ZLOŻONOŚĆ OBLICZENIOWA - WYK. 2

1. **Twierdzenie Sipsera**: Dla dowolnej maszyny M działającej w pamięci $S(n)$ istnieje maszyna M' taka, że:

   - $L(M) = L(M')$,
   - M' działa w pamięci $S(n)$,
   - M' ma własność stopu.

   Dowód: ze skryptu Damiana Niwińskiego.

   - jak wygląda graf konfiguracji maszyny M na konkretnym słowie wejściowym w (pojedyncze cykle i drzewa; fragmenty z konfiguracjami akceptującymi to drzewa).
   - jak sprawdzić czy M akceptuje słowo w? Można po prostu wykonać maszynę M, czyli przeglądać graf wzdłuż krawędzi poczynając od konfiguracji początkowej. *Problem:* Jak wykryć pętlenie?
     - Pomysł I: wypisywać wszystkie konfiguracje po kolei i porównywać aktualną z poprzednimi. *Problem:* to zużywa o wiele więcej pamięci niż M.
     - Pomysł II: liczyć kroki. Skoro maszyna M używa $f(n)$ pamięci, to jej konfiguracje można zapamiętać w pamięci rozmiaru $f(n)$. Można więc zrobić licznik o tym rozmiarze. *Problemy:*
       (i) Widząc maszynę M być może nie umiemy stwierdzić jaka jest funkcja $f(n)$ (NB. z pomysłu Sipsera okaże się że umiemy). To jest problem pozorny: wiemy że taka funkcja istnieje, więc istnieje maszyna, która na początku rezerwuje licznik o tym rozmiarze. Być może nie umiemy jej obliczyć z M, ale to nie szkodzi.
       (ii) Ale czy na pewno taka maszyna istnieje? Funkcja $f(n)$ musi być pamięciowo konstruowalna (powiemy o tym później).
       (iii) Wymogu pamięciowej konstruowalności można uniknąć: maszyna symulująca może nie rezerwować całego licznika od razu, tylko zwiększać go w miarę potrzeby, tak by zawsze był taki duży jak pamięć użyta do tej pory przez maszynę symulowaną. Jeżeli taki licznik się w pewnej chwili przepełni, to maszyna symulowana się pętli. Ale jest jeszcze jeden problem:
       (iv) Jeżeli $f(n)$ jest mniejsza od $\log n$, to konfiguracji maszyny M nie zmieścimy w pamięci $f(n)$, bo do konfiguracji należy położenie głowicy na taśmie wejściowej. Języki rozpoznawalne w pamięci mniejszej niż $\log n$ nie są zbyt interesujące, ale istnieją.
     - Pomysł III (Sipser): przeglądać graf od tyłu, zaczynając od konfiguracji akceptujących.

        * *zaleta:* problem cykli w ogóle znika, bo nie ma ich w tych fragmentach grafu.
        * *wada:* konfiguracji akceptujących jest nieskończenie wiele, a w ich drzewach są nieskończone ścieżki.
    - Procedura $Search(C, k, w)$: działa na taśmie wejściowej z zapisanym słowem w, i z taśmą roboczą, na której zaznaczono k pozycji. Zaczynając od konfiguracji C, przechodzi graf DFS-em szukając konfiguracji początkowej, przy czym ogranicza się tylko do konfiguracji z pamięcią roboczą co najwyżej k.
    - Jeżeli z C nie ma kroku do innej k-konfiguracji, to ta procedura ma własność stopu i działa w pamięci k. (Wyjaśnić jak zrobić DFS w pamięci k: trzeba pamiętać czy przyszliśmy z dołu czy z góry drzewa, a jeśli z dołu to z którego syna.)
    - oto algorytm symulujący maszynę M od tyłu:
        - dla kolejno rosnących k wykonaj następujące kroki:
            * przejrzyj wszystkie k-konfiguracje i wybierz te, z których jest przejście do $k+1$-konfiguracji.
            * dla każdej takiej C poszukaj $Search(C, k, w)$. Jeśli się udało, to zwiększ k i od początku.
        - Po tej pętli wiemy, że M na słowie w używa dokładnie k pamięci. Teraz trzeba tylko sprawdzić czy akceptuje.
        - w tym celu robimy $Search(C, k, w)$ z wszystkich k-konfiguracji akceptujących.

2. Wniosek z tw. Sipsera: jeśli język L jest częściowo obliczalny, ale nie rozstrzygalny, to każda maszyna M, która go rozpoznaje, na pewnym słowie w zużywa nieskończenie wiele pamięci.

   Dowód: Gdyby na każdym słowie M zużywała tylko skończoną pamięć, to istniałaby funkcja $S(n)$ taka, że M działa w pamięci $S(n)$. Z tw. Sipsera, język L byłby rozstrzygalny.

3. Funkcja $f(n)$ jest *czasowo konstruowalna* jeśli istnieje maszyna M, która na słowie wejściowym 1^n:

   - produkuje słowo o długości dokładnie $f(n)$,
   - działa w czasie $\mathcal{O}(f(n))$.

   Ćwiczenia: jeśli f i g są konstruowalne to $f+g$, $f \cdot g$, f^g też. Funkcje n, $n \log n$, n^k, k^n są konstruowalne. Uwaga: $\log n$, ani żadna funkcja asymptotycznie mniejsza od n, nie jest czasowo konstruowalna.

4. Funkcja $f(n)$ jest *pamięciowo konstruowalna* jeśli istnieje maszyna M, która na słowie wejściowym 1^n:

   - produkuje słowo o długości dokładnie $f(n)$,

- działa w pamięci $\mathcal{O}(f(n))$.

Ćwiczenia: Każda funkcja czasowo konstruowalna jest pamięciowo konstruowalna. Jeśli f i g są konstruowalne to $f+g$, $f \cdot g$, f^g też. Funkcje n, $\log n$, n^k, k^n są konstruowalne. Funkcja $\log\log n$ nie jest pamięciowo konstruowalna; kolosalnie szybko rosnące funkcje też nie.

5. Mogłoby się wydawać, że jeśli jakaś maszyna działa w pamięci $S(n)$, to funkcja $S(n)$ jest pamięciowo konstruowalna. Tymczasem to jest nieprawda. Jeśli maszyna M działa w pamięci dokładnie $S(n)$, to dla każdego n istnieje słowo w długości n takie, że M zużywa $S(n)$ pamięci. Ale próbując skonstruować funkcję $S(n)$, dostajemy słowo z 0^n. I nie wiadomo jak skonstruować to "najgorsze" słowo w. Jeśli $S(n) > n$ to spoko, możemy przejrzeć wszystkie słowa długości n i na każdym uruchomić M, ale jeśli $S(n)$ jest małe, to nie możemy tego zrobić.

   W istocie:

   - *Ćwiczenie:* Istnieje język, który nie jest regularny, ale rozpoznawalny w pamięci $\log\log n + 1$.
   - *Twierdzenie:* Funkcja $\log\log n$ nie jest pamięciowo konstruowalna.

     Dowód: Załóżmy że maszyna M działa w pamięci $\mathcal{O}(\log\log n)$; niech dla dużych długości n używa co najwyżej $c\log\log n$ komórek. Wtedy różnych konfiguracji "wewnętrznych", tj. z pominięciem pozycji głowicy na taśmie wejściowej, ma $\log^d n$, dla pewnej stałej d. Weźmy takie m, że $\log^d m < m$.

     Rozważmy bieg maszyny M na słowie 0^m. Weźmy jakiś jego kawałek, który zaczyna się na początku słowa, kończy na końcu słowa, a po drodze nie odwiedza ani początku ani końca. Gdzieś na tym kawałku powtarzają się konfiguracje wewnętrzne. Jeśli odległość między wystąpieniami jest $k > 0$, to ten kawałek biegu na słowie 0^{m+nk} (dla dowolnego n) wygląda tak samo jak na słowie 0^m.

     Na różnych takich kawałkach mogą być różne k, ale na słowie $0^{m+m!}$ wszystkie te kawałki wyglądają tak samo; w szczególności używają takiej samej pamięci roboczej. To samo na słowach

     $$0^m, 0^{m+m!}, 0^{m+2m!}, \ldots$$

     Wobec tego M nie używa zawsze $\log\log n$ pamięci, nawet asymptotycznie.

     Uwaga: To pokazuje, że żadna funkcja rosnąca w $o(\log n)$ nie jest pamięciowo konstruowalna.
   - ALE: istnieje nieograniczona funkcja w $O(\log\log n)$ pamięciowo konstruowalna. (Hint: nie jest rosnąca). *Rozwiązanie:*

     $$S(n) = \log(\min\{i \mid i \text{ nie jest dzielnikiem } n\})$$

     .

6. Twierdzenie o hierarchii pamięciowej: jeśli funkcja $g(n)$ jest pamięciowo konstruowalna, $g(n) \geq \log n$ i $f(n) = o(g(n))$, to

$$\mathbf{DSPACE}(f(n)) \subsetneq \mathbf{DSPACE}(g(n)).$$

Dowód: Jest to wariant dowodu Turinga o nierozstrzygalności problemu stopu. Rozważmy język:

$$L = \{([M], w) \mid M \text{ odrzuca słowo } ([M], w) \text{ w pamięci } \leq g(|[M], w|)\}$$

Po pierwsze, $L \notin \mathbf{DSPACE}(f(n))$. Jeśli bowiem maszyna M' działa w pamięci $f(n)$, to dla pewnego odpowiednio długiego słowa w na słowie $[M'], w$ używa mniej niż $g(|[M'], w|)$ pamięci i dostajemy sprzeczność: M' akceptuje słowo $[M'], w$ wtw. to słowo nie jest w L, więc M' nie rozpoznaje języka L.

Po drugie, L można rozpoznać w pamięci $\mathcal{O}(g(n))$. Robimy to tak:

- na słowie o długości n, korzystając z pamięciowej konstruowalności, rezerwujemy $g(n)$ pamięci roboczej.
- sprawdzamy czy słowo jest postaci $[M], w$; w przeciwnym razie odrzucamy. Do tego potrzeba $\log n$ pamięci.
- nie mamy gwarancji czy M ma własność stopu, więc musimy wykrywać pętlenie. W tym celu rezerwujemy licznik rozmiaru $O(g(n))$.
- symulujemy maszynę M na słowie w w zarezerwowanej pamięci, za każdym krokiem inkrementując licznik. Jeśli M spróbuje wyjść poza pamięć lub czas, to odrzucamy. Jeśli M zaakceptuje, odrzucamy. Jeśli odrzuci, akceptujemy.

7. Twierdzenie o hierarchii czasowej: jeśli funkcja $g(n)$ jest czasowo konstruowalna i $f(n) = o(g(n))$, to

$$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n)^2).$$

Dowód: Podobnie jak poprzednio, rozważmy język:

$$L = \{([M], w) \mid M \text{ odrzuca słowo } ([M], w) \text{ w czasie } \leq g(|[M], w|)\}$$

Jak poprzednio, $L \notin \mathbf{DTIME}(f(n))$. Argument jest taki sam.

Po drugie, L można rozpoznać w czasie $O(g(n)^2)$. Robimy to tak:

- sprawdzamy czy słowo jest postaci $[M], w$; w przeciwnym razie odrzucamy. Do tego potrzeba czasu $O(n)$.
- korzystając z czasowej konstruowalności, na jednej z taśm roboczych rezerwujemy licznik rozmiaru $c \cdot g(n)$, dla odpowiedniej stałej c.

- symulujemy maszynę M na słowie w na taśmie wejściowej, za każdym krokiem inkrementując licznik (zapisany unarnie). Jeśli M spróbuje wyjść poza czas, to odrzucamy. Jeśli M zaakceptuje, odrzucamy. Jeśli odrzuci, akceptujemy. Symulacja każdego kroku zajmuje czas $g(n)$: na słowie wejściowym trzeba odszukać w kodzie maszyny M następny krok do wykonania, a potem wrócić głowicą we właściwe miejsce.

8. Tak naprawdę twierdzenie o hierarchii czasowej jest mocniejsze:

$$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n)\log(g(n))).$$

   To samo działa nawet wtedy, gdy do definiowana **DTIME** użyje się maszyn jednotaśmowych, chociaż dowód jest trudniejszy. Jedyny problem polega na sprytnym ograniczaniu narzutu na symulacje maszyn maszyną uniwersalną.

9. Wnioski z twierdzeń o hierarchii: $\mathbf{DTIME}(n^k) \subsetneq \mathbf{DTIME}(n^{k+1})$, $\mathbf{DSPACE}(n^k) \subsetneq \mathbf{DSPACE}(n^{k+1})$, $\mathbf{L} \subsetneq \mathbf{PSPACE}$, $\mathbf{P} \subsetneq \mathbf{EXPTIME}$. Tę ostatnią nierówność pokazuje się tak:

$$\mathbf{P} \subseteq \mathbf{DTIME}(2^n) \subsetneq \mathbf{DTIME}(2^{2n}) \subseteq \mathbf{EXPTIME}$$

10. Twierdzenie o luce czasowej: istnieje taka obliczalna funkcja $f(n)$, że $\mathbf{DTIME}(f(n)) = \mathbf{DTIME}(2^{f(n)})$.

    Dowód: Wymyślamy funkcję $f(n)$ taką, że żadna maszyna nie zatrzymuje się między $f(n)$ a $2^{f(n)}$ kroków.

    - Ponumerujmy wszystkie maszyny Turinga w sposób obliczalny.
    - niech $P(i,k)$ będzie spełnione wtw. gdy każda z pierwszych i maszyn, na każdym wejściu o długości co najwyżej i, zatrzymuje się po mniej niż k krokach lub po więcej niż 2^k krokach (albo się pętli). Relacja P jest oczywiście obliczalna.
    - dla liczby i, zdefiniuj $k_1 = i$, a potem $k_{j+1} = 2^{k_j}$.
    - każde wejście długości co najwyżej i może sfalsyfikować tylko jedno $P(i,k_j)$, więc istnieje takie $j < |\Gamma|^i$ że $P(i,k_j)$ jest prawdziwe.
    - definiujemy $f(i) = k_j$. Ta funkcja jest obliczalna.

    Weźmy teraz dowolną maszynę (o numerze M_k) działającą w czasie $\mathbf{DTIME}(2^{f(n)})$. Dla każdego wejścia x dłuższego niż k maszyna nie może się zatrzymać między $f(|x|)$ a $2^{f(|x|)}$ krokami, więc musi się zatrzymać co najwyżej po $f(|x|)$ krokach. Wejść krótszych niż k jest stała liczba. W sumie język tej maszyny można rozpoznać w $\mathbf{DTIME}(f(n))$.

    Oczywiście funkcja $f(n)$ nie jest czasowo konstruowalna. Można ją obliczyć, ale w o wiele dłuższym czasie.

    Takie samo twierdzenie o luce zachodzi dla złożoności pamięciowej.