Université des Sciences et de la Technologie Houari Boumediène
Faculté d'Electronique et d'Informatique
Département d'Informatique
LMD Master1 RSD 2009/2010
Module ''Algorithmique Avancée et Complexité''

TP 1
Illustration du Non-déterminisme Polynomial

Le but du TP est de montrer à l'étudiant, à travers un programme résolvant le problème SAT, ce qu'est un algorithme non-déterministe polynomial.

Le problème SAT (satisfiabilité) :

Une proposition atomique est une variable booléenne, c'est-à-dire prenant ses valeurs dans l'ensemble BOOL={VRAI,FAUX}. Un littéral est une proposition atomique ou la négation d'une proposition atomique. Une proposition atomique est aussi appelée littéral positif ; et la négation d'une proposition atomique littéral négatif. Une clause est une disjonction de littéraux.

Etant données m propositions atomiques $p_1, \ldots, p_m$, une instanciation du m-uplet $(p_1, \ldots, p_m)$ est un élément de $\{VRAI,FAUX\}^m$. Une instanciation $(e_1, \ldots, e_m)$ de $(p_1, \ldots, p_m)$ satisfait une clause c (noté $(e_1, \ldots, e_m) \models c$) si et seulement si l'une des conditions suivantes est satisfaite :

1. il existe $i \in \{1,\ldots,m\}$ tel que (e_i=VRAI) et (p_i occurre dans c)
2. il existe $i \in \{1,\ldots,m\}$ tel que (e_i=FAUX) et ($\neg p_i$ occurre dans c)

Une instanciation satisfait une conjonction de clauses si et seulement si elle satisfait chacune de ses clauses. Une conjonction de clauses est satisfiable si et seulement si il existe une instanciation la satisfaisant. Une instanciation satisfaisant une conjonction est dite solution ou modèle de la conjonction.

Le problème SAT est maintenant défini comme suit :

Entrée : une conjonction C de n clauses construites à l'aide de m propositions atomiques $p_1,\ldots,p_m$

Sortie : la conjonction C est-elle satisfiable ?

Version 1 : SAT comme problème de décision (oui/non)

début
Cond=FAUX
i=0
Tant que (non Cond) **et** ($i<=2^m-1$){

Si (i solution de C) **alors**{
Imprimer « OUI »
Cond=VRAI
}
i++
}
Si (non Cond) **alors** imprimer « NON »
Fin

Version 2 : impression d'une solution si satisfiabilité

début
Cond=FAUX
i=0
Tant que (non Cond) **et** (i<=2^m-1){
    Si (i solution de C) **alors**{
                Imprimer i
                Cond=VRAI
                }
    i++
    }
Si (non Cond) **alors** imprimer « AUCUNE SOLUTION »
fin

Version 3 : impression de toutes les solutions

début
Cond=FAUX
i=0
Tant que (i<=2^m-1){
    Si (i solution de C) **alors**{
                Imprimer i
                Cond=VRAI
                }
    i++
    }
Si (non Cond) **alors** imprimer « AUCUNE SOLUTION »
Fin

Programme C pour les versions 1 et 2

```
main(){
int C[100][100],inst[100],
    n,m,i,j,k,dividende,c_satisfaite,i_solution,max ;
printf(« saisie du nombre de clauses : ») ;
scanf(« %d\n »,&n) ;
printf(« Saisie du nombre de propositions atomiques : ») ;
scanf(« %d\n »,&m) ;
for(i=0 ;i<n ;i++){
    printf("Saisie de la clause %d : »,i) ;
    for(j=0 ;j<m;j++){
        printf(« C[%d][%d]= ? ») ;
        scanf(« %d\ n»,&C[i][j]) ;
        }
    }
    i=0;
cond=0 ;
max=2^m;
while( !cond && i<max){
    //Mise à zéro du tableau inst//
    for(j=0 ;j<m ;j++)inst[j]=0 ;
    dividende=i ;
    j=m-1 ;
    while(dividende !=0){
        inst[j]=dividende%2 ;
        dividende=dividende/2 ;
        j-- ;
        }
    i_solution=1 ;
    k=0 ;
    while(i_solution && 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)i_solution=0 ;
        k++ ;
        }
    cond=i_solution ;
    i++;
    }
```

//VERSION 1

```
if(cond)printf("oui");
else printf("non");
```

//VERSION 2

```
if(cond){
    printf("L'instance du problème SAT est satisfiable par l'instanciation ");
    for(j=0;j<m;j++)
        if(inst[j]==0)prinf("F");
        else printf("V");
    printf("qui en est donc solution");
    }
```

else printf("L'instance du problème SAT n'est pas satisfiable");
}

Un algorithme pour la version 3 s'en déduit facilement …

Discussion

Le programme, dans ses versions 1 et 2, procède par énumération des différentes instanciations possibles ($e_1,\ldots,e_m$) du m-uplet ($p_1,\ldots,p_m$) de propositions atomiques. Quand il y a satisfiabilité, l'énumération s'arrête à la toute première rencontre d'une instanciation satisfaisant la conjonction de clauses : il y a alors sortie de la boucle la plus externe et impression du résultat. S'il n'y a pas satisfiabilité, l'énumération des 2^m instanciations possibles et vérification, pour chacune d'entre elles, qu'elle ne satisfait pas toutes les clauses de la conjonction, sont faites avant de se rendre compte qu'il n'y a pas satisfiabilité. Pour une instanciation donnée, représentée par le tableau *inst*, le programme parcourt, dans le pire des cas, chacune des lignes de la matrice C pour tester si la clause qu'elle représente est satisfaite par l'instanciation : le coût d'un tel test est en O(n*m). Le programme est donc non-déterministe polynomial, de complexité O(n*m). Cela implique deux choses :

- si on est suffisamment chanceux pour 'tomber' sur la bonne instanciation dès le départ, on répond (positivement) à la question d'existence d'une solution, et on imprime une telle solution, au bout d'un temps polynomial (en la taille de l'instance en entrée), en O(n*m) plus précisément
- si, par contre, l'instance n'a pas de solution, ou en a une seule consistant en la toute dernière dans l'énumération, le programme entrera dans la boucle la plus externe 2^m fois dont le coût de chacune est en O(n*m) : le coût du programme sur de telles instances est exponentiel