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''
Date : 16/02/2010

Corrigé de l'examen de rattrapage

Exercice 1 (10 points : <5,2,3>) :

On considère le problème de décision TSP(k) suivant :

- **Description :** un graphe non orienté dont les arêtes sont étiquetées par des entiers positifs ; et un entier k positif
- **Question :** le graphe admet-il un cycle hamiltonien de coût inférieur ou égal à k ?

1. Donnez un algorithme polynomial de validation pour le problème TSP(k) :
   - Expliquez les différents paramètres de l'algorithme
   - Expliquez la polynomialité de l'algorithme

Réponse :

L'algorithme de validation est comme suit. Il est écrit sous forme d'une fonction booléenne à quatre arguments n, C, k et T. Le triplet (n,C,k) donne le codage de l'instance du problème :

- L'instance est constituée d'un graphe non orienté étiqueté G=<V,E,c> (V={u_0,…,u_{n-1}} est l'ensemble des sommets de G, de cardinal n ; E est l'ensemble de ses arêtes ; et c est la fonction associant à chaque arête (u,v) de G son coût c(u,v))
- C est la matrice d'adjacence et des coûts de G, de taille n*n, définie comme suit

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

L'argument T est un certificat consistant en un tableau de taille n, contenant exactement une fois chacun des entiers de 0 à n-1 (tableau de permutation). L'algorithme retourne VRAI si et seulement si le certificat T valide l'instance, c'est-à-dire si et seulement si le n+1-uplet ($u_{T[0]}$,…,$u_{T[n-1]}$,$u_{T[0]}$) est un cycle hamiltonien de G de coût inférieur ou égal a k :

- ($u_{T[i]}$,$u_{T[i+1]}$) est arête de G, pour tout i dans {0,…,n-2}
- ($u_{T[n-1]}$,$u_{T[0]}$) est arête de G
- ($\sum_{i=0}^{n-2} c(u_{T[i]}, u_{T[i+1]})$+c($u_{T[n-1]}$,$u_{T[0]}$)≤k

Si un entier est représenté sur p bits, le triplet (n,C,k) peut être vu comme un mot de $\{0,1\}^*$ de longueur 2p+(p+1)*n^2 : les 2p premiers bits (0 ou 1) coderont le nombre n de sommets de l'instance et l'entier k, les (p+1)*n suivants coderont la 1ère ligne de la matrice C, …, les (p+1)*n derniers bits coderont la toute dernière ligne de la matrice C. L'élément C[i][j] de la matrice C sera codé sur p+1 bits :

- Le premier bit dira si oui ou non (u_i,u_j) est arête de G (1 si oui, 0 sinon)
- Les p bits restants donneront, dans le cas où (u_i,u_j) est arête de G, le coût c(u_i,u_j)

Booléen validation_h(n,C,k,T){

```
somme=0;
i=0;
while(i<n-1){
        if(C[T[i]][T[i+1]]=+∞) alors retourner FAUX
        sinon{
                somme=somme+C[T[i]][T[i+1]] ;
                i=i+1 ;
                }
        }
if(C[T[n-1]][T[0]]=+∞) alors retourner FAUX
sinon{
        somme=somme+C[T[n]][T[0]] ;
        i=i+1 ;
        }
Si (somme≤k) alors retourner VRAI
Sinon retourner FAUX ;
}
```

La boucle de l'algorithme parcourt les (n-1) premiers éléments du certificat T (indicés de 0 à n-2), jusqu'à éventuellement en rencontrer un, T[i], vérifiant C[T[i]][T[i+1]]=+∞ (il n'y a pas d'arête joignant les sommets $u_{T[i]}\ et\ u_{T[i+1]}$), auquel cas l'algorithme retourne FAUX, signifiant que le certificat ne constitue pas un cycle Hamiltonien. A la sortie de la boucle, l'algorithme teste s'il y a arête reliant les sommets $u_{T[n-1]}\ et\ u_{T[0]}$: si non, le certificat ne constitue pas un cycle Hamiltonien ; si oui, le certificat constitue un cycle Hamiltonien et l'algorithme teste alors si son coût est inférieur ou égal à k (si oui, l'algorithme retourne VRAi ; si non, il retourne FAUX).

2. Ecrivez un algorithme donnant, pour toute instance de TSP(k), 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 :

L'algorithme est écrit sous forme d'une fonction booléenne existe_ch à trois arguments n, C et k représentant une instance de TSP(k). L'algorithme retourne OUI si et seulement si l'instance en entrée, représentée par le triplet (n,C,k), admet une solution ; c'est-à-dire si et seulement si le graphe étiqueté donné par la n*n-matrice d'adjacence et des coûts C possède un chemin Hamiltonien de coût inférieur ou égal à k :

```
Booléen existe_ch(n,C,k){
        cond=0 ;
        i=0;
        j=0;
        max=n!;
                //il y a n! permutations possibles de n objets//
        while( !cond && i<max){
                for(k=0 ;k<n ;k++)T[k]=0 ;
                        //Mise à zéro du tableau T//
                dividende=j ;
                k=n-1 ;
```

```
while(dividende !=0){
        T[k]=dividende%n ;
        dividende=dividende/n ;
        j-- ;
        }
        //écriture de i dans le système de numération de base n (n chiffres, 0,…,n-1)//
        //résultat dans le tableau T//
    if(T tableau de permutation){
                //un certificat est un tableau de permutation//
        i++ ;
                //i compte le nombre de certificats  déjà passés en revue//
        if(validation_h(n,C,k,T))cond=1 ;
                //si le certificat valide l'instance, on ne parcourt pas les certificats restants//
        }
    j++ ;
    }
if(cond)printf("oui");
else printf("non");
}
```

3. Expliquez comment utiliser le problème de décision TSP(k) pour résoudre le problème d'optimisation TSP du voyageur de commerce :
    - **Description :** un graphe non orienté dont les arêtes sont étiquetées par des entiers positifs
    - **Question :** quel est le coût du cycle hamiltonien minimal du graphe ?

Le problème TSP est-il, à un facteur polynomial près, plus difficile à résoudre que le problème TSP(k) ?

Réponse :

Le codage d'une instance de TSP est une paire (n,C) :

- L'instance est constituée d'un graphe non orienté étiqueté G=<V,E,c> ($V=\{u_0,\ldots,u_{n-1}\}$ est l'ensemble des sommets de G, de cardinal n ; E est l'ensemble de ses arêtes ; et c est la fonction associant à chaque arête (u,v) de G son coût c(u,v))
- C est la matrice d'adjacence et des coûts de G, de taille n*n, définie comme suit

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

Tout cycle Hamiltonien de l'instance G contient n arêtes. Soient n_1 le plus grand des coûts des arêtes de G, et n_2 la somme des coûts des arêtes de G : le coût de tout chemin Hamiltonien (en d'autres termes, la somme des coûts de ses arêtes) est inférieur ou égal à $\min(n*n_1, n_2)$. Donc pour résoudre une instance de TSP de codage (n,C), il suffit de résoudre successivement les instances (n,C,0), (n,C,1), …, (n,C, min(n*n1, n2)) des problèmes TSP(0), TSP(1), …, TSP(min(n*n1, n2)), respectivement. Si la solution (booléenne) de chacune de ces instances est NON alors l'instance (n,C) de TSP n'admet aucun chemin Hamiltonien. Sinon, soit k le plus petit entier tel que l'instance (n,C,k) admet OUI comme solution : l'instance (n,C) de TSP admet au moins un chemin Hamiltonien ; de plus, le coût du chemin Hamiltonien minimal est k.

La résolution de l'instance (n,C) TSP peut ainsi être faite par l'appel de la résolution de TSP(k) un nombre de fois borné par min(n*n1, n2).

Exercice 2 (6 points) :

Donnez un algorithme linéaire permettant de tester si un tableau d'entiers de taille n est un tableau de permutation (c'est-à-dire dont tous les éléments sont distincts et compris entre 1 et n). Expliquez la linéarité de l'algorithme.

Indication : utilisez un tableau auxiliaire.

Réponse :

Soit T le tableau d'entiers de taille n dont on veut savoir s'il est un tableau de permutation. On utilise un tableau auxiliaire S de même type et de même taille.

```
Booléen tab_de_perm(T){
        pour i=1 à n faire S[i]=0 fait
                //On suppose que les éléments du tableau T sont indicés de 1 à n.//
                //Il faut montrer que les éléments de T sont deux à deux distincts,//
                //et compris entre 1 et n//
        i=1 ;
        tant que(i≤n){
                si(T[i]≥1 et T[i]≤n) alors
                        si(S[T[i]]=T[i]) alors retourner FAUX
                                //L'élément T[i] est compris entre 1 et n mais occurre au//
                                //moins deux fois dans T : T n'est donc pas un tableau//
                                //de permutation//
                        sinon{
                                S[T[i]]=T[i] ;
                                i=i+1 ;
                                }
                                // L'élément T[i] est compris entre 1 et n et on est à sa première//
                                //rencontre : on mémorise T[i] dans S[T[i]] et on avance//
                sinon retourner FAUX ;
```

//L'élément T[i] n'est pas compris entre 1 et n : T n'est donc pas un//
//tableau de permutation//

}
retourner VRAI ;
}

Exercice 3 (4 points) :

Ecrivez des algorithmes de recherche d'un élément, d'insertion d'un élément et de suppression d'un élément dans une liste simplement chaînée. On supposera que les éléments de la liste sont de type enregistrement à deux champs :

- un champ clé de type entier
- un champ successeur de type pointeur sur un élément de la liste

Réponse :

RECHERCHE-LISTE(L,k){
x=TETE(L);
tant que(x!=NIL et *clé*(x)!=k)x=*successeur*(x);
retourner x;
//si x vaut NIL, la valeur k n'est pas présente dans la liste. Sinon, le champ clé//
//de l'élément de la liste pointé par x vaut k//
}
INSERTION-LISTE(L,x){
successeur(x)=TETE(L);
TETE(L)=x;
}
SUPPRESSION-LISTE(L,x){
Si x=TETE(L) **alors** TETE(L)=successeur(x)
sinon{
y=TETE(L);
tant que successeur(y)<>x **faire** y=successeur(y);
successeur(y)=successeur(x);
}
}