Université Lille 1 *IEEA*

Rapport de Travaux Pratiques

Algorithmes et ComplexiTé (ACT)
TP4 : La classe NP

Auteur :

Salla DIAGNE
Anis TELLO

Professeur :

Sophie TISON

12 Novembre 2014

Listing des dossiers et fichiers

donnees/ : contient les données de test

certificat_bin_pack.ml : fichier d'implémentation du module Certificat_bin_pack

certificat_bin_pack.mli : fichier d'interface du module Certificat_bin_pack

Makefile

probleme_bin_pack.ml : fichier d'implémentation du module Probleme_bin_pack

probleme_bin_pack.mli : fichier d'interface du module Probleme_bin_pack

probleme_partition.ml : fichier d'implémentation du module Probleme_partition

probleme_partition.mli : fichier d'interface du module Probleme_partition

probleme_sum.ml : fichier d'implémentation du module Probleme_sum

probleme_sum.mli : fichier d'interface du module Probleme_sum

solve_bin_pack.ml : fichier contenant le programme de résolution du problème de binpacking

solve_partition.ml : fichier contenant le programme de résolution du problème de partitionnement

solve_sum.ml : fichier contenant le programme de résolution du problème sum

Mode opératoire

Techniquement, il n'y a que 4 lettres à taper pour compiler le projet, le Makefile se charge du reste. Plus précisément :

- ✶ se placer à la racine du projet (dossier tp4_diagne-tello) et exécuter la commande `make` pour générer les fichiers `cmi, cmx, o` et exécutables du projet. 3 exécutables sont créés :

  ⇛ `solve_bin_pack` : ./solve_bin_pack <fichier> (-v | -nd | -exh) lit l'instance du problème de bin-packing dans fichier et recherche un certificat par le ou les modes choisis. Une solution est affichée, s'il en existe.
  -v = mode vérification : l'utilisateur entre un certificat, et le programme vérifie que

c'est une solution au problème
-nd = mode non déterministe : un certificat aléatoire est générée et passé au programme
-exh = mode exhaustif : tous les certificats possibles sont passés au programme. Le premier qui satisfait le problème est affiché.

⇛ `solve_partition` : ./solve_partition <fichier> : lit l'instance du problème de partitionnement dans fichier puis le réduit en problème de bin-packing avec 2 sacs et affiche un certificat si la somme des entiers du sac 0 est égale à celle des entiers du sac 1.

⇛ `solve_sum` : ./solve_sum <fichier> lit l'instance du problème sum puis le réduit en problème de partitionnement.

✶ tester.

Réponses aux questions

1 Qu'est-ce qu'une propriété NP

Q1. La propriété est NP

Définir une notion de certificat

Un certificat est une information que l'on ajoute à une donnée du problème que la réponse au problème pour cette donnée est positive ou négative.
Dans ce problème, le certificat pourrait être un tableau de n entiers, qui pour chaque `i` appartenant à `[1..n]` associe un numéro de sac `j` appartenant à `[1..k]`.
La taille du certificat est donc `n`.

Comment penser vous l'implémenter ?

Nous pensons l'implémenter sous la forme d'un tableau à une dimension dont les indices correspondront aux objets et les valeurs aux numéros de sac.

Quelle sera la taille d'un certificat ?

La taille d'un certificat est le nombre d'objets * la taille de la représentation binaire d'un objet <=> $\boxed{T \simeq n * (logn)}$

La taille des certificats est-elle bien bornée polynômialement par rapport à la taille de l'entrée ?

Oui, la taille des certificats bornée polynômialement par rapport à la taille de l'entrée.

Proposez un algorithme de vérification associé

```
boolean verifieCertificat(Certificat c, Probleme u) {
     int[] capacitesSacs = new int[u.k];
     pour i de 0 à u.n - 1
          int nbSac = cert[i];
          capacitesSac[nbSac] = u.p[i];
          si (capacitesSac[nbSac] > u.c) alors
               retourner faux;
          fin si
     fin pour
     retourner vrai;
}
```

Est-il bien polynômial ?

Oui, il est polynômial car sa complexité est de l'ordre de `O(n)`.

Q2. NP = Non-déterministe Polynômial

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

Une méthode non-déterministe consisterait à choisir aléatoirement, pour chaque objet, le numéro de sac dans lequel le mettre.

```
boolean aUneSolutionNonDeterministe(int n, int[] p, int c, int k) {
     int[n] objToSacs;
     pour i de 0 à n - 1 faire
          objToSacs[i] = rand(k);
     fin pour
```

```
    U = (n, p, c, k);
    C = objToSacs;
    A(C, U);
}
```

Q3. NP $\subset$ ExpTime

Q3.1.

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

Pour `k` et `n` fixés, un certificat peut prendre k^n valeurs.

Q3.2 Énumération de tous les certificats

Quel ordre proposez-vous ?

Le premier certificat serait celui pour lequel tous les objets sont dans le premier sac. Puis on incrémentera le numéro de sac du dernier objet, comme pour une énumération de nombres binaires.
Exemple : Pour `n = 3` et `k = 2`, on aura :

```
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
```

Q3.3 L'algorithme du British Museum

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

Il est plus facile de mettre en place un itérateur qui transforme le certificat passé en paramètre en son suivant (l'algorithme a aussi besoin de k pour savoir s'arrêter au (k - 1)e sac et passer à l'indice suivant).

```
void certificatSuivant(Certificat certificat, int k) {
     int[] donnees = certificat.donnees;
     int indice = donnees.length - 1 et fait == false;
     tant que (indice >= 0 et !fait) faire
          si (donnees[indice] < k - 1) alors
               donnees[indice]++;
               fait = true;
          sinon
               donnees[indice] = 0;
               indice--;
          fin si
     fin tant que
     si (indice < 0) alors
          exception Pas_de_certificat_suivant;
     fin si
}
```

L'algorithme déterministe se présenterait comme suit :

```
boolean aUneSolutionDeterministe(Probleme probleme) {
     int n = probleme.nbre_objets;
     int k = probleme.nbre_sacs;
     int[] donnees = new int[n];
     certificat = { donnees = donnees };
     trouve = false;
     try {
          tant que (!trouve) faire
               certificat_suivant certificat k;
               trouve = verifie_certificat certificat probleme;
          fin tant que
          si (trouve) alors
               affiche_certificat
          fin si
     }    catch (ArrayIndexOfBoundsException) {
          printf "Pas de solution";
     }
```

```
}
```

Quelle complexité a cet algorithme ?

Cet algorithme est de complexité `O(n * k)`.

2 Réductions polynômiales

Q4.

BinPack est défini comme ceci :

Donnée:

n - un nombre d'objets

$x_1, ..., x_n$ - n entiers, les poids des objets

c - la capacité d'un sac

k - le nombre de sacs

Sortie:

Oui, s'il existe une mise en sachets possible, i.e.:

aff : $[1..n] \to [1..k]$ - à chaque objet, on attribut 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

Partition est défini comme ceci :

Donnée:

n - un nombre d'entiers

$x_1, ..., x_n$ - les entiers

Sortie:

Oui, s'il existe un sous-ensemble $[1..n]$ 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

$\sum_{i \in J} x_i = \sum_{i \notin J} x_i = \frac{\sum_{i=1}^{n} x_i}{2}$

Q4.1.

Montrer que Partition se réduit polynômialement en BinPack

Partition peut se réduire polynômialement en BinPack en prenant 2 sacs de capacité $c = \frac{\sum_{i=1}^{n} x_i}{2}$, le tableau des entiers de *Partition* correspondant au tableau des poids de *BinPack*.
De ce fait, une instance de *Partition* est positive si la somme des entiers du sac 0 = la somme des entiers du sac 1 = somme des entiers de l'instance / 2.

BinPack	*Partition*
`nbre\_objets`	`nbre\_entiers`
`poids[]`	`entiers[]`
`capacite`	$\sum_{i=1}^{n} x_i/2$
`nbre_sacs`	2

Q4.2.

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

BinPack est NP-complet.

Q5.

Sum est défini comme ceci :

Donnée:

n - un nombre d'entiers

$x_1, ..., x_n$ - les entiers

c - un entier cible

Sortie:

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

Non sinon

Q5.1

Entre Sum et Partition lequel des deux problèmes peut être presque vu comme un cas particulier de l'autre ?

Partition peut être vu comme un cas particulier de *Sum*.

Q5.2

Montrer que Sum se réduit en Partition

Nous savons que *Partition* est un cas particulier de *Sum*, la valeur de la cible de étant la somme des entiers de l'instance. Pour réduire *Sum* en *Partition*, il suffit de "changer" la somme de ces entiers. C'est-à-dire, rajouter au tableau d'entiers de l'instance de *Partition* la valeur absolue de $(2 * cible) - sommedesentiers$.
Appelons *oldsigma* l'ancienne somme des entiers et *newsigma* la nouvelle. De ce fait, la nouvelle somme des entiers est :

$$\begin{aligned} newsigma &= oldsigma + abs((2 * cible) - oldsigma) \\ &= oldsigma + (2 * cible) - oldsigma \\ &= 2 * cible \end{aligned}$$

L'instance de *Partition* obtenue aura alors pour objectif de partitionner son tableau d'entiers en 2 parties, la somme de chaque partie étant $(2 * cible)/2 = cible$. L'ensemble recherché sera le premier ensemble de l'instance de *Partition* (on retire l'entier ajouté s'il s'y trouve).

Partition	*Sum*
`nbre\_entiers`	`nbre\_entiers`
`entiers[]::(2 * cible - oldsigma)`	`entiers[]`

Q7. Question bonus : deux réductions un peu plus dures

Q7.1

Montrer que BinPackDiff se réduit polynômialement en BinPack

La seule différence entre les problèmes *BinPackDiff* et *BinPack* est que la capacité des sacs du premier est variable, alors que les sacs du deuxième sont de capacité fixe. Pour transformer le premier en le deuxième, il suffit donc d'équilibrer les capacités. C'est-à-dire, choisir comme capacité fixe le ou les sacs avec la plus grande capacité, et rajouter des objets dans les sacs de plus faibles capacité. Un exemple permet de mieux comprendre. Prenons celui du fichier `donnees/data_reduction/exBinPackDiff4` :
Nous avons 8 objets de poids de poids respectifs : 2, 6, 5, 2, 5, 5, 2, 8 et 5 sacs de capacités respectives : 8, 8, 8, 7, 6
L'instance de *BinPackDiff*, que nous appellerons `BPD` construite est :
nbre_objets = 8

poids = [|2;6;5;2;5;5;2;8|]
capacites = [|8;8;8;5;6|]
nbre_sacs = 5

L'instance de *BinPack* (`BP`) que l'on aura après réduction sera :
nbre_objets = 8
poids = ?
capacites = ?
nbre_sacs = 5

Comme nous l'avions dit plus haut, il faut rajouter des objets dans les sacs de plus faible capacité pour équilibrer la capacité fixe et ne pas fausser le problème. Rappelons que les capacités du problème `BPD`