DS4 d'informatique MPSI
1^{er} juin 2012

Devoir surveillé

Durée : 2 heures

Aucun document n'est autorisé. **On pourra définir des fonctions non demandées explicitement, si cela facilite la programmation. Inutile de prouver la correction et la terminaison des fonctions écrites, sauf si on le demande explicitement. Les fautes de syntaxe seront sanctionnées.**

Ordonnancement

Seule la première partie comporte des questions de programmation en Caml.
Étant donné un ensemble fini X, on notera $|X|$ son cardinal.
Étant donné deux ensembles A et B, on notera $B \setminus A$ l'ensemble des éléments de B qui ne sont pas dans A.

Partie A – 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 $T_1, \ldots, T_n$ qu'il faut exécuter dans cet ordre pour obtenir la pièce désirée.

Pour effectuer cette tâche, nous avons à notre disposition plusieurs ateliers de fabrication, notés $A_1, \ldots, A_p$. 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 (on aura $c_{i,j} = c_{j,i}$). Les matériaux nécessaires à la première tâche T_1 sont disponibles dans tous les ateliers, donc T_i peut être effectuée dans n'importe quel atelier sans transport préalable.

Concrètement, on supposera que le temps d'exécution des tâches sera donné par une fonction fabrique de type `int -> int -> int` prenant en argument k (la tâche T_k), i (l'atelier A_i), et renvoyant $t_{k,i}$.

De même, une fonction transport déjà fournie, de type `int -> int -> int`, renverra $c_{i,j}$ lorsqu'elle prendra i et j en arguments.

A.1 Écrire une fonction chemin de type `int list -> int`, qui, à une liste d'entiers indiquant la suite des ateliers parcourus en sens inverse (le dernier atelier est en tête de liste), associe le temps nécessaire à l'élaboration de la pièce en suivant ce chemin.
Remarque : on autorise ici l'utilisation de `list_length` (mais on a le droit de s'en passer).

A.2 On suppose qu'il y a trois tâches dans la fabrication de la pièce et deux ateliers, dont les temps de traitement sont donnés par le tableau suivant. Le transport d'une pièce d'un atelier à un autre coûte deux 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
A_1	2	3	5
A_2	6	2	2

A.3 Donner le nombre total de chemins possibles pour fabriquer une pièce nécessitant n tâches, lorsqu'on dispose de p ateliers.

A.4 On appelle $t_{min}(k, i)$ le temps de traitement minimal de la suite de termes $T_1, \ldots, T_k$ de telle sorte que la tâche T_k soit traitée par l'atelier A_i. Donner une formule de récurrence permettant de calculer $t_{min}(k, i)$ en fonction des valeurs de $t_{min}(k', i')$, avec $k' < k$.

A.5

a Écrire une fonction temps_minimal de type `int vect -> int -> int_vect`, qui, lorsqu'elle prend en argument le tableau des $t_{min}(k-1, i)$ (i variant de 1 à p) et k, renvoie celui des $t_{min}(k, i)$.

Pour simplifier la programmation Caml, nous ne tiendrons pas compte du terme d'indice 0 des tableaux : par exemple, temps_minimal prendra en premier argument [|0; 10; 14; 12|] et en second argument 4, ce qui signifiera que $t_{min}(3, i)$ vaut 10 si $i = 1$, 14 si $i = 2$, et 12 si $i = 3$. En particulier, la taille du tableau pris en argument par temps_minimal est $p + 1$.

b En déduire un algorithme efficace temps tel que temps n p calcule le temps minimal nécessaire au traitement de toutes les n tâches par les p ateliers.

c Donner la complexité de votre algorithme (on ne demande pas un calcul détaillé, mais un résultat précédé d'une brève explication).

Partie B – Théorie des matroïdes

Définition : on appelle **matroïde** tout couple (S, I), où S est un ensemble fini, et I un ensemble non vide de parties de S tels que

1. *(Hérédite)* si $X \in I$, alors pour tout $Y \subset X$, $Y \in I$.
2. *(Échange)* si A et B sont deux éléments de I et si $|A| < |B|$, alors il existe $x \in B \setminus A$ tel que $A \cup \{x\} \in I$.

Un élément de I est appelé **indépendant**.

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

B.1 Montrer que tous les indépendants maximaux d'un matroïde (S, I) ont le même cardinal.

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

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

B.2 Soit s_k l'élément de S d'indice minimal tel que $\{s_k\}$ soit un indépendant (on suppose qu'un tel s_k existe). Montrer qu'il existe un indépendant de poids maximal qui contient s_k.

B.3 Montrer que l'algorithme Glouton ci-dessous donne un indépendant de poids maximal.

```
Début
        A <- emptyset
        pour i = 1, 2, ..., |S| faire
                si A U {s_i} est indépendant, alors
                        A <- A U {s_i}
        retourner A
Fin
```

Partie C – 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 y 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 :

1. toutes les tâches à l'heure sont effectuées avant les tâches en retard.
2. les tâches à l'heure sont exécutées dans l'ordre des dates limites croissantes.

C.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 est inférieure ou égale à P.

On s'intéresse à l'algorithme ordonnance_tâches suivant :

```
Début
        Calculer la liste L des tâches triées par p_i décroissants
        A <- emptyset (* la liste des tâches à l'heure *)
        B <- emptyset (* 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
```

C.2 Exécuter cet algorithme sur l'exemple suivant (on donnera les listes A et B après chaque itération de la boucle pour) :

	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 dit qu'un ensemble de tâches est **indépendant** si les tâches qui le composent peuvent être exécutées en étant toutes à l'heure.

C.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 :

1. l'ensemble des tâches A est indépendant.
2. Pour tout $t \in [\![1, n]\!]$, $N_t(A) \leqslant t$.
3. si on exécute les tâches de A dans l'ordre canonique, il n'y a aucune tâche en retard.

C.4 Soit A et B deux ensembles de tâches indépendantes, avec $|A| < |B|$. Montrer qu'il existe une tâche T_i dans $B \setminus A$ telle que $A \cup \{T_i\}$ soit indépendant.

C.5 Montrer que l'algorithme précédent calcule un ordonnancement de pénalité minimale.