Examen Algorithmique I (2020-21)

On insistera sur la correction et la clarté des réponses. Toute affirmation devra être justifiée explicitement, par un théorème du cours, par un numéro de question, par une référence à une règle. Pour tout raisonnement par récurrence, on devra dire sur quoi est effectuée la récurrence. N'inventez pas vos propres notations, et réutilisez les miennes, même si elles ne vous plaisent pas. Je veux une copie au propre, pas un brouillon : pas de rature, pas de questions dans le désordre, notamment. Je corrigerai très probablement toutes les copies en corrigeant d'abord toutes les questions 1 de tout le monde, puis toutes les questions 2, et ainsi de suite : ne comptez pas sur le fait que je me souvienne de ce que vous avez écrit dans une question précédente.

Je me réserve le droit de ne pas chercher à comprendre une réponse illisible, insuffisamment claire, ou plus longue ou compliquée que nécessaire.

1 Algorithme de Johnson

Soit $G \stackrel{\text{def}}{=} (S, A, c\colon A \to \mathbb{R})$ un graphe valué. On fabrique un nouveau graphe valué $G_* \stackrel{\text{def}}{=} (S_*, A_*, c_*)$ où :

- S_* est l'union disjointe de S et d'un nouveau sommet $*$;
- $A_* \stackrel{\text{def}}{=} A \cup \{(*, u) \mid u \in S\}$, autrement dit les arcs de G_* sont ceux de G, plus un arc de $*$ à chacun des sommets de G ;
- $c_*(e) \stackrel{\text{def}}{=} c(e)$ pour tout arc $e \in A$, et $c_*(*, u) \stackrel{\text{def}}{=} 0$ pour tout $u \in S$. (Note : j'écris $c_*(u, v)$ plutôt que l'inélégant $c_*((u, v))$.)

Question 1 Montrer que si G n'a pas de circuit de coût strictement négatif, alors G_* non plus.

Dans la suite, on suppose que G, et donc G_*, n'a pas de circuit de coût strictement négatif. On note $h(u)$, pour tout $u \in S_*$, le coût minimum d'un chemin de $*$ à u dans G_*. On note que ceci est bien défini.

On définit un nouveau graphe $G_{**} \stackrel{\text{def}}{=} (S_*, A_*, c_{**})$, de mêmes sommets et mêmes arcs que G_*, mais avec $c_{**}(u, v) \stackrel{\text{def}}{=} c_*(u, v) + h(u) - h(v)$.

Question 2 Montrer que, sous l'hypothèse que nous avons faite que G n'a pas de circuit de coût strictement négatif, G_{**} n'a pas d'*arc* de coût strictement négatif.

Question 3 Soit d la fonction qui à tout couple (u, v) de sommets de G associe le coût minimum d'un chemin de u à v dans G. Soit D la fonction qui à tout couple (u, v) de sommets de G_{**} associe le coût minimum d'un chemin de u à v dans G_{**}. Montrer que, pour tous $u, v \in S$, $d(u, v) = D(u, v) + h(v) - h(u)$.

Question 4 En déduire un algorithme de calcul de la table $(d(u,v))_{u,v\in S}$ de tous les coûts minimums de u à v dans G. On rappelle que G est supposé sans circuit de coût strictement négatif. Nommez explicitement les algorithmes du cours que vous utilisez, mais ne les recopiez pas. Vous devez obtenir un algorithme d'une complexité meilleure que la complexité $O(n^3)$ de Roy-Warshall/Floyd, et strictement meilleure lorsque G est clairsemé (c'est-à-dire si $m = O(n)$). Donnez cette complexité explicitement.

Question 5 On ne suppose plus que G soit sans circuit de coût strictement négatif. Peut-on modifier l'algorithme précédant, sans augmenter sa complexité asymptotique, de sorte qu'il détecte en plus si G a un circuit de coût strictement négatif ? Justifier.

2 Graphes et-ou

Un *graphe et-ou* est un graphe orienté $G \stackrel{\text{def}}{=} (S, A)$, où l'ensemble des sommets S est partitionné en deux parties disjointes $S_\wedge$ et $S_\vee$.

L'ensemble des *sommets accessibles* de G est par définition le plus petit ensemble de sommets $R_0 \subseteq S$ tel que :

— pour tout sommet $v \in S_\vee$, s'il existe un sommet $u \in R_0$ et un arc $u \to v$, alors $v \in R_0$;

— pour tout sommet $v \in S_\wedge$, si tout arc $u \to v$ d'extrémité v a sa source u dans R_0, alors $v \in R_0$.

Question 6 On se donne le graphe et-ou décrit par :

— $S_\wedge \stackrel{\text{def}}{=} \{a, b, c\}$;

— $S_\vee \stackrel{\text{def}}{=} \{u_1, u_2, u_3, u_4, u_5\}$;

— $A \stackrel{\text{def}}{=} \{(a, u_1), (u_1, u_2), (u_1, b), (u_3, b), (b, u_4), (u_2, c), (c, u_3), (c, u_5), (u_5, c)\}$.

Quel est l'ensemble de ses sommets accessibles ? Indication : ce n'est pas l'ensemble vide ! Il est conseillé de bien comprendre comment ceci fonctionne avant de passer à la suite.

On supposera dans la suite que `succ` (u) donne la liste des successeurs d'un sommet u. Chaque sommet v dispose aussi d'un champ (modifiable) `count`(v), qui est un entier naturel. L'idée est que `count`(v) est le nombre de prédécesseurs de v qu'il reste à marquer avant de savoir que tous les prédécesseurs de v sont marqués. Ceci n'aura de sens que lorsque $v \in S_\wedge$.

Plus précisément, on souhaite calculer l'ensemble R_0 des sommets accessibles d'un graphe et-ou donné en entrée. Pour ceci, on imite la procédure `reach_set` vue en cours, et on va accumuler dans un ensemble `marked`, initialement vide, les sommets de R_0. Les sommets de `marked` sont les sommets *marqués*. On maintient aussi une pile `work` de sommets *en attente*. On maintiendra comme invariants :

Inv1 pour tout $v \in S_\wedge$, il y a exactement `count`(v) prédécesseurs de v qui ne sont pas dans `marked` ;

Inv2 pour tout $v \in S_\wedge$, si `count`$(v) = 0$ alors $v \in$ `marked` $\cup$ `work` ;

Inv3 pour tout $v \in S_\vee$, s'il existe un prédécesseur de v dans `marked`, alors $v \in$ `marked`$\cup$`work` ;

Inv4 tous les sommets de `marked` $\cup$ `work` sont dans R_0.

Question 7 Écrire les lignes de code d'initialisation : elles doivent initialiser `marked` à $\emptyset$ et `work` à la pile vide, puis établir les invariants ci-dessus. Vous pouvez utiliser des instructions de la forme `for each` $v \in S$ pour parcourir les sommets, et de même avec $S_\wedge$ et $S_\vee$ et avec la liste `succ` (u) des successeurs d'un sommet u. Vous pouvez aussi tester par $v \in S_\wedge$ si un sommet v est dans $S_\wedge$, et similairement pour $S_\vee$. Je ne demande pas une preuve formelle en logique de Hoare, mais vous devez expliquer votre code, clairement et succinctement, et notamment quelles lignes établissent quels invariants.

Question 8 Écrire une procédure `mark(`u`)` qui marque un sommet u passé en argument, sous les hypothèses, valables en entrée :

(a) $u \notin$ `marked` ;

(b) pour tout $v \in S_\wedge$, il y a exactement `count`(v) prédécesseurs de v qui ne sont pas dans `marked` ;

(c) pour tout $v \in S_\wedge$, si `count`$(v) = 0$ alors $v \in$ `marked` $\cup$ `work` $\cup \{u\}$;

(d) pour tout $v \in S_\vee$, s'il existe un prédécesseur de v dans `marked`, alors $v \in$ `marked` $\cup$ `work` $\cup \{u\}$;

(e) tous les sommets de `marked` $\cup$ `work` $\cup \{u\}$ sont dans R_0.

En sortie, `marked` doit être égal à l'ensemble `marked` donné en entrée, union $\{u\}$, et les invariants **Inv1**–**Inv4** doivent être vérifiés. Justifier, en citant méticuleusement les invariants utilisées, et les propriétés établies.

Question 9 On considère l'algorithme :

```
⟨ code d'initialisation de la Question 7 ⟩
while not(emptyp(work)) do
      let u = pop(work) in
          if u ∉ marked then mark(u); (* Question 8 *)
```

(Ici, `emptyp` teste la vacuité de son argument, et `pop(work)` dépile un élément de `work`.) En raisonnant sur les invariants, que l'on devra citer explicitement, justifier que, à la sortie de l'algorithme, `marked` $= R_0$.

Question 10 Comme d'habitude, on note n le nombre de sommets et m le nombre d'arcs de G. Pourquoi l'algorithme de la **Question 9** ne peut-il faire qu'au plus $n + m$ tours de sa boucle `while` ? Je ne demande pas une preuve à coups d'invariants, mais une justification convaincante. Bonus : en raffinant (éventuellement) un peu vos algorithmes, montrer qu'on peut descendre à n tours de boucle `while` (ce que l'on admettra dans la suite).

Question 11 On suppose que G nous est donné dans la représentation suivante : d'abord, un graphe orienté usuel décrit par liste de successeurs, et pour chaque sommet u, un champ `et_ou`(u), qui vaut 1 si $u \in S_\wedge$ et 0 sinon. Montrer que l'algorithme de la **Question 9**, avec l'amélioration suggérée à la **Question 10**, fonctionne en temps $O(n + m)$. On précisera notamment la complexité des opérations de la **Question 7** et de la **Question 8**, les façons d'implémenter les diverses opérations non données dans la représentation de G (`for each` $v \in S_\wedge$, $v \in S_\wedge$, la gestion de l'ensemble `marked`, etc.).

Question 12 En guise d'application, on considère le problème de vacuité du langage d'une grammaire algébrique. Soit Σ un alphabet fini fixé. Une grammaire algébrique $\mathcal{G}$ est un triplet (NT, S, P) où :

— NT est un ensemble fini dit de *non-terminaux*, disjoint de Σ (les éléments de Σ sont parfois appelés les *terminaux*) ;

— S est un élément de NT, appelé *non-terminal de départ* ;

— P est un ensemble fini de *productions* ; une production est un couple formé d'un non-terminal X et d'un mot fini W sur $(\Sigma \cup NT)^*$, et on l'écrira $X \to W$. En général, W s'écrit de façon unique comme la concaténation $w_0 Y_1 w_1 Y_2 \cdots w_{k-1} Y_k w_k$ où les w_i sont des mots de Σ^* et les Y_i sont des non-terminaux, notation que l'on reprendra implicitement plus bas.

La *taille* d'une règle $X \to w$ est égale à la longueur $|w|$ de w plus 1, et la taille d'une grammaire algébrique est la somme des tailles de ses règles.

On définit le système de preuve suivant, permettant de dériver des jugements $\vdash w' \in Y$, où $w' \in \Sigma^*$ et $Y \in NT$, dont la signification intuitive est « le mot w' est produit par la grammaire au non-terminal Y ». Pour chaque production $X \to w_0 Y_1 w_1 Y_2 \cdots w_{k-1} Y_k w_k$ dans P, on a une règle de la forme :

$$\frac{\vdash w'_1 \in Y_1 \quad \cdots \quad \vdash w'_k \in Y_k}{\vdash w_0 w'_1 w_1 w'_2 \cdots w_{k-1} w'_k w_k \in X}$$

Le *langage* $L(\mathcal{G})$ de la grammaire algébrique $\mathcal{G}$ est par définition l'ensemble des mots $w \in \Sigma^*$ tels qu'on peut dériver $\vdash w \in S$ dans ce système.

Montrer que, étant donnée une grammaire algébrique $\mathcal{G}$ en entrée, on peut décider si $L(\mathcal{G})$ est vide ou non en temps linéaire en la taille de $\mathcal{G}$.

3 Parcours en profondeur de graphes non orientés

Un graphe non orienté $G \stackrel{\text{def}}{=} (S, A)$ est vu, comme en cours, comme un graphe orienté dont les arcs sont les couples (u, v) (notés $u \to v$) tels que $u - v$ soit une arête de G. (On note $u - v$ une arête, c'est-à-dire la paire $\{u, v\} \in A$.) Comme $u - v$ et $v - u$ sont la même arête, ceci signifie que si $u \to v$ est un arc de G, alors $v \to u$ aussi.

Dans la suite, on suppose G connexe, en tant que graphe non orienté, et on se fixe un sommet $*$ quelconque de G.

Question 13 Pourquoi $*$ est-il une source de G vu comme graphe orienté ? Autrement dit, pourquoi tout sommet de G est-il accessible depuis $*$?

Question 14 On se fixe un parcours en profondeur L partant de $*$ quelconque de G, vu comme graphe orienté. Montrer que L ne contient aucun arc transverse.

4 Ordonnancement biprocesseur

On souhaite résoudre le problème de l'allocation statique de tâches et de bases de données sur un système à deux processeurs communiquant entre eux.

En entrée, on nous donne une liste de tâches à effectuer et une liste de bases de données. On appellera les tâches et les bases de données des *modules*. Formellement, on nous donne $t+d$ modules $M_1, \ldots, M_{t+d}$, les t premiers étant appelés des tâches, les d derniers des bases de données. On nous donne aussi les temps T_{i1} et T_{i_2} que mettrait chaque tâche M_i, $1 \leq i \leq t$ à être exécutée sur chacun des deux processeurs, ainsi que les temps de communication C_{ij} entre

les modules M_i et M_j lorsqu'ils sont alloués sur des processeurs différents (communication entre tâches, accès d'une tâche à une base de donnée, notamment). Lorsque M_i et M_j sont alloués sur le même processeur, on suppose que leur temps de communication est nul.

Une *allocation* des modules est un sous-ensemble E de $\{1, \cdots, t+d\}$, qui représente les numéros des modules placés sur le processeur numéro 1. Les autres sont placés sur le processeur numéro 2. Le *coût* d'une telle allocation est la somme :

$$c(E) \stackrel{\text{def}}{=} \sum_{i \in E \cap \{1,\cdots,t\}} T_{i1} + \sum_{i \in \{1,\cdots,t\} \setminus E} T_{i2} + \sum_{i \in E, j \in \{1,\cdots,t+d\} \setminus E} C_{ij}.$$

On souhaite trouver une allocation E de coût $c(E)$ minimal. Pour ceci, on forme un réseau dont les sommets sont numérotés $1, \ldots, t+d$, en plus d'une source s et d'un puits p.

Question 15 Compléter la description du réseau, et montrer que l'on peut résoudre algorithmiquement la question du coût $c(E)$ minimal en temps $O((t+d)^4)$ (ou mieux !).