Épreuve de Mathématiques-Informatique

(Épreuve commune aux ENS de Lyon et de Cachan)

(Durée : 3 heures)

L'usage de calculatrices électroniques de poche à alimentation autonome, non imprimantes et sans document d'accompagnement, est autorisé pour toutes les épreuves d'admissibilité, sauf pour les épreuves de français et de langues. Cependant, une seule calculatrice à la fois est admise sur la table ou le poste de travail, et aucun échange n'est autorisé entre les candidats.

Important

Les parties 2 et 3 sont indépendantes. En revanche, nous conseillons au candidat de bien se familiariser avec les différentes notions introduites dans la partie 1 avant d'aborder les deux autres parties, a priori plus difficiles. Nous signalons qu'il sera particulièrement tenu compte des questions comportant la mise en oeuvre d'algorithmes dans la notation du sujet. Ceux-ci seront décrits en utilisant les structures de données et fonctions définies dans l'annexe "Mise en oeuvre des algorithmes".

Introduction et terminologie

Ordonnancer des tâches, c'est déterminer les dates d'exécution de ces tâches et les ressources nécessaires à leur exécution. Les problèmes d'ordonnancements apparaissent dans des domaines très divers, comme par exemple en informatique pour la compilation de programmes (les tâches peuvent être des sous-programmes, les ressources des processeurs), dans l'industrie pour la gestion d'ateliers (les tâches peuvent être des pièces à fabriquer, les ressources des ouvriers ou des machines), dans l'administration pour la mise au point d'emplois du temps (les tâches peuvent être des réunions à planifier, les ressources des salles disponibles), etc.

Les tâches et les ressources peuvent être modélisées de multiples façons. Dans ce sujet, on s'intéressera à deux modèles très simples pour lesquels les tâches sont dites **non morcelables** (une fois qu'une tâche est commencée, elle doit être terminée sans interruption) et les ressources sont dites **renouvelables** (une ressource ne peut être utilisée que pour une tâche à la fois, mais devient disponible dès que cette tâche est terminée). De plus, on supposera que l'exécution d'une tâche ne nécessite qu'une seule ressource et que toutes les ressources sont équivalentes. Ainsi pour le cas de l'organisation de réunions, on ne souhaite pas interrompre une réunion, chaque réunion ne nécessite qu'une salle (et n'importe laquelle) et cette salle devient libre dès que la réunion se termine. Enfin, on supposera que les tâches sont liées par des contraintes dites de **précédence**, indiquant que certaines tâches ne peuvent pas commencer tant que d'autres n'ont pas été achevées. La formulation précise de ces contraintes sera détaillée ultérieurement.

En général, pour la plupart des modèles de tâches, trouver un ordonnancement de durée totale minimale est difficile : les seuls algorithmes connus sont de complexité[1] super-polynomiale et l'existence d'un algorithme de complexité polynomiale remettrait en cause certaines conjectures fondamentales de l'informatique. Dans ce sujet, on étudiera quelques cas particuliers pour lesquels il existe des algorithmes de complexité polynomiale fournissant soit des ordonnancements de durée totale minimale, soit des ordonnancements dont la durée totale est minimale à un facteur multiplicatif constant près.

1. Les notions de complexité (complexité polynomiale, super-polynomiale, etc.) utilisées dans ce sujet sont précisées dans l'annexe "Mise en oeuvre des algorithmes".

Définitions et notations

On représente les tâches et les contraintes entre tâches par une structure $G = (S, A)$ appelée **graphe de tâches** et définie par deux ensembles finis S et A où $A \subset S \times S$. Les éléments de S sont appelés **sommets** et les éléments de A sont appelés **arcs**. Les couples définissant les arcs sont ordonnés, c'est-à-dire que le couple (x, y) est différent du couple (y, x). Pour bien marquer cette orientation, on notera parfois $(x \to y)$ pour signifier $(x, y) \in A$.

On appelle **prédécesseur** (resp. **successeur**) de $x \in S$ tout sommet $y \in S$ tel que $y \to x$ (resp. $x \to y$). Un **chemin** P de G de longueur $k \geq 1$ est une séquence de k arcs de G, $(e_1, \ldots, e_k)$, de la forme $e_i = (x_i, x_{i+1})$. On dit que P est un chemin de x_1 à x_{k+1} ou que x_1 est l'**origine** de P et x_{k+1} l'**extrémité** de P. Un **cycle** est un chemin dont l'origine et l'extrémité sont identiques.

On attribue une valeur (appelée **poids**) à chacun des sommets et/ou à chacun des arcs. On note $\delta(x)$ le poids d'un sommet x et $\omega(e)$ le poids d'un arc e. Par extension, pour un chemin P, on note $\omega(P)$ la somme des poids des arcs de P et $\delta(P)$ la somme des poids des sommets de P : ainsi, si $P = (e_1, \ldots, e_k)$ et $e_i = (x_i, x_{i+1})$, $\omega(P) = \sum_{i=1}^{k} \omega(e_i)$ et $\delta(P) = \sum_{i=1}^{k+1} \delta(x_i)$.

Enfin, si X est un ensemble fini, on note $|X|$ son cardinal.

Partie 1. Contraintes de précédence et contraintes de ressources.

Dans les parties 1 et 2, on suppose que les tâches sont liées par des **contraintes de précédence** définies par un graphe de tâches $G = (S, A)$ de la façon suivante : chaque élément x de S correspond à une tâche particulière qu'on identifie au sommet x lui-même. Le poids $\delta(x)$ de x, un entier strictement positif, est la **durée** de la tâche x. Un arc $x \to y$ de A indique que la tâche y ne peut pas commencer tant que la tâche x n'est pas terminée, ce qui se traduit par la contrainte :

$$[x \to y] \Rightarrow [d(y) \geq d(x) + \delta(x)] \tag{1}$$

où $d(x) \in \mathbf{N}$ est la date de début d'exécution (à déterminer) de x. Le poids d'un arc n'est pas défini. On suppose qu'on dispose de m ressources identiques; et on note $r(x)$ (à déterminer également) la ressource utilisée par x. $r(x)$ est un entier compris entre 1 et m. Les fonctions r et d sont liées par les **contraintes de ressources** suivantes :

$$[x \neq y \text{ et } r(x) = r(y)] \Rightarrow [d(x) + \delta(x) \leq d(y) \text{ ou } d(y) + \delta(y) \leq d(x)] \tag{2}$$

Déterminer un **ordonnancement** $\Omega = (d, r)$ d'un graphe de tâches G, c'est déterminer deux fonctions d et r satisfaisant les contraintes (1) et (2). La **durée totale** $D(\Omega)$ de $\Omega = (d, r)$ est la date la plus tardive de terminaison d'une tâche, c'est-à-dire : $D(\Omega) = \max_{x \in S}[d(x) + \delta(x)]$. On note $D_{\text{opt}}(m)$ la durée totale minimale d'un ordonnancement de G pour m ressources.

Question 1.1. *Étude d'un exemple.*

Considérons 5 tâches, T_1, T_2 et T_3 de durée 1, T_4 et T_5 de durée 2. La tâche T_1 précède les tâches T_2 et T_3, elles-mêmes précèdent la tâche T_4. La figure 1 décrit le graphe de tâches correspondant et un ordonnancement possible de durée totale 5 pour $m = 2$ ressources.

a) *Le graphe de tâches de la figure 1 a-t-il un cycle? Quel est la longueur du plus long chemin? Quels sont les chemins P de poids $\delta(P)$ maximal?*

b) *Pour le graphe de tâches de la figure 1, calculer $D_{\text{opt}}(1)$, $D_{\text{opt}}(2)$, $D_{\text{opt}}(3)$, puis $D_{\text{opt}}(m)$ pour tout entier $m \geq 3$. Dans chacun des cas, on décrira, par une table similaire à celle de la figure 1, un ordonnancement de durée totale minimale.*

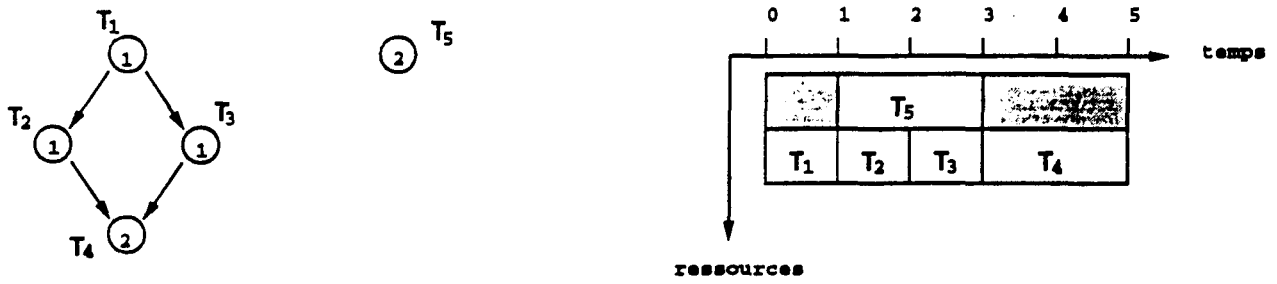

FIG. 1 - Un exemple de graphe de tâches et un ordonnancement possible.

Question 1.2. *Graphe de tâches acyclique et ordre total.*

a) Soit G un graphe de tâches. Montrer que s'il existe un ordonnancement de G, alors G est acyclique, c'est-à-dire sans cycles.

b) Montrer que si $G = (S, A)$ est acyclique (et S non vide), l'ensemble des sommets de G sans prédécesseurs est non vide. En déduire un algorithme (cf Annexe) de complexité $O(|S| + |A|)$ permettant de représenter l'ensemble S sous forme d'une liste L dont l'ordre total $\preceq_L$ associé vérifie $(x \to y) \Rightarrow (x \preceq_L y)$. On pourra utiliser, pour chaque sommet x, une variable comptant le nombre de prédécesseurs de x non encore placés dans la liste L.

On suppose dorénavant, et jusqu'à la fin de la partie 2, que le graphe de tâches G est **acyclique**.

Question 1.3. *Cas particuliers polynomiaux : $m = 1$ et $m \geq |S|$.*

a) Soit Ω un ordonnancement de G. Montrer que, pour tout chemin P de G, $D(\Omega) \geq \delta(P)$. Montrer que $D(\Omega) \geq \frac{1}{m} \sum_{x \in S} \delta(x)$. Interpréter ce résultat comme une inégalité portant sur la surface d'un certain rectangle.

b) Quelle est la valeur de $D_{opt}(1)$? De $D_{opt}(m)$ lorsque $m \geq |S|$? Justifier en donnant pour chacun des cas un algorithme de complexité polynomiale fournissant un ordonnancement de durée totale minimale, c'est-à-dire, pour chaque sommet x, des valeurs $d(x)$ et $r(x)$ adéquates.

Étant donné un ordonnancement $\Omega = (d, r)$, on dit qu'une tâche x est **prête à l'instant** t ($t \in \mathbb{N}$) si $d(x) \geq t$ et si, pour tout prédécesseur y de x, y est terminé à l'instant t, c'est-à-dire $d(y) + \delta(y) \leq t$. On dit qu'une ressource r est **libre à l'instant** t ($t \in \mathbb{N}$) si quelque soit la tâche x telle que $r(x) = r$, on a soit $d(x) + \delta(x) \leq t$, soit $d(x) > t$. On suppose que les sommets de G sont donnés sous forme d'une liste L dont l'ordre total $\preceq_L$ associé est **compatible**, c'est-à-dire vérifie, comme à la question 1.2.b., $(x \to y) \Rightarrow (x \preceq_L y)$. On appelle alors **ordonnancement de liste**, un ordonnancement vérifiant les deux propriétés suivantes :

- s'il existe une ressource libre à l'instant t alors l'ensemble des tâches x prêtes à l'instant t et non exécutées à l'instant t (c'est-à-dire telles que $d(x) \neq t$) est vide.
- si x et y sont deux tâches prêtes à l'instant t et telles que $x \preceq_L y$, alors $d(x) \leq d(y)$.

Question 1.4. *Mise au point d'un ordonnancement de liste.*

a) Donner, pour le graphe de tâches de la figure 1, le nombre d'ordres $\preceq_L$ compatibles et tous les ordonnancements de liste possibles si $m = 2$. Même question si on remplace la tâche T_5 de durée 2 par deux tâches T_6 et T_7 de durée 1, telles que T_6 précède T_7.

b) Donner un algorithme de complexité $O(|S|^2+|A|)$ qui, étant donné un ordre $\preceq_L$ compatible, permet de construire un ordonnancement de liste. Pour cela, on ordonnancera les sommets les uns après les autres en s'aidant de deux listes L_1 et L_2: L_1 sera la liste des sommets non encore ordonnancés, et L_2 sera une liste de m sommets, comprenant pour chaque ressource r, le dernier sommet x déjà ordonnancé et affecté à la ressource r. On supposera, pour simplifier, que $m \leq |S|$ et que la liste L_2 est initialisée à une liste de m sommets fictifs $(x_i)_{1\leq i\leq m}$ tels que $\delta(x_i) = 1$, $d(x_i) = -1$ et $r(x_i) = i$.

Partie 2. Quelques propriétés des ordonnancements de liste.

Un ordonnancement de liste n'est pas toujours de durée totale minimale, c'est le cas par exemple pour le graphe de tâches de la figure 1 pour $m = 2$. L'objet de cette partie est de montrer que, néanmoins, les ordonnancements de liste ont des propriétés intéressantes: ils sont de durée totale minimale pour certains graphes et si l'ordre $\preceq_L$ compatible est bien choisi, et ils sont, dans le cas général, de durée totale minimale à un facteur multiplicatif constant près.

On étudie tout d'abord le cas particulier où le graphe de tâches $G = (S, A)$ est une **anti-arborescence** (tout sommet x de G a au plus un successeur) et où toutes les tâches ont **même durée** (pour simplifier les notations, on suppose que $\delta(x) = 1$ pour tout $x \in S$). On note niveau(x) la longueur maximale d'un chemin d'origine x (on pose niveau$(x) = 0$ si x n'a pas de successeur). On note $h = \max_{x\in S}[\text{niveau}(x)]$ le niveau maximal dans G.

Question 2.1. *Extension des résultats des questions 1.2.b. et 1.3.a.*

a) Expliquer comment on pourrait construire un ordre total $\preceq_L$ sur S tel que $[\text{niveau}(x) \geq \text{niveau}(y)] \Rightarrow [x \preceq_L y]$. Montrer qu'un tel ordre $\preceq_L$ est compatible, c'est-à-dire vérifie également $(x \to y) \Rightarrow (x \preceq_L y)$.

b) En s'inspirant de la question 1.3.a., montrer que $\forall i,\ 0 \leq i \leq h,\ D_{opt}(m) \geq \frac{|N_i|}{m} + i$ où $N_i = \{x \in S \mid \text{niveau}(x) \geq i\}$ est l'ensemble des sommets de niveau supérieur ou égal à i.

Soit $\Omega = (d, r)$ un ordonnancement de liste construit à partir d'un ordre total $\preceq_L$ vérifiant la condition $[\text{niveau}(x) \geq \text{niveau}(y)] \Rightarrow [x \preceq_L y]$. Pour $0 \leq t < D(\Omega)$, on note S_t l'ensemble des tâches exécutées à l'instant t pour l'ordonnancement Ω : $S_t = \{x \in S \mid d(x) = t\}$.

Question 2.2. *Optimalité pour les anti-arborescences à durées unitaires (algorithme de Hu).*

On suppose pour commencer qu'il existe un entier t, $0 \leq t < D(\Omega)$ tel que S_t contienne exactement m tâches de même niveau. On note k le plus grand de ces entiers et L le niveau des tâches de S_k. On suppose de plus que $k < D(\Omega) - 1$ et on note L' le niveau maximal des tâches x non encore exécutées à l'instant k, c'est-à-dire telles que $d(x) > k$.

a) En étudiant le niveau maximal d'un élément de S_t, pour $k < t < D(\Omega)$, montrer que $D(\Omega) = k + L' + 2$.

b) Montrer que pour tout entier t, $0 \leq t \leq k$, $|S_t| = m$. On utilisera le fait que, G étant une anti-arborescence, tout sommet a au plus un successeur.

c) Déduire de b) et c) que $D(\Omega) = D_{opt}(m)$. Montrer que si $k = D(\Omega) - 1$ (auquel cas L' n'est pas défini) ou si k n'existe pas, le résultat $D(\Omega) = D_{opt}(m)$ reste vrai.

On revient à présent au cas général, celui d'un graphe de tâches quelconque acyclique, pour lequel chaque sommet x a un poids $\delta(x)$, entier strictement positif quelconque.

Question 2.3. *Garantie de performances pour les algorithmes de liste (borne de Coffman).*

Soit x une tâche terminale d'un ordonnancement de liste $\Omega = (d, r)$, c'est-à-dire une tâche telle que $d(x) + \delta(x) = D(\Omega)$.

a) Soit $t < d(x)$ le plus grand entier, s'il existe, tel qu'une ressource soit libre à l'instant t. Montrer qu'il existe une tâche y, en cours d'exécution à l'instant t, c'est-à-dire telle que $d(y) \leq t < d(y) + \delta(y)$, et telle qu'il existe un chemin dans G de y à x.

*b) Déduire de **a)** que si I est l'ensemble des entiers t, $0 \leq t < D(\Omega)$, tels qu'il existe une ressource libre à l'instant t, alors il existe un chemin P_0 dans G tel que $|I| \leq \delta(P_0)$.*

c) Démontrer l'inégalité $mD(\Omega) \leq \sum_{x \in S} \delta(x) + (m-1)\delta(P_0)$. On pourra, comme à la question 1.3, interpréter ce résultat comme une inégalité portant sur la surface d'un certain rectangle. En déduire finalement l'inégalité de Coffman: $D_{opt}(m) \leq D_{liste}(m) \leq (2 - \frac{1}{m}) D_{opt}(m)$.

Partie 3. Problème cyclique sans contraintes de ressources : quelques propriétés des ordonnancements cycliques.

Dans cette partie, on généralise le modèle de précédence des parties 1 et 2 à des ensembles de tâches **infinis**. On suppose que les tâches sont définies par un graphe de tâches $G = (S, A)$ de la façon suivante : chaque élément x de S correspond, non plus à une tâche particulière, mais à une infinité de **tâches génériques** indexées par $\mathbf{N}$, qu'on notera (x, i) pour $i \in \mathbf{N}$. Pour chaque x, les tâches (x, i) sont toutes de même durée $\delta(x)$, entier strictement positif. Chaque arc $e = (x, y)$ de A possède un poids $\omega(e)$, entier positif ou nul, qui indique que la tâche $(y, i + \omega(e))$ ne peut pas commencer tant que la tâche (x, i) n'est pas terminée, ce qui se traduit par la contrainte :

$$[e = (x, y) \in A] \Rightarrow [\forall i \in \mathbf{N},\ d(y, i + \omega(e)) \geq d(x, i) + \delta(x)] \tag{3}$$

où $d(x, i) \in \mathbf{N}$ est la date de début d'exécution (à déterminer) de la tâche (x, i). Au contraire des deux parties précédentes, on s'intéresse ici, pour simplifier, à un problème d'ordonnancement **sans** contraintes de ressources. Déterminer un **ordonnancement** Ω, c'est donc déterminer une fonction d_Ω de $S \times \mathbf{N}$ dans $\mathbf{N}$, satisfaisant la contrainte (3).

On définit la quantité $D(\Omega, n) = \max\{d_\Omega(x, i) + \delta(x) \mid x \in S,\ 0 \leq i \leq n\}$. $D(\Omega, n)$ est la durée totale de l'ordonnancement Ω lorsqu'on ne s'intéresse qu'aux tâches (x, i) pour $0 \leq i \leq n$.

Question 3.1. *Fréquence asymptotique.*

Montrer que $P(\Omega) = \lim_{N \to \infty} \left[\sup_{n \leq N} \dfrac{D(\Omega, n)}{n} \right]$ existe dans $\mathbb{R} \cup \{+\infty\}$.

$P(\Omega)$ est la **fréquence asymptotique** de Ω. On note P_{opt} la fréquence asymptotique minimale d'un ordonnancement de G.

Question 3.2. *Borne inférieure pour P_{opt}.*

a) Montrer que, si G a un cycle C tel que $\omega(C) = 0$, il n'existe pas d'ordonnancement de G.

b) Soit C un cycle de G tel que $\omega(C) \neq 0$. En étudiant la durée totale pour les multiples de $w(C)$, c'est-à-dire $D(\Omega, n\omega(C))$, montrer que $P(\Omega) \geq \dfrac{\delta(C)}{\omega(C)}$.

On suppose dorénavant que G a au moins un cycle et que $\omega(C) > 0$ pour tout cycle C de G. On note $\rho(C) = \frac{\delta(C)}{\omega(C)}$ pour tout cycle C et on pose $\rho = \sup\{\rho(C) \mid C \text{ cycle de } G\}$. L'objet des questions suivantes est de montrer que $\rho < +\infty$ et de montrer qu'on peut construire un ordonnancement de fréquence asymptotique égale à ρ.

Question 3.3. *Importance des cycles élémentaires, c'est-à-dire des cycles dont seules l'origine et l'extrémité sont identiques.*

a) Montrer que si a, b, c et d sont des réels strictement positifs, alors $\frac{a+b}{c+d} \leq \max\left(\frac{a}{c}, \frac{b}{d}\right)$.

b) Soit C un cycle de G de longueur $> |S|$. Montrer qu'on peut construire un cycle C' de longueur strictement plus petite que la longueur de C et tel que $\rho(C') \geq \rho(C)$.

c) En déduire que $\rho < +\infty$ et qu'il existe un cycle C de longueur $\leq |S|$ tel que $\rho = \rho(C)$.

On appelle ordonnancement **cyclique**, un ordonnancement tel que $d_\Omega(x, i) = \lambda i + c(x)$ où $\lambda \in \mathbf{N}$ et pour tout $x \in S$, $c(x) \in \mathbf{N}$. On note un tel ordonnancement $\Omega = (\lambda, c)$.

Exemple La figure 2 représente un graphe de tâches cyclique à 4 sommets : $(T_j)_{1 \leq j \leq 4}$. Les poids des sommets sont indiqués dans les cercles, les poids des arcs le long des flèches. La table de droite fournit un ordonnancement cyclique $\Omega = (\lambda, c)$ pour lequel $\lambda = 6$, et $c(T_1) = c(T_3) = 0$, $c(T_2) = 2$, $c(T_4) = 3$. Ω a une forme particulière : pour tout $i \in \mathbf{N}$ et pour tous les sommets x et y du graphe, (x, i) se termine avant que $(y, i+1)$ ne commence. Attention, ceci n'est pas une propriété générale.

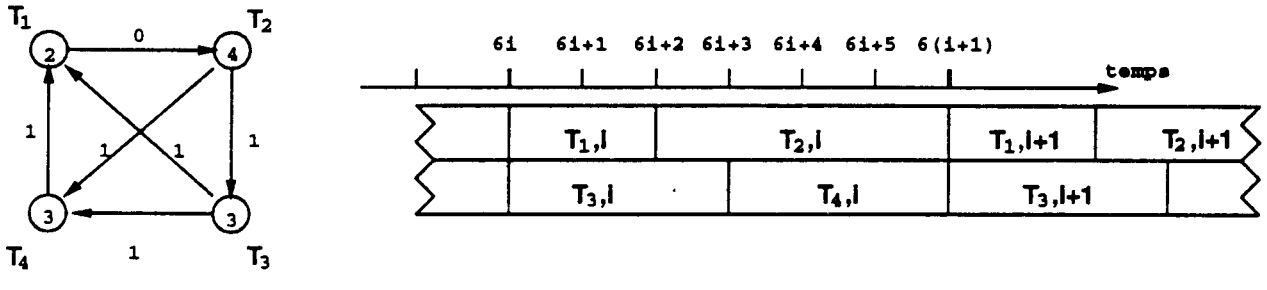

FIG. 2 - *Un exemple de graphe et un ordonnancement cyclique possible.*

Question 3.4. *Existence d'ordonnancements cycliques.*

Étant donné $\lambda \in \mathbb{Q}$, on définit, pour tout arc $e = (x, y) \in A$, la quantité $\omega_\lambda(e) = \delta(x) - \lambda\omega(e)$.

a) Pour tout ordonnancement cyclique $\Omega = (\lambda, c)$, calculer $P(\Omega)$ et exprimer la contrainte (3) en fonction de ω_λ et de c. Vérifier qu'on a bien $P(\Omega) \geq \rho$.

b) Soit $\lambda \in Q$ tel que $\omega_\lambda(C) \leq 0$ pour tout cycle C de G. Pour tout $x \in S$, on définit la quantité $dist(x) = \max(0, \sup\{\omega_\lambda(P) \mid P \text{ chemin d'extrémité } x\})$. Montrer que $dist(x) < +\infty$ et que pour tout $e = (x, y) \in A$, $dist(y) \geq dist(x) + \omega_\lambda(e)$.

c) Montrer que pour tout entier $\lambda \geq \rho$, il existe un ordonnancement cyclique de fréquence asymptotique égale à λ. Pour l'exemple du graphe de tâches de la figure 2, construire un ordonnancement cyclique $\Omega = (\lambda, c)$ tel que $\lambda = 5$.

d) Lorsque ρ n'est pas un entier, comment peut-on définir un ordonnancement (non cyclique) de fréquence asymptotique égale à ρ?

Question 3.5. *Construction d'ordonnancements cycliques.*

On note $dist(x)[i] = \max(0,\ \max\{\omega_\lambda(P) \mid P$ *chemin de longueur* $\leq i$ *d'extrémité* $x\})$.

a) Soit $x \in S$. *Exprimer* $dist(x)[i+1]$ *en fonction de* $dist(x)[i]$ *et des valeurs* $dist(y)[i]$ *des prédécesseurs* y *de* x. *En déduire un algorithme permettant de calculer* $dist(x)[i]$ *pour tout* $x \in S$, i *étant fixé.*

b) Donner un algorithme de complexité polynomiale qui permet de répondre à la question "existe-t-il dans G *un cycle* C *tel que* $\omega_\lambda(C) > 0$*?" et qui, si la réponse est non, permet de calculer* $dist(x)$ *pour tout* $x \in S$. *Préciser sa complexité.*

c) On note $\delta_{\max} = \max_{x \in V} \delta(x)$. *Donner un algorithme de complexité polynomiale en* $|S|$, $|A|$ et $\log(\delta_{\max})$ *permettant de calculer* $\overline{\rho}$, *le plus petit entier supérieur ou égal à* ρ.

d) Comment pourrait-on calculer ρ *lorsque* ρ *n'est pas entier?*

Annexe: mise en oeuvre des algorithmes

On décrira tous les algorithmes demandés en utilisant les structures et fonctions suivantes.

Une **liste** L de taille n est un ensemble de n éléments ordonnés par un ordre total $\preceq_L$. On manipulera les éléments d'une liste à l'aide des fonctions suivantes :

Taille(L) désigne le nombre d'éléments de la liste L.

Vide() désigne une liste sans élément.

Extraire_min(L) (resp. **Extraire_max**(L)) retourne le plus petit (resp. plus grand) élément x, au sens de $\preceq_L$, de la liste L. La liste L est modifiée en la liste $L' = L \setminus \{x\}$. L'ordre $\preceq_{L'}$ est l'ordre $\preceq_L$ induit sur L' : $y \preceq_{L'} z \Leftrightarrow y \preceq_L z$ pour tout $(y, z) \in L' \times L'$.

Insérer_max(L, x) (resp. **Insérer_min**(L, x)) modifie la liste L en la liste $L' = L \cup \{x\}$. L'ordre $\preceq_{L'}$ est l'ordre $\preceq_L$ étendu à L' en considérant x supérieur (resp. inférieur) à tous les autres : $y \preceq_{L'} z \Leftrightarrow y \preceq_L z$ pour tout $(y, z) \in L \times L$ et $y \preceq_{L'} x$ (resp. $x \preceq_{L'} y$) pour tout $y \in L$.

Si la liste L est l'ensemble vide, les fonctions Extraire_min(L) et Extraire_max(L) ne sont pas définies. Si x appartient déjà à L, les fonctions Insérer_max(L, x) et Insérer_min(L, x) ne sont pas définies.

On manipulera un **graphe de tâches** $G = (S, A)$ à l'aide des fonctions suivantes :

Sommet(G) désigne une liste dont les éléments sont les sommets de G.

Si x est un sommet de G, **Arc_Succ**(x) désigne une liste dont les éléments sont les arcs de G de la forme $e = (x, y)$, **Arc_Pred**(x) désigne une liste dont les éléments sont les arcs de G de la forme $e = (y, x)$ et **Poids_Som**(x) désigne le poids de x.

Si e est un arc de x à y, c'est-à-dire de la forme $e = (x, y)$, **Origine**(e) désigne le sommet x, **Extrem**(e) désigne le sommet y et **Poids_Arc**(e) désigne le poids de e.

Le candidat pourra définir des fonctions similaires aux fonctions Poids_Som et Poids_Arc permettant de manipuler des quantités relatives aux sommets ou aux arcs, par exemple pour calculer pour chaque sommet x les valeurs $d(x)$ et $r(x)$, ou toute autre quantité que le candidat jugera nécessaire.

L'algorithme suivant donne un exemple d'utilisation des fonctions permettant de manipuler listes et graphes. À partir d'un graphe $G = (S, A)$, il fournit, sous forme d'une liste, l'ensemble des sommets possédant une boucle, c'est-à-dire l'ensemble des sommets $x \in S$ tel que $(x, x) \in A$.

```
Algorithme Boucle
Entrée: Graphe G
Sortie: Liste L_b
début
    L_b = Vide()
    L_s = Sommet(G)
    T_s = Taille(L_s)
    pour i = 1 à T_s faire
        x = Extraire_min(L_s)
        Insérer_max(L_s,x) /* facultatif ici, permet de lire la liste sans la vider */
        L_a = Arc_Succ(x)
        T_a = Taille(L_a)
        pour j = 1 à T_a faire
            e = Extraire_min(L_a)
            Insérer_max(L_a,e) /* Idem, reconstruction pour réutilisation éventuelle */
            Si Extrem(e) = x alors Insérer_max(L_b,x)
        finpour
    finpour
    Retourner L_b
fin
```

La **complexité** C d'un algorithme est définie comme le nombre d'appels aux fonctions décrites précédemment et permettant de manipuler listes et graphes. Cette complexité est en général fonction de plusieurs paramètres $(p_i)_{1\leq i\leq n}$ définissant la taille des données de l'algorithme. On dit que la complexité est **polynomiale** s'il existe un polynôme P à n variables tel que $C = O(P)$, c'est-à-dire s'il existe une constante $K > 0$ telle que C, exprimé en fonction de $p_1, \ldots, p_n$, vérifie $C(p_1, \ldots, p_n) \leq KP(p_1, \ldots, p_n)$ lorsque les paramètres p_i sont suffisamment grands. Dans le cas contraire, la complexité de l'algorithme est dite super-polynomiale.

Dans l'ensemble du sujet, la taille d'un graphe sera représentée par les paramètres $|S|$ et $|A|$, et également, lorsque cela s'avérera nécessaire (c'est le cas à la question 3.5.c), par le paramètre $\log(\delta_{\max})$ où $\delta_{\max}$ est le poids maximal d'un sommet de G. Par exemple, l'algorithme Boucle donné ci-dessus est de complexité $O(|S| + |A|)$, donc de complexité polynomiale.

www.tunisie-etudes.info

Ce document a été téléchargé depuis
www.tunisie-etudes.info

Des documents gratuits, devoirs, examens, cours, exercices, corrigés... Ainsi que toute une rubrique pour vous aider à trouver un emploi sans oublier les avis de concours en direct

Notre page Twitter :
http://www.twitter.com/TunisieEtudes
Notre page FaceBook :
http://www.facebook.com/TunisieEtudes



Merci d'avoir choisi www.tunisie-etudes.info
Bonne lecture et bon travail