ZLOŻONOŚĆ OBLICZENIOWA - WYK. 8

1. Problem osiągalności w grafie skierowanym jest **NL**-zupełny.

   Dowód: Ten dowód już w zasadzie robiliśmy. Weźmy język L rozpoznawalny maszyną niedeterministyczną M w pamięci $c \cdot \log n$. Dla danego słowa w, wygenerujmy pełen graf konfiguracji M na słowie w: po kolei generujmy wszystkie konfiguracje pamięci roboczej i wypisujmy wszystkie przejścia z niej. Każda konfiguracja jest logarytmicznie długa, więc można ją wypisać w pamięci logarytmicznej. Cały graf jest wielomianowo duży.

   Na tym grafie wystarczy rozwiązać problem osiągalności.

2. Fakt: jeśli jakiś problem $\mathcal{C}$-zupełny jest w klasie $\mathcal{D}$ oraz $\mathcal{D}$ jest zamknięta na składanie z funkcjami obliczalnymi w **L**, to $\mathcal{C} \subseteq \mathcal{D}$.

   Dowód: Oczywiste.

   Wniosek: Jeśli osiągalność w grafie skierowanym jest w **coNL**, to **NL** = **coNL**.

3. Twierdzenie Immermana-Szelepcsenyiego (1987): *nie*osiągalność w grafie skierowanym jest w **NL**.

   Dowód: Trzeba pokazać niedeterministyczny algorytm działający w pamięci logarytmicznej, który sprawdza *nie*osiągalność w grafie. Idea: w **NL** potrafimy nie tylko sprawdzić osiągalność, ale także policzyć osiągalne wierzchołki.

   - Wyjaśnić, dlaczego nie jest jasne, jak w **NL** wypisać wszystkie wierzchołki osiągalne z s. Naiwna idea (dla każdego wierzchołka spróbuj zgadnąć ścieżkę) nie działa, bo nieosiągalny wierzchołek od razu kończy obliczenie i odrzuca.
   - Najpierw taki algorytm w **NL**: mając dane liczby k i q, wypisz q różnych wierzchołków osiągalnych z wierzchołka startowego s w k krokach i zaakceptuj, albo - jeśli osiągalnych wierzchołków jest mniej niż q - odrzuć.

     Rozwiązanie: Taka pętla: najpierw ustal licznik wypisanych na 0, potem dla każdego wierzchołka w grafie, niedeterministycznie: albo go zignoruj, albo zgadnij ścieżkę do niego z v, wypisz i inkrementuj licznik, a kiedy przekroczy q, zaakceptuj.
   - Teraz główny trik: korzystając z powyższego algorytmu, indukcyjne obliczamy q_k, liczby wierzchołków osiągalnych z s w k krokach.

     Rozwiązanie: Oczywiście $q_0 = 1$. Mając dane q_k, obliczamy q_{k+1} tak: ustal q_{k+1} na 1 (s jest zawsze osiągalny), potem dla wszystkich wierzchołków v po kolei zrób taką pętlę: dla każdej krawędzi (u, v) wypisz q_k wierzchołków osiągalnych w k krokach. Jeżeli kiedykolwiek wypisałeś odpowiednie u, to inkrementuj q_{k+1}.

     Uwaga: to jest niedeterministyczny algorytm (bo wypisywanie jest niedeterministyczne), ale jeśli akceptuje, to zawsze wypisuje te same

wierzchołki. Dlatego nasz algorytm, w obliczeniach akceptujących, zawsze obliczy dobre wartości q_k.

- Teraz już łatwo. Jak?

  Rozwiązanie: Oblicz q_n i wypisz odpowiednią liczbę wierzchołków osiągalnych w n krokach. Jeśli wypisałeś wierzchołek docelowy, to odrzuć, w przeciwnym razie akceptuj.

4. **Pytanie:** Dlaczego ten sam trick nie działa w dowodzie że **NP**=**coNP?** Dlaczego nie umiemy pokazać że np. *SAT* jest w **coNP**?

   Odpowiedź: Powyższy argument opiera się na zliczaniu: umiemy w **NL** nie tylko sprawdzić osiągalność, ale i policzyć liczbę wierzchołków osiągalnych. Tymczasem nie umiemy, nawet niedeterministycznie, policzyć w czasie wielomianowym spełniających wartościowań danej formuły. Proste: takich wartościowań może być wykładniczo wiele, więc gdybyśmy chcieli je liczyć "po jednym", to nie zmieścimy się w wielomianowym czasie.

5. **Ćwiczenie:** wniosek z tw. Im-Szel: dla każdej funkcji pamięciowo konstruowalnej $S(n) \geq \log n$,

   $$\mathbf{NSPACE}(S(n)) = \mathbf{coNSPACE}(S(n))$$

   Jest to ilustracja techniki zwanej *paddingiem*.

6. Tw. Ladnera (1975): jeśli $\mathbf{P} \neq \mathbf{NP}$, to istnieje problem, który nie jest w **P** i nie jest **NP**-trudny względem redukcji w czasie wielomianowym.

   Dowód: Zakładając że SAT $\notin \mathbf{P}$ pokażemy język $L \in \mathbf{NP}$, taki że:

   - L nie jest w **P** i
   - SAT nie redukuje się do L w czasie wielomianowym.

   Wybierzmy enumerację:

   - $M_1, M_2, M_3, \ldots$ maszyn Turinga taką, że M_i działa w czasie $\mathcal{O}(n^i)$ i każdy język w **P** jest rozpoznawany przez którąś M_i.

   Język L będzie językiem SAT spaddowanym na tyle, żeby przestał być **NP**-zupełny, ale nie aż na tyle, żeby był w **P**. Definicja będzie taka:

   $$L = \{w01^k \mid |w| + k + 1 = f(|w|), w \in \text{SAT}\}$$

   dla funkcji $f(n)$, którą definiujemy przez indukcję takim programem:

   (a) weź $i = 1$ i $n = 1$.

   (b) ustal $f(n) = n^i$.

   (c) jeśli istnieje słowo w o długości $\leq \log n$ takie że M_i źle rozpoznaje należenie w do L, to $i := i + 1$.

   (d) $n := n + 1$; przejdź do kroku (b).

Fakt 1: Można sprawdzić w czasie wielomianowym, czy słowo jest właściwej postaci (tj. czy ma odpowiednio wiele jedynek). *Dowód:* Aby obliczyć $f(n)$ robimy n obrotów pętli, w każdym obrocie sprawdzamy wielomianową liczbę słów (logarytmicznej długości), a na każdym słowie uruchamiamy maszynę M_i, która działa w czasie $\mathcal{O}(\log^i n)$, a tyle czasu mamy, bo wejście ma mieć długość $f(n) \geq n^i$ (ewentualnie przerywamy wykonanie M_i i odrzucamy, jeśli jedynek jest za mało).

Fakt 2: $L \notin \mathbf{P}$. *Dowód:* Gdyby $L \in \mathbf{P}$, to któraś maszyna M_i rozpoznawałaby L, więc od pewnego momentu $f(n) = n^i$. Ale wtedy istniałaby redukcja w czasie wielomianowym z SAT do L, więc SAT byłby w $\mathbf{P}$, sprzeczność.

Pokazaliśmy przy tym, że funkcja f jest (rosnąca i) nieograniczona z góry.

Fakt 3: L nie jest $\mathbf{NP}$-zupełny. *Dowód:* Załóżmy, że SAT redukuje się do L za pomocą funkcji g obliczalnej w czasie n^k. Pokażemy wielomianowy algorytm dla SAT.

Wiemy, że istnieje n_0 takie, że dla $n \geq n_0$ mamy $f(n) > n^k$. Dla formuł o długości mniejszej niż n_0 spełnialność stablicujmy. Dla formuły ϕ długosci $n \geq n_0$, rozważmy słowo $g(\phi)$; ma ono długość co najwyżej n^k. Jeśli $g(\phi)$ nie jest postaci $\psi 011\ldots1$, to nie jest w L i odrzucamy ϕ. Jeśli jest tej postaci i jedynek jest tyle że $|g(\phi)| = f(|\psi|)$, to wiemy że ψ jest w SAT wtw. gdy ϕ jest w SAT.

Teraz: jeśli ψ jest krótsza niż n_0 to jej spełnialność mamy stablicowaną. Jeśli jest dłuższa to $|\phi^k| \geq |g(\phi)| = f(|\psi|) > |\psi|^k$, więc ψ jest krótsza niż ϕ i można ten algorytm zaaplikować w pętli aż długość formuły spadnie poniżej n_0. To daje wielomianowy algorytm dla SAT.

7. Problemy CSP i hipoteza o dychotomii.

8. Dużo dowodów w teorii złożoności używa maszyn Turinga jako czarnych skrzynek. Dowody są postaci:

   - Załóżmy, że istnieje maszyna M działająca w czasie ..., rozpoznająca ...
   - Zróbmy z niej maszynę M', która wykona M wiele razy w pętli ...
   - ...a potem zaneguje wynik, wykona się na każdej maszynie Turinga ...
   - w końcu dostajemy maszynę M'''''', o której wiadomo że nie może istnieć, więc M nie istnieje.

   Takie dowody się *relatywizują*, tzn. działają także wtedy, jeśli każda maszyna na świecie ma dostęp do pewnej ustalonej wyroczni.

   Przykłady dowodów relatywizujących się: twierdzenie Turinga o nierozstrzygalności, twierdzenia o hierarchii, twierdzenia o luce, twierdzenia

Ladnera, Immermana-Szelepcsenyiego, Savitcha... Z drugiej strony, rozmaite dowody oparte na obwodach się nie relatywizują (bo nie jest jasne co to w ogóle jest wyrocznia dla obwodu).

Kolejne twierdzenie pokazuje, że żadnym rozumowaniem, które się relatywizuje, nie można rozstrzygnąć problemu **P** vs. **NP**.

9. Tw. Baker-Gill-Solovay (1975): Istnieją języki A i B takie, że:

$$\mathbf{P}^A = \mathbf{NP}^A \qquad \text{i} \qquad \mathbf{P}^B \neq \mathbf{NP}^B.$$

Dowód: Za A wystarczy wziąć QBF. Porachujmy:

$$\mathbf{NP}^{\mathrm{QBF}} \subseteq \mathbf{NPSPACE} = \mathbf{PSPACE} = \mathbf{P}^{\mathrm{QBF}}.$$

Po kolei kroki:

- zamiast pytać wyroczni QBF o słowo, maszyna może sama obliczyć odpowiedź: pytanie jest wielomianowo długie, więc mieści się w wielomianowej pamięci, a QBF jest rozwiązywalny w wielomianowej pamięci.
- tw. Savitcha,
- ostatnia inkluzja wynika bezpośrednio z **PSPACE**-zupełności (w sensie Karpa) problemu QBF.
- ćwiczenie: sprawdzić co się rozsypuje w tym dowodzie, jeśli zamiast QBF weźmiemy SAT.

Teraz skonstruujemy pewną wyrocznię B i rozważymy język

$$L = \{1^n \mid \text{pewne słowo } w \text{ długości } n \text{ jest w } B\}.$$

Po pierwsze, $L \in \mathbf{NP}$, bo maszyna może zgadnąć właściwe słowo w.

Deterministyczna maszyna rozpoznająca L ma problem: może tylko pytać wyroczni o kolejne słowa i nie starczy jej czasu by sprawdzić wszystkie. Trzeba tylko zaprojektować B tak, by faktycznie nie dało się zrobić nic mądrzejszego.

Wybierzmy enumerację $M_1, M_2, M_3, \ldots$ wszystkich maszyn Turinga z wyrocznią, działających w czasie wielomianowym (wyrocznia nie jest częścią definicji maszyny), podobnie jak w dowodzie tw. Ladnera. Będziemy konstruować język B po trochu, oszukując kolejne maszyny.

Skonstruujemy języki $B_1 \subseteq B_2 \subseteq B_3 \cdots B$ i długości $n_1 \leq n_2 \leq n_3 \leq \cdots$ takie, że:

- $B = \bigcup_{i \in \mathbb{N}} B_i$,
- $M_i^{B_i}$ źle rozpoznaje słowo 1^{n_i},
- M_i^B zgadza się z $M_i^{B_i}$ na słowie 1^{n_i}.

Zaczynamy od $B_0 = \emptyset$. Potem dla $i = 1, 2, \ldots$:

- wybieramy n_i takie duże, żeby żadna maszyna M_j (dla $j < i$) na słowie 1^{n_j} nie była w stanie wyprodukować zapytania długości n_i (to gwarantuje, że oszukane wcześniej maszyny nadal są oszukane), oraz żeby M_i na słowie 1^{n_i} działała na pewno w mniej niż 2^{n_i} krokach.
- uruchom M_i z wyrocznią B_{i-1} na słowie 1^{n_i}. Zauważmy, że ta wyrocznia na każde pytanie długości n_i odpowiada negatywnie.
- jeśli M_i zaakceptowała, to weź $B_i = B_{i-1}$. Wtedy $1^{n_i} \notin L$ i oszukaliśmy maszynę M_i.
- jeśli M_i odrzuciła, to znajdź słowo w długości n_i, o które maszyna M_i nie zapytała (takie słowo musi być, bo M_i zrobiła mniej niż 2^{n_i} kroków) i ustal $B_i = B_{i-1} \cup \{w\}$. Wtedy $1^{n_i} \in L$ i oszukaliśmy maszynę M_i.

Język B jest obliczalny (w czasie wykładniczym), ale dla tego twierdzenia to nieważne.