Complexité algorithmique, quelques preuves détaillées

Stéphane Le Roux, leroux@lsv.fr

December 10, 2022

N'hésitez pas à me signaler les erreurs de tout type ou à me dire que vous avez du mal à comprendre certains passages. Je compte enrichir ce document avec de nouvelles preuves au fur et à mesure, en partie en fonction de vos suggestions.

Question 1 : composition de fonction logspace

Soient $f : A^* \to B^*$ et $g : B^* \to C^*$ deux fonctions logspace. Montrer que la composée $g \circ f$ est aussi logspace.

Solution 1

Soit M_f et M_g deux machines de Turing qui réalisent f et g, respectivement, en espace logarithmique en la taille de l'entrée.

On décrit d'abord une manière naturelle de procéder, qui n'est pas la bonne, mais qu'il nous suffira de modifier pour obtenir le résultat. Soit M une machine qui a une bande d'entrée E_M, une bande de sortie S_M, les bandes de travail de M_f et M_g plus une bande de travail d'interface I. M n'a pas la bande de sortie de M_f, c'est I qui va la remplacer, et M n'a pas la bande d'entrée de M_g, c'est aussi I qui va la remplacer. Sur son entrée, M donne la main à M_f et lui dit que sa bande d'entrée est E_M et que sa bande de sortie est I. Quand M_f termine, M donne la main à g et lui dit que sa bande d'entrée est I et sa bande de sortie est S_M. Le problème est que l'espace utilisé sur I peut être polynomial en la taille de l'entrée, cette façon de "composer" M_f et M_g n'est donc pas logspace

À la place de I, on va utiliser trois bandes pour simuler I : I_p sera la bande de position de la tête de lecture, I_c la bande de contenu de la case correspondante, et I_d une bande de décrément d'un compteur.

Le controleur de M a deux parties principales : M'_f qui se comporte presque comme le controleur de M_f, et M'_g presque comme celui de M_g. Ces deux parties effectuent séquentiellement des opérations puis se (re)donnent la main.

La machine M commence par écrire 0 sur I_p et écrire \$′ sur la première case de I_c, pour symboliser à M'_g le début de bande de sortie de M_f sans utiliser un vrai \$. (M'_g est programmée de telle sorte qu'elle comprend \$′.) La main est ensuite passée à Mg'.

1. M'_g lit le contenu de I_c et effectue alors sa transition comme M_g, sauf qu'au lieu de déplacer la tête de lecture à droite/gauche, elle incrémente/décrémente le contenu de I_p.

    - Si la transition faisait terminer M_g/M'_g, alors M termine.

- Sinon, M'_g passe la main à M'_f, cf point 2.

2. M'_f lit le contenu de I_p, calcule (comme expliqué au point 3) le contenu de la case correspondante dans la sortie de M_f, puis l'écrit sur I_c en écrasant le contenu précédent. Elle passe ensuite la main à M'_g, cf point 1. Notons que M'_f ne termine jamais.

3. Calcule d'une "case de sortie". M'_f copie le contenu de I_p sur I_d. Puis calcule (depuis le début à chaque fois) comme M_f sur l'entrée de M, sauf que, à chaque fois que M_f aurait envie d'écrire une lettre sur sa bande de sortie, M'_f fait la chose suivante :

   - Si le contenu de I_d est au moins 2, elle le décrémente de 1 et reprend son calcul comme si M_f avait écrit la lettre en question.
   - Si le contenu est 1, la lettre en question sera écrite sur I_c comme indiqué dans la suite du point 2.

Sur une entrée w, l'espace utilisé sur les bandes de travail de M est du type :

- L'espace utilisé pendant le calcul de M_f sur w, donc logartihmique en $|w|$
- L'espace utilisé pendant le calcul de M_g sur $f(w)$, donc logarithmique en $|f(w)|$, donc logarithmique en $|w|$, car $|f(w)|$ est borné par un polynome fixe en $|w|$.
- L'espace utilisé par I_c, de taille 1.
- L'espace utilisé par I_p, logarithmique en $|f(w)|$ donc logarithmique en $|w|$.
- L'espace utilisé par I_d, borné par celui utilisé par I_p, car le décrément du compteur peut se faire "sur place", sans espace auxiliaire.

Question 2 : Connexité dans un graphe en $NSPACE(\log)$

Montrer que le problème suivant est dans $NSPACE(\log)$.

- Entrée : un graphe orienté et deux de ses sommets.
- Sortie : y a-t-il un chemin du premier sommet vers le deuxième.

Solution 2

En entrée, soit $G = (V, E)$ un graphe, qu'on suppose donné sous la forme d'une matrice d'adjacence, et s, t deux sommets. On pose $n := |V|$, et on remarque que s'il existe un chemin de s à t, il en existe un de longueur au plus n.

Une première tentative naturelle de résoudre le problème serait de deviner une suite de n sommets ou moins de manière non-déterministe, puis de vérifier si c'est un chemin de s à t. Le problème est que stocker une suite de n sommets peut prendre un espace linéaire. (Cela ce voit mieux si on considère les listes d'adjacence.) Une solution est alors de ne deviner le chemin qu'au fur et à mesure, en ne conservant que les noms de deux sommets à chaque fois.

Un espace logarithmique suffit pour vérifier que l'entrée est bien la matrice d'un graphe et que s, t sont bien inférieurs ou égaux à n. Le processus calcule n et l'écrit sur une bande de

travail $T_{|V|}$. On utilise plusieurs bandes de travail supplémentaires T_{blabla} qu'on introduit au fur et à mesure dans la description de la procédure ci-dessous. La machine M commence par écrire s sur T_u, écrire t sur T_{cible}, et écrire n sur T_c.

1. Une phase non-déterministe commence : au moyen d'un compteur décroissant de $\log_2 n$ (environ) à 0 sur une bande de travail auxiliaire, la machine M devine et écrit bit par bit le nom d'un sommet suivant sur T_v. La phase non-déterministe est terminée.

2. M vérifie si (u, v) est une arête, où u est le sommet représenté sur T_u et v sur T_v. Cela s'effectue en lisant la case $[u, v]$ de la matrice d'entrée, à laquelle on accède au moyen de deux compteurs décroissant de u et v, respectivement, à 0.

3. Si (u, v) n'est pas une arête, M rejette (ce chemin non-déterministe). Sinon, on continue ci-dessous.

4. Si $v = t$, déterminé en comparant les contenus de T_v et T_{cible}, on accepte.

5. Le compteur sur T_c est décrémenté de 1. Si la valeur 0 est atteinte, M rejette (ce chemin non-déterministe). Sinon, on continue ci-dessous.

6. Le contenu de T_v est copié sur T_u en écrasant le contenu précédent.

Le contenu de chaque bande n'excède jamais $\log_2 n$ (environ), l'espace utilisé est donc logarithmique en la taille de l'entrée.

Question 3 : Connexité dans un graphe en $SPACE(\log^2)$

Montrer que le problème suivant est dans $SPACE(\log^2)$.

- Entrée : un graphe orienté et deux de ses sommets.
- Sortie : y a-t-il un chemin du premier sommet vers le deuxième.

Solution 3

Remarques prélliminaires dans un graphe $G = (V, E)$ à n sommets.

- Pour tout $s, t \in V$, il existe un chemin de s à t ssi il en existe un de longueur au plus $n - 1$.
- Pour tout $u, v \in V$, il existe un chemin de longueur au plus i de u à v ssi il existe un sommet w tel qu'il existe un chemin de longueur au plus $\lfloor \frac{i}{2} \rfloor$ de u à w et un chemin de longueur au plus $\lceil \frac{i}{2} \rceil$ de w à v.

Les remarques ci-dessus motivent l'algorithme haut niveau ci-dessous pour décider s'il existe un chemin entre deux sommets donnés d'un graphe

- Entrée : un graphe $G = (V, E)$ et deux sommets $s, t \in V$.
- Sortie : OUI ou NON.
- On commence par calculer $n := |V|$.

- On renvoie $C(s,t,n)$, où C est décrit ci-dessous.
- Algorithme récursif C.
  - Si $n=0$: si $s=t$ renvoyer OUI, sinon renvoyer NON.
  - Si $n=1$: si $s=t$ ou $(s,t)\in E$ renvoyer OUI, sinon renvoyer NON.
  - Pour tout $v\in V$ faire : si $C(s,v,\lfloor\frac{n}{2}\rfloor)$ et $C(s,v,\lceil\frac{n}{2}\rceil)$ renvoyer OUI.
  - Renvoyer NON.

L'algorithme ci-dessus fait appel à la récursion. Or, la récursion n'est pas primitive dans une machine de Turing, nous allons devoir l'implémenter nous-même dans l'algorithme bas-niveau ci-dessous.

En entrée, soit $G=(V,E)$ un graphe, qu'on suppose donné sous la forme d'une liste d'arête, et s,t deux sommets. On pose $n:=|V|$. Les sommets sont $v_1,\dots,v_n$ où v_k est la représentation binaire de k.

- On écrit $(s,t,n,v_1,?,?)$ sur une bande de travail qu'on appelle la pile.
- Ce tuple sera toujours en fond de pile, i.e. en début de bande de travail, et de la forme (s,t,n,v_k,b_g,b_d), avec $k\in[n]$, et $b_g,b_d\in\{?,0,1\}$. Par choix, $b_g=1$ signifie qu'il existe un chemin de s à v_k de longueur au plus $\lfloor\frac{n}{2}\rfloor$ et 0 signifie que non, et ? signifie qu'on ne sait pas encore. La sémantique de b_d est similaire. Les objets s,t,n seront constants le long du calcul, mais v_k,b_g,b_d pourront être modifiés plusieurs fois.
- Les autres éléments de la pile seront de la forme (d,u,v,l,v_k,b_g,b_d), où comme pour le fond de pile $u,v,v_k\in V$ et $l\in[n]$ et $b_g,b_d\in\{?,0,1\}$, avec la même sémantique que ci-dessus. Les objets u,v,l seront constants pendant la vie de ce tuple dans la pile.
- Ci-dessus, $d\in\{g,d\}$ où g signifie que le tuple correspond à un appel récursif "gauche" dans l'algorithme de haut niveau.
- Quatre types d'opérations peuvent avoir lieu en fonction du haut de la pile.
  - Retour du résultat.
    - Si le haut de pile est de la forme $(s,t,n,v_k,1,1)$, la machine accepte
    - Si le haut de pile est de la forme (s,t,n,v_n,b_g,b_d) avec $b_g=0$ ou $b_d=0$, la machine rejette.
  - Appels récursifs.
    - Si le haut de pile est de la forme $(\dots,u,v,l,v_k,?,?)$ avec $l>1$, alors on le remplace par $(\dots,u,v,l,v_k,?,?)(g,u,v_k,\lfloor\frac{l}{2}\rfloor,v_1,?,?)$.
    - Si le haut de pile est de la forme $(\dots,u,v,l,v_k,?,?)$ avec $l>1$, alors on le remplace par $(\dots,u,v,l,v_k,?,?)(d,v_k,v,\lceil\frac{l}{2}\rceil,v_1,?,?)$.
  - Modification du haut de pile.
    - Si le haut de pile est de la forme $(\dots,u,v,0,v_k,?,?)$, alors si $u=v$ on le remplace par $(\dots,u,v,0,v_k,1,1)$, sinon par $(\dots,u,v,0,v_1,0,0)$.
    - Si le haut de pile est de la forme $(\dots,u,v,1,v_k,?,?)$ alors si $u=v$ ou $(u,v)\in E$, on le remplace par $(\dots,u,v,0,v_k,1,1)$, sinon par $(\dots,u,v,0,v_1,0,0)$.

    - Si le haut de pile est de la forme $(\ldots, u, v, l, v_k, 0, 0)$ avec $k < n$, on le remplace par $(\ldots, u, v, l, v_{k+1}, ?, ?)$.
  - Propagation de l'information vers le bas de la pile, i.e. vers le haut de l'arbre de récursion. Plus précisément :
    - Si le haut de pile est de la forme $(\ldots, ?, ?)(g, \ldots, 1, 1)$ on le remplace par $(\ldots, 1, ?)$.
    - Si le haut de pile est de la forme $(\ldots, ?)(d, \ldots, 1, 1)$ on le remplace par $(\ldots, 1)$.
    - Si le haut de pile est de la forme $(\ldots, ?, ?)(g, \ldots, v_n, b_g, b_d)$ avec $b_g = 0$ ou $b_d = 0$, on le remplace par $(\ldots, 0, ?)$.
    - Si le haut de pile est de la forme $(\ldots, ?)(d, \ldots, v_n, b_g, b_d)$ avec $b_g = 0$ ou $b_d = 0$, on le remplace par $(\ldots, 0)$.

On se convainc que ce programme (ou plutôt sa version sans bug), termine et renvoie la réponse correcte à notre question.

En terme d'espace utilisé :

- Dans la pile, à chaque instant il y a au plus $\lceil \log_2 n \rceil$ éléments.
- Chacun d'entre eux a une taille logarithmique en celle de l'entrée.
- L'espace utilisé par la pile est donc en $(log|G|)^2$.
- Les calculs de type $l \mapsto \lfloor fracl2 \rfloor$ peuvent s'effectuer en espace logarithmique sur une autre bande de travail.
- l'incrément $v_k \mapsto v_{k+1}$ peut s'effectuer directement dans la pile sans espace supplémentaire.
- De plus, le "pattern matching" sur le haut de pile et les tests auxiliaires s'effectuent aussi sans espace supplémentaire.
- Globalement, l'espace utilisé par la pile est donc en $(log|G|)^2$.