TP - La classe NP

1) Qu'est-ce qu'une propriété NP ?

Question :

Définir une notion de certificat pour le problème

Le certificat c , représente un ensemble de n job comportant pour chaque job une date de début qu'on nommera d et une machine sur laquelle elle s'exécute qu'on nommera m où m < nombre de machine totale.

Comment pensez-vous l'implémenter ?

- On crée une liste de n jobs.
- Chaque job possède 2 attributs :
  - d : un entier date de début
  - m : un entier numéro de machine

Quelle sera la taille d'un certificat ?

|c|, taille du certificat : La taille du certificat dépend du codage de la donnée.

|c| = n * |taille de donnée d| + n* |taille de la donnée m|
=> n * log2(x) + n* log2(x)
=> 2n*log2(x)

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

La taille du certificat ne dépasse pas le nombre de tâches passées en entrée. Il est borné par le nombre de tâche n. On peut donc dire qu'il est borné polynomiquement.

Proposez un algorithme de vérification associé.

```
//Pour chaque job contenu dans jobs
        //Pour chaque job2 contenu dans jobs
                // Si job1 != job2 && job1 et job2 ont la même machine
                // Si la date de début de job1 appartient à l'intervalle d'exécution de job2 et vice versa
                        //return false;
                // Si la date de début est supérieur de la date de début d'origine du job+ du délais d'attente max du job: return false

//return true.
```

Est-il polynomial ?

La complexité de cet algorithme est en O(n 2) il est donc polynomial.

Q2) *Génération aléatoire d'un certificat*
Algorithme génération de certificat :

```
D = durée d'attente max
m = nombre de machine
jobs= liste de jobs
for job in jobs
        job.machine = random(0,m)
        // il peut y avoir un temps de décalage entre l'arrivée
        // et le lancement de la tâche de max D
        job.arrive += random(0,D+1)
```

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

Ce serait donc l'algorithme de la génération aléatoire d'un certificat associé à l'algorithme qui vérifie si la taille du certificat est bornée polynomialement par rapport à la taille d'entrée.

3.1 Pour une instance donnée, pouvez-vous donner un ordre de grandeur du nombre de certificats ?

$$m^{(n/2)} * D^{(n/2)}$$

Il est exponentiel.

3.2. Enumération de tous les certificats ou l'algorithme du British Museum

une liste de certificat, pour chaque certificat on vérifie si il est correct alors return true sinon on passe au suivant

```
//pour chaque certificat dans la liste des certificats trié par ordre croissant :
        // vérifier certificat
        //si certificat est valide
                //return true
        //itération certificat suivant
//Quand il n'y a plus d'itération suivante
//return false
```

On parcourt toutes les conditions de certificats possible à partir de l'élément le plus petit jusqu'à l'élément suivant.

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

2) Réduction polynomiale

Q 2. Montrer que Partition se réduit polynomialement en JSP.

Pour cela on cherche une fonction de réduction permettant de résoudre le problème Partition en un problème JSP.

Pour cela il faut transformer les données du problème pour qu' elles soit utilisable par JSP.

Donnée : n –un nombre d'entiers
x1, · · · , xn –les entiers
Sortie : Oui, si il existe un sous-ensemble de [1..n] tel que la somme des xi correspondants soit exactement la moitié de la somme des xi , i.e. J ⊂ [1..n], tel que P i∈J xi = P i /∈J xi = Pn i=1 xi

Partition	JSP
n+k*(nombre d'entiers + nombre de partition ou k > 0)	n (nombre d'entiers)
0	ai (date d'arrivée d'une tâche)
k- nombre de machine	m (le nombre de machine)

Pour pouvoir réduire polynomialement Partition en JSP on doit mettre les entiers sous formes de tâches. Les tâches ne peuvent s'exécuter que sur deux machines pour différencier les deux sommes d'entiers. Toutes les tâches commencent à zéro. En ajoutant un temps d'attente maximum de Sommexi/2 il est impossible d'avoir un tâche qui commence à une valeur supérieure à Sommexi/2. Cependant si on fait la somme des durées elle peut encore être supérieur à sommexi/2. Pour résoudre se problème on ajoute 2 tâches (0, Sommexi/2) qui vérifieront que la somme des entiers sur une machine est bien égale à la (somme des xi)/2

Q.2.3) Voir la classe PblPartition dans le répertoire tp6

Q.3.1) on peut facilement vérifier si un certificat est correct, Sum est donc NP.
En effet, il suffit de vérifier que la somme des entiers fournis dans le certificat est égale à l'entier cible. Cela se calcule en un temps polynomial (un parcours de n valeur et une comparaison)
Q.3.2) Montrer que Sum se réduit polynomialement en JSP

Partition	Sum
n+k (nombre d'entiers + nombre de partition ou k > 0)	n (nombre d'entiers)
X1,...,Xn (les entiers)	X1,...,Xn (les entiers)

s (entier cible)	s = somme(xi)/2

Q.3.3) voir la classe PblSum dans le répertoire tp6
Q.4) voir la classe PblSum dans le répertoire tp6

3 Optimisation versus Décision

Q 1. Montrer que si JSP Opt1 (resp. JSP Opt2) était P, la propriété JSP le serait aussi ; qu'en déduire pour JSP Opt1 (resp. JSP Opt2) ?

Si JSP OPT1 était P cela voudrait dire qu'il existe un algorithme de résolution polynomial pour ce problème. On sait que pour résoudre JSPOpt1 il faut pouvoir calculer :

- Un ordonnancement correct
- Trouver l'ordonnancement avec le plus petit D

On sait que pour résoudre JSPOpt1 il faut pouvoir calculer :

- Un ordonnancement correct

On voit bien ici que lorsqu'on calcul JSPOpt1 on calcul JSP. La complexité de JSP sera donc inférieure ou égale à celle de JSPOpt1. Or on sait que JSP se résout en un temps polynomial. On peut donc dire que :
Si JSPOpt1 est P alors JSP sera P.

Q 2. Montrer que si la propriété JSP était P, JSP Opt1 le serait aussi.

Si JSP était P on pourrait écrire un algorithme de ce type pour calculer JSPOpt1.

D = l'attente maximale du problème d'origine.
for (i=0; i< D; i++)
    PblJSP jspOpt= new PblJSP(n_old,jobs_old, i)
    if (jspOpt.aUneSolution) return i
return D;

On peut dire que la complexité de JSPOpt1 est dans le pire des cas O(D*(complexité de JSP))
On peut donc dire que JSPOpt1 se résout en temps polynomial et est donc P si JSP est P.

Q 3. Plus dur... Montrer que si la propriété JSP était P, JSP Opt2 le serait aussi

Dans un premier temps, on applique JSPOpt1 pour trouver le temps d'attente minimal. On sait, parce que on l'a montré à la question précédente que JSPOpt1 est P. Puis on itère sur les tâches tant qu'on a pas trouvé d'ordonnancement correct de longueur minimal. De ce fait, on construit l'ordonnancement optimal en tant polynomial.