TP - La classe NP

Objectif: Le but du TP est de "concrétiser" les notions de propriété NP et de réduction polynomiale vues en cours.

A faire: Le TP a deux sections, une sur la notion de NP, un sur celle de réduction polynomiale. Le TP est à rendre la semaine du 21 novembre sous Prof sous forme d"une archive contenant un rapport (bref) avec les réponses aux questions et vos choix d'implémentation et le code. Vous disposez des exemples de données pour tester.

1 Qu'est-ce qu'une propriété NP?

On va illustrer la notion de propriété NP via le problème de la mise en sachets. Le problème est de placer un certain nombre d'objets dans un certain nombre de sacs. Les objets sont de poids divers, chaque sac acceptant une charge maximale. La propriété -ou problème de décision- $BINPACK$ associée est définie formellement comme suit:

Donnée:
n –un nombre d'objets
$x_1, \cdots, x_n$ – n entiers, les poids des objets
c –la capacité d'un sac (entière)
k –le nombre de sacs
Sortie:
Oui, si il existe une mise en sachets possibles, i.e.:
$aff : [1..n] \rightarrow [1..k]$ –à chaque objet, on attribue un numéro de sac
tel que:
$\sum_{i/aff(i)=j} x_i \leq c$, pour tout numéro de sac j, $1 \leq j \leq k$. – aucun sac n'est trop chargé
Non, sinon

Par exemple, soit $n = 5$, $x_1 = 3, x_2 = 2, x_3 = 4, x_4 = 3, x_5 = 3$; si $c = 5$, il y a une solution pour $k = 4$ mais pas pour $k = 3$; si $c = 4$, il faut au moins $k = 5$ pour avoir une solution.

A faire avant d'implémenter!

Q 1. La propriété est NP.

> **NP**
>
> L est dit NP si il existe un polynôme Q, et un algorithme polynômial à deux entrées et à valeurs booléennes tels que:
> $$L = \{u / \exists c, A(c, u) = Vrai, |c| \leq Q(|u|)\}$$

Définir une notion de certificat.

Comment pensez-vous l'implémenter?

Quelle sera la taille d'un certificat? La taille des certificats est-elle bien bornée polynomialement par rapport à la taille de l'entrée?

Proposez un algorithme de vérification associé. Est-il bien polynomial?

Q 2. NP= Non déterministe polynomial

Quel serait le schéma d'un algorithme non-déterministe polynomial pour le problème?

Q 3. $NP \subset ExpTime$

Pour vérifier si le problème a une solution, on va donc énumérer tous les certificats.

Q 3.1. Pour k et n fixés, combien de valeurs peut-prendre un certificat?

Q 3.2. Enumération de tous les certificats

Pour énumérer les certificats (soit définir un itérateur sur les certificats), on va s'appuyer sur un ordre total sur les certificats associés à un problème. Donc, on définit le certificat de départ "le plus petit" pour l'ordre, et la notion de successeur qui permet de passer d'un certificat au suivant.

Quel ordre proposez-vous?

Q 3.3. L'algorithme du British Museum

Comment déduire de ce qui précède un algorithme pour tester si le problème a une solution? Quelle complexité a cet algorithme?

Implémentation

On veut donc implémenter les notions et algorithmes évoqués ci-dessus. On devra donc être capable de lire une instance d'un problème de BinPack, lire une proposition de certificat, vérifier si un certificat est valide, vérifier si le problème a une solution en essayant tous les certificats, "vérifier aléatoirement" si le problème a une solution.

A la fin du sujet est proposé un embryon d'architecture en Java. **Vous pouvez en choisir une autre ou utiliser d'autres langages (C, CAML)**. Vous veillerez cependant à documenter votre code, à respecter l'esprit et à faciliter le test en respectant le schéma ci-dessous.

Pour tester, on pourra donc avoir un programme qui lit l'instance du problème dans un fichier et :

. en mode "vérification" propose à l'utilisateur de saisir un certificat et vérifie sa validité.

. en mode "non-déterministe", génère aléatoirement un certificat, le teste et retourne Faux si il n'est pas valide, "Vrai" sinon (avec éventuellement la valeur du certificat).

. en mode "exploration exhaustive" génère tous les certificats jusqu'à en trouver un valide, si il en existe un et retourne Faux si il n'en existe pas de valide -la propriété n'est donc pas vérifiée- , "Vrai" sinon (avec éventuellement la valeur du certificat trouvé).

Par exemple en java, l'usage sera : java solve <files> <mode> avec comme modes (au moins) -v (verif), -nd (non déterministe), -exh (exhaustif).

Attention! Ne pas l'utiliser sur des problèmes de grande taille! Vous avez à votre disposition des exemples.

2 Réductions polynomiales

On va maintenant illustrer la notion de réduction polynomiales d'une propriété à l'autre.

Q 4. Une première réduction très simple

Soit le problème de décision $Partition$ défini par:

Donnée:

n –un nombre d'entiers

$x_1, \cdots, x_n$ –les entiers

Sortie: Oui, si il existe un sous-ensemble de $[1..n]$ tel que la somme des x_i correspondants soit exactement la moitié de la somme des x_i, i.e. $J \subset [1..n]$, tel que $\sum_{i \in J} x_i = \sum_{i \notin J} x_i = \frac{\sum_{i=1}^{n} x_i}{2}$

Q 4.1. Montrer que $Partition$ se réduit polynomialement en $BINPACK$.

Q 4.2. $Partition$ est connue $NP-$dure. Qu'en déduire pour $BinPack$?

Q 4.3. Implémenter cette réduction et utilisez-là pour vérifier si une instance de $Partition$ est positive ou négative. Vous disposez d'exemples d'instances de Partition pour tester.

On définira donc la classe/notion de `PblPartition` et une méthode de complexité polynomiale qui associera à une instance de `PblPartition` une instance équivalente (i.e. positive Ssi la première l'était) de $BinPack$. On utilisera cette méthode pour coder `aUneSolution` pour `PblPartition`. Par exemple en Java:

```
//la classe des instances de Partition
public class PblPartition extends PblDec{
...
//associe à l'instance de PblPartition une instance équivalente de PblBinPAck
//doit être de complexité polynomiale
public PblBinPack redPolyTo(){
// A définir       ...   }
public boolean aUneSolution() {
 // à définir à partir de  redPolyTo() et aUneSolution() de PblBinPack. }
}
```

Q 5. Une deuxième réduction

Soit maintenant le problème *Sum* défini par:

Donnée:

n –un nombre d'entiers

$x_1, \cdots, x_n$ –les entiers

c –un entier cible

Sortie: Oui, si il existe un sous-ensemble de $[1..n]$ tel que la somme des x_i correspondants soit exactement c, i.e. $J \subset [1..n]$, tel que $\sum_{i \in J} x_i = c$

Q 5.1. Entre *Sum* et *Partition* lequel des deux problèmes peut être presque vu comme un cas particulier de l'autre? Qu'en déduire en terme de réduction?

Q 5.2. Montrer que *Sum* se réduit en *Partition* et implémenter la réduction.

Q 6. Composition de réductions

En utilisant les deux réductions précédentes, implémenter une réduction de *Sum* dans *BinPack*?

Q 7.*Question bonus: deux réductions un peu plus dures*

Q 7.1. Soit le problème de mise en sachets où les sacs ont potentiellement des capacités différentes, soit $BinPackDiff$:

Donnée:

n –un nombre d'objets

$x_1, \cdots, x_n$ – n entiers, les poids des objets

k –le nombre de sacs

$c_1, \cdots, c_k$ –les capacités des sacs (entières)

Sortie:

Oui, si il existe une mise en sachets possibles, i.e.:

$aff : [1..n] \to [1..k]$ –à chaque objet, on attribue un numéro de sac

tel que:

$\sum_{i/aff(i)=j} x_i \leq c_j$, pour tout numéro de sac j, $1 \leq j \leq k$. – aucun sac n'est trop chargé

Non, sinon

Bien sûr, $BinPack$ est un cas particulier de $BinPackDiff$.

Montrez que $BinPackDiff$ se réduit polynomialement en$BinPackDiff$ (il n'est pas demandé d'implémenter).

Q 7.2.

Soit le problème de décision *KnapSackExact* :

Donnée:

n –un nb d'objets

$p_1, \cdots, p_n$ – n entiers, les poids des objets

$v_1, \cdots, v_n$ – n entiers, les poids des objets

p –une cible poids

v –une cible valeur

Sortie: Oui, si il existe un remplissage complet du sac qui a pour valeur exactement v, soit:

$choix : [1..n] \to [0..1]$ –à chaque objet, on attribue 0 si on ne le prend pas, 1 sinon

tq:

$\sum_{i/choix(i)=1} p_i = p$, $\sum_{i/choix(i)=1} v_i = v$,

Proposer une réduction de *KnapSackEqu* dans *Sum* (il n'est pas demandé d'implémenter).

```
/*****************************************************************************************************
              EXEMPLE D'ARCHITECTURE DE CODE POUR LA PARTIE 1-  CE N'EST QU'UNE PROPOSITION
*****************************************************************************************************/
//la classe abstraite des problèmes de décision...
abstract  class PblDec {
 public PblDec(){}
}

//la classe des problèmes BinPack
...class PblBinPack extends PblDec{
    ... int nb_objets; //nb d'objets
    ... int poids[];  //poids des objets
    ... int cap;   //capacité du sac
    ...  int nb_sacs;   //nb de sacs

    public Pbl_Bin_Pack(int n, int p[], int c, int nbs ){
    //juste le constructeur A ECRIRE
    }

   //Ecrivez Les accesseurs que vous souhaitez!
    ...
    //retourne Vrai ssi il existe une mise en sachets possible i.e. si l'instance du pb est positive
    public boolean aUneSolution() {
    //fonctionnera par recherche exhaustive
    // A ECRIRE
    }

    //teste au hasard une mise en sachets et retourne Vrai si elle est valide
    //chaque mise en sachets doit pouvoir être générée par une exécution
    public boolean aUneSolutionNonDéterministe() {
    // A ECRIRE
    }
}

interface Certificat{
   //retourne True SSi le certificat est valide pour le problème
   //doit être de complexité polynomiale par rapport à la taille du certificat et du problème
   public boolean correct();  //algo de vérification A de la définition NP du cours!

   // pour l'énumération on utilisera un ordre total sur les certificats
   //par ex. le constructeur initialisera au plus petit  élément
   //transforme le certificat en son successeur pour l'ordre
    public void suivant();
   //retourne True Ssi le certificat n'a pas de successeur pour l'ordre
   public boolean estDernier();

   //modifie aléatoirement la  valeur du certificat
   //chaque valeur doit pouvoir être générée par au moins une exécution
   public void alea();

   //affiche un certificat
   public void affiche();

  /*FACULTATIFS*/
```

```
  //réinitialise le certificat au plus petit pour l'ordre
  public void reset();

  //permet la saisie d'un certificat
  public void saisie();
}

//la notion de certificat pour le problème Bin_Pack
... class  CertificatBinPack implements Certificat{
private PblBinPack pb; //le problème associé au certificat
//A compléter
}
```