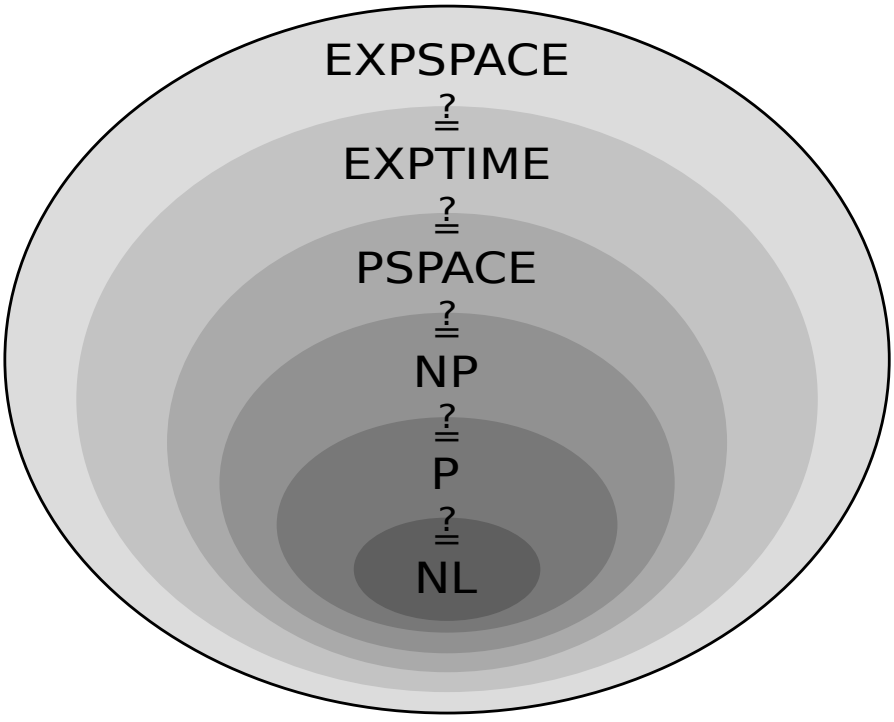
Rapport de TP

Algorithmes et Complexité

La classe NP



Hamani Rayane
Nguyen Kévin

Master 1 d'Informatique
2020 - 2021

Introduction

Objectif

xxx

Partie 1 : Qu'est-ce qu'une propriété NP ?

Q1 :

Le certificat que nous avons choisi est : Pour chaque tâche, sa date de lancement.

L'algorithme de vérification placera les tâches sur une machine disponible à la date proposée s'il en trouve une.

La taille de notre certificat sera : nombre de tâches * log2(sa date de lancement).

La taille de l'entrée est : log2(nombre de tâches) + log2(nombre de machines) + log2(temps d'attente maximum) + (nombre de tâches * (log2(date d'arrivée) + log2(temps d'exécution)))

La date de lancement et la date d'arrivée sont de mêmes types. De cette façon, on voit bien que la taille de notre certificat est bien bornée par la taille de l'entrée.

Notre algorithme de vérification est la fonction « verification » dans notre code.

Q2.1:

Notre algorithme de génération aléatoire d'un certificat est la fonction « generationAleatoire » dans notre code.

Q2.2 :

Le schéma d'un tel algorithme serait de générer un par un la totalité des certificats possibles et de vérifier la validité du certificat généré avant de passer au suivant. La génération s'arrêterait dès lors qu'un certificat venait à être valide, sinon elle continuerait jusqu'à en trouver un de valide. Si aucun venait à être valide à la fin de la génération de la totalité des certificats, alors les données en entrée n'admettent aucune solution.

Q3.1 :

L'ordre de grandeur de notre nombre de certificat est exponentielle car on génère exactement ((temps d'attente maximum + 1) ^ nombre de tâches) certificats au maximum.

Q3.2 :

L’ordre que nous proposons pour être sûr de générer tous les certificats est de commencer du plus petit petit jusqu’au plus grand.

On va générer tout les certificats, si le dernier certificat a était généré mais n'est pas valide alors le problème n'a pas de solution.

La complexité temporelle de notre algorithme va étre exponentielle car on effectue des conbinaisons de taches, l'exponentielle étant dépendant
du nombre de taches.

La complexité spatiale de notre algorithme du british museum, est très petite O(n), car on génère un certificat, on le vérifie puis remplace
certificat actuel par son successeur, permettant de limiter la l'espace mémoire a uniquement la taille du tableau par le nombre de tache
et la taille des valeurs que chaque case du tableau contient. Ainsi on évite de stocker tout les certificats possible de générer, consommant
un espace mémoire très grand et rendent l'algorithme inutilisable.

Partie 2 : Réductions polynomiales

Q2 :

Partition est un problème NP-complet.

Pour montrer que partition se réduit polynomialement en JSP, il faut pouvoir effectuer une transformation du problème de partition en JSP. Une réduction polynomiale possible du problème est la suivante :

- Pour chaque entier du problème de partition, on peut lui associer une tâche.
- Chaque sous ensemble d'entiers va être associer à une machine
- A noter qu'il faut que chaque sous-ensemble soit égal a la moitié de la somme totale des xi, ce qui signifie qu'il y a exactement 2 sous-ensembles, et donc qu'il faut qu'il y ait obligatoirement 2 machines.

Le but de Partition est de répartir équitablement les tâches en deux groupes distincts. Celui de JSP est de trouver un arrangement de tâches tel que la répartition entre les deux machines soient équitables. On voit clairement qu’il y a match.

La construction est également polynomiale (elle est en O(n)) puisque l’on parcourt la liste des tâches qu’une seule et unique fois.

Le problème de Partition étant NP-complet et donc par conséquent NP-dur, Partition se réduit polynomialement en JSP. JSP est donc ainsi aussi NP-dur et tout aussi difficile que Partition.

Q2.1 :

Nous avons montré dans la partie 1 que JSP est un problème NP et nous venons à l'instant de prouver que JSP est aussi un problème NP-dur si le nombre de machines est égal à 2. Ainsi, puisque JSP est NP et NP-dur, JSP est NP -omplet.

Q2.2 :

Non car si nous avons une instance où le nombre de machine est égal à 5 par exemple, on ne peut pas réduire JSP en Partition.

Q2.3 :

Notre implémentation de la réduction polynomiale de Partition en JSP est notre fonction « reductionPartitionEnJSP » dans notre code.

Q3.1 :

Pour le certificat on peut considérer qu'il ce compose d'un sous ensemble d'entier parmi une liste d'entiers.

L'algorithme de vérification ce chargera de vérifier que la somme des entiers de ce sous ensemble d'entiers est égale a l'entier cible s renvoyant vrai, sinon faux.

Il faut donc parcourir ce sous ensemble d'entiers que l'on peut implémenter par une liste, la complexité de l'algorithme est donc en O(n).

Il faut regarder que la taille du certificat soit égale ou inférieure a la taille de la donnée.
La taille des données étant :

taille de n (un entier) : log2(n)
taille de s (un entier) : log2(s)
x1,...,xn (des entiers) : n*log2(x)

la taille des données est : log2(n) + log2(s) + n*log2(x).

La taille du certificat étant :

taille du sous ensemble d'entiers composé au total de m entiers : m*(log2(x)

taille de s (un entier) : log2(s)

la taille des données est : log2(s) + m*log2(x)

Ainsi le sous ensemble d'entiers est forcément inférieur ou égale a l'ensemble d'entiers de départ, n => m

Donc log2(s) + m*log2(x) <= log2(n) + log2(s) + n*log2(x)

Donc la taille du certificat est inférieure ou égale a la taille des données en entrées.

En conclusion, nous venons de prouver que SUM est un problème NP.

Q3.2 :

On peut voir des similitudes entre le problème de partition est SUM, car on peut associer l'entier cible s du problème SUM
a la moitié de la somme des entiers du problème partition

Ainsi nous avons :

- le nombre d'entiers n du problème de partition qui s'associe au nombre d'entiers n du problème SUM.

- x1,...,xn les entiers du problème de partition qui s'associe au entiers x1,...,xn du problème SUM.

- la moitié de la somme des entiers x1,...,xn du problème de partition qui s'associe a l'entier s du problème SUM.

Q3.3 :

Notre algorithme de réduction dans JSP est la fonction « reductionSumEnPartition» dans notre code.

Q4 :

Notre algorithme de réduction dans JSP est la fonction « reductionSumEnPartition» dans notre code.

Partie 3 : Optimisation versus Décision

Nous n’avons pas traité cette partie.

Annexes

Le code de notre programme est inclus dans l'archive du rendu.

Pour compiler notre code en exécutable, vous pouvez utiliser la commande suivante :

```
g++  classeNP.cpp -g -o classeNP -Wall -Wextra
```

Pour utiliser notre programme sur l'un des fichiers de test (ici le fichier donnee1) de l'archive, vous pouvez utilisez la commande suivante :

```
./classeNP -v 4 < donneesTestNP/donnee1
```