**Composition d'Informatique (4 heures), Filière MP**
**(XULCR)**

## 1. L'épreuve

Il s'agissait dans ce problème de calculer des ordonnancements pour un ou plusieurs processeurs à partir de graphes de tâches. L'épreuve alternait des questions de programmation avec des questions relatives à la compréhension de la structure des chemins de dépendances dans ces graphes. Elle amenait le candidat à calculer des ordonnancements optimaux sur des graphes généraux pour un seul processeur, puis sur des classes spécifiques de graphes pour un nombre arbitraire de processeurs.

La partie 1 présentait le contexte du problème et introduisait les concepts nécessaires à sa résolution. La partie 2 montrait quelques propriétés de base des graphes de tâches admettant des ordonnancements, en particulier l'acyclicité dont découle l'existence de racines et de feuilles. Elle amenait ensuite le candidat à programmer quelques fonctions basiques de dénombrement des tâches et des racines du graphe. La partie 3 introduisait le concept de chemins critiques amont d'une tâche dans un graphe, fournissait un algorithme de parcours permettant de calculer leur longueur et démontrait la terminaison et la correction de cet algorithme. Ensuite, une procédure d'ordonnancement fondée sur un tri des tâches par longueurs de chemins critiques amont croissantes était proposée, dont l'optimalité dans le cas d'un seul processeur était démontrée. La partie 4 introduisait le concept dual de chemin critique aval et proposait une autre procédure d'ordonnancement, introduite initialement par T.C. Hu, dans laquelle la mesure de priorité des tâches est donnée cette fois par les longueurs de leurs chemins critiques aval. L'optimalité de cette procédure était ensuite démontrée pour la classe des graphes dits arborescents entrants et ce pour un nombre arbitraire de processeurs. Enfin, la partie 5 concluait le propos avec quelques références historiques et bibliographiques.

## 2. Remarques générales

Il s'agissait de la première épreuve consécutive à la réforme du programme de l'option informatique. L'occasion a donc été saisie d'exploiter des nouveautés du programme, à savoir ici des graphes.

Les notes se répartissent selon le tableau suivant, avec une moyenne de 9,51 et un écart-type de 3,16. Le nombre de notes éliminatoires, c'est-à-dire inférieures à 2, est de 7.

| | | |
|---|---|---|
| $0 \leqslant N < 4$ | 32 | 3,30 % |
| $4 \leqslant N < 8$ | 282 | 28,70 % |
| $8 \leqslant N < 12$ | 439 | 44,80 % |
| $12 \leqslant N < 16$ | 216 | 22,00 % |
| $16 \leqslant N \leqslant 20$ | 12 | 1,20 % |
| Total | 981 | 100 % |
| Nombre de copies : 981 | | |
| Note moyenne : 9,51 | | |
| Écart-type : 3,16 | | |

Les notes se répartissent selon le tableau suivant, avec une moyenne de 9,91 et un écart-type de 3,08 (pour les candidats français de l'École polytechnique) :

**X - ENS**

| | | |
|---|---|---|
| $0 \leqslant N < 4$ | 17 | 2,26 % |
| $4 \leqslant N < 8$ | 184 | 24,44 % |
| $8 \leqslant N < 12$ | 351 | 46,61 % |
| $12 \leqslant N < 16$ | 191 | 25,37 % |
| $16 \leqslant N \leqslant 20$ | 10 | 1,33 % |
| Total | 753 | 100 % |
| Nombre de copies : 753 | | |
| Note moyenne : 9,91 | | |
| Écart-type : 3,08 | | |

Concernant les questions de programmation, les candidats doivent être conscients du fait qu'une réponse longue doit être expliquée en détail et que très souvent un programme très long contient un grand nombre d'erreurs. Découper un programme un peu long en plusieurs fonctions est une bonne initiative, mais appeler ces différentes fonctions aux, aux2, aux3, etc., ne l'est pas. Certains candidats utilisent une couleur différente pour écrire leurs programmes et cette initiative est très appréciée des correcteurs. Il en va de même pour le soin apporté à l'indentation du code, qui en facilite grandement la compréhension.

## 3. Commentaire détaillé

Pour chaque question, sont indiqués entre crochets le pourcentage de candidats ayant traité la question et le pourcentage de candidats ayant obtenu la totalité des points. Pour obtenir la note maximale, il était nécessaire de traiter le problème en entier. Les parties I et V ne contenaient pas de question.

## Partie II

**Question 1 [99% - 80%].** Question très facile et dans l'ensemble bien traitée.

**Question 2 [98% - 26%].** Question facile. La plupart des candidats ont identifié que l'absence de racine (ou de feuille) permettait la construction d'un chemin de dépendance de longueur arbitraire et aboutissait à une contradiction. La rédaction est néanmoins imprécise dans de nombreuses copies. Un certain nombre de candidats ont essayé de faire une preuve par récurrence sur le nombre de tâches dans le graphe de dépendance. L'utilisation de cette technique résultait en une preuve beaucoup plus complexe et peu de candidats ont mené ce raisonnement au bout sans erreur.

**Question 3 [99% - 68%].** Il s'agissait d'écrire deux fonctions relativement simples. Cette question a été bien traitée dans l'ensemble.

**Question 4 [99% - 93%].** La fonction demandée est assez similaire à la précédente mais beaucoup de candidats n'ont pas respecté le fait que le tableau devait être uniquement complété par `empty_task`, s'autorisant à insérer des tâches vides au milieu du tableau.

## Partie III

**Question 5 [99% - 78%].** Comme de nombreux candidats l'ont fait remarquer, l'énoncé contenait une coquille. Il manquait le graphe en argument de la fonction demandée. Cette question de programmation a été bien traitée dans l'ensemble.

**Question 6 [96% - 23%].** Des candidats ne distinguent pas le ramassage et l'étiquetage, réalisant les deux actions en une seule passe, ce qui est incorrect. Ils ont donc dû manquer, ou ignorer, l'indication située sur le début de la page suivante.

**Question 7 [98% - 16%].** Beaucoup de candidats oublient de montrer que $P_u \neq \emptyset$, se contentant de montrer que les longueurs des chemins amonts sont majorées.

**Question 8 [92% - 12%].** Relativement peu de candidats ont proposé une preuve par récurrence juste. Une erreur classique consistait à supposer que tous les prédécesseurs d'une tâche étiquetée $k$ étaient étiquetés par $k-1$.

**Question 9 [95% - 4%].** Il s'agissait ici d'une question en 3 parties, à savoir montrer deux implications et fournir un exemple. Beaucoup de candidats se sont contentés d'invoquer les questions 7 et 8 pour établir les 2 implications. Ces deux questions supposent que le graphe est acyclique et ne peuvent être utilisées que pour l'une des implications.

**Question 10 [83% - 30%].** Question facile mais pour laquelle les candidats ont souvent fait preuve de négligence dans la rédaction.

**Question 11 [89% - 12%].** La difficulté de cette question résidait principalement dans l'affichage par lots de $p$ et, en particulier, dans le bon placement des retours-chariots. Quelques candidats ont proposé une solution élégante de complexité linéaire.

**Question 12 [74% - 48%].** Pas de difficulté particulière. Cette question a été dans l'ensemble bien traitée par les candidats ayant proposé un code à la question 11.

**Question 13 [64% - 7%].** Bien que relativement facile, pratiquement aucun candidat n'a justifié correctement l'optimalité de l'ordonnancement obtenu en pensant à invoquer le fait que lors de l'étiquetage, on ne « saute » pas d'étiquette.

Certains candidats ont fait un contresens et ont tenté de justifier l'optimalité de la complexité de l'algorithme d'ordonnancement proposé.

**Question 14 [86% - 44%].** Comme plusieurs candidats l'ont fait remarquer, le but était plutôt d'appliquer l'algorithme 2 (et non l'algorithme 1). Cependant, les candidats ayant uniquement appliqué l'algorithme 1, correctement, n'ont pas été sanctionnés.

Cette question a été bien traitée mais beaucoup de candidats oublient de répondre à une partie de la question, à savoir fournir la durée totale d'exécution, ou se trompent lors de ce calcul.

## Partie IV

**Question 15 [80% - 74%].** Ici, aucun code n'était demandé et la plupart des candidats ont correctement traité cette question.

**Question 16 [66% - 23%].** Question relativement facile et dans le même esprit que la question 10. Mais là encore, beaucoup de copies manquent de rigueur.

**Question 17 [73% - 50%].** Il s'agissait d'appliquer l'algorithme d'ordonnancement de Hu. Cette question a été plutôt bien traitée. Plusieurs candidats ont noté avec raison que les ordonnancements obtenus ici avaient déjà pu servir de contre-exemples à la question 14.

**Question 18 [64% - 47%].** Question de programmation dont le code attendu était semblable à celui de la question 5. Dans l'ensemble, les candidats ayant traité cette question ont bien réussi.

**Question 19 [45% - 4%].** Beaucoup de candidats oublient de tester si la tâche est déjà dans l'état `Done`, se contentant du test `is_ready t`. Plusieurs candidats se sont contentés d'adapter légèrement le code de la question 11, sans réaliser qu'ici la liste des tâches prêtes à être affichées doit être recalculée après chaque affichage d'un bloc de tâches.

**Question 20 [31% - 4%].** Parmi les candidats ayant abordé cette question, peu d'entre eux ont bien identifié le fait qu'un appel à la fonction `is_ready t` avait un coût $O(n)$ et que la fonction permettant le calcul des tâches prêtes avait donc un coût $O(n^2)$.

**Question 21 [31% - 7%].** Parmi les candidats ayant abordé cette question, très peu ont démontré le résultat attendu, se contentant d'expliquer que, vue la forme du graphe, une tâche exécutée ne peut « libérer »qu'une tâche.

**Question 22 [36% - 25%].** Question facile puisqu'il s'agissait là encore d'appliquer l'algorithme de Hu. Cependant, peu de candidats ont finalement abordé cette question. Parmi les copies traitant cette question, on a vu beaucoup d'erreurs que l'on pourrait qualifier d'inattention.

**Question 23 [11% - 2%].** Question très difficile pour laquelle quelques candidats seulement se sont risqués à tenter une explication. Aucune preuve correcte n'a été donnée mais les points de cette question ont tout de même été attribués aux tentatives les plus élaborées.