Egzamin ze złożoności obliczeniowej – 6.6.2011. Komentarz

Komentarz dotyczy *zadania teoretycznego* (nr 4) oraz części teoretycznej egzaminu.

Zadanie 4

Wykaż, że jeśli $P = NP$, to problem faktoryzacji może być rozwiązany w czasie wielomianowym.

Funkcyjny problem *faktoryzacji* polega na tym, by dla liczby naturalnej danej w postaci binarnej znaleźć jej rozkład na czynniki pierwsze.

Hipoteza $P = NP$ oznacza, że równe są dwie klasy języków: języki rozpoznawane przez deterministyczne maszyny Turinga w czasie wielomianowym oraz języki rozpoznawane w czasie wielomianowym przez maszyny *niedeterministyczne*, lub, co jest równoważne, prezentowalne jako

$$\exists R = \{x : (\exists y)\, R(x, y)\}$$

dla relacji wielomianowych R.

Problem faktoryzacji nie jest jednak językiem, ale problemem funkcyjnym. Przy założeniu $P = NP$ skonstruujemy deterministyczny algorytm, który dla liczby złożonej n znajduje w czasie wielomianowym jej nietrywialny dzielnik $1 \neq k \neq n$. Przejście od tego algorytmu do algorytmu znajdującego pełny rozkład na czynniki pierwsze pozostawiamy Czytelnikowi.

Wspomniany algorytm można zaprojektować na kilka sposobów, naszkicujemy dwa.

(A) Rozwiązanie ogólniejsze. Dla dowolnej relacji wielomianowej R, pokażemy algorytm, który dla $x \in \exists R$, znajduje y, takie że $R(x, y)$.

Korzystamy ze znanego z wykładu algorytmu, który dla danej spelnialnej formuły φ znajduje spełniające ją wartościowanie używajac wyroczni odpowiadającej na pytanie, czy $\psi \in SAT$ (*Lemma 3* w notatkach). Jeśli $P = NP$, to wyrocznię możemy zastąpić deterministycznym algorytmem. Z kolei wiemy, że problem SAT jest zupełny w sensie Levina, a zatem, gdy redukujemy $x \mapsto \varphi$, to z wartościowania spełniającego φ (gdy $x \in \exists R$) potrafimy odtworzyć „‚świadka dla x", czyli takie y, że $R(x, y)$.

(B) Rozwiązanie dedykowane problemowi podzielności. Rozważamy język L

$$L \ni \langle n, a, b\rangle \Leftrightarrow n \text{ ma nietrywialny dzielnik w przedziale } [a, b].$$

Łatwo widzieć, że język L jest w NP, zatem przy założeniu, że $P = NP$, jest również P. Stosując wyszukiwanie binarne możemy zmniejszać przedział $[a, b]$ tak, by po $\mathcal{O}(\log n)$ krokach znaleźć poszukiwany dzielnik.

Trzeba jednak podkreślić, że w ogólności z faktu, że $L = \exists R \in P$, nie potrafimy wysnuć algorytmu, który dla $x \in L$ znajdowałby y, takie że $R(x, y)$. Przykładem jest własnie zbiór liczb złożonych: znamy deterministyczny test pierwszości/złożoności AKS, a nie znamy wielomianowego algorytmu faktoryzacji; co więcej bezpieczeństwo wielu funkcji kryptograficznych (jak np. RSA) opiera się na założeniu, że taki algorytm nie istnieje.

Teoria

Należało wskazać i wyjaśnić błędy w przytoczonych rozumowaniach.

1. Aby sprawdzić, czy liczba m jest dzielnikiem n, wystarczy dodawać m do siebie i badać, czy w końcu trafimy w n, czy też je przekroczymy. Do tego wystarczy $\mathcal{O}(\log n)$ pamięci. Zatem, testując kolejne $m < n$, możemy zbadać, czy liczba n jest pierwsza w pamieci jedynie logarytmicznej, a wiemy, że $L \subseteq P$. Dlaczego zatem tyle hałasu wokół algorytmu AKS wielomianowego rozpoznawania liczb pierwszych (z 2002 r.) ?

Algorytm Agrawala, Kayala i Saxeny był pierwszym znanym algorytmem rozstrzygającym, czy liczba n jest pierwsza czy złożona w czasie zależnym wielomianowo od rozmiaru zapisu liczby n w systemie binarnym, czyli od $\log n$. Nasz algorytm działa w pamięci $\mathcal{O}(\log n)$, czyli w pamięci zależnej liniowo od rozmiaru n i jego czas działania jest wykładniczy względem $\log n$.

2. Niech $\varphi = \alpha_1 \wedge \ldots \wedge \alpha_k$, gdzie każde α_i jest klauzulą zawierającą *dwa* literały. Rozważmy

$$\begin{aligned} \varphi' &= (\alpha_1 \vee \neg x_1) \wedge \ldots \wedge (\alpha_k \vee \neg x_k) \wedge \\ &\quad (x_1 \vee x_1 \vee x_1) \wedge \ldots \wedge (x_k \vee x_k \vee x_k) \end{aligned}$$

Łatwo sprawdzić, że φ jest spełnialna wtedy i tylko wtedy, gdy φ' jest spełnialna, ale φ' ma już w kazdej klauzuli 3 literały, a wiemy, że problem 3-CNF SAT jest *NP*-zupełny. Ale właśnie opisaliśmy redukcję $\varphi \mapsto \varphi'$, zatem problem 2-CNF SAT jest także *NP*-zupełny. Jednak pamiętamy z ćwiczeń, że ten ostatni problem jest w P. Zatem wykazaliśmy, że $P = NP$.

Istotnie, problem 3-CNF SAT jest *NP*-zupełny a problem 2-CNF SAT jest w P, ale opisana redukcja działa „w złą stronę", tzn. redukuje problem łatwiejszy do trudniejszego, co nie wnosi żadnej informacji. Skądinąd, jeśli przyjąć, że w postaci 3-CNF SAT klauzula może zawierać *co najwyżej* 3 literały, to problem 2-CNF SAT redukuje się do 3-CNF SAT w sposób trywialny przez funkcję identycznościową.

3. Na wykładzie pokazano algorytm, która dla maszyny Turinga M i liczby n konstruuje obwód logiczny o n wejściach rozpoznający dokładnie słowa długości n akceptowane przez M. Ale każdą funkcję boolowską $\{0,1\}^n \to \{0,1\}$ można obliczyć obwodem o $\mathcal{O}(2^n)$ bramkach. Ewaluacji obwodu można dokonać w czasie wielomianowym od jego rozmiaru. A zatem każdą maszynę Turinga (przynajmniej, jeśli się zawsze zatrzymuje) można symulować przez maszynę pracującą w czasie wykładniczym. Co to więc za ściema z tym *Twierdzeniem o hierarchii czasowej* ?

Istotnie, jak to zauważyliśmy na wykładzie, każdy w ogóle język $L \subseteq \{0,1\}^*$ można rozpoznać ciągiem obwodów C_n, gdzie C_n ma rozmiar $\mathcal{O}(2^n)$. Jeśli L jest rozpoznawalny przez maszynę Turinga działającą w czasie $T(n)$, to odpowiedni obwód C_n można nawet efektywnie skonstruować z n, jednak czas tej konstrukcji zależy od $T(n)$ i nie potrafimy go ograniczyć z góry. Możliwa konstrukcja polega na zastosowania algorytmu z wykładu do znalezienia obwodu D_n zależnego of M (o ktorym jest mowa w treści zadania), a następnie poprzez porównywanie znalezienie obwodu rozmiaru wykładniczego (czas tej operacji jest jednak $\mathcal{O}(2^{2^n} \cdot T(n)^2)$).

Uwaga. „Twierdzenie o hierarchii czasowej" nie było dokładnie przedstawione na wykładzie, jednak kontekst zadania i analogia z *Twierdzeniem o hierarchii pamięciowej* wskazują na główny wniosek: istnieją języki o dowolnie dużej złożoności czasowej. To ostatnie stwierdzenie można też wyprowadzić z Twierdzenia o hierarchii pamięciowej i zależności pomiędzy czasem a pamięcią.

4. Przypuśćmy, że algorytm probabilistyczny rozstrzyga język L z prawdopodobienstwem błedu $p \leq \frac{1}{4}$, dając odpowiedzi *tak* lub *nie*. Zatem, jeśli dla wejścia x otrzymamy na wyjściu odpowiedź *tak*, to z prawdopodobieństwem $1-p$ zachodzi $x \in L$, a jeśli otrzymamy odpowiedź *nie*, to z prawdopodobieństwem $1-p$ zachodzi $x \notin L$.

(Np. przypuśćmy, że algorytm rozstrzyga, czy $1 = 1$, rzucając 5 razy monetą. Jeśli wypadnie 5 reszek, mówi *nie*, poza tym *tak*. Przypuśćmy, że otrzymaliśmy wynik: *nie*. Czy zaczniemy wątpić, że $1 = 1$?)

Przypomnijmy warunek algorytmu probabilistycznego klasy *BPP*

$$(\forall x)\ x \in L \Rightarrow \Pr(A(x) = \text{tak}) \geq \frac{3}{4}$$
$$(\forall x)\ x \notin L \Rightarrow \Pr(A(x) = \text{tak}) < \frac{1}{4}.$$

W powyższym tekście, zdradliwe było użycie słowa *Zatem*. Stwierdzenia następujace po tym słowie dotyczą innego zdarzenia niż to, o którym jest mowa w przytoczonej definicji algorytmu *BPP*, a mianowicie, że $x \in L$ przy założeniu, że algorytm mówi *tak*, a nie odwrotnie (podobnie dla odpowiedzi *nie*).

Prawdopodobieństwu, o którym mowa w zadaniu, można nadać pewien sens, ale jedynie jeśli coś wiemy o $\Pr(x \in L)$. Wynik eksperymentu (czyli odpowiedź algorytmu) zależy od dwóch zdarzeń losowych: wyboru x i bitów losowych algorytmu. Wtedy

$$\begin{aligned}\Pr(x \in L | A(x) = \text{tak}) &= \frac{\Pr(x \in L \wedge A(x) = \text{tak})}{\Pr(A(x) = \text{tak})} \\ &= \frac{\Pr(A(x) = \text{tak} | x \in L) \cdot \Pr(x \in L)}{\Pr(A(x) = \text{tak} | x \in L) \cdot \Pr(x \in L) + \Pr(A(x) = \text{tak} | x \notin L) \cdot \Pr(x \notin L)}\end{aligned}$$

i podobnie

$$\Pr(x \in L | A(x) = \text{nie}) = \frac{\Pr(A(x) = \text{nie} | x \in L) \cdot \Pr(x \in L)}{\Pr(A(x) = \text{nie} | x \in L) \cdot \Pr(x \in L) + \Pr(A(x) = \text{nie} | x \notin L) \cdot \Pr(x \notin L)} \quad (1)$$

Szczególna sytuacja ma miejsce, gdy $\Pr(x \in L) = 1$, a tak właśnie było w przytoczonym przykładzie (algorytm, niezależnie od wejscia x, ma rozstrzygnąć, czy $1 = 1$). Wtedy we wzorze (1) składnik $\Pr(A(x) = \text{nie} | x \notin L) \cdot \Pr(x \notin L)$ znika i otrzymujemy

$$\begin{aligned}\Pr(x \in L | A(x) = \text{nie}) &= \frac{\Pr(A(x) = \text{nie} | x \in L) \cdot \Pr(x \in L)}{\Pr(A(x) = \text{nie} | x \in L) \cdot \Pr(x \in L)} \\ &= 1,\end{aligned}$$

a zatem nie ma powodu do zwątpienia, że $1 = 1$!

5. Na wykładzie pokazano wielomianowy dowód interakcyjny, w wyniku którego Matematyk przekonuje Algorytm[1], że dwa grafy *nie* są izomorficzne. Zaadaptujmy ten protokół do sprawdzania, że dwie maszyny Turinga M_1 i M_2 są nierównoważne.

Dokładniej, Matematyk chce przekonać Algorytm, że $L(M_1) \neq L(M_2)$. W tym celu Algorytm losuje $i \in \{1, 2\}$, a następnie, dokonując kilku dalszych losowań, modyfikuje M_i, tak że maszyna liczy wprawdzie to samo, ale „wygląda" zupełnie inaczej niż M_i. Tak otrzymaną maszynę N Algorytm przedstawia Matematykowi z pytaniem: *co to jest ?* (tzn. M_1 czy M_2).

Jeśli M_1 i M_2 są nierównoważne, Matematyk podaje jedyną dobrą odpowiedź. Jeśli jednak są równoważne, to Matematyk (który nie zna wyników losowań Algorytmu) może co najwyżej zgadnąć oczekiwane i z prawdopodobieństwem $\frac{1}{2}$.

Tym samym otrzymaliśmy dowód interakcyjny, rozstrzygajacy w czasie wielomianowym, czy dwie maszyny Turinga są nierównoważne, co jak wiadomo jest problemem *nierozstrzygalnym*. Tymczasem na wykładzie twierdzono (podając nawet odpowiednio zaciemniony „dowód"), że języki rozpoznawane przez dowody interakcyjne są nie tylko rozstrzygalne, ale nawet w *PSPACE* !

W powyższym tekście błędne jest właściwie jedno zdanie:

> Jeśli jednak [M_1 i M_2] są równoważne, to Matematyk może co najwyżej zgadnąć oczekiwane i z prawdopodobieństwem $\frac{1}{2}$.

Przypomnijmy, że w przedstawionym na wykładzie dowodzie interakcyjnym dla problemu nie-izomorfizmu grafów, dla danych G_1 i G_2 o n wierzchołkach, Algorytm losuje najpierw $i \in \{1, 2\}$, a nastepnie permutację zbioru $\{1, \ldots, n\}$ i tak otrzymaną kopię przedstawia Matematykowi z pytaniem *co to jest ?* Nietrudno jest sprawdzić, że każdą izomorficzną kopię G_i można otrzymać z tym samym prawdopodobieństwem i dlatego w przypadku izomorficznych G_1 i G_2 Matematyk może jedynie „zgadnąć", jakie było i.

W naszym przykładzie, jakkolwiek niewątpliwie można wygenerować pewne modyfikacje maszyny nie zmieniające języka, to jednak sytuacja jest trudniejsza dla Algorytmu, choćby dlatego że równoważnych kopii jest nieskończenie wiele, a ponadto sprawdzenie równoważności dwóch maszyn jest zadaniem nierozstrzygalnym. Dlatego Matematyk, ktory widzi maszyny M_1 i M_2 oraz zna zasady Algorytmu, może na ogół odróżnić kopię wygenerowaną z M_1 od kopii wygenerowanej z M_2 nawet, gdy maszyny M_1 i M_2 są równoważne. Przypomnijmy, że Matematyk (który w sensie matematycznym jest po prostu strategią) nie jest związany żadnymi ograniczeniami obliczalności. W hipotetycznym protokole Matematyk jest „za silny" (a nie za słaby !) i przez to właśnie cały protokoł jest za słaby, żeby rozstrzygać równoważność maszyn Turinga. Inaczej istotnie popadlibyśmy w sprzeczność pomiędzy Twierdzeniem Shamira a nierozstrzygalnością problemu stopu.

[1] W literaturze anglo-językcznej te postaci przedstawione są jako *Prover* i *Verifier*.