SESSION 2008

2nd concours

INFORMATIQUE

École normale supérieure de Lyon

Durée : 3 heures

Ce livret comprend 4 pages numérotées de 1 à 4

Les calculatrices sont interdites.

Quelques problèmes d'ordonnancement

Ce sujet comporte trois parties. Il est recommandé de lire l'ensemble du sujet avant de commencer la rédaction. Il est également conseillé de traiter les questions dans l'ordre de l'énoncé. On pourra cependant aborder une question en admettant les résultats des questions précédentes. Les algorithmes demandés pourront être écrit en pseudo-code ou dans un langage au choix du candidat, en utilisant les structures de contrôle habituelles.

Notations

- Dans la description des algorithmes, nous utiliserons la notation $a \leftarrow b$ pour signifier que «a prend la valeur de b».
- Étant donnés deux ensembles A et B, on notera $A \backslash B$ l'ensemble des éléments de A qui ne sont pas dans B.

Partie 1 Ateliers de fabrication

On étudie dans cette partie comment répartir les tâches nécessaires à l'élaboration d'une pièce dans plusieurs ateliers de fabrication. Pour réaliser une pièce, plusieurs traitements sont nécessaires : on peut par exemple imaginer qu'il faut d'abord découper un matériau, puis l'emboutir, le percer, etc. Ces différents traitements forment une suite de tâches, notées $T_1, T_2, \ldots T_n$ qu'il faut effectuer **dans cet ordre** pour obtenir la pièce désirée.

Pour effectuer ces tâches, nous avons à notre disposition plusieurs ateliers de fabrication, notés $A_1, A_2, \ldots A_p$. Chaque tâche doit être effectuée dans un de ces ateliers. Ces ateliers ne disposent pas des mêmes machines, et même s'ils sont tous capables d'effectuer l'ensemble des tâches, certains sont plus efficaces que d'autres pour certaines tâches. On note $t_{k,i}$ le temps que prend le traitement de la tâche T_k dans l'atelier A_i.

Les différents ateliers ne sont pas tous localisés au même endroit. Si une partie du traitement de la pièce est fait dans un atelier A_i et que la suite du traitement est fait dans un autre atelier A_j, il faut prendre en compte le temps nécessaire au transport de la pièce inachevée entre ces deux ateliers. On note $c_{i,j}$ le temps nécessaire pour transporter une pièce entre A_i et A_j. Les matériaux nécessaires à la première tâche T_1 sont disponibles dans tous les ateliers, donc T_1 peut être effectuée dans n'importe quel atelier sans transport préalable.

Question 1.1. On décrit le chemin de traitement d'une pièce sous la forme d'une fonction $f : A_{f(k)}$ est l'atelier qui exécutera la tâche T_k. Donner un algorithme qui calcule le temps total nécessaire pour terminer la pièce en suivant l'ordonnancement f.

Question 1.2. On suppose qu'il y a trois tâches dans la fabrication de la pièce ($n = 3$), et deux ateliers ($p = 2$) dont les temps traitements sont donnés par le tableau suivant. Le transport d'une pièce d'un atelier à l'autre coûte 2 unités de temps : $c_{1,2} = c_{2,1} = 2$. Donner un chemin de traitement optimal et son temps de traitement.

	T_1	T_2	T_3
atelier A_1	2	3	5
atelier A_2	6	2	2

Question 1.3. Donner le nombre total de chemins possibles pour fabriquer une pièce nécessitant n tâches, lorsqu'on dispose de p ateliers.

Question 1.4. On appelle $t_{\min}[k,i]$ le temps de traitement minimal de la suite de tâches $T_1,\ldots,T_k$, de telle sorte que la tâche T_k soit traitée par l'atelier A_i. Donner et justifier une formule de récurrence permettant de calculer $t_{\min}[k,i]$ en fonction des valeurs de $t_{\min}[k',i']$ avec $k' < k$.

Question 1.5. En déduire un algorithme efficace qui calcule le temps minimal nécessaire au traitement de toutes les tâches. Donner la complexité de votre algorithme.

Question 1.6. Comment compléter l'algorithme de la question précédente afin de pouvoir afficher un chemin de traitement optimal, c'est-à-dire la liste des ateliers utilisés, par exemple `1 3 2 2`.

Partie 2 Théorie des matroïdes

On introduit dans cette partie un outil pour l'étude des algorithmes : la théorie des matroïdes. Un matroïde est composé d'un ensemble d'éléments et d'une collection de sous-ensembles de ces éléments, nommés *indépendants*, qui ont des propriétés d'échange et d'hérédité.

Définition 1. *$(S,\mathcal{I})$ est un **matroïde** si S est un ensemble de n éléments et $\mathcal{I}$ est un ensemble de parties de S vérifiant ces deux propriétés :*
- *l'hérédité : si $X \in \mathcal{I}$ alors pour tout $Y \subset X$, $Y \in \mathcal{I}$,*
- *l'échange : si A et B sont deux éléments de $\mathcal{I}$ et que $|A| < |B|$, alors il existe $x \in B\backslash A$ tel que $A \cup \{x\}$ est dans $\mathcal{I}$*

Si $X \in \mathcal{I}$, on dit que X est un indépendant.

Dans cet partie, on va chercher à construire un indépendant de façon incrémentale, en rajoutant des éléments à un indépendant existant. Lorsqu'on ne peut plus rajouter d'éléments, on dit que l'indépendant est maximal :

Définition 2. *Un indépendant F est dit **maximal** si pour tout $x \in S\backslash F$, $F \cup \{x\}$ n'est pas un indépendant.*

Question 2.1. Montrer que tous les indépendants maximaux d'un matroïde $(S,\mathcal{I})$ ont le même cardinal.

On pondère les éléments de S avec une fonction de poids $w : \forall x \in S, w(x) \in \mathbb{N}$. On étend cette fonction de poids aux parties de S :

$$\forall X \subset S, \quad w(X) = \sum_{x \in X} w(x)$$

On cherche à calculer un indépendant de poids maximal. On suppose que les éléments de S sont triés par poids décroissants :

$$s_1 \geqslant s_2 \geqslant \cdots \geqslant s_n$$

Question 2.2. Soit s_k l'élément de S d'indice minimal tel que $\{s_k\}$ est un indépendant (c'est-à-dire que s_k est le plus grand élément de S qui forme un singleton indépendant). Montrer qu'il existe un indépendant de poids maximal qui contient s_k.

Question 2.3. Montrer que l'algorithme Glouton ci-dessous donne un indépendant de poids maximal.

Algorithme 1 : Algorithme Glouton

```
début
    A ← ∅;
    pour i = 1, 2, ..., |S| faire
        si A ∪ {s_i} est indépendant alors
            A ← A ∪ {s_i}
    retourner A
fin
```

Partie 3 Retards et pénalités

On s'intéresse maintenant à la fabrication de plusieurs exemplaires d'une même pièce dans un atelier. La fabrication d'un exemplaire se fait sans interruption, et prend un temps fixe (une unité de temps). L'atelier fabrique un seul type de pièce, mais il a plusieurs commandes pour cette pièce. Chaque commande précise une date limite à laquelle l'exemplaire de la pièce doit être fabriqué. Si cette date n'est pas respectée, l'atelier devra payer une pénalité fixée dans la commande. Le but est de trouver un ordre de fabrication des pièces de telle sorte que la somme des pénalités à payer soit minimale.

On suppose que l'atelier doit fabriquer n exemplaires. On note T_k la tâche correspondant à la fabrication de l'exemplaire k. Toutes ces tâches sont identiques, et prennent un temps 1. Pour une tâche T_k, on note d_k sa date limite, et p_k la pénalité à payer si cette date n'est pas respectée.

Dans un ordonnancement donné, on appelle **tâches à l'heure** les tâches dont la date limite est respectée, et **tâches en retard** les tâches dont la date limite n'est pas respectée, et pour lesquelles il faudra payer une pénalité.

On dit qu'un ordonnancement vérifie l'**ordre canonique** si dans cet ordonnancement :

(i) toutes les tâches à l'heure sont effectuées avant les tâches en retard ;

(ii) les tâches à l'heure sont exécutées dans l'ordre des dates limites croissantes.

Question 3.1. Montrer qu'à partir d'un ordonnancement des tâches donné par la fonction $f : T_{f(1)}, \ldots, T_{f(n)}$, de pénalité totale P, on peut construire un ordonnancement qui vérifie l'ordre canonique et dont la pénalité totale inférieure ou égale à P.

Question 3.2. Exécuter l'algorithme Ordonnance_Tâches (décrit page suivante) sur l'exemple suivant :

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
d_i	4	2	4	3	1	4	6
p_i	7	6	5	4	3	2	1

On donnera les listes A et B après chaque itération de la boucle **pour**.

Algorithme 2 : Ordonnance_Tâches

```
début
    Calculer la liste L des tâches triées par p_i croissants
    A ← ∅        (la liste des tâches à l'heure)
    B ← ∅        (la liste des tâches en retard)
    pour chaque tâche T_i de la liste L faire
        insérer T_i dans la liste A en utilisant l'ordre canonique, cela forme la liste A'
        si toutes les dates limites sont respectées dans A' alors
            A ← A'
        sinon
            ajouter T_i à la fin de la liste B
fin
```

Définition 3. *On dit qu'un ensemble de tâches est indépendant si ces tâches peuvent être exécutées en étant toutes à l'heure.*

Question 3.3. Soit A un ensemble de tâches. On note $N_t(A)$ le nombre de tâches dans A dont la date limite est inférieure ou égale à t. Montrer que les trois propriétés suivantes sont équivalentes :
(a) l'ensemble de tâches A est indépendant ;
(b) $\forall t = 1, 2, \ldots, n, \quad N_t(A) \leqslant t$;
(c) si on exécute les tâches de A dans l'ordre canonique, il n'y a aucune tâche en retard.

Question 3.4. Soit A et B deux ensemble de tâches indépendants, avec $|A| < |B|$. Montrer qu'il existe une tâche T_i dans $B \backslash A$ telle que $A \cup \{T_i\}$ est indépendant.

Question 3.5. Montrer que l'algorithme précédent calcule un ordonnancement de pénalité minimale.

Partie 4 Gestion de projet

On considère dans cette partie un projet constitué de plusieurs tâches $T_1, T_2, \ldots, T_n$. Chacune des tâches doit être traitée par une personne pendant un certain temps. Les différentes tâches composant le projet ne sont pas indépendantes : certaines tâches utilisent les résultats d'autres tâches. Pour chaque tâche T_k, on note $D[i,k] = 1$ s'il y a une dépendance entre T_i et T_k, c'est-à-dire si T_i doit être terminée avant T_k, et $D[i,k] = 0$ s'il n'y a pas de dépendance. Enfin, on note t_k la durée de traitement de la tâche T_k par une personne.

Question 4.1. On suppose pour l'instant qu'il n'y a qu'une seule personne affectée à ce projet. Donner un algorithme calculant un ordre de traitement possible de ces tâches, lorsque cela est possible. L'algorithme calculera un tableau L, de telle sorte que $L[i] = k$ si la $i^{\text{ème}}$ tâche traitée est T_k. Est-ce toujours possible ?

Question 4.2. On suppose maintenant qu'on dispose d'autant de personnel qu'on le souhaite : on peut en particulier affecter une personne différente à chaque tâche. Donner un algorithme calculant pour chaque tâche T_k, la date minimale $D_{\min}[k]$ à laquelle l'exécution de cette tâche peut commencer, ainsi que le temps de traitement minimal de toutes les tâches.