Université des Sciences et de la Technologie Houari Boumediène
Faculté d'Electronique et d'Informatique
Département d'Informatique
LMD Master 1ère Année IL 2011/12
Module ''Algorithmique Avancée et Complexité''
Date : 30/01/2012

Corrigé de l'examen de rattrapage
<Corrigé préparé par le responsable du module, Mr ISLI>

Exercice 1 (NP-complétude : 7 points=4+1,5+1,5) :
On considère le problème de décision ISO_GRAPHES suivant :

- **Description :** deux graphes orientés $G_1=(V_1,E_1)$ et $G_2=(V_2,E_2)$ de même nombre n de sommets, avec $V_1=\{X_1,\ldots,X_n\}$ et $V_2=\{Y_1,\ldots,Y_n\}$
- **Question :** existe-t-il un isomorphisme f : $V_1 \rightarrow V_2$ tel que pour tous i,j dans {1,…,n}, (X_i,X_j) est arc de G_1 si et seulement si $(f(X_i),f(X_j))$ est arc de G_2 ?

Le but de l'exercice est de montrer que le problème ISO_GRAPHES appartient à la classe de complexité NP. Pour ce faire, il vous est demandé de trouver un algorithme polynômial de validation pour le problème, que vous appellerez **validation_ig**. Procédez comme suit :

1. Donnez l'algorithme sous forme d'une fonction booléenne dont il est important que vous expliquiez les paramètres (*numérotez les lignes de votre algorithme*).
2. Calculez le nombre d'opérations élémentaires de l'algorithme en fonction d'une taille n à préciser. Appelez ce nombre T(n).
3. Montrez que $T(n)=\Theta(n^k)$, pour une certaine constante k à préciser.

Solution :

1. L'algorithme de validation est comme suit. Il est écrit sous forme d'une fonction booléenne à quatre entrées n, C1, C2 et c. La paire (n,C1,C2) donne le codage de l'instance du problème :
    - L'instance consiste en deux graphes orientés G1=<V1,E1> et G2=<V2,E2>, chacun à n sommets (V1={X[1],…,X[n]}, V2={Y[1],…,Y[n]})
    - C1 est la matrice d'adjacence de G1, de taille n*n, booléenne
        - C1[i][j]=1 si et seulement si (X[i],X[j]) est arc de G1
    - C2 est la matrice d'adjacence de G2, de taille n*n, booléenne
        - C2[i][j]=1 si et seulement si (Y[i],Y[j]) est arc de G2

    L'entrée c est un certificat consistant en un tableau de taille n, dont les éléments sont tous différents et appartiennent à l'ensemble {1,…,n} (le certificat est un tableau de permutation de taille n : le certificat représente un isomorphisme f :V1→V2 ; il doit être interprété comme suit : pour tout i=1..n, f(X[i])=Y[c[i]]). L'algorithme de validation retournera VRAI ssi pour tout i allant de 1 à n, pour tout j=1..n, (X[i],X[j]) est arc de G1 si et seulement si (Y[c[i]],Y[c[j]]) est arc de G2 ; en d'autres termes, ssi pour tous i,j, C1[i][j]=C2[c[i]][c[j]].

    Si un entier est représenté sur p bits, la paire (n,C1,C2) peut être vue comme un mot de $\{0,1\}^*$ de longueur $p*(2n^2+1)$.

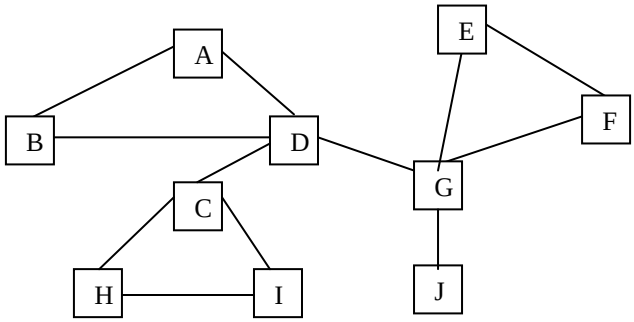
```
Booléen validation_is(n,C1,C2,c)
    début
1.      pour i=1 à n faire
2.          pour j=1 à n faire
3.              Si C1[i][j]≠C2[c[i]][c[j]] alors retourner FAUX finsi
4.          fait
5.      fait
6.      retourner VRAI
    fin
```

2. Deux boucles imbriquées : T(n) est clairement un polynôme de degré 2 en n, le nombre commun de sommets des deux graphes de l'instance : $T(n)=k_2n^2+k_1n+k_0$
3. Tout nombre d'opérations élémentaires polynôme de degré p en n est en $\Theta(n^p)$: donc $T(n)=\Theta(n^2)$.

Exercice 2 <Comparaison d'algorithmes : 3 points> :
Soient A1, A2 et A3 trois algorithmes conçus pour la même tâche T, et dont les nombres d'opérations élémentaires du pire cas sont, respectivement, $f_1(n)=3n^{20}\log(n)$, $f_2(n)=2n^{22}$ et $f_3(n)=90n^{17}+n^{20}$. Comparez les complexités des trois algorithmes. Expliquez.

Solution :
$f_1(n)=\Theta(n^{20}\log(n))$; $f_2(n)=\Theta(n^{22})$; $f_3(n)=\Theta(n^{20})$. Comme) $\Theta(n^{20})<\Theta(n^{20}\log(n))<\Theta(n^{22})$, l'algorithme A3 est meilleur que l'algorithme A1, qui à son tour est meilleur que l'algorithme A2.

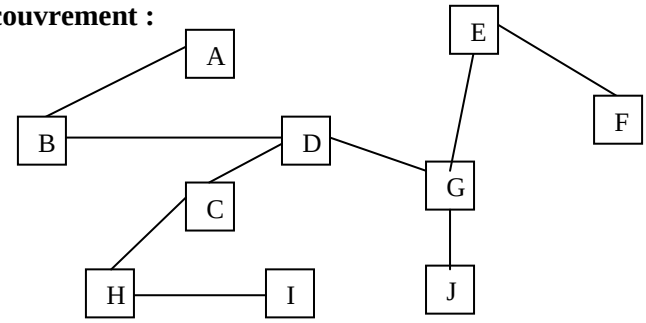
Exercice 3 (parcours en profondeur d'abord : 5 points=2+2+1) :
On considère le graphe suivant :



1. Donnez un arbre de recouvrement issu d'un parcours en profondeur d'abord du graphe
2. Donnez l'ordre dans lequel le parcours considéré a visité les sommets du graphe
3. Déduire des questions précédentes l'arbre de recouvrement ordonné issu du parcours considéré : le parcours en profondeur d'abord de l'arbre ordonné doit visiter les sommets dans l'ordre dans lequel ils l'ont été par le parcours considéré du graphe

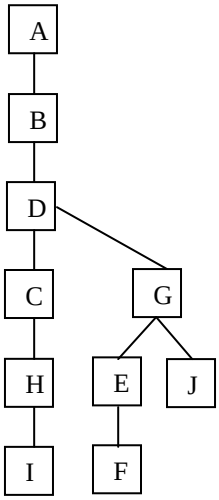
Solution :

1. **Arbre de recouvrement :**



2. **Ordre de visite des sommets :** ABDCHIGEFJ

3. **Arbre de recouvrement ordonné :**



Exercice 4 (dérécursivation : 5 points=2+3) : Dérécursivation du parcours en profondeur d'abord d'un arbre binaire : le cas postfixe

1. Donnez un algorithme récursif de parcours postfixe d'un arbre binaire.
2. Dérécursivation du parcours : donnez un algorithme itératif équivalent à l'algorithme récursif de la question 1.

Solution :

1. **Algorithme récursif :**

```
Postfixe(A)
début
    Si A≠NIL alors
            Postfixe(A.sa-gauche);
            Postfixe(A.sa-droit);
            Afficher(A.valeur);
    finsi
fin
```

2. **Algorithme itératif équivalent :**

Soit A un arbre dont il faut faire le parcours postfixe (A est plus précisément un pointeur sur la racine de l'arbre). L'algorithme utilise une pile P initialisée à vide. Les éléments de la pile sont des enregistrements à trois champs : un champ champ0 de type booléen (prenant ses valeurs dans l'ensemble {''clef'','pointeur''}), un champ champ1 de type pointeur sur un nœud de l'arbre à parcourir, et un champ champ2 de même type que les clefs des différents nœuds de l'arbre. La variable temporaire temp utilisée par l'algorithme est du même type que les éléments de la pile P. Je laisse le soins à vous mes chers étudiantes et étudiants, et aux lecteurs du corrigé en général, de deviner comment les trois champs de temp et des éléments de P sont exploités par l'algorithme (c'est toujours bon de laisser un peu de flou dans un corrigé, un peu de non-dit, un peu de suspense, mais juste suffisamment pour éveiller l'intelligence du lecteur).

```
Postfixe_it(A)
début
    Initialisation d'une pile P à vide
    temp.champ0= ''pointeur''
    temp.champ1=A
    EMPILER(P,temp)
    tant que (non PILE-VIDE(P)) faire
        B=DEPILER(P)
        si B.champ0=''pointeur'' alors
            C=B.champ1
            si C≠NIL alors
                temp.champ0=''clef''
                temp.champ2=C.clef
                EMPILER(P,temp)    //empiler la clef
                temp.champ0=''pointeur''
                temp.champ1=C.sa-droit
                EMPILER(P,temp);   //empiler pointeur sur sous-arbre droit
                temp.champ1=C.sa-gauche
                EMPILER(P,temp);   //empiler pointeur sur sous-arbre gauche
                //on empile la clef
                //puis le sous-arbre droit
                //puis le sous-arbre gauche !
                //dans l'ordre inverse dans lequel la racine (clef), le sous-arbre
                //gauche et le sous-arbre droit seront visités, l'ordre inverse étant
                //(racine,sous-arbre droit,sous-arbre gauche))
            finsi
        sinon Afficher(B.champ2)
        finsi
    fait
fin
```