C31211

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94230 CACHAN

Concours d'admission en **3ème année**
Informatique
Session 2011

INFORMATIQUE 1

Durée : **5 heures**

Aucun document n'est autorisé
Aucun dictionnaire n'est autorisé
L'usage de toute calculatrice est interdit

Des chiffres entre parenthèses indiquent la difficulté des questions. Le but est d'aider le candidat à apprécier le niveau des questions et non de le décourager à s'y atteler. Il ne faut par contre pas y voir un barème au sens strict du terme.

Première partie

Réseaux de tri

Cette partie étudie les réseaux de tri. Il s'agit d'une classe de circuits pour trier une séquence de données numériques à l'aide d'un seul type d'élément logique : le comparateur.

Les réseaux de tri représentent également un exemple de calcul parallèle. Alors que tout algorithme de tri séquentiel pour n éléments est de complexité $\Omega(n \log n)$, un calcul parallèle peut être plus efficace : il est possible de construire un réseau de profondeur $O(\log n)$ où la profondeur correspond à la durée du passage des données à travers le réseau. Cependant, la construction optimale est très complexe. Le but de cette partie est moins ambitieux : on va construire des réseaux de profondeur $\log^2 n$.

On commence par quelques définitions et questions basiques mais utiles pour le reste de cette partie, par exemple le principe 0-1. Ensuite on étudie le tri d'une classe de séquences dites bitoniques. Cette étude mène à une solution générale.

Un réseau de tri est un circuit avec une séquence de n entrées en nombres entiers (pour un n quelconque) et une séquence de n sorties. Le réseau est composé de *comparateurs* reliés par des câbles. Un comparateur a deux entrées et deux sorties. La sortie "haute" émet le minimum et la sortie "basse" émet le maximum des deux entrées. Un réseau de tri consiste en n lignes connectées par des comparateurs. Chaque ligne est composée d'un ou plusieurs câbles reliant les comparateurs.

La figure 1 (a) montre la représentation graphique d'un seul comparateur, et (b) montre un réseau de tri pour $n = 4$ avec des entrées et les sorties correspondantes.

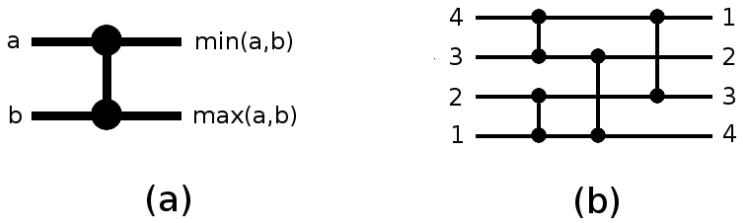

FIGURE 1 – (a) Un comparateur. (b) Exemple d'un réseau de tri.

Un réseau de tri est dit *correct* pour une séquence d'entrées si les sorties correspondantes sont triées dans l'ordre croissant. Par exemple, le réseau de la figure 1 (b) est correct pour la séquence $4, 3, 2, 1$.

Un réseau est simplement dit correct s'il l'est pour toutes les séquences d'entrées.

Question 1 (1) Trouver une séquence pour laquelle le réseau de la figure 1 (b) n'est pas correct.

Question 2 (2) Construire un réseau de tri correct pour $n = 4$.

La *taille* d'un réseau est définie comme le nombre de comparateurs dans le réseau.

Question 3 (3) Quelle est la taille minimale d'un réseau de tri pour $n = 4$? Justifier votre réponse.

Une fonction f est dite *croissante* si $f(x) \leq f(y)$ pour toute paire vérifiant $x \leq y$. Elle est dite *décroissante* si $f(x) \geq f(y)$ pour toute paire vérifiant $x \leq y$.

Question 4 (2) Soit f croissante. Montrer que si un réseau de tri transforme la séquence $x_1, \ldots, x_n$ en $y_1, \ldots, y_n$, alors $f(x_1), \ldots, f(x_n)$ sera transformée en $f(y_1), \ldots, f(y_n)$.

Une séquence est *binaire* si elle est composée uniquement de zéros et de uns.

Question 5 (2) *(Principe 0-1)* Montrer qu'un réseau de tri est correct pour toute séquence d'entrées si et seulement si il est correct pour toute séquence binaire.

Pour les quatre questions suivantes, prouver ou réfuter l'assertion. Il ne suffit pas de répondre vrai ou faux, il faut justifier votre réponse.

Question 6 (2) Vrai ou faux : Dans tout réseau correct il existe au moins un comparateur reliant chaque paire de lignes voisines.

Question 7 (2) Vrai ou faux : Tout réseau ayant des comparateurs entre toutes les paires de lignes est correct.

Question 8 (1) Vrai ou faux : Un réseau correct le reste si on ajoute un comparateur à la fin du réseau.

Question 9 (3) Vrai ou faux : Un réseau correct le reste si on ajoute un comparateur n'importe où dans le réseau.

Dans ce qui suit, on suppose que n est une puissance de 2, c'est à dire $n = 2^k$ pour un $k \geq 0$ quelconque.

Une séquence est dite *bitonique* si elle est binaire et s'il y a au maximum deux paires d'entrées voisines qui sont différentes. Par exemple, les séquences $0, 0, 1, 0$ ou $1, 1, 0, 1$ ou $0, 0, 1, 1$ ou $1, 1, 1, 1$ sont toutes bitoniques, mais $1, 0, 1, 0$ ne l'est pas. (Une séquence bitonique est donc soit croissante, soit décroissante, soit croissante puis décroissante, soit décroissante puis croissante.) Un réseau est dit bitonique s'il est correct pour toute séquence bitonique.

Un *demi-nettoyeur* est un réseau de tri avec $n/2$ comparateurs qui connectent la i-ème ligne et la $(i + n/2)$-ème ligne, pour $1 \leq i \leq n/2$, comme le montre la figure 2.

Question 10 (4) Soit $y_1, \ldots, y_n$ le résultat du demi-nettoyeur sur une séquence bitonique $x_1, \ldots, x_n$. Montrer que
- pour tout $i \in \{1, \ldots, n/2\}$ et $j \in \{n/2 + 1, \ldots, n\}$ on a $y_i \leq y_j$;
- les deux séquences $y_1, \ldots, y_{n/2}$ et $y_{n/2+1}, \ldots, y_n$ sont bitoniques.

La profondeur d'un câble est définie comme suit :
- si le câble est relié à une entrée, alors sa profondeur est 0 ;

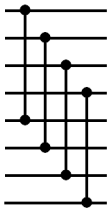

FIGURE 2 – Un demi-nettoyeur pour $n = 8$.

– si les câbles d'entrée d'un comparateur sont de profondeur p_1 et p_2, alors la profondeur des câbles de sortie est $1 + max(p_1, p_2)$.

La profondeur d'un réseau est la profondeur maximale de ses câbles. Exemple : La profondeur du réseau de la figure 1 (b) est 2, la profondeur du demi-nettoyeur est 1.

Question 11 (2) Décrire la construction d'un réseau bitonique pour $n = 2^k$ de profondeur k. Donner le résultat de cette construction pour $n = 8$.

Question 12 (3) Soit $n = 2^k$. Construire un réseau de profondeur $k + 1$ qui est correct pour toute séquence binaire $x_1, \ldots, x_n, y_1, \ldots, y_n$ telle que $x_1, \ldots, x_n$ et $y_1, \ldots, y_n$ sont croissantes.

Question 13 (4) Soit $n = 2^k$. Décrire la construction d'un réseau correct dont la profondeur est bornée par k^2. Donner le résultat de cette construction pour $n = 8$.

Deuxième partie

Accessibilité répétée

Cette partie étudie le principe du parcours en profondeur pour résoudre le problème de l'accessibilité répétée. Simplement parlant il s'agit de trouver des boucles autour de certains sommets dans un graphe orienté fini. Si le graphe représente le comportement d'un système, une telle boucle peut représenter une exécution qui satisfait certaines propriétés, par exemple que le système exécute une certaine action importante de temps en temps.

D'abord, on étudie une solution simple à l'aide des composantes fortement connexes. On verra que cette solution n'est pas toujours satisfaisante. Pour cette raison, on étudie ensuite une solution plus complexe mais aussi plus efficace.

Un graphe est un couple $G = (S, A, \iota)$ où S est un ensemble fini de sommets, $A \subseteq S \times S$ est un ensemble d'arcs, et $\iota \in S$ est un sommet dit *initial*. On dit que t est un successeur de s si $(s, t) \in A$. La figure 3 donne la représentation graphique usuelle d'un graphe $G = (S, A, \iota)$ avec $\iota = a$.

Pour $k \geq 1$, une suite de sommets $s_0, s_1, \ldots, s_k$ tel que $(s_i, s_{i+1}) \in A$ pour $0 \leq i < k$ est dite un *chemin de s_0 à s_k de longueur k*. Un tel chemin est appelé *circuit* si $s_0 = s_k$. On note $s \rightarrow_G^+ t$ si et seulement s'il existe un chemin de s à t, et $s \rightarrow_G^* t$ si et seulement si $s \rightarrow_G^+ t$ ou $s = t$. Si $s \rightarrow_G^* t$, on dit également que t est accessible depuis s.

On suppose désormais que tous les sommets sont accessibles depuis le sommet initial ι. En plus, on suppose que tout sommet possède au moins un successeur.

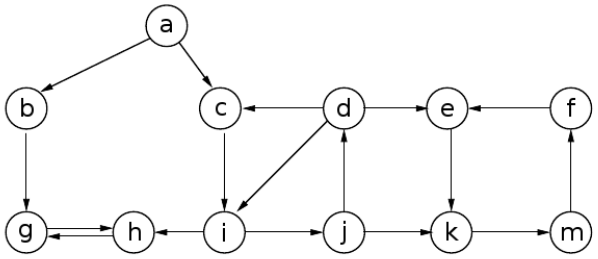

FIGURE 3 – Exemple d'un graphe

On pose $s \equiv_G t$ si s est accessible depuis t et t est accessible depuis s. On appelle *composantes fortement connexes* (ou simplement composantes) les classes de cette relation d'équivalence. On note $[\![s]\!]_G$ la classe d'équivalence du sommet s. Une composante est dite *simple* si elle ne contient qu'un seul sommet s tel que $(s, s) \notin A$ et *complexe* sinon. S'il n'y a pas d'ambiguïté sur le graphe G considéré, on peut omettre l'indice dans $\to^*$, $\to^+$, $\equiv$ et $[\![\cdot]\!]$.

Question 14 (1) Quelles sont les composantes du graphe G représenté dans la figure 3 ?

Dans ce qui suit, $G = (S, A, \iota)$ est un graphe quelconque.

Question 15 (2) Le graphe $V_G = (S', A', \iota')$ est défini comme suit :
- $S' = \{ [\![s]\!] \mid s \in S \}$;
- $A' = \{ ([\![s]\!], [\![t]\!]) \mid (s, t) \in A \text{ et } [\![s]\!] \neq [\![t]\!] \}$;
- $\iota' = [\![\iota]\!]$.

Montrer que V_G ne contient aucun circuit.

On considère des exécutions dans G, c'est à dire des suites infinies qui commencent avec ι, puis un successeur de ι etc. Formellement, une *exécution* est une fonction $\rho \colon \mathbb{N} \to S$ avec $\rho(0) = \iota$ et $(\rho(i), \rho(i+1)) \in A$ pour tout $i \geq 0$.

Soit $F \subseteq S$ un ensemble de sommets. Pour une exécution ρ on définit $Occ(\rho, F) := \{ i \mid \rho(i) \in F \}$, c'est à dire l'ensemble des positions des éléments de F dans ρ. On dit que ρ est *valide pour* F si et seulement si $Occ(\rho, F)$ est un ensemble infini. Le problème de l'*accessibilité répétée* est de déterminer, étant donné un graphe G et un ensemble de sommets F, si G contient une exécution valide pour F.

Question 16 (2) Montrer que G contient une exécution valide pour F si et seulement si G contient une composante C telle que C est complexe et $C \cap F \neq \emptyset$. (On rappelle que tous les sommets sont accessibles depuis ι.)

Suivant le résultat de la question 16, une solution naïve pour l'accessibilité répétée s'obtient en calculant les composantes d'un graphe $G = (S, A)$ et en testant ensuite s'il existe une composante complexe qui intersecte F. Calculer les composantes d'un graphe est possible en temps proportionnel à $|S| + |A|$; cette solution est alors efficace, et dans le pire des cas on ne peut pas faire mieux.

Cependant, dans de nombreuses instances il est possible de trouver une réponse positive beaucoup plus rapidement. Considérons le graphe de la figure 4. Pour déterminer que ce

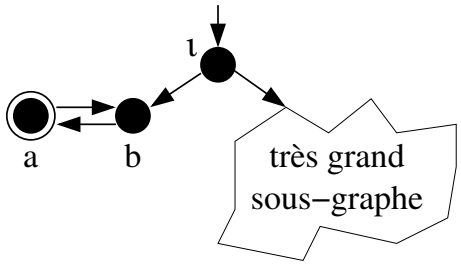

FIGURE 4 – Exemple d'un graphe avec une exécution valide pour $\{a\}$.

graphe contient une exécution valide pour $\{a\}$, il suffit de regarder la petite partie avec a, b, ι sans jamais regarder le sous-graphe de droite.

Le but de cette partie est de construire un algorithme qui exécute un parcours en profondeur sur un graphe orienté en cherchant une exécution valide. Cet algorithme marque tous les sommets et tous les arcs qu'il examine et donne une réponse positive dès que l'ensemble des sommets et arcs marqués contient une telle exécution. Dans la figure 4, cet algorithme pourra donner une exécution positive très rapidement s'il décide d'explorer a et b avant le très grand sous-graphe. On note que plusieurs parcours en profondeur différents sont possibles et que la rapidité de réponse (dans le cas positif) dépend du parcours choisi.

Une *pile* est une structure de données qui stocke une suite finie (de sommets) avec les opérations suivantes :

- *tester* si la pile est vide ou non ;
- *empiler* : ajoute un sommet en le mettant en haut de la pile ;
- *dépiler* : enlève et renvoie le sommet en haut de la pile ;
- *obtenir* le sommet en haut de la pile sans l'enlever.

Un *parcours en profondeur* affecte un numéro $n(s)$ à chaque sommet s de G. L'algorithme utilise une pile P. Au début, P est vide et aucun sommet ni arc n'est marqué. On marque ι, l'empile sur P et lui affecte le numéro $n(\iota) := 1$. Ensuite, à chaque itération on teste d'abord si P est vide et dans ce cas l'algorithme termine. Sinon, on obtient le sommet s en haut de P. Si tous les arcs issus de s sont marqués, on dépile s. Sinon, on choisit aléatoirement un arc (s, t) non marqué et on le marque. Si t est marqué, on continue. Sinon, on marque et empile t, et on lui donne le prochain numéro non encore utilisé (2, 3, 4, etc). La figure 5 donne une implémentation de cet algorithme.

Question 17 (1) Donner deux résultats possibles d'un parcours en profondeur sur le graphe de la figure 3, c'est à dire deux affectations n_1 et n_2 (partiellement) différentes qui en résultent.

Question 18 (1) Soient s, t deux sommets tels que $s \to^* t$ et $[\![s]\!] \neq [\![t]\!]$. A-t-on nécessairement $n(s) < n(t)$ après le parcours en profondeur ? Justifier votre réponse.

Soit $M \subseteq S$ un ensemble de sommets et n une affectation possible de l'algorithme de la figure 5. On appelle *racine de M sous n* et on note $r_{M,n}$ le sommet de M réalisant le minimum de n dans M, c'est à dire $n(r_{M,n}) \leq n(t)$ pour tout $t \in M$. Si n est sous-entendu, on écrira simplement r_M.

Pour une exécution de l'algorithme, on note $T(s)$ le moment où le sommet s est dépilé (on appelle cette action le *retour sur trace* de s).

```
1: Retirer les marques de tous les sommets et arcs
2: P ← ∅ ;  cnt ← 1 ;
3: n(ι) ← 1 ;  Empiler(P, ι) ;  Marquer(ι) ;
4: tant que P n'est pas vide faire
5:    Obtenir(P, s) ;
6:    si ¬(∃(s, t) ∈ A tel que (s, t) n'est pas marqué) alors
7:       Dépiler(P, s) ;
8:    sinon
9:       Soit t tel que (s, t) ∈ A n'est pas marqué ;
10:      Marquer (s, t) ;
11:      si t n'est pas marqué alors
12:         cnt ← cnt + 1 ;
13:         n(t) ← cnt ;  Empiler(P, t) ;  Marquer(t) ;
14:      fin si
15:   fin si
16: fin tant que
```

FIGURE 5 – Algorithme de parcours en profondeur

Question 19 (3) Soit s la racine d'une composante C. Montrer que $T(s) \geq T(t)$ pour tout sommet $t \in C$.

A un moment donné pendant le parcours en profondeur, l'ensemble des sommets et arcs marqués forme le sous-graphe G_M de G dit le graphe *marqué*. Egalement à un moment donné, une composante C' de $G_M(!)$ est dite *active* si $r_{C'}$ se trouve sur la pile P. Le graphe G_A dit *actif* est le sous-graphe de G_M induit par les composantes actives. Un sommet est actif lorsqu'il fait partie de G_A.

Les questions suivantes visent à établir certaines propriétés de G_M et G_A à travers le parcours en profondeur. En particulier, le but est de prouver que

1. *une composante de G_M devient inactive lorsqu'elle a été complètement explorée ;*
2. *les numéros affectés aux sommets actifs vérifient la propriété suivante : soit $r_1, \ldots, r_m$ la sous-séquence des racines actives dans P, alors un sommet actif s appartient à $[\![r_i]\!]_{G_M}$ si et seulement si $n(r_i) \leq n(s) < n(r_{i+1})$ pour $1 \leq i < m$ et à $[\![r_m]\!]_{G_M}$ si et seulement si $n(r_m) \leq n(s)$.*

Les composantes actives sont alors celles qui ont été explorées partiellement, et on peut alors utiliser le numéro d'un sommet actif pour déterminer sa composante. La figure 6 visualise cette idée.

Question 20 (3) Supposons que l'algorithme est à la ligne 5 alors que la pile contient $v_1, \ldots, v_m$ (avec v_1 au fond et v_m en haut de la pile). Montrer que pour tous $1 \leq i < j \leq m$ on a

- $n(v_i) < n(v_j)$;
- $v_i \rightarrow^+_{G_M} v_j$.

Question 21 (1) Montrer que si, à un moment donné, le sommet s est dans la pile et que $n(s) < n(t)$ alors il y a un chemin marqué de s à t : $s \rightarrow^+_{G_M} t$.

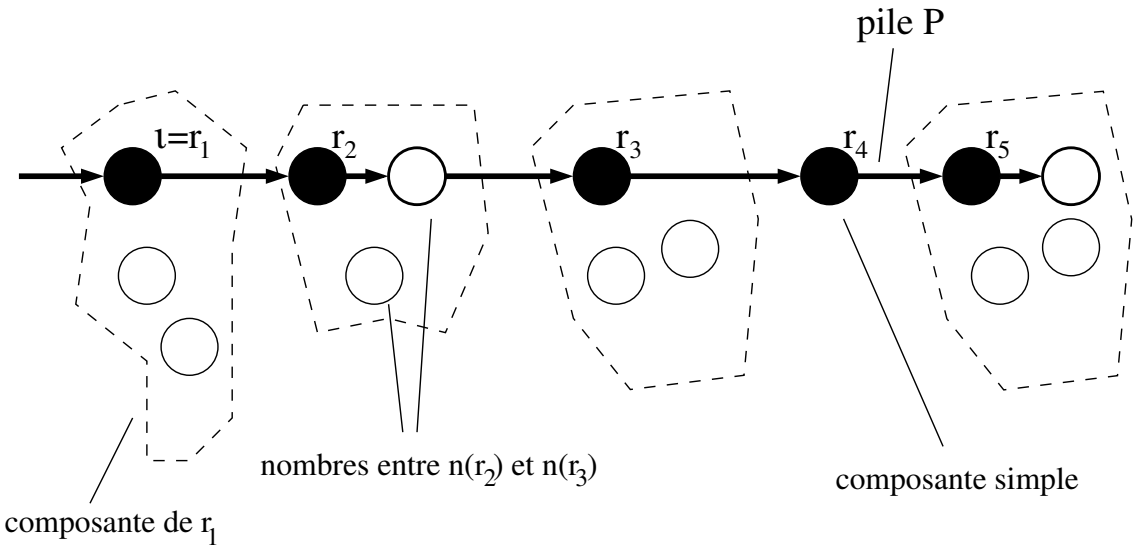

FIGURE 6 – Structure de G_A et distribution des numéros des sommets actifs. Les éléments de la pile P sont indiqués en gras et les racines actives en noir.

Question 22 (4) Soit $s \to_G^+ t$ un chemin entre deux sommets s et t de G. Montrer qu'au moment du retour sur trace de s au moins une des deux propriétés suivantes est satisfaite :

- $s \to_{G_M}^+ t$, c'est à dire que le chemin est entièrement marqué ;
- le chemin passe par un sommet $t' \neq s$ qui est encore dans la pile et $s \to_{G_M}^+ t'$.

Question 23 (1) Soit s un sommet actif (à un moment donné). Montrer que $[\![s]\!]_{G_A} = [\![s]\!]_{G_M} \subseteq [\![s]\!]_G$.

Question 24 (3) Soit s un sommet marqué inactif. Montrer que $[\![s]\!]_{G_M} = [\![s]\!]_G$.

Question 25 (4) A un moment donné, soit n l'affectation effectuée, s un sommet actif et t une racine active avec $n(t) \leq n(s)$. Montrer que $s \in [\![t]\!]_{G_M}$ si et seulement s'il n'existe aucune racine active u satisfaisant $n(t) < n(u) \leq n(s)$.

Note : Les résultats des questions 24 et 25 impliquent les propriétés 1 et 2 mentionnées ci-dessus. On va prouver quelques autres propriétés qui seront utiles avant de passer à un algorithme efficace.

Question 26 (2) Soient s, t deux sommets actifs tels que $n(s) \leq n(t)$. Montrer que $s \to_{G_M}^* t$.

Question 27 (1) Soient C, C' deux composantes actives et $t \in C$ et $s \in C'$ deux sommets tels que $n(t) \leq n(s)$. Supposons qu'on ajoute un arc $(s, t) \in A$ à G_M. Montrer qu'après cette addition, $C \cup C'$ se retrouve dans la même composante de G_M.

Question 28 (2) Au moment $T(s)$ du retour sur trace d'une racine active s, montrer que $s \to_{G_A}^* t$ si et seulement si $t \in [\![s]\!]_G$.

```
1: Retirer les marques de tous les sommets et arcs
2: P ← ∅ ; Q ← ∅ ; cnt ← 1 ;
3: n(ι) ← 1 ; Empiler(P, ι) ; Empiler(Q, ι) ; Marquer(ι)
4: tant que P n'est pas vide faire
5:    Obtenir(P,s) ;
6:    si ¬(∃(s, t) ∈ A tel que (s, t) n'est pas marqué) alors
7:       Dépiler(P, s) ;
8:       si s est en haut de Q alors
9:          Dépiler(Q, s) ;
10:      fin si
11:   sinon
12:      Soit t tel que (s, t) n'est pas marqué ;
13:      Marquer (s, t) ;
14:      si t n'est pas marqué alors
15:         cnt ← cnt + 1 ;
16:         n(t) ← cnt ; Empiler(P, t) ; Empiler(Q, t) ; Marquer(t) ;
17:      sinon si t est actif et n(t) ≤ n(s) alors
18:         répéter
19:            Dépiler(Q, u) ;
20:         jusqu'à n(u) ≤ n(t) ;
21:         Empiler(Q, u) ;
22:      fin si
23:   fin si
24: fin tant que
```

FIGURE 7 – Algorithme pour identifier les composantes d'un graphe

La figure 7 donne un autre algorithme, plus précisément une extension du parcours en profondeur qui maintient une deuxième pile Q. Cet algorithme, avec quelques modifications faciles, donnera la solution au problème de l'accessibilité répétée qu'on souhaite.

Dans les questions 29 et 30, il faut donner une justification exacte en analysant toutes les actions et cas possibles.

Question 29 (2) Montrer que l'algorithme de la figure 7 satisfait l'invariant

Q contient une sous-séquence de P.

c'est à dire que cette condition est toujours satisfaite au début de la boucle principale.

Question 30 (5) Montrer que l'algorithme de la figure 7 maintient un autre invariant :

Les éléments de Q sont exactement les racines actives.

Question 31 (3) L'algorithme donné dans la figure 7 utilise une condition "t est actif". Montrer que cette condition peut être testée par un algorithme travaillant en temps proportionnel à $|S| + |A|$?

Question 32 (3) Modifier l'algorithme de la figure 7 pour qu'il détermine si G contient une exécution valide pour F. Dans le cas affirmatif, la réponse doit être renvoyée au moment précis où le graphe marqué G_M contient une telle exécution.