

Option Informatique en Spé MP et MP*

Ordonnancement de tâches : le corrigé

Question 1 • La somme des durées des tâches est 45, donc on ne peut espérer faire moins que $45/3 = 15$. Cette valeur est réalisée en affectant (par exemple) les tâches 1, 2 et 6 à la machine m_1 , les tâches 3 et 5 à la machine m_2 et les tâches 4, 7, 8 et 9 à la machine m_3 .

Question 2 • $\tau_{min}(m, d)$ est au moins égal à la durée de la plus longue tâche : $\tau_{min}(m, d) \geq \max_{1 \leq i \leq n} d_i$. Par ailleurs, la somme des durées des tâches est au moins égale à $m \tau_{min}(m, d)$, donc $\tau_{min}(m, d) \geq \left\lceil \frac{d_1 + \dots + d_n}{m} \right\rceil$.

Conclusion : $\tau_{min}(m, d)$ est au moins égal à la plus grande de ces deux valeurs.

Question 3 • Les charges des trois machines sont respectivement 14, 18 et 13 ; donc $\tau_{seq}(m, d) = 18$.

Question 4 • Choisissons une tâche tardive ; notons d_k sa durée, μ_j la machine à laquelle elle est attribuée et δ la somme des durées des tâches exécutées jusque là par μ_j . Au moment où l'on attribue d_k à μ_j , les $m - 1$ autres machines sont encore en activité, donc $\tau_{min}(m, d) > \delta$. Par ailleurs, $\tau_{min}(m, d) \geq d_k$. Nous en déduisons $\tau_{seq}(m, d) = \delta + d_k < 2\tau_{min}(m, d)$.

Question 5 • Prenons $m = \max(1, \lceil 1/\varepsilon \rceil)$. Soit alors d la suite constituée de $m(m-1)$ tâches de durée 1, suivie d'une tâche de durée m . Nous aurons $\tau_{seq}(m, d) = 2m - 1$, car à la date $m - 1$ toutes les machines terminent de travailler, sauf une à laquelle on confie la dernière tâche. Par ailleurs, $\tau_{min}(m, d) = m$ car on peut allouer m tâches de durée 1 à chacune des $m - 1$ premières machines, et la tâche de durée m à la dernière. Concluons : $\frac{\tau_{seq}(m, d)}{\tau_{min}(m, d)} = 2 - \frac{1}{m} \geq 2 - \varepsilon$.

Question 6 • La machine m_1 exécute les tâches 1, 5 et 2 ; la machine m_2 exécute les tâches 8, 6 et 9 ; enfin, la machine m_3 exécute les tâches 3, 7 et 4. On trouve $\tau_{decr}(m, d) = 16$.

Question 7 • Comme $n > m$, nous n'avons à considérer que les cas $n = 3$ et $n = 4$.

- Cas $n = 3$: si $d_1 = d_2$, le résultat est clair. Sinon, m_1 reçoit d_1 , tandis que m_2 reçoit d_2 et d_3 ; donc $\tau_{decr}(2, d) = d_2 + d_3$ ce qui est clairement optimal.

- Cas $n = 4$: ici encore, nous pouvons supposer $d_1 > d_2$. m_1 reçoit d_1 , puis m_2 reçoit d_2 et d_3 . À ce stade, deux cas se présentent :

1. si $d_2 + d_3 < d_1$, alors d_4 est elle aussi affectée à m_2 et le temps requis est $\max(d_1, d_2 + d_3 + d_4)$. Un ordonnancement qui affecterait une deuxième tâche à m_1 demanderait un temps au moins égal à $d_1 + d_4$, lequel majore à la fois d_1 et $d_2 + d_3 + d_4$.
2. sinon, d_4 est affectée à m_1 et le temps requis est $\max(d_1 + d_4, d_2 + d_3)$. Tout autre ordonnancement requiert un temps supérieur : si seule d_1 est affectée à m_1 , le temps requis est $d_2 + d_3 + d_4$; sinon, le temps requis est au moins égal à $d_1 + d_3$.

Question 8 • Prenons $d = (3, 3, 2, 2, 2)$: l'ordonnancement séquentiel décroissant affecte les tâches 1, 3 et 5 à m_1 (pour une durée totale de 7) et les tâches 2 et 4 à m_2 (pour une durée totale de 5) ; donc $\tau_{seq}(m, d) = 7$. Or $\tau_{min}(m, d) = 6$, réalisé en affectant les tâches 1 et 2 à m_1 et les trois autres tâches à m_2 .

Question 9 • Si les deux machines terminent en même temps, alors $\tau_{decr}(2, d) = \tau_{min}(2, d)$. Sinon, considérons une machine tardive ; notons c la durée de la dernière tâche exécutée sur cette machine, b la durée de l'avant-dernière tâche, et a la somme des durées des tâches précédentes : alors $a \geq b \geq c$ et $\tau_{decr}(2, d) = a + b + c$. À la date $a + b$, l'autre machine est encore au travail, donc la somme des durées est au moins $2a + 2b + c$, si bien que $\tau_{min}(2, d) > a + b$. Mais $c \leq \frac{a+b}{2}$, donc $c < \frac{\tau_{min}(2, d)}{2}$, d'où $\tau_{decr}(2, d) < \frac{3}{2} \tau_{min}$.

Question 10 • Reprenons la démarche de la question précédente : notons b la durée de l'avant-dernière tâche confiée à la machine tardive, et a la somme des durées des tâches qui l'ont précédée. Alors $a \geq b \geq c$. À la date $a + b$, les $m - 1$ autres machines sont au travail, donc chacune a travaillé pendant un temps au moins égal à $a + b \geq 2c$.

Question 11 • Considérons une des k machines qui n'ont reçu qu'une tâche : la durée de celle-ci était au moins égale à $2c$. Considérons maintenant une des $m - k - 1$ autres machines : elle a reçu au moins deux tâches, chacune de durée au moins égale à c , puisque la durée de la dernière tâche était de durée c exactement. Chaque machine tardive a reçu au moins trois tâches, de durée au moins égale à c . Ceci nous donne au moins $k + 2(m - k - 1) + 3 = 2m - k + 1$ tâches de durée au moins égale à c .

Question 12 • Montrons qu'aucun ordonnancement ne réalise une durée strictement inférieure à $3c$, en examinant la situation à la date $2c$: si l'on a lancé toutes les tâches de durée au moins égale à $2c$, elles ont occupé (ou occupent encore) k processeurs, qui ne pourront donc exécuter en plus une des $2m - k + 1$ autres tâches sans dépasser la date $3c$. Il nous reste $m - k$ processeurs, auxquels il faudrait confier $2m - 2k + 1$ tâches de durée au moins c : or on ne peut confier plus de deux de ces tâches à l'un d'eux sans dépasser la date $3c$. Donc m processeurs ne peuvent suffire.

Question 13 • Nous avons donc $\tau_{min}(m, d) > 3c$. Comme toutes les machines fonctionnent à la date $a + b$, nous avons $\tau_{min}(m, d) > a + b$. Alors $\tau_{decr}(m, d) = a + b + c < \frac{4}{3}\tau_{min}(m, d)$

Question 14 • Soit μ une machine tardive. Si μ n'a reçu qu'une tâche, la durée de celle-ci est $\max_{1 \leq j \leq n} d_j$, qui minore $\tau_{min}(m, d)$ (cf. la question 2): dans ce cas, $\tau_{decr}(m, d) = \tau_{min}(m, d)$.

• Examinons maintenant le cas où μ a reçu deux tâches, de durées a et b , avec $a \geq b$; nous allons montrer que l'affectation de la tâche de durée b à une autre machine μ' ne peut diminuer la durée totale; pour le voir, plaçons-nous à la date a : si μ' n'a reçu qu'une tâche, la durée de celle-ci est $a' \geq a$ et le résultat est clair. Sinon, μ' a reçu $q \geq 2$ tâches de durée au moins égale à b , avec $qb \geq a$; si on lui assigne en plus la tâche de durée b (et même si on la soulage de toutes les tâches de durée inférieure à b), μ' va travailler pendant un temps au moins égal à $(q + 1)b \geq a + b$, ce qui n'améliore rien. Conclusion: dans ce cas encore, $\tau_{decr}(m, d) = \tau_{min}(m, d)$.

Question 15 • Notons $p = \max(1, \lceil 1/\varepsilon \rceil)$. Nous utiliserons $m = 2p + 1$ processeurs; la liste des tâches d est constituée de:

- deux tâches de durée k , pour chaque $k \in \llbracket 2p + 1, 4p \rrbracket$;
- trois tâches de durée $2p$.

Pour $k \in \llbracket 1, m \rrbracket$, l'ordonnancement séquentiel décroissant affecte au processeur k des tâches de durées $4p + 1 - \lceil k/2 \rceil$ et $2p - 1 + \lceil k/2 \rceil$. Puis, à la date $6p$, il lui reste à affecter la troisième tâche de durée $2p$ à l'un quelconque des m processeurs; ainsi $\tau_{decr}(m, d) = 8p$. Il existe un ordonnancement plus économique, qui consiste à affecter des tâches de durées respectives $4p + 1 - \lceil k/2 \rceil$ et $2p + \lceil k/2 \rceil$ au processeur k , pour $k \in \llbracket 1, m - 1 \rrbracket$; les trois tâches restantes, de durée $2p$, sont affectées au processeur m . Avec cet ordonnancement, la durée est $6p + 1$, donc $\tau_{min}(m, d) \leq 6p + 1$. Ainsi $\frac{\tau_{decr}(m, d)}{\tau_{min}(m, d)} \geq \frac{8p}{6p + 1} = \frac{4}{3} - \frac{1}{3(6p + 1)} \geq \frac{4}{3} - \varepsilon$ puisque $3(6p + 1) > 18p \geq 1/\varepsilon$.

Question 16 • Pas de remarques particulières pour cette partie du programme, qui ne présentait aucune difficulté. L'algorithme d'insertion dans une liste triée fait partie du programme de première année.

```
let précède e1 e2 = e1.charge < e2.charge ;;

let rec insère e = fonction
| [] -> [e]
| t::q when précède e t -> e::t::q
| t::q -> t::(insère e q) ;;

let mise_à_jour_état (i,d) = fonction
| [] -> failwith "aucune machine!"
| {machine = m; charge = c; tâches = t}::q ->
  let e' = {machine = m ; charge = c+d ; tâches = i::t} in insère e' q ;;
```

Question 17 • Quelques explications: le programme demandé a été découpé en deux parties. La première comporte les éléments suivants:

- `créer_état_initial` crée l'état initial (scoop!): chaque machine est en attente;
- `créer_charge` crée une liste des couples (i, d_i) où i est un numéro de tâche et d_i sa durée;
- `répartir_charge` répartit la charge de travail entre les machines, en respectant l'algorithme d'ordonnement séquentiel.

```

let crée_état_initial m =
  let crée_machine j = {machine = j; charge = 0; tâches = []} in
  map crée_machine (intervalle 1 m) ;;

let crée_charge d = combine (intervalle 1 (list_length d),d) ;;

let rec maj e = fonction
  | [] -> e
  | t::q -> maj (mise_à_jour_état t e) q ;;

let répartit_charge m d =
  maj (crée_état_initial m) (crée_charge d) ;;

```

`répartit_charge` applique à l'état initial la fonction $\varphi_n \circ \varphi_{n-1} \circ \dots \circ \varphi_1$, où φ_i est la fonction qui modifie l'état en affectant la tâche numéro i à la première machine disponible (dans la liste). Le fragment de session Caml qui suit illustre le fonctionnement de `répartit_charge`; on notera que la liste des tâches affectées à une machine est présentée en ordre décroissant de numéros.

```

#répartit_charge 3 [8;27;5;12;19;9;17] ;;
- : état list =
  [{machine = 2; charge = 27; tâches = [2]};
   {machine = 1; charge = 27; tâches = [5; 1]};
   {machine = 3; charge = 43; tâches = [7; 6; 4; 3]}]

```

La deuxième partie exploite l'état final: nous disposons, pour chaque machine, de la liste (classée par ancienneté croissante) des tâches qui lui ont été confiées. Nous associons à chaque tâche le numéro de la machine à laquelle elle est affectée: ceci nous donne, pour la machine j , une liste de couples de la forme (i, j) où i est le numéro d'une tâche. Nous mettons «à plat» la liste de listes de couples ainsi obtenue, que nous trions par numéro de tâche; il ne reste plus qu'à projeter sur la deuxième composante pour avoir le résultat demandé.

```

let tri_1_up x y = fst x < fst y ;;

let numérote t_list j = map (fun x -> (x,j)) t_list ;;

let flat = it_list (prefix @) [] ;;

let exploite y =
  let f x = numérote x.tâches x.machine in
  map snd (sort__sort tri_1_up (flat (map f y))) ;;

let ordonnancement_séquentiel m d =
  exploite (répartit_charge m d) ;;

```

Question 18 • Il suffit, dans ce qui précède, de modifier la fonction `crée_charge` pour qu'elle trie la liste par durées décroissantes. Nous nous rendons compte qu'il suffisait de transmettre en paramètre une fonction de classement pour ne rien à avoir à refaire...

```

let tri_2_down x y = snd x > snd y ;;

let crée_charge_décroissante d =
  sort__sort tri_2_down (crée_charge d) ;;

let répartit_charge_décroissante m d =
  maj (crée_état_initial m) (crée_charge_décroissante d) ;;

let ordonnancement_séquentiel_décroissant m d =
  exploite (répartit_charge_décroissante m d) ;;

```

Question 19 • Une seule remarque pour cette dernière question : pour arrondir un quotient à l'entier immédiatement supérieur, nous utilisons la formule $\lceil p/q \rceil = \lfloor (p+q-1)/q \rfloor$.

```
let max_of_list l = it_list max (hd l) (tl l) ;;
let sum_of_list = it_list (prefix +) 0 ;;

let qualité m d =
  let r = répartit_charge_décroissante m d in
  let tosd = max_of_list (map (fun z -> z.charge) r) in
  let min1 = max_of_list d in
  let min2 = (m - 1 + sum_of_list d) / m in
  let borninf = max min1 min2 in
  printf__printf "ordonnancement séquentiel décroissant: %d" tosd;
  print_newline() ;
  printf__printf "borne inférieure: %d" borninf;
  print_newline() ;
  let rapport = float_of_int((100*tosd)/borninf) /. 100. in
  printf__printf "rapport: %f" rapport;
  print_newline() ;
;;
```

L'extrait d'une session Caml ci-dessous est intéressant :

```
#qualité 3 [8;27;5;12;19;9;17] ;;
ordonnancement séquentiel décroissant: 35
borne inférieure: 33
rapport: 1.060000
```

On peut réaliser la durée 34 avec l'affectation (1, 2, 2, 3, 3, 1, 1). On ne peut faire mieux : les tâches 2 et 3 doivent être affectées à une même machine ; puis la tâche 5 ne peut être associée qu'à la tâche 4.

Références bibliographiques

- ▶ L'idée de départ de ce sujet est le chapitre 10 du livre *Approximation algorithms*, de Vijay V. VAZIRANI (éd. Springer).
 - ▶ L'étude de ces questions d'ordonnancement remonte à l'article *Bounds for certain multiprocessing anomalies*, de Ronald. L. GRAHAM, paru dans le *Bell System Technical Journal* en 1966.
 - ▶ La NP-complétude du problème d'ordonnancement se prouve par réduction du problème de l'équipartition : étant donné une famille finie $(x_i)_{i \in I}$ de nombres, existe-t-il une partition $I = J \cup K$ telle que $\sum_{j \in J} x_j = \sum_{k \in K} x_k$.
- Il est clair que ce problème revient à chercher un ordonnancement optimal pour $(x, 2)$ et à regarder si le temps associé est la moitié du temps total.
- ▶ Merci à Évelyne LATRÉMOLIÈRE et Philippe ESPÉRET pour leur efficace participation.

FIN