Binôme : Philippot Grégoire - Pelage François-Xavier

1: Qu’est-ce qu’une propriété N P ?

Q 1)La propriété est N P.

donnée :
  m, un entier positif (nombre de machines) => taille = log2(m)
  n, un entier positif (nombre de tâches)=> taille = log2(n)
  Liste l(tuple (date arrivée, durée)) => ∑(log2(arrivée)) + ∑(log2(durée)) = borne max
  d,entier (attente max autorisée) => taille = log2 (D)

Certificat :
  -liste de tuple (départ_machine, numéro_machine)

Ici on cherche pour chaque tâches, la distribution sur chaque machine on a donc une liste de tuple dont la dimensions de borne supérieur est $\sum_{i=0}^{n-1} log2(arrivée) + \sum_{i=0}^{n-1} log2(durée)$

polynomiale en n (nombre de tâches) donc m * $\sum_{i=0}^{n-1} log2(arrivée) + \sum_{i=0}^{n-1} log2(durée)$
bornée polynomialement en la taille de la donnée

algo:
entrée : Certificat
        données

sortie : oui le certificat est valide
        non le certificat n'est pas valide

        //tableau somme temps Certificat (temps d’arrivé + durée) pour chaque machine
        entier temps_certificat[m]
        //tableau somme temps list tâche (temps d’arrivé + durée) pour chaque machine
        entier temps_start[m]
        //parcours tâches
        pour i allant de 0 à n
                pour j allant de 0 à n
                        temps_certificat+= l[j][0] // ici on incrémente le temps avec la date de départ sur la machine de la tâche
                        temps_start += Certificat[j][0] // ici on incrémente le temps avec la date d'arrivée de la tâche
                        Si (temps_start[Certificat[j][0]] - temps_certificat[Certificat[j][0]] > d)
                                retourner non
                        Fin Si
                        Si (tâche ne se chevauche pas)   //alors le certificat n'est pas valide

```
                    retourner non
                Fin Si
                temps_certificat +=l[j][1] // incrémente le temps avec la durée de la
tâche
                temps_start +=l[j][1]
                fin pour
            fin pour
        retourner oui
```

le parcours de chaque tâche est majorée par m*n*($\sum_{i=0}^{n-1} log2(arrivée)$ + $\sum_{i=0}^{n-1} log2(durée)$)
donc l'algo est bien polynomiale en la taille de l'entrée
Donc le problème est bien en NP.

Q2) N P = Non déterministe polynomial

2.1
algo:
entrée : données

sortie : certificat aléatoire

```
 //durée écoulée qui sera mise à jour à chaque tâches passées pour chaque machine
 entier temps[m]
 pour i allant de 0 à taille de list_tache
        machine_alea = random(0,m-1)
        tache_alea = random(0,n-1)
        // Si date d'arrivée > au temps alors on incrémente le temps
            Si temps[machine_alea] < tache_alea[0] alors
                temps[machine_alea]+= tache_alea[0] - temps[machine_alea]
            Fin Si

        // on affiche la machine et la tache qui va s'exécuter

        // on incrémente le temps de la durée de la tâche
        temps[machine_alea]+= tache_alea[2]

        // on supprime la tâche de la liste des tâches
        remove(list_tache[tache_alea])

 fin pour
```

2.2
il faudrait générer à partir de données, un certificat aléatoire, puis le vérifier avec l'algorithme de vérification, on ne saurait donc pas si le certificat est bon ou non

Q3) N P ⊂ ExpTime

3.1
$n!^m$ est une borne supérieure du problème

3.2
le plus petit certificat serait de mettre toutes les tâches sur la machine 0.
Le dernier serait toute les tâches incrémenté au plus de l'attente maximale sur la dernière machine.
Pour vérifier si le problème à une solution, on test pour chaque certificat l'algorithme de vérfication, la complexité temporelle sera donc bornée par $n!^m$.

2: Réductions polynomiales:

Q 1) Une première réduction

Soit le problème de décision Partition défini par :
Donnée :
np –un nombre d'entiers
x 1 , · · · , x n –les entiers
Sortie : Oui, si il existe un sous-ensemble de [1..n] tel que la somme des P x i corres

Q2) Montrer que Partition se réduit polynomialement en JSP .
x_i étant les entiers de Partition et n_p le nombre d'entier

Donnée :
m = 2
n = n_p
a_i = 0 i, 1<= i <= n
t_i = x_i i, 1 <= i <= n
D >= Somme_totale(list_tache)/2 //ici on prendra comme temps d'attente max la moitié de la somme de toutes les dates d'arrivées .

La valeur m est fixe (2)
la valeur n ne bouge pas d'un problème à un autre
la valeur de a_i est fixe (0)
la valeur t_i reprendra toutes les valeurs de Partition
La valeur D prendra la somme totale de toutes les dates d'arrivées et on les diviseras par 2.
La transformation est bien calculable par un algorithme polynomiale

Si le problème TSP a une solution, on obtiendra alors après transformation un problème JSP qui a une solution et donc ici: une liste dont les tâches sont disposée sur les deux machines
et ces deux machines auront le même temps d'exécution équivalent à D donc la moitié de la somme des arrivées des tâches.

Si le problème TSP n'a pas de solution, on obtiendra alors après transformation un problème JSP qui n'a pas de solution et donc ici: une liste dont les tâches sont disposée sur les deux machines et ces dernières auront un temps d'exécution différent dont au moins l'un au dessus de D donc la moitié de la somme des arrivées des tâches.

Q2.1
JSP un NP-dur
JSP étant NP-dure et NP, on peut dire qu'elle est donc NP-complète

Q2.2
JSP est NP-dur
Partition est NP-dur
Partition se réduit en JSP on peut donc en déduire que JSP se réduit en Partition

Q2.3
voir Partition.java

Q3)

Q3.1
x'_i étant les entiers de SUM, n_s le nombre d'entier et s l'entier cible

Donnée :

$n_p = n_s$
$x_i = x'_i$ i,1<= i <= np -1
x_{np} = s

SUM utilise l'algo de vérification de JSP le parcours de chaque tâche est majorée par
2*np*($\sum_{i=0}^{n-1} log2(x_i)$)
donc l'algo est bien polynomiale en la taille de l'entrée
Donc le problème est bien en NP.

Q3.2
la valeur np prendra la
les valeur x_i prendront les valeurs x'_i

la dernière valeur de la liste (x_{n_p}) sera la valeur cherchée.

La transformation est bien calculable par un algorithme polynomiale

Si le problème SUM a une solution, on obtiendra alors après transformation un problème Partition qui a une solution et donc ici: une liste dont les tâches sont disposée sur les deux machines et ces deux machines auront le même temps d'exécution équivalent à D donc la valeur recherchée.

Si le problème SUM n'a pas de solution, on obtiendra alors après transformation un problème Partition qui n'a pas de solution et donc ici: une liste dont les tâches sont disposée sur les deux machines et ces dernières auront un temps d'exécution différent dont au moins l'un au dessus de D donc la valeur recherchée.

Q3.3.

Voir Sum.java

3: Optimisation versus Décision

Q1)
S'il existe un algo polynomiale (JSPOpt1) permettant de trouver un ordonnancement correcte avec une attente minimale on peut donc en déduire qu'il existe un algorithme polynomiale qui permettrait de trouver un ordonnancement correct (ici JSP) .On peut donc en conclure que si JSPOpt1 est en P alors JSP serait aussi au moins P.

S'il existe un algo polynomiale (JSPOpt2) permettant de trouverun ordonnancement correcte avec une attente optimale on peut donc en déduire qu'il existe un algorithme polynomiale qui permettrait de trouver un ordonnancement correct (ici JSP) .On peut donc en conclure que si JSPOpt2 est en P alors JSP serait aussi au moins P.

Q2)

Si on peut tester en temps polynomial qu'il existe un ordonnancement correct pour un D minimal, alors nous avons JSPOpt2

On doit juste regarder pour chaque Certificat valide le temps d’attente de chaque tâche et voir s’il dépasse un D, et comparer ce D aux autres Certificats jusqu’à en trouver le plus petit.