ZLOŻONOŚĆ OBLICZENIOWA - WYK. 7

1. Redukcje.
   - Idea: problem A redukuje się do B jeżeli wiemy jak, umiejąc zrobić B, łatwo zrobić A.
   - Jeśli B jest łatwy, to A jest łatwy.
   - Jeśli A jest trudny, to B jest trudny.
2. Redukcja w sensie Turinga:
   - Maszyna Turinga M z wyrocznią dla języka K:
     - osobna taśma do zapisywania zapytań dla wyroczni (tylko do zapisu, tj. głowica chodzi tylko w prawo)
     - specjalny stan $q_?$ do pytania wyroczni
     - po wejściu w stan $q_?$, pierwsza komórka taśmy zapytań jest nadpisywana przez 0 lub 1, zależnie od tego czy słowo na taśmie zapytań należy do K. Głowica na taśmie zapytań przeskakuje na początek taśmy.
   - język L redukuje się do K jeśli istnieje maszyna M z wyrocznią dla K, która rozpoznaje L.
   - ograniczając zasoby maszynie M można mówić o redukcji w czasie wielomianowym, pamięci logarytmicznej itd.
3. Redukcja w sensie Karpa:
   - język $L \subseteq \Sigma^*$ redukuje się do $K \subseteq \Gamma^*$ jeśli istnieje funkcja obliczalna (w czasie wielomianowym, pamięci logarytmicznej itd.) $f : \Sigma^* \to \Gamma^*$ taka, że $w \in L \iff f(w) \in K$.
4. Fakt: jeśli L redukuje się do K w sensie Karpa, to w sensie Turinga też (przy tych samych ograniczeniach zasobów).

   Dowód: weźmy maszynę obliczającą funkcję f. Potraktujmy ją jako maszynę z wyrocznią, która na sam koniec zadaje jedno pytanie.
5. Które pojęcie redukcji jest lepsze?
   - redukcja Turinga lepiej odpowiada intuicji, np. jeśli umiemy szukać cykli Hamiltona w podanych grafach, to potrafimy łatwo szukac cykli Hamiltona w *dwóch* podanych grafach.
   - redukcję Turinga czasem zbyt łatwo znaleźć, np. pozwala zredukować język do jego dopełnienia, co zaciera różnice między **NP** i **coNP**.
   - w praktyce umiemy pokazywać redukcję Karpa, więc skoro jest trudniejsza, to stosujemy to pojęcie.
   - na tej samej zasadzie, wolimy redukcje w pamięci logarytmicznej niż w czasie wielomianowym.

6. Język L jest *$\mathcal{C}$-zupełny* (ze względu na redukcje Karpa w pamięci logarytmicznej) jeśli:
   - $L \in \mathcal{C}$, oraz
   - każdy język w klasie $\mathcal{C}$ redukuje się do L w sensie Karpa w pamięci logarytmicznej (czyli L jest *$\mathcal{C}$-trudny*.
7. To zadziwiające, że problemy zupełne w ogóle istnieją.
8. Twierdzenie Cooka: Problem SAT jest **NP**-zupełny.
   - SAT: dla danej formuły zdaniowej ze zmiennymi (zapisanej infiksowo z pełnym nawiasowaniem, zmienne jako ich numery) stwierdzić, czy jest spełnialna.
   - *Dowód:* Po pierwsze, SAT $\in$ **NP**: zgadujemy wartościowanie spełniające.

     Teraz trudność: Dla danego języka L rozpoznawalnego niedeterministyczną maszyną M w czasie ograniczonym przez wielomian $p(n)$ podajemy, jak w pamięci logarytmicznej ze słowa w skonstruować formułę ϕ taką, że $w \in L$ wtw. ϕ jest spełnialna.

     Są trzy rodzaje zmiennych:
     - t_{icn}: w kroku n w miejscu taśmy i jest litera c.
     - s_{qn}: w kroku n maszyna jest w stanie q.
     - g_{in}: w kroku n głowica jest na pozycji i.

     Tych zmiennych jest wielomianowo wiele: zmiennych t i g jest $\mathcal{O}(p^2(n))$, zmiennych s tylko $\mathcal{O}(p(n))$.

     Formuła to (wielomianowo) wielka koniunkcja rzeczy do sprawdzenia:
     - początkowa zawartość taśmy, pozycja głowicy i stan są takie jak trzeba:
       $$s_{q_0 0} \wedge q_{01} \wedge t_{10w_1} \wedge \cdots \wedge t_{n0w_n}$$
     - najwyżej jeden stan na raz
       $$\bigwedge_{1 \le n \le p(n)} \bigwedge_{q \neq q'} (\neg s_{qn} \vee \neg s_{q'n})$$
     - najwyżej jedna pozycja głowicy na raz ($g_{in} \to \neg g_{jn}$ dla $i \neq j$)
     - najwyżej jeden symbol na raz w jednym miejscu
     - symbol się nie zmienia jeśli głowica jest w innym miejscu
       $$\bigwedge_{i \neq j} \bigwedge_{n} \bigwedge_{c} g_{jn} \wedge t_{icn} \to t_{(i+1)cn}$$
     - możliwe przejścia:
       $$(t_{icn} \wedge s_{qn} \wedge g_{in}) \to \bigvee (t_{(i \pm 1)c'(n+1)} \wedge s_{q'(n+1)} \wedge g_{(i \pm 1)(n+1)})$$

– akceptacja: $\bigvee_{q \in Q_A} s_{qp(n)}$.

Z tych definicji od razu widać że tę formułę można wypisać w pamięci logarytmicznej.

9. Problem HORNSAT: spełnialność formuł CNF, w którym każda klauzula ma co najwyżej 1 literał pozytywny.

   Fakt: HORNSAT jest **P**-zupełny (ze względu na redukcje w pamięci logarytmicznej).

   Dowód: Problem jest w **P** (saturacja a'la Prolog). Trudność: jeśli w dowodzie tw. Cooka maszyna M jest deterministiczna, to wszystko wygląda tak samo, ale nie ma dużej alternatywy w przedostatnim punkcie. Wtedy cała formuła jest w postaci HORN-CNF.

10. Ćwiczenie: klasa **polyL** nie ma problemów zupełnych ze względu na redukcje w **L**. Wniosek: $\mathbf{P} \neq \mathbf{polyL}$.

11. Prawie każdy problem w **L** jest **L**-zupełny.

12. Problem QBF jest **PSPACE**-zupełny.

    - Problem QBF: dana formuła zdaniowa z poumerowanymi zmiennymi $\phi(x_1, \ldots, x_k)$. Rozstrzygnąć, czy prawdziwe jest zdanie

      $$\exists x_1 . \forall x_2 . \exists x_3 \forall x_4 . \cdots \phi(x_1, \ldots, x_k).$$

    - Uwaga: problem pozostaje **PSPACE**-zupełny nawet jeśli ograniczymy ϕ do postaci CNF. Dowód jest trochę trudniejszy, ale podobny.

    Dowód: Najpierw pokażmy QBF $\in$ **PSPACE**: instancję można potraktować jako obwód logiczny (bramki binarne) o liniowej głębokości. Można ten obwód przejrzeć backtrackingiem, ze stosem liniowej głębokości.

    Teraz **PSPACE**-trudność. Stosujemy ten sam trik co w dowodzie tw. Savitcha. Niech język L będzie rozpoznawalny maszyną M w pamięci wielomianowej. Niech $p(n)$ będzie takim wielomianem, że konfiguracje maszyny M na słowie długości n można zakodować w $p(n)$ bitach.

    Dla danego słowa w długości n, dla każdej liczby naturalnej i napiszemy formułę $\psi_i(x_1, \ldots, x_{p(n)}, y_1, \ldots, y_{p(n)})$ prawdziwą dokładnie wtedy, gdy z konfiguracji zakodowanej bitami x można się dostać do konfiguracji zakodowanej bitami y za pomocą co najwyżej 2^i kroków maszyny M. Skoro konfiguracji jest $2^{p(n)}$, to maszynę M można przetłumaczyć na formułę $\psi_{p(n)}(\bar{x}, \bar{y})$ gdzie $\bar{x}$ to wektor kodujący konfigurację początkową, a $\bar{y}$ końcową.

    Formułę $\phi_0(\bar{x}, \bar{y})$ napisać łatwo; albo $\bar{x} = \bar{y}$ albo $\bar{y}$ można uzyskać z $\bar{x}$ jednym krokiem maszyny M, co się zapisuje (długą) formułą zdaniową, bez kwantyfikatorów. Tę formułę można wypisać w pamięci logarytmicznej: wystarczy w licznikach trzymać pozycje głowic i tam manipulować konfiguracjami, a poza tym wypisywać same równości między $\bar{x}$ i $\bar{y}$.

Naiwny pomysł: $\phi_{i+1}(\bar{x},\bar{y}) = \exists\bar{z}.(\phi_i(\bar{x},\bar{z}) \wedge \phi_i(\bar{z},\bar{y}))$. To nie działa, bo formuła rośnie wykładniczo. Sprytny trik to użyć formuły ϕ_i tylko raz:

$$\phi_{i+1}(\bar{x},\bar{y}) = \exists\bar{z}.\forall\bar{r}.\forall\bar{t}.\big((\bar{r}=\bar{x}\wedge\bar{t}=\bar{z})\vee(\bar{r}=\bar{z}\wedge\bar{t}=\bar{y})\rightarrow\phi_i(\bar{r},\bar{t})\big)$$

To nie jest w postaci QBF, ale kwantyfikatory w ϕ_i można przesunąć na początek, jeżeli zadbamy żeby za każdym razem kwantyfikowały po świeżych zmiennych (np. indeksując je poziomami i).

Pytanie jak tę formułę wypisać w pamięci logarytmicznej? (Uwaga: przy **PSPACE** zwykle jednak rozluźniamy rygory i pozwalamy na redukcje w **P**)

- najpierw blok kwantyfikatorów (zmienne indeksowane poziomami i),
- potem blok takich samych prefiksów zdaniowych (na początku trzeba porównać z konfiguracjami początkowymi i końcowymi)
- na końcu formuła ϕ_0 i nawiasy zamykające.

13. 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**.

14. Twierdzenie Immermana-Szelepcsenyiego: *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.

    - 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.

15. **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.