

Option Informatique en Spé MP et MP*

Devoir surveillé du samedi 31 janvier 2009

Langages rationnels Ordonnancement de tâches

Résumé

Le sujet se compose de deux problèmes, de tailles inégales.

Le premier problème porte sur les langages rationnels : il s'agit de déterminer, pour un langage L rationnel non vide, la longueur minimale d'un mot de L , puis l'ensemble des mots de longueur minimale.

Le deuxième problème porte sur l'algorithmique : on s'intéresse à l'optimisation de la durée d'exécution d'un ensemble de tâches, sur un ensemble de processeurs identiques. Avec la multiplication du nombre de «cœurs» dans les processeurs récents (deux, puis quatre, puis six), ce problème est tout à fait d'actualité.

Dans chacun des problèmes, des questions de programmation seront pour vous l'occasion de montrer vos talents en Caml.

Veillez rédiger chaque partie sur une copie séparée.

Table des matières

1	Premier problème : langages rationnels	2
2	Deuxième problème : ordonnancement de tâches	3
2.1	Présentation du problème	3
2.2	Ordonnancement séquentiel	3
2.3	Ordonnancement séquentiel décroissant	3
2.4	Programmation	4
2.5	Annexe: conseils de programmation	5

Consignes de programmation

► L'usage des mots `ref`, `for`, `if`, `then`, `else` est interdit. Vous ne devez utiliser ni vecteurs, ni chaînes de caractères. Comptent pour une ligne : l'en-tête d'une fonction ; chaque motif ; une fonction définie par un seul motif (`let f x = x + 1;;`) ne compte que pour une ligne.

1 Premier problème : langages rationnels

► Fixons un alphabet Σ non vide. Nous nous intéressons au problème suivant : étant donné un langage rationnel L non vide, comment déterminer la longueur minimale d'un mot de L ; et comment obtenir l'ensemble des mots de longueur minimale de L .

Question 1 Soit L un langage décrit par une expression rationnelle e ne faisant pas intervenir le symbole \emptyset . Montrez que L n'est pas vide.

Question 2 Soit L un langage rationnel non vide ; montrez qu'il peut être décrit par une expression rationnelle e ne faisant pas intervenir le symbole \emptyset .

► Nous noterons $\mathcal{ER}(\Sigma)$ l'ensemble des expressions rationnelles sur l'alphabet Σ , ne faisant pas intervenir le symbole \emptyset .

Question 3 Définissez par induction structurelle une fonction μ de $\mathcal{ER}(\Sigma)$ dans \mathbb{N} , spécifiée comme suit : $\mu(e)$ est la longueur minimale d'un mot du langage décrit par e .

► Nous définissons un type Caml pour les expressions rationnelles sur l'alphabet des caractères de l'ordinateur :

```
type exprat = Epsilon | Lettre of char | Somme of exprat * exprat
              | Produit of exprat * exprat | Etoile of exprat ;;
```

Voici par exemple le codage en Caml de l'expression rationnelle $e_1 = a^*b + c$:

```
let e1 = Somme(Produit(Etoile(Lettre 'a'),(Lettre 'b')), (Lettre 'c')) ;;
```

Question 4 Rédigez en Caml une fonction de signature :

```
longueur_min : exprat -> int
```

spécifiée comme suit : `longueur_min e` calcule la longueur minimale d'un mot du langage décrit par l'expression rationnelle e . Par exemple, `longueur_min e1` doit rendre la valeur 1. Objectif : six lignes.

Question 5 Rédigez en Caml une fonction de signature :

```
minimots : exprat -> string list
```

spécifiée comme suit : `minimots e` dresse la liste des mots de longueur minimale du langage décrit par l'expression rationnelle e . Par exemple, `minimots e1` doit rendre (à l'ordre près) la liste `["b";"c"]`. Vous utiliserez la fonction `mu` de la question précédente. Objectif : huit lignes.

► Nous étudions maintenant le cas où le langage L (toujours supposé non vide) est reconnu par un automate fini \mathcal{A} , que vous pourrez supposer déterministe et émondé.

Question 6 Décrivez un algorithme calculant la longueur minimale $\mu(L)$ d'un mot de L .

Question 7 Décrivez un algorithme construisant la liste des mots de L de longueur $\mu(L)$.

► Notez bien que, dans ces deux dernières questions, on vous demande un algorithme, c'est-à-dire une description de son fonctionnement, utilisant un formalisme mathématique. En conséquence, vous ne devez pas écrire un programme.

2 Deuxième problème : ordonnancement de tâches

2.1 Présentation du problème

► Nous disposons de m machines identiques, et nous devons organiser l'exécution de $n > m$ tâches, de durées d_1, \dots, d_n de manière à minimiser la durée d'exécution. Plus précisément, nous devons minimiser le délai qui s'écoule entre le lancement de la première tâche et la fin de l'exécution de la dernière tâche.

► Une *instance* de notre problème est donc un couple (m, d) où m est un naturel au moins égal à 2, et d est une liste de $n > m$ naturels non nuls.

► Nous nous plaçons dans la situation simplifiée où les tâches sont indépendantes les unes des autres, et où l'on peut lancer une nouvelle tâche sur une machine aussitôt que la tâche qui lui était précédemment affectée est terminée.

► Donnons un exemple pour fixer les idées : $m = 3$, et $d = (7, 3, 9, 1, 6, 5, 4, 8, 2)$. Nous pourrions assigner les trois premières tâches à la machine m_1 , les trois suivantes à la machine m_2 et les trois dernières à la machine m_3 . Le planning aurait l'allure suivante, chaque case représentant une unité de temps, et indiquant quelle tâche est en train de s'exécuter :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
m_1	1	1	1	1	1	1	1	2	2	2	3	3	3	3	3	3	3	3	3
m_2	4	5	5	5	5	5	5	6	6	6	6	6							
m_3	7	7	7	7	8	8	8	8	8	8	8	8	9	9					

La durée associée à ce planning est égale à 19.

► Nous dirons qu'une machine est *tardive* si sa durée de fonctionnement est maximale ; dans notre exemple, seule la machine m_1 est tardive. Nous dirons qu'une tâche est *tardive* si elle est la dernière affectée à une machine tardive : sa date de terminaison est donc maximale. Dans notre exemple, seule la tâche 3 est tardive.

► Nous noterons $\tau_{\min}(m, d)$ la durée minimale associée à l'instance (m, d) .

Question 1 • Pour l'exemple que nous venons de donner, déterminez la durée minimale, et un ordonnancement permettant de réaliser cette durée.

Question 2 • Donnez, en fonction de d et de m , un minorant de $\tau_{\min}(m, d)$.

2.2 Ordonnancement séquentiel

► Nous considérons dans cette partie la méthode d'ordonnancement séquentiel : les tâches 1 à m sont assignées aux machines 1 à m ; par la suite, la prochaine tâche en attente est assignée à la première machine disponible. Nous noterons $\tau_{\text{seq}}(m, d)$ la durée d'exécution obtenue avec l'ordonnancement séquentiel.

Question 3 • Déterminez $\tau_{\text{seq}}(m, d)$ pour l'exemple donné plus haut.

Question 4 • Prouvez l'inégalité $\tau_{\text{seq}}(m, d) < 2\tau_{\min}(m, d)$. Indication : considérez une tâche tardive.

Question 5 • Soit $\varepsilon > 0$. Construisez une instance (m, d) pour laquelle $\frac{\tau_{\text{seq}}(m, d)}{\tau_{\min}(m, d)} > 2 - \varepsilon$.

2.3 Ordonnancement séquentiel décroissant

► La construction de l'exemple demandé à la question 5 semble indiquer qu'il vaut mieux donner la priorité aux tâches les plus longues. L'ordonnancement séquentiel décroissant consiste à trier les tâches par durées décroissantes, puis à appliquer l'ordonnancement séquentiel. Nous noterons $\tau_{\text{decr}}(m, d)$ la durée d'exécution correspondante.

Question 6 • Déterminez $\tau_{\text{decr}}(m, d)$ pour l'exemple donné plus haut.

Question 7 • Montrez qu'avec deux machines, l'ordonnancement séquentiel décroissant donne une solution minimale si $n \leq 4$.

Question 8 • Construisez un exemple avec deux machines et cinq tâches, pour lequel l'ordonnancement séquentiel décroissant ne donne pas une solution minimale.

Question 9 • Prouvez la majoration $\tau_{decr}(2, d) < \frac{3}{2} \tau_{min}(2, d)$. Indication : considérez les deux dernières tâches exécutées sur une machine tardive.

► Nous nous proposons d'établir l'égalité suivante :

$$\sup \frac{\tau_{decr}(m, d)}{\tau_{min}(m, d)} = \frac{4}{3}$$

où le sup est pris sur l'ensemble des instances.

► Soit (m, d) une instance vérifiant la propriété suivante : lorsqu'on lui applique l'ordonnancement séquentiel décroissant, il existe au moins une machine tardive qui a eu au moins trois tâches à exécuter. Notons c la durée de la dernière tâche affectée à cette machine.

Question 10 • Montrez que chacune des $m - 1$ autres machines a travaillé pendant une durée au moins égale à $2c$.

► Plaçons-nous à l'instant τ où la machine tardive considérée se voit affecter sa troisième tâche : aucune des autres machines ne s'est arrêtée de travailler au cours de l'intervalle de temps $[0, \tau[$. Notons k le nombre de celles qui, au cours de ce même laps de temps, ne se sont vu confier qu'une seule tâche.

Question 11 • Montrez que, parmi les n tâches, il y en a au moins $2m - k + 1$ de durée au moins égale à c , dont k de durée au moins égale à $2c$.

Question 12 • En déduire $\tau_{min}(m, d) \geq 3c$.

Question 13 • Établissez alors la majoration $\frac{\tau_{decr}(m, d)}{\tau_{min}(m, d)} < \frac{4}{3}$.

Question 14 • Examinez le cas où aucune machine tardive n'a reçu plus de deux tâches à exécuter.

Question 15 • Montrez que le majorant $\frac{4}{3}$ est effectivement la borne supérieure. Pour ce faire, vous construirez, pour $\varepsilon > 0$, une instance (m, d) telle que $\tau_{decr}(m, d) \geq \left(\frac{4}{3} - \varepsilon\right) \tau_{min}(m, d)$.

2.4 Programmation

► Nous utiliserons le type Caml suivant pour décrire l'état d'une machine à un instant donné :

```
type état = { machine : int ; charge : int ; tâches : int list } ;;
```

Le champ `machine` donne le numéro de la machine (entre 1 et m). Le champ `tâches` donne la liste des numéros des tâches qui lui ont été affectées (la plus récente en tête) ; enfin, le champ `charge` donne la somme des durées de ces tâches. Le programme générera une liste des états des machines, triée par charge croissante ; ainsi, la machine en tête de liste sera celle à laquelle on affectera la prochaine tâche.

Question 16 • Rédigez en Caml une fonction de signature

```
mise_à_jour_état : int * int -> état list -> état list
```

spécifiée comme suit : `mise_à_jour_état (j, d) v` affecte la tâche numéro j , de durée d , à la machine qui est en tête de la liste v , puis réordonne cette liste.

Question 17 • Rédigez en Caml une fonction de signature

```
ordonnancement_séquentiel : int -> int list -> int list
```

spécifiée comme suit : `ordonnancement_séquentiel m d` applique l'algorithme d'ordonnancement séquentiel ; la fonction rend une liste a de même longueur que d , et telle que a_i indique à quelle machine doit être assignée la tâche d_i .

Question 18 • Rédigez en Caml une fonction de signature

```
ordonnancement_séquentiel_décroissant : int -> int list -> int list
```

spécifiée comme suit : `ordonnement_séquentiel_décroissant m d` applique l'algorithme d'ordonnement séquentiel décroissant à l'instance (m, d) . Le résultat suit la même spécification que pour la fonction précédente.

Question 19 • Rédigez en Caml une fonction de signature

```
qualité : int -> int list -> unit
```

spécifiée comme suit : `qualité m d` affiche les renseignements suivants :

- durée requise par l'ordonnement séquentiel décroissant ;
- minorant établi à la question 2 ;
- rapport entre ces deux valeurs (exprimé avec deux décimales).

2.5 Annexe : conseils de programmation

► Vous pourrez utiliser les fonctions Caml suivantes :

- `filtre` et `intervalle`, décrites en classe ;
- `map`, `combine`, `it_list`, `fst` et `snd` qui font partie de la bibliothèque Caml ;
- `sort__sort` qui fait également partie de la bibliothèque Caml.

Les signatures de ces fonctions sont rappelées ci-dessous :

```
intervalle: int -> int -> int list = <fun>
filtre : ('a -> bool) -> 'a list -> 'a list = <fun>
map: ('a -> 'b) -> 'a list -> 'b list = <fun>
combine: 'a list * 'b list -> ('a * 'b) list = <fun>
it_list: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
fst: 'a * 'b -> 'a = <fun>
snd: 'a * 'b -> 'b = <fun>
sort__sort: ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

► En ce qui concerne le type produit état, le fragment de programme suivant montre son utilisation :

```
let état_initial m = {machine = m; charge = 0; tâches = []} ;;

let ajoute_tâche (j,d) e =
  {machine = e.machine; charge = e.charge + d; tâches = d::e.tâches} ;;

let e = état_initial 3 in
  let e = ajoute_tâche (3,27) e in
  let e = ajoute_tâche (5,43) e in
  print_int e.charge; print_newline(); print_int (list_length e.tâches) ;;
```

Lorsque l'on exécute ce programme, il affiche (sur deux lignes successives) les valeurs 70 (somme des durées des tâches affectées à la machine 3) et 2 (nombre de ces tâches).

► L'emploi de références et/ou de structures de données mutables est interdit.

FIN