1 Qu'est-ce qu'une propriété NP ?

Q1.

Q1.1 Définir une notion de certificat pour le problème.

Une notion de certificat pour ce problème est une liste dont chaque indice représente une tâche et dont chaque entier dans la liste est le numéro de la machine dans laquelle mettre la tâche correspondante.

Q1.2 Comment pensez-vous l'implémenter ?

Nous pensons l'implémenter en JAVA en faisant une liste de taille n (n étant le nombre de tâches) dont chaque case contient un numéro de machine.

Q1.3 Quelle sera la taille d'un certificat ?

La taille d'un certificat est de n car il y a n cases dans la liste.

Q1.4 La taille des certificats est-elle bien bornée polynomialement par rapport à la taille de l'entrée ?

Oui, la taille de l'entré est toujours supérieur a la taille du certificat. En effet la taille d'un certificat est n < n +3 (nombre de machine, nombre de tâche et d) qui correspond à la taille de l'entrée. Donc il est bien polynomial.

Q1.5 Proposez un algorithme de vérification associé. Est-il bien polynomial ?

(1) On prend un certificat C et une entrée U.

(2) On vérifie que toute les taches de U sont présentes dans C et que chacune de ces taches est associées à une machine.

(3) Pour chaque tâche du certificat associé à une machine, on vérifie si la disposition des taches est compatible avec le délai maximal fixé dans U.

(4) Si l'association tache machine de C est compatible avec U alors on renvoie ture sinon false.

code:

```
public static int[] divise(String certificat) {
```

```
                char charArray[] = new char[str.length()];
                int[] index_machines = new int[str.length()];
                for (int i = 0; i < certificat.length(); i++) {
                        charArray[i] = certificat.charAt(i);
                        index_machines[i] = Integer.parseInt(String.valueOf(charArray[i]));
                }
        return index_machines;
        }
```

//Changer la forme du certificat pour qu'elle corresponde à l'entrée de U.

```
        public static void verifier(int d, String s, int num_machine, ArrayList<tache> taches) {
        Object[] taches_en_machines = new Object[num_machine];

                int[] index_machines = divise(s);
                String resultat = "";
                for (int i = 0; i < taches.size(); i++) {
                        taches.get(i).setNum_machine(index_machines[i]);}
```

// vérifie que toute les taches de U sont présentes dans C et que chacune de ces taches est associées à une machine.

```
    public static String compare(ArrayList<tache> taches, int d) {
        String analyse = "oui";
        if (taches.size() > 0) {
                int y = taches.get(0).debut + taches.get(0).duree;
                for (int i = 1; i < taches.size(); i++) {
                        if (y <= taches.get(i).debut) {
                                y = taches.get(i).debut + taches.get(i).duree;
                        } else if ((y > taches.get(i).debut) && ((y - taches.get(i).debut)) <= d) {//
taches.get(i+1).setDebut(taches.get(i).debut+taches.get(i).duree);
                                y = y + taches.get(i).duree;
                        } else {
                                analyse = "non";
                        }
                }
        }
        return analyse;
    }
```

// Pour chaque tâche du certificat associé à une machine, on vérifie si la disposition des taches est compatible avec le délai maximal fixé dans U.

Q 2. NP = Non déterministe polynomial

Q 2.1. Génération aléatoire d'un certificat.

Proposez un algorithme de génération aléatoire de certificat

```
    for (int t = 0; t < num_taches; t++) {
                certificat = certificat + ((int) (Math.random() * num_machine) + "");
        }
```

On utilise le Math.random() pour donner un certificat par hasard en String. Par exemple "0101011" C'est à dire que pour chaque indice la tache correspondante est assigne à la machine correspondant à l'entier de l'indice.

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

(1) Obtenir un certificat comme Q2.1.

(2) Pour chaque tâche de ce certificat associé à une machine, on vérifie si la disposition des taches est compatible avec le délai maximal fixé dans U.

(3) Si le certificat n'est pas compatible on recommence avec un nouveau certificat génère au hasard jusqu'a obtenir un résultat positif.

Q 3. NP → ExpTime

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

m^n

Avec n le nombre de taches et m le nombre de machines.

Q 3.2. Enumération de tous les certificats

Une méthode classique pour énumérer les certificats .Quel ordre proposez-vous ?

On peut représenter par une liste de 0 de taille n (nombre de taches) et on l'incrémente de 1 en 1 comme un nombre en base m (nombre de machines)

Q 3.3. Comment déduire de ce qui précède un algorithme pour tester si le problème a une solution ? Quelle complexité a cet algorithme ?

On teste l'algorithme de vérification sur tous les certificats jusqu'à avoir une vérification qui renvoie vrai. La complexité est

(1) Pour obtenir tous les possibilités(m^n)

(2) Pour vérifier chaque certificat n

(3) La complexité est (m^n)*n

Implémentation

Bien fini en code.

2 Réductions polynomiales

Q 4. Une première réduction Partition

Bien fini en code.

Q 5. Montrer que Partition se réduit polynomialement en JSP.

Oui, parce que le problème de JSP est une situation spéciale de Partition, où il y a juste deux machines et on ne consulte pas l'arrivé de la tâche. Quand les sommes des durées de chaque machine sont pareils, retourne true, sinon retourne false.

Q 5.1. Qu'en déduire pour JSP ?

Puisque la propriété Partition est NP-complète, c'est-à-dire que Partition est NP-dure. Puisque Partition se réduit polynomialement en JSP, alors on peut en déduire que JSP est aussi NP-dure. Puisque nous avons vu précédemment que JSP est aussi de la classe NP, alors JSP est NP-complète.

Q 5.2. Pensez-vous que JSP se réduise polynomialement dans Partition ? Pourquoi ?

Non, nous ne pensons pas que JSP se réduit polynomialement en Partition parce que dans l'énoncé´ du problème de Partition, on a un nombre fixe de partitions (ici 2) alors que le nombre de machines dans le problème de JSP peut changer en fonction de l'instance du problème.

Q 5.3. [à coder] Implémenter la réduction polynomiale de Partition dans JSP.

```
ArrayList<String> total_possibilites = Partition.total_possibilite(taches.size(),
index_machines);
```

Bien fini les autres fonctions en code.

Q 6. Une deuxième réduction SUM

Q 6.1. Montrer que SUM est NP.

Parce que on peut vérifier un certificat de sum dans le temps polynomial. On ajoute tous les chiffres du certificat ensemble, et puis on compare le résultat avec la valeur de sum. (La complexité est n). Si ils sont pareils retourne true, sinon retourne false.

Q 6.2. Montrer que SUM se réduit polynomialement en Partition.

Partition est aussi un cas spécial de Sum, puisque c'est un problème de SUM mais la cible est la moitié de la somme de tous les chiffres. En termes de réduction, Sum se réduit donc polynomialement en Partition.

Q 6.3. Programmez la réduction ci-dessus de SUM dans Partition.

Bien fini en code

Q 7. Composition de réductions

En utilisant les deux réductions précédentes, implémenter une réduction de SUM dans JSP. La réduction de SUM dans JSP est déjà faite, en effet SUM se réduit en Partition qui se réduit en JSP.

3 Optimisation versus Décision

Au problème de décision JSP, on peut associer deux problèmes d'optimisation : 3

Q 8. Montrer que si JSPOpt1 (resp. JSPOpt2) était P, la propriété JSP le serait aussi ; qu'en déduire pour JSPOpt1 (resp. JSPOpt2)?

Parce que JSP est plus simple que JSPOpt1, parce que JSPOpt1 cherche le plus D possible ce qui peut prendre plus d'itération que pour JSP.
JSP étant moins exigeant il serait donc P aussi.

Il en est de même pour JSPOpt2 mais on recherche l'attente optimale.

Q 9. Montrer que si la propriété JSP était P, JSPOpt1 le serait aussi.

JSPOpt1 n'est autre que JSP dans une boucle, donc si JSP est P alors x*JSP avec x un entier alors JSPOpt1 est P aussi.