question associée à l'instance J de PLUS-COURT-CHEMIN (trouver le plus court chemin entre les sommets u_d et u_a) peut être trouvée en testant au plus (n2-n1+1) fois si l'algorithme de validation validation_chemin valide une instance de CHEMIN. Ceci peut être fait par une procédure comme celle-ci :

```
plus_court_chemin(n,C,d,a)
        début
                pour k=n1 à n2 faire
                        si validation_chemin vailde l'instance  (n,C,d,a,k) de CHEMIN
                                alors retourner k
                        fsi
                fait
                retourner +∞ (inexistence de chemin entre les sommets ud et ua)
        fin
```

Morale : Le problème d'optimisation PLUS-COURT-CHEMIN n'est pas plus difficile à résoudre que le problème de décision associé CHEMIN, à un facteur polynomial près (le facteur polynomial étant ici (n2-n1+1)).

Exercice 4 (Arbres binaires de recherche)<6 points> :

1. Donnez une structure de données permettant la représentation d'un arbre binaire.
2. Ecrivez un algorithme permettant de fusionner deux arbres binaires de recherche, t1 et t2, de telle sorte que le résultat t de la fusion soit un arbre binaire de recherche.

Solution 1 (je mettrai d'autres solutions prochainement !) :

1)
```
typedef struct arbreb{
  int clef;
  arbreb *sag,*sad;
  };
```

2)
```
inserer(r,x){
    if(r!=NULL){
            pere=r ;
            if(x->clef≤r->clef) inserer(r->sag,x)
            else inserer(r->sad,x)
            }
    else
            if(x->clef≤pere->clef)pere->sag=x
            else pere->sad=x
    }

void fusionner_postfixe(arbreb *t1,arbreb t2){
    if(t2!=NULL){
            fusionner_postfixe(t1,t2->sag);
            fusionner_postfixe(t1,t2->sad);
            t2->sag=NULL ;
            t2->sad=NULL ;
            isererer(t1,t2);
            }
    }
```