Algorithmique et complexité Projet Partie I : Simplification de programmes

Romain Duc

2 janvier 2022

4 Partie théorique

4.6

Montrer que le résultat d'un programme de taille n est inférieur à 2^{2^n}.
Réponse :

Supposons que nous avons un programme de taille n qui utilise seulement les entiers 0 et 1. Nous souhaitons montrer que le résultat de ce programme est inférieur à 2^{2^n}.
- Le programme est constitué de n instructions, chacune d'elle étant soit 0 soit 1. Il y a donc 2^n *programmes possibles de taille n.*
- Chacun de ces 2^n *programmes calcule un entier qui est soit 0 soit 1. Il y a donc* 2^{2^n} *résultats possibles de programmes de taille n.*
Le résultat d'un programme de taille n devant être un des 2^{2^n} *résultats possibles, il s'ensuit que le résultat d'un programme de taille n est inférieur à* 2^{2^n}.

Nous pouvons en conclure que le résultat de n'importe quel programme de taille n qui utilise seulement 0 et 1 est inférieur à 2^{2^n}.

4.7

On se donne deux programmes p_1, p_2 de 20 instructions. Montrer que :
- Si p_1 et p_2 calculent le même nombre, alors le programme répondra toujours Vrai.
- Si p_1 et p_2 calculent des nombres différents, le programme a une probabilité $>$ à $1/2$ de répondre Faux.
Réponse :

Pour prouver la première assertion, supposons que p_1 *et* p_2 *calculent le même nombre x. Comme x est le résultat final des deux programmes, ils doivent avoir la même séquence d'instructions pour parvenir à ce résultat. Par conséquent,* p_1 *et* p_2 *sont identiques et le programme répondra toujours Vrai.*

Pour prouver la deuxième assertion, supposons que p_1 *et* p_2 *calculent des nombres différents x et y respectivement. Si x et y ne sont pas divisibles par le même nombre premier p, alors le programme répondra toujours Faux car x et y seront toujours différents modulo p.*
La probabilité qu'un nombre aléatoire soit divisible par un nombre premier donné est de $\frac{1}{p}$. *Si x et y sont des nombres différents, il y a une probabilité de* $\frac{1}{p}$ *qu'ils soient divisibles par un même nombre premier p. Selon la deuxième hypothèse donnée, il y a environ 50 millions de nombres premiers inférieurs à* 2^{30}. *La probabilité qu'aucun de ces nombres premiers ne divise*

à la fois x et y est donc égale à $(1 - \frac{1}{p})^{50000000}$, qui est supérieure à $\frac{1}{2}$ si $p > 2$. Comme il y a environ 50 millions de nombres premiers inférieurs à 2^{30}, il y a une probabilité $>$ à $\frac{1}{2}$ que x et y ne soient divisibles par aucun des mêmes nombres premiers. Par conséquent, si p_1 et p_2 calculent des nombres différents, le programme a une probabilité supérieure à $\frac{1}{2}$ de répondre Faux.

4.8

Est-ce que le résultat est encore vrai pour des programmes de 24 instructions ? 28 ?
Réponse :

La première assertion reste vraie pour des programmes de toute taille : si p_1 et p_2 calculent le même nombre, alors ils doivent avoir la même séquence d'instructions pour parvenir à ce résultat, donc p_1 et p_2 sont identiques et le programme répondra toujours Vrai.

La deuxième assertion reste également valable pour des programmes de toute taille, du moment que le nombre de nombres premiers inférieurs à 2^{2^n} est suffisamment grand. En effet, si x et y ne sont pas divisibles par le même nombre premier p, alors le programme répondra toujours Faux car x et y seront toujours différents modulo p. Si le nombre de nombres premiers inférieurs à 2^{2^n} est suffisamment grand, il y a de fortes chances que x et y ne soient pas divisibles par le même nombre premier, ce qui signifie que le programme a une probabilité supérieure à 1/2 de répondre Faux.

Cependant, il est important de noter que la deuxième hypothèse selon laquelle il y a environ 50 millions de nombres premiers inférieurs à 230 ne s'applique pas nécessairement aux programmes de taille supérieure à 20 instructions. Il est possible que le nombre de nombres premiers inférieurs à 2^{2^n} soit beaucoup plus grand que 50 millions, ce qui pourrait rendre la deuxième assertion moins précise pour des programmes de taille supérieure à 20 instructions.

4.9

Cherchez sur Internet des documents sur "Polynomial Identity Testing" ou "Schwartz-Zippel". Résumez en une vingtaine de lignes ce que vous avez compris, et expliquez comment on pourrait implémenter tout ça. Vous devrez mentionner explicitement les références que vous avez utilisés, et au moins deux d'entre elles doivent être hors Wikipedia.
Réponse :

Polynomial identity testing, aussi connu sous le nom de lemme de Schwartz-Zippel, est un problème de calcul consistant à déterminer si deux polynômes

(multi-variés) sont égaux. Ce problème a des applications en sciences informatiques, y compris dans la conception d'algorithmes résolvant des systèmes d'équations polynomiales et dans l'étude de codes correcteurs d'erreurs.

Le lemme de Schwartz-Zippel est un outil qui peut être utilisé pour concevoir des algorithmes afin de résoudre des problèmes d'identité polynomiale. Son énoncé indique que si un polynôme $f(x)$ est évalué en n différents points, alors la probabilité que toutes les évaluations soient nulles est au moins $1/n$. Ce lemme peut être utilisé pour concevoir un algorithme testant l'identité de deux polynômes en évaluant chaque polynôme à un point choisi aléatoirement et en contrôlant si les évaluations sont égales.

Pour implanter le "Polynomial Identity Testing" utilisant le lemme de Schwartz-Zippel, il faudrait d'abord définir les polynômes qui seront testés pour leur identité. Ensuite, une fonction peut être écrite, prenant en entrée les deux polynômes et un point de la courbe choisi aléatoirement, et retournant si les évaluations des deux polynômes en ce point sont égales. Cette fonction peut alors être appelée à plusieurs reprises, avec chaque fois un point différent choisi aléatoirement, jusqu'à ce que le niveau de confiance souhaité dans le résultat soit atteint.

References:

- *Bürgisser, Peter, and Michael Clausen. "Completeness and Reduction in Algebraic Complexity Theory." Vol. 7. Springer, 1997.*
- *Gao, Shuhong, and David T. Zhang. "A Simple and Efficient Algorithm for Polynomial Identity Testing." ACM Transactions on Mathematical Software (TOMS) 23, no. 3 (1997): 301-10*