Wykłady ze Złożoności Obliczeniowej *

Antoni Kościelski

1 Wprowadzenie

1.1 Przykład wielomianowego problemu decyzyjnego: problem osiągalności

Przypuśćmy, że mamy dany graf skierowany $G = \langle V, E\rangle$ (a więc $E \subseteq V^2$) oraz dwa wierzchołki tego grafu $s, t \in V$. Interesuje nas, czy istnieje w grafie G ścieżka od s do t, a więc, czy istnieje ciąg $v_0, v_1, \ldots, v_n$ wierzchołków grafu G spełniający warunek $(v_i, v_{i+1}) \in E$ dla wszystkich $i < n$ taki, że $v_0 = s$ oraz $v_n = t$. Jeżeli istnieje taki ciąg, to mówimy, że t jest osiągalne z s w grafie G.

Problemem osiągalności nazywamy zadanie polegające na ustaleniu, czy w danym skończonym grafie skierowanym istnieje ścieżka między danymi elementami. Problem ten formalizujemy jako zbiór wszystkich trójek złożonych z grafu G (skończonego i skierowanego) i jego dwóch wierzchołków s, t takich, że t jest osiągalne w grafie G z s.

Następujący algorytm rozwiązuje problem osiągalności:

$S := \{s\}$;
$M := \{s\}$;
powtarzaj
  wybierz $x \in S$;
  $S := (S \setminus \{x\}) \cup \{y \in V : (x, y) \in E \wedge y \notin M\}$;
  $M := M \cup \{y \in V : (x, y) \in E\}$
aż $S = \emptyset$;
jeżeli $t \in M$,
  to zwróć *'tak'*, **w przeciwnym razie zwróć** *'nie'*

*Wykłady są prowadzone według książki Computational Complexity Christosa Papadimitriou

W podanym algorytmie próbujemy przejść od s do t idąc wzdłuż krawędzi danego grafu. Aby sprawdzić, czy z x można dojść do t, przesuwamy się wzdłuż krawędzi z x do y i sprawdzamy, czy można dojść z y do t. Oczywiście, sprawdzanie to musi zostać powtórzone dla wszystkich y osiągalnych z x wzdłuż pojedyńczej krawędzi. Zbiór S składa się z tych wierzchołków, które będziemy sprawdzać badając, czy można z nich dojść do t. Zbiór M składa się z wierzchołków, o których wiemy, że są osiągalne z s. Uznajmy za oczywiste, że podany algorytm rozwiązuje problem osiągalności.

Nietrudno zauważyć, że zawsze $S \subseteq M$ poza momentami, w którym S lub M są zmieniane przez wykonanie podanych podstawień, a także, że każdy wierzchołek może zostać wybrany podczas wykonywania instrukcji **wybierz** najwyżej jeden raz.

1.2 Przykład problemu optymalizacyjnego: problem maksymalnego przepływu

Siecią będziemy nazywać graf skierowany $G = (V, E)$ z funkcją $c : E \to N$ i dwoma wyróżnionymi elementami $s, t \in V$. Element s nazywamy źródłem sieci, element t – odpływem. Zakładamy, że w grafie G nie ma krawędzi wchodzących do s i wychodzących z t. W tym kontekście wierzchołki grafu nazywamy węzłami. Krawędzie takiego grafu możemy interpretować jako odcinki rurociągów, a cały graf – jako sięć rurociągów. Poszczególnym odcinkom funkcja c przyporządkowuje maksymalny przepływ wzdłuż danego odcinka. Taka sieć rurociągów umożliwia przesyłanie cieczy od węzła s do t.

Przypuśćmy, że mamy daną sieć G z funkcją c, źródłem s i odpływem t. Funkcję f taką, że $f(x, y) \le c(x, y)$ dla dowolnej krawędzi $(x, y) \in E$ nazywamy przepływem, jeżeli dla dowolnego $v \in V \setminus \{s, t\}$ zachodzi następująca równość:

$$\sum_{(x,v)\in E} f(x, v) = \sum_{(v,y)\in E} f(v, y).$$

Równość tę możemy interpretować jako stwierdzenie, że ilość cieczy dopływającej do dowolnego węzła $v \ne s, t$ jest równa ilości cieczy odpływającej. Wielkością przypływu f nazywamy liczbę

$$s(f) = \sum_{(s,x)\in E} f(s, x).$$

Przypływ nazywamy maksymalnym, jeżeli jego wielkość jest maksymalna.

Problem maksymalnego przepływu to zadanie polegające na znalezieniu maksymalnego przepływu dla danej sieci lub wielkości maksymalnego przepływu.

Przypuśćmy, że mamy sieć $G = (V, E)$ z funkcją c, źródłem s i odpływem t, oraz przepływ f. Zdefiniujemy sieć G_f z funkcją c_f, źródłem s i odpływem t. Załóżmy, że dowolne dwa wezły w grafie G mogą być połączone najwyżej w jedną stronę (a więc jeżeli $(x, y) \in E$, to $(y, x) \notin E$). Jeżeli ten warunek nie jest spełniony, to każdej pary węzłów x, y takich, że $(x, y) \in E$ i $(y, x) \in E$ dodajemy węzeł z, krawędź (y, x) zastępujemy przez krawędziami (y, z) oraz (z, x) i określamy nową funkcję c' równą c dla dotychczasowych krawędzi i spełniającą warunek $c'(y, z) = c'(z, x) = c(y, x)$. Zakładamy także, że w grafie G nie ma krawędzi prowadzących od pewnego do wierzchołka do niego ($(x, x) \notin E$).

Graf G_f definiujemy jako $(V, E \cup \{(y, x) : (x, y) \in E\})$. Funkcję c_f definiujemy wzorem

$$c_f(x, y) = \begin{cases} c(x, y) - f(x, y) & \text{jeśli } (x, y) \in E \\ f(y, x) & \text{w przeciwnym razie.} \end{cases}$$

Nietrudno zauważyć, że jeżeli w grafie G_f istnieje ścieżka $x_0, x_1, \ldots x_n$ (bez zapętleń) taka, że $x_0 = s$, $x_n = t$ i $c_f(x_i, x_{i+1}) > 0$ dla wszystkich $i < n$, to możemy zdefiniować przepływ w sieci G wielkości większej niż $s(f)$. Niech $d = \min\{c_f(x_i, x_{i+1}) : i < n\}$. Interesujący nas przepływ f' definiujemy wzorem

$$f'(x, y) = \begin{cases} f(x, y) + d & \text{jeśli } \exists i < n \ (x_i, x_{i+1}) \in E \wedge (x_i, x_{i+1}) = (x, y) \\ f(x, y) - d & \text{jeśli } \exists i < n \ (x_i, x_{i+1}) \notin E \wedge (x_i, x_{i+1}) = (y, x) \\ f(x, y) & \text{w przeciwnym razie.} \end{cases}$$

Tak zdefiniowana funkcja f' jest przepływem i $s(f') = s(f) + d$.

Powyższe spostrzeżenie sugeruje następujący algorytm znajdujący maksymalny przepływ:

$f :=$ **funkcja określona w zbiorze E, stale równą 0**;
dopóki w G_f **jest ścieżka od** s **do** t (o wyżej podanych własnościach)
wykonaj podstawienie:
  $f := f'$; (gdzie f' jest funkcją zdefiniowaną tak, jak to zostało podane)
zwróć f.

Twierdzenie 1.1 *Podany algorytm znajduje maksymalny przepływ.*

Dowód. Przypuśćmy, że mamy sieć $G = (V, E)$ z funkcją c, źródłem s i odpływem t. Zbiór S nazywamy przekrojem sieci, jezeli $s \in S$ i $t \notin S$. Wielkość maksymalnego wypływu $w(S)$ przez przekrój S definiujemy wzorem

$$w(S) = \sum_{x \in S} \sum_{y \notin S \wedge (x,y) \in E} c(x, y).$$

Niech f będzie dowolnym przepływem przez rozważaną sieć. Zauważmy, że z definicji przepływu wynika następująca równość:

$$\sum_{x \in S \wedge x \neq s} \sum_{\substack{y \\ (x,y) \in E}} f(x,y) = \sum_{y \in S \wedge y \neq s} \sum_{\substack{x \\ (x,y) \in E}} f(x,y).$$

Ponieważ nie ma krawędzi prowadzących do s, dla $y = s$ mamy $\sum_{(x,y) \in E} f(x,y) = 0$ i sumowanie po prawej stronie równości można rozciągnąć na wszystkie $y \in S$. Mamy więc

$$\sum_{x \in S \wedge x \neq s} \sum_{\substack{y \\ (x,y) \in E}} f(x,y) = \sum_{y \in S} \sum_{\substack{x \\ (x,y) \in E}} f(x,y).$$

Do tej równości dodajemy stronami

$$\sum_{(s,y) \in E} f(s,y) = s(f).$$

W ten sposób otrzymujemy

$$\sum_{x \in S} \sum_{\substack{y \\ (x,y) \in E}} f(x,y) = \sum_{(s,y) \in E} f(s,y) + \sum_{x \in S \wedge x \neq s} \sum_{\substack{y \\ (x,y) \in E}} f(x,y) = s(f) + \sum_{y \in S} \sum_{\substack{x \\ (x,y) \in E}} f(x,y).$$

Dalej przekształcamy pierwszy i ostatni wzór. W wewnętrznych sumach tych wzorów grupujemy składniki wyznaczone przez elementy nienależace do S i pozostałe. W ten sposób otrzymujemy

$$\sum_{\substack{y \\ (x,y) \in E}} f(x,y) = \sum_{y \notin S \wedge (x,y) \in E} f(x,y) + \sum_{y \in S \wedge (x,y) \in E} f(x,y)$$

oraz

$$\sum_{\substack{x \\ (x,y) \in E}} f(x,y) = \sum_{x \notin S \wedge (x,y) \in E} f(x,y) + \sum_{x \in S \wedge (x,y) \in E} f(x,y).$$

Jeżeli zsumujemy drugie składniki tych sum, to otrzymamy równe liczby:

$$\sum_{x \in S} \sum_{y \in S \wedge (x,y) \in E} f(x,y) = \sum_{y \in S} \sum_{x \in S \wedge (x,y) \in E} f(x,y).$$

Równe są więc także sumy pozostałych składników:

$$\sum_{x \in S} \sum_{y \notin S \wedge (x,y) \in E} f(x,y) \quad = \quad s(f) + \sum_{y \in S} \sum_{x \notin S \wedge (x,y) \in E} f(x,y). \tag{1}$$

Dalej będziemy korzystać z wzoru (1). Po pierwsze zauważmy, że lewa strona tego wzoru jest niewiększa od wielkości maksymalnego wypływu z przekroju S, a druga strona jest niemniejsza od $s(f)$.

Wniosek 1.2 *Wielkość dowolnego przypływu w danej sieci nie przekracza wielkości maksymalnego wypływu przez dowolny przekrój tej sieci.*

Zauważmy, że w wyniku działania analizowanego algorytmu otrzymujemy przepływ f taki, że w grafie G_f nie istnieje ścieżka od s do t wzdłuż krawędzi, którym funkcja c_f przyporządkowuje liczby dodatnie. Niech V_f oznacza zbiór krawędzi w grafie G_f. Zdefiniujmy zbiór S następującym wzorem

$$S = \{v \in V : \exists x_0, \ldots, x_m \; x_0 = s \wedge x_n = v \wedge \forall i < m \; (x_i, x_{i+1}) \in V_f \wedge c_f(x_i, x_{i+1}) > 0\}.$$

Zbiór S jest oczywiście przekrojem sieci. Przypuśćmy, że mamy $x \in S$, $y \notin S$ takie, że $(x, y) \in E$. Gdyby $c_f(x, y) > 0$, to ścieżkę od s do x możnaby wydłużyć do y, i w konsekwencji y należałby do S. Mamy więc dla takich x i y

$$0 = c_f(x, y) = c(x, y) - f(x, y).$$

Przypuśćmy jeszcze, że mamy $y \in S$ i $x \notin S$ takie, że $(x, y) \in E$. W grafie G_f mamy także krawędź $(y, x) \in V_f$, a funkcja c_f przyporządkowuje jej liczbę $f(x, y)$. Założenie $c_f(y, x) > 0$ możemy sprowadzić do sprzeczności tak, jak w poprzednim przypadku. Dla rozważanych teraz y i x zachodzą więc równości $c_f(y, x) = f(x, y) = 0$. Z przeprowadzonych rozważań wynika, że we wzorze (1), po lewej stronie wyrażenie $f(x, y)$ możemy zastąpić przez $c(x, y)$, a wyrażenie $f(x, y)$ po prawej stronie możemy zastąpić przez 0. W ten sposób otrzymujemy, że

$$w(S) = \sum_{x \in S} \sum_{y \notin S \wedge (x,y) \in E} c(x, y) = s(f) + \sum_{y \in S} \sum_{x \notin S \wedge (x,y) \in E} 0 = s(f).$$

Oznacza to, że analizowany algorytm konstruuje przepływ, którego wielkość jest równa wielkości maksymalnego wypływu przez przekrój S. Wobec podanego wniosku, nie istnieje przepływ o większej wartości. Tak więc rozważany algorytm zwraca maksymalny przepływ. □

2 Formalizacja pojęcia algorytmu: maszyny Turinga

2.1 Definicja maszyny Turinga

Maszyną Turinga nazywamy układ $M = (K, \Sigma, \delta, s)$ taki, że

- K jest zbiorem skończonym i jest nazywany zbiorem stanów maszyny M; oprócz stanów ze zbioru K maszyna Turinga posługuje się trzema stanami końcowymi: *'yes'*, *'no'* i *'h'*,
- Σ jest skończonym zbiorem znaków i jest nazywamy alfabetem maszyny M; zakładamy, że do alfabetu dowolnej maszyny Turinga należą przynajmniej trzy znaki, w tym $\triangleright$ i $\sqcup$,
- δ jest (całkowitą) funkcją taką, że
$$\delta : K \times \Sigma \to (K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}) \times \Sigma \times \{\to, \leftarrow, -\}$$
taką, że dla dowolnego stanu $q \in K$ równość
$$\delta(q, \triangleright) = (q', \triangleright, \to)$$
zachodzi dla pewnego $q' \in K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}$ (symbol $\triangleright$ jest symbolem początku taśmy i nie może być przekraczany w lewo),
- $s \in K$ i jest nazywany stanem początkowym.

Konfiguracją maszyny Turinga M nazywamy trójkę (q, u, v) taką, że $q \in K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}$, $u, v \in \Sigma^*$ oraz u jest słowem zaczynającym się znakiem $\triangleright$. W zbiorze konfiguracji maszyny M definiujemy relację przejścia $\to$ jako najmniejszą relację, która dla wszystkich $q \in K$, $q' \in K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}$, $u, v \in \Sigma^*$ oraz $a, a', b \in \Sigma$ spełnia następujące warunki:

- jeżeli $\delta(q, a) = (q', a', -)$, to $(q, ua, v) \to (q', ua', v)$,
- jeżeli $\delta(q, a) = (q', a', \to)$, to $(q, ua, \varepsilon) \to (q', ua'\sqcup, \varepsilon)$,
- jeżeli $\delta(q, a) = (q', a', \to)$, to $(q, ua, bv) \to (q', ua'b, v)$,
- jeżeli $\delta(q, a) = (q', a', \leftarrow)$, to $(q, ua, v) \to (q', u, a'v)$.

Zauważmy, że symbol $\rightarrow$ jest używany w dwóch znaczeniach. Symbolem $\stackrel{*}{\rightarrow}$ będziemy oznaczać przechodnie domknięcie relacji $\rightarrow$.

Zdefiniowane maszyny Turinga będziemy nazywać jednonapisowymi (lub jednotaśmowymi). Będziemy też posługiwać się maszynami k-napisowymi (lub k-taśmowymi). Aby zdefiniować takie maszyny wystarczy rozszerzyć podanę definicję żądając, by

$$\delta : K \times \Sigma^k \rightarrow (K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}) \times (\Sigma \times \{\rightarrow, \leftarrow, -\})^k$$

a także, by funkcja δ spełniała warunek analogiczny do podanego w definicji (jednonapisowej) maszyny Turinga zawsze wtedy, gdy wśród argumentów tej funkcji znajdzie się przynajmniej jeden znak $\triangleright$. Konfigurację maszyny k-taśmowej definiujemy jako $(q, u_1, v_1, \ldots, u_k, v_k)$, gdzie $q \in K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}$, $u_i, v_i \in \Sigma^*$ oraz u_i zaczynają się znakiem $\triangleright$ dla $i = 1, \ldots, k$. Relację przejścia maszyny k-taśmowej definiujemy tak, aby warunki analogiczne do podanych w definicji relacji przejścia dla zwykłych maszyn zachodziły dla wszystkich $i = 1, \ldots, k$.

Będziemy też rozważać maszyny z wejściem i/lub wyjściem (z taśmą wejściową i/lub taśmą wyjściową). W takich przypadkach będziemy zakładać, że taśmą wejściową jest pierwsza, a taśmą wyjściową – ostatnia taśma. Maszyny z taśmą wejściową spełniają dodatkowo następujące warunki:

- jeżeli $\delta(q, a, \ldots) = (q', b, r, \ldots)$, to $a = b$ (a więc napis na taśmie wejściowej nie ulega zmianie),
- jeżeli $\delta(q, \sqcup, \ldots) = (q', \sqcup, r, \ldots)$, to $r = \leftarrow$ (znak $\sqcup$ oznacza koniec napisu na taśmie wejściowej i nie jest przekraczany przy ruchu w prawo)

dla dowolnych $a, b \in \Sigma$, $q \in K$, $q' \in k \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}$ i $r \in \{\rightarrow, \leftarrow, -\}$. Maszyny z taśmą wyjściową nie wykonują na taśmie wyjściowej ruchów w lewo, a więc

- jeżeli $\delta(q, \ldots, a) = (q', \ldots, b, r)$, to $r \neq \leftarrow$

dla dowolnych $a, b \in \Sigma$, $q \in K$, $q' \in K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}$ i $r \in \{\rightarrow, \leftarrow, -\}$.

Maszyny Turinga wykorzystujemy najczęściej do rozwiązywania problemów decyzyjnych lub do obliczania funkcji. Zwykle zakładamy (a więc możliwe są wyjątki), że maszyna rozwiązująca problem decyzyjny nie posługuje się stanem *'h'*, a maszyna obliczająca funkcję nie posługuje się stanami *'yes'* i *'no'*. Jeżeli $M = (K, \Sigma, \delta, s)$ jest maszyną Turinga, to dla $x \in \Sigma^*$ definiujemy 'wynik' pracy $M(x)$ maszyny M

przyjmując

$$M(x) = \begin{cases} \text{'yes'} & \text{jeżeli } \exists u_1, v_1, \ldots, u_k, v_k \in \Sigma^* \\ & \qquad (s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{*} (\text{'yes'}, u_1, v_1, \ldots, u_k, v_k) \\ \text{'no'} & \text{jeżeli } \exists u_1, v_1, \ldots, u_k, v_k \in \Sigma^* \\ & \qquad (s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{*} (\text{'no'}, u_1, v_1, \ldots, u_k, v_k) \\ y & \text{jeżeli } y \in \Sigma^* \text{ i nie kończy się znakiem } \sqcup \text{ oraz} \\ & \qquad \exists u_1, v_1, \ldots, u_k, v_k \in \Sigma^* \; \exists n \in N \; u_k v_k = \triangleright y \sqcup^n \wedge \\ & \qquad\quad (s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{*} (\text{'h'}, u_1, v_1, \ldots, u_k, v_k) \\ \nearrow & \text{w pozostałych przypadkach.} \end{cases}$$

Język

$$\{x \in (\Sigma \setminus \{\triangleright, \sqcup\})^* : M(x) = \text{'yes'}\}$$

nazywamy językiem akceptowanym (rozpoznawanym) przez maszynę M. Języki akceptowane przez maszyny Turinga nazywamy rekurencyjnie przeliczalnymi. Maszyna $M = (K, \Sigma, \delta, s)$ rozstrzyga język L, jeżeli dla dowolnego $x \in (\Sigma \setminus \{\triangleright, \sqcup\})^*$ warunek $x \in L$ implikuje $M(x) = $ *'yes'*, a warunek $x \notin L$ implikuje $M(x) = $ *'no'*. Języki rozstrzygane przez maszyny Turinga nazywamy rekurencyjnymi. Maszyna $M = (K, \Sigma, \delta, s)$ oblicza funkcję $f : (\Sigma \setminus \{\triangleright, \sqcup\})^* \to \Sigma^*$, jeżeli dziedziną funkcji f jest zbiór $\{x \in (\Sigma \setminus \{\triangleright, \sqcup\})^* : M(x) \neq \nearrow\}$ oraz $f(x) = M(x)$ dla wszystkich x z dziedziny funkcji f. Funkcje obliczane przez maszyny Turinga nazywamy rekurencyjnymi.

Maszyny Turinga, które zostały zdefiniowane, nazywamy deterministycznymi. Niedeterministyczną maszyną Turinga nazywamy układ $M = (K, \Sigma, \Delta, s)$ taki, że K, Σ i s mają własności podane w definicji (deterministycznej) maszyny Turinga, a ponadto

- Δ jest zbiorem takim, że

  $$\Delta \subseteq (K \times \Sigma) \times ((K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\})$$

  i jeżeli $((q, \triangleright), (q', a, r)) \in \Delta$, to $a = \triangleright$ oraz $r = \rightarrow$, dla dowolnych $q \in K$, $q' \in K \cup \{\text{'yes'}, \text{'no'}, \text{'h'}\}$, $a \in \Sigma$ oraz $r \in \{\rightarrow, \leftarrow, -\}$.

Będziemy rozważać także wielonapisowe maszyny niedeterministyczne. Wtedy Δ będzie zbiorem spełniającym zawieranie

$$\Delta \subseteq (K \times \Sigma^k) \times ((K \cup \{\text{'yes'}, \text{'no'}, h\}) \times (\Sigma \times \{\rightarrow, \leftarrow, -\})^k)$$

i warunki dotyczące znaków $\triangleright$ analogiczne do podanego.

Wiekszość pojęć dotyczących maszyn niedeterministycznych definiujemy tak, jak dla maszyn deterministycznych: wystarczy (w przypadku maszyn jednonapisowych) równości $\delta(q,a) = (q', a', r)$ zastąpić przez warunki $((q,a),(q',a',r)) \in \Delta$. W ten sposób można zdefiniować relację przejścia oraz maszynę z taśmą wejściową lub wyjściową. Dla maszyn niedeterministycznych nie definiujemy wyniku pracy $M(x)$. Aby otrzymać definicję języka akceptowanego przez maszynę niedeterministyczną wystarczy w definicji tego pojęcia w przypadku deterministycznym zastąpić warunek $M(x) =$ *'yes'* przez równoważny mu warunek z definicji $M(x)$. Funkcje będziemy obliczać za pomocą maszyn niedeterministycznych w wyjątkowych przypadkach.

2.2 Hierarchia czasowa

Niech M będzie maszyną Turinga i niech $\to$ oznacza realcję przejścia maszyny M. Symbolem $\stackrel{n}{\to}$ oznaczamy n-krotne złożenie relacji $\to$.

Niech $f : N \to N$. Mówimy, że maszyna $M = (K, \Sigma, \delta, s)$ działa w czasie f, jeżeli dla dowolnego $n \in N$ i dla dowolnego słowa x długości n nie istnieje konfiguracja $(q, u_1, v_1, \ldots, u_k, v_k)$ maszyny M taka, że

$$(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \stackrel{f(n)+1}{\to} (q, u_1, v_1, \ldots, u_k, v_k).$$

Fakt 2.1 *Niech $M = (K, \Sigma, \delta, s)$ będzie maszyną Turinga rozwiązującą problem decyzyjny i działającą w czasie f. Załóżmy, że $f(n) = 1$ dla pewnego $n \in N$. Wtedy język akceptowany przez maszynę M jest pusty, albo jest równy $(\Sigma \setminus \{\triangleright, \sqcup\})^*$.*

Dowód. Niech n będzie liczbą taką, że $f(n) = 1$ i weźmy dowolne słowo x długości n. Maszyna M po uruchomieniu w konfiguracji $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon)$ wykonuje pewien rozkaz (wynika to stąd, że $s \in K$). Działanie maszyny w tej konfiguracji zależy od wartości $\delta(s, \triangleright, \ldots, \triangleright)$. Z definicji maszyny Turinga mamy, że

$$\delta(s, \triangleright, \ldots, \triangleright) = (q, \triangleright, \to, \ldots, \triangleright, \to)$$

dla pewnego stanu q. Gdyby $q \in K$, to maszyna M także w konfiguracji ze stanem q wykonałaby jakiś rozkaz i działałaby w czasie dłuższym niż $f(n) = 1$. Wobec tego $q =$ *'yes'* lub $q =$ *'no'*. Oznacza to, że funkcja δ albo spełnia równość

$$\delta(s, \triangleright, \ldots, \triangleright) = (\text{'yes'}, \triangleright, \to, \ldots, \triangleright, \to),$$

albo

$$\delta(s, \triangleright, \ldots, \triangleright) = (\text{'no'}, \triangleright, \to, \ldots, \triangleright, \to).$$

W pierwszym przypadku maszyna M zawsze po wykonaniu pierwszego rozkazu znajdzie się w stanie akceptującym, a więc zaakceptuje wszystkie słowa. W drugim – zawsze znajdzie się w stanie odrzucającym, a więc nie zaakceptuje żadnego słowa. $\square$

Niech $f : N \to R$. Przyjmijmy, że $\mathit{TIME}(f)$ jest klasą języków akceptowanych przez (deterministyczne) maszyny Turinga działające w czasie f', gdzie $f'(n) = \lfloor f(n) \rfloor$. Podobnie definiujemy klasę $\mathit{NTIME}(f)$. Jest to klasa języków akceptowanych przez niedeterministyczne maszyny Turinga działające w czasie f' zdefiniowanym jak wyżej.

Fakt 2.2 *Jeżeli $f : N \to R$ i $f(n) < 1$ dla pewnego $n \in N$, to*

$$\mathit{TIME}(f) = \mathit{NTIME}(f) = \emptyset.$$

Jeżeli $f(n) \geq 1$ dla wszystkich $n \in N$ oraz $f(n) < 2$ dla pewnego $n \in N$, to

$$\mathit{TIME}(f) = \mathit{NTIME}(f) = \{\Sigma^* : \Sigma \text{ jest alfabetem}\} \cup \{\emptyset\}.$$

2.3 Palindromy

Palindromem nazywamy słowo $c_1 c_2 \ldots c_n$ takie, że

$$c_1 c_2 \ldots c_n = c_n c_{n-1} \ldots c_1,$$

gdzie $c_1, c_2, \ldots, c_n$ są znakami rozważanego alfabetu. Symbolem P będziemy oznaczać zbiór palindromów (nad pewnym alfabetem). Nietrudno wymyślić trzy algorytmy badające, czy dane słowo jest palindromem.

Rozważmy maszynę Turinga $M = (\{s, s_0, s_1, v_0, v_1, c\}, \{0, 1, \sqcup, \triangleright\}, \delta, s)$ z funkcją przejścia zdefiniowaną w następujący sposób:

$$\begin{array}{lll}
\delta(s,\triangleright) = (s,\triangleright,\to) & \delta(s_0,\triangleright) = (\text{'no'},\triangleright,\to) & \delta(s_1,\triangleright) = (\text{'no'},\triangleright,\to) \\
\delta(s,0) = (s_0,\triangleright,\to) & \delta(s_0,0) = (s_0,0,\to) & \delta(s_1,0) = (s_1,0,\to) \\
\delta(s,1) = (s_1,\triangleright,\to) & \delta(s_0,1) = (s_0,1,\to) & \delta(s_1,1) = (s_1,1,\to) \\
\delta(s,\sqcup) = (\text{'yes'},\sqcup,-) & \delta(s_0,\sqcup) = (v_0,\sqcup,\leftarrow) & \delta(s_1,\sqcup) = (v_1,\sqcup,\leftarrow) \\
 & & \\
\delta(v_0,\triangleright) = (\text{'yes'},\triangleright,\to) & \delta(v_1,\triangleright) = (\text{'yes'},\triangleright,\to) & \delta(c,\triangleright) = (s,\triangleright,\to) \\
\delta(v_0,0) = (c,\sqcup,\leftarrow) & \delta(v_1,0) = (\text{'no'},0,-) & \delta(c,0) = (c,0,\leftarrow) \\
\delta(v_0,1) = (\text{'no'},1,-) & \delta(v_1,1) = (c,\sqcup,\leftarrow) & \delta(c,1) = (c,1,\leftarrow) \\
\delta(v_0,\sqcup) = (\text{'no'},\sqcup,-) & \delta(v_1,\sqcup) = (\text{'no'},\sqcup,-) & \delta(c,\sqcup) = (\text{'no'},\sqcup,-)
\end{array}$$

Jeżeli maszyna M w stanie s obserwuje 0 lub 1, to zapamiętuje obserwowany znak, zastępuje go symbolem $\triangleright$, znajduje ostatni znak analizowanego słowa, porównuje go z zapamiętanym. Jeżeli oba znaki są identyczne, to wymazuje ostatni znak, cofa się na początek danego słowa i ponownie przechodzi do stanu s. Czynności te powtarza tak długo, aż stwierdzi różne znaki na odpowiadających pozycjach, lub aż wymaże całe słowo.

Nietrudno sprawdzić, że jeżeli maszyna M znajduje się w stanie s, obserwuje znak różny od $\triangleright$ i poprzedzony znakiem początku taśmy, jeżeli po tym znaku i przed pierwszym znakiem $\sqcup$ znajduje m znaków 0 i 1, to po raz kolejny znajdzie się stanie s po wykonaniu $2m+1$ ruchów. To spostrzeżenie pozwala przekonać się, że maszyna M uruchomiona ze słowem długości n wykona najwyżej

$$\frac{n^2 + 3 \cdot n + 4}{2}$$

ruchów (uruchomiona z palindromem wykonuje dokładnie tyle ruchów). Jest oczywiste, że wtedy utworzy słowo o długości $n+2$.

Kolejny algorytm wymaga maszyny dwutaśmowej. Najpierw przepisuje dane na drugą taśmę, cofa jedną głowicę na początek taśmy i porównuje napisy na obu taśmach czytając je w przeciwnych kierunkach. Maszyna realizująca taki algorytm, po uruchomieniu ze słowem długości n wykonuje $3 \cdot n + 5$ ruchów i tworzy dwa napisy długości $n+2$.

Ostatni algorytm da się zaimplementować na maszynach o trzech taśmach. Na pierwszej taśmie będzie przechowywane dane słowo, na drugiej – przedstawienia binarne pewnych liczb naturalnych, początkowo przedstawienie liczby 1. Jeżeli na drugiej taśmie będzie zapisana liczba k, to maszyna realizująca ten algorytm znajdzie i zapamięta (przez przejście do odpowiedniego stanu) k-tą literę danego słowa, a następnie znajdzie k-tą literę, ale tym razem licząc od końca. Jeżeli znalezione litery będą różne, maszyna zatrzyma się w stanie odrzucającym. W przeciwnym razie zwiększy o 1 liczbę na drugiej taśmie. Postępowanie to będzie kontynuowane tak długo, aż na drugiej taśmie znajdzie się przedstawienie liczby o długości większej od długości danego słowa. Trzecia taśma będzie wykorzystywana do zliczania ruchów podczas szukania k-tego znaku. Trudno ustalić liczbę ruchów takiej maszyny. Powinna być rzędu $O(n^2 \cdot \log n)$. Łatwo ustalić długość napisów tworzonych podczas pracy. Na pierwszej taśmie długość napisu nie powinna przekracać $n+2$, a na pozostałych – $4 + \lfloor \log n \rfloor$ ($1 + \lfloor \log n \rfloor$ to długość przedstawienia liczby n, na drugiej taśmie może się pojawić przedstawienie liczby $n+1$ i może być o 1 dłuższe od przedstawienia n, dodatkowo na taśmie są zapisane dwa znaki).

Twierdzenie 2.3 *Jeżeli jednotaśmowa maszyna Turinga wykonuje po uruchomieniu ze słowem długości n nie więcej niż $f(n)$ ruchów i akceptuje zbiór palindromów $P \subseteq \{0,1\}^*$, to*

$$\limsup_{n\to\infty} \frac{f(n)}{n^2} > 0,$$

a więc dla pewnej stałej $c > 0$ i dla nieskończenie wielu liczb naturalnych n zachodzi nierówność $f(n) > c \cdot n^2$.

Dowód. Niech M będzie jednotaśmową maszyną Turinga akceptującą zbiór P, o złożoności czasowej f. Dodatkowo o M możemy założyć, że w każdym ruchu przesuwa głowicę w lewo lub w prawo. Można spełnić to założenie zwiększając dwukrotnie czas pracy maszyny. Nietrudno zauważyć, że jeżeli teza jest spełniona dla funkcji $2 \cdot f$, to jest także spełniona dla f.

Ustalmy liczbę $n \in N$. Będziemy rozważać palindromy długości $2 \cdot n$. Najpierw jednak zdefiniujemy ciągi przejść.

Niech w będzie słowem długości n. Weźmy ciąg konfiguracji utworzony przez maszynę M uruchomioną w konfiguracji początkowej ze słowem ww^R. Analizując ten ciąg konfiguracji zdefiniujemy ciągi $\alpha_w(i)$ stanów maszyny M. Zrobimy to podając algorytm pozwalający obliczyć interesujące nas ciągi. Ciągi te tworzymy przeglądając kolejno konfiguracje w rozważanym ciągu. Dla każdej konfiguracji jeden z ciągów $\alpha_w(i)$ wydłużamy przez dopisanie na końcu jednego stanu. Początkowo wszystkie ciągi $\alpha_w(i)$ są puste. Później postępujemy zgodnie z następującymi zasadami:

- jeżeli M w konfiguracji (q, u, w) wykonuje ruch w prawo i przechodzi do stanu p, to dopisujemy p do ciągu $\alpha_w(|u| - 1)$,
- jeżeli M w konfiguracji (q, u, w) wykonuje ruch w lewo i przechodzi do stanu p, to dopisujemy p do ciągu $\alpha_w(|u| - 2)$.

Niech $s_i(w)$ oznacza słowo w pozbawione i pierwszych liter ($i = 1, \ldots, n$). Potrzebną nam własność ciągów przejść wyraża następujący fakt:

Fakt 2.3.1 *Dla dowolnego ciągu α stanów maszyny M, funkcja s_i jest różnowartościowa na zbiorze $\{w \in \{0,1\}^n : \alpha_i(w) = \alpha\}$, a więc zbiór ten ma nie więcej niż 2^{n-i} elementów.*

Dowód. Przypuśćmy, że mamy dwa (różne) słowa w_1, w_2 długości n takie, że $\alpha_i(w_1) = \alpha = \alpha_i(w_2)$ oraz $s_i(w_1) = s_i(w_2)$. Niech v oznacza prefiks długości i

słowa w_1. Będziemy porównywać pracę maszyny M po uruchomieniu jej ze słowami $w_1w_1^R$, $w_2w_2^R$ oraz $vs_i(w_2)w_2^R$. Słowa w_1 i w_2 różnią się jedną z i pierwszych liter. Wobec tego $vs_i(w_2)w_2^R$ nie jest palindromem. Uzyskamy sprzeczność pokazując, że M akceptuje $vs_i(w_2)w_2^R$.

Pracę maszyny z tymi danymi dzielimy na etapy, w których maszyna znajduje się w konfiguracjach (q, u, w) takich, że $|u| \leq i + 1$ lub w konfiguracjach takich, że $|u| > i + 1$.

W pierwszym etapie maszyna obserwuje jedną z $i+1$ początkowych liter swojego napisu (w konfiguracji początkowej złożonego z symbolu początku taśmy oraz i liter danego słowa). Etap ten trwa krócej lub dłużej, zależnie od danych, ale dla wszystkich rozważanych danych zakończy się w tym samym stanie. W przypadku danych $w_1w_1^R$ i $w_2w_2^R$ wynika to z założenia o identyczności ciągów przejść. Dla słowa $vs_i(w_2)w_2^R$ jest to konsekwencją tego, że praca maszyny M w tym etapie zależy tylko od pierwszych $i+1$ liter napisu, a więc fragmentu, ktory jest identyczny dla słów $vs_i(w_2)w_2^R$ i $w_1w_1^R$. Dla obu podanych słów maszyna M będzie w pierwszym etapie pracować dokładnie w ten sam sposób, tak samo zmieni początkowy fragment napisu, będzie znajdować się kolejno w tych samych stanach, a także po zakończeniu etapu znajdzie się w tym samym stanie.

Podobnie będzie w drugim etapie. We wszytkich przypadkach etap ten zacznie się w tym samym stanie i po jego zakończeniu maszyna M znajdzie się w tym samym stanie z analogicznych powodów. Tym razem jednak maszyna M z daną $vs_i(w_2)w_2^R$ będzie pracować tak samo jak z daną $w_2w_2^R$, gdyż praca M będzie w tym etapie zależeć od końcowego fragmentu danych, który w obu przypadkach jest taki sam. Ponadto, podczas pracy z daną $vs_i(w_2)w_2^R$ maszyna M będzie przechodzić przez te same stany, co z daną $w_2w_2^R$.

Zauważmy jeszcze, że trzeci etap zależy od początkowego fragmentu napisu, fragmentu utworzonego w pierwszym etapie. Dla danych $vs_i(w_2)w_2^R$ i $w_1w_1^R$ ten fragment jest identyczny. Powtórzy się więc sytuacja z pierwszego etapu. Z przeprowadzonych rozważań wynika między innymi, że maszyna M uruchomiona z daną $vs_i(w_2)w_2^R$ podczas pracy znajduje się w stanach, w ktorych znajduje się także po uruchonieniu z danymi $w_1w_1^R$ lub $w_2w_2^R$. W szczególności, jeżeli zakończy pracę, to znajdzie się w stanie końcowym, w którym także znalazła się po uruchomieniu z danymi $w_1w_1^R$ lub $w_2w_2^R$, a więc w stanie *'yes'*. $\diamond$

Teraz będziemy szacować liczbę ruchów maszyny M, które zależą od tego fragmentu napisu, który początkowo zawiera pierwszą połowę danego słowa. Zauważmy, że z definicji ciągów $\alpha_i(w)$ wynika natychmiast, że

$$\sum_{i=1}^{n} |\alpha_i(w)| \leq \sum_{i\in N} |\alpha_i(w)| = f(2\cdot n).$$

Wprowadźmy następujące oznaczenie:

$$p(i) = \frac{\sum\limits_{|w|=n} |\alpha_i(w)|}{2^n}.$$

Tak więc $p(i)$ oznacza średnią długość ciągu $\alpha_i(w)$ dla słów w długości n.

Fakt 2.3.2 *Niech $(x_0, x_1, \ldots, x_{m-1})$ będzie dowolnym ciągiem liczb nieujemnych, a s – średnią wartością jego elementów. Wtedy przynajmniej połowa wyrazów tego ciągu ma wartości mniejsze od $2\cdot s$.*

Dowód. Niech $X = \{i < m : x_i < 2\cdot s\}$. Załóżmy, że $\#X < \frac{m}{2}$, a więc że $\#\{i < m : i \notin X\} > \frac{m}{2}$. Wtedy

$$s = \frac{\sum\limits_{i=0}^{m-1} x_i}{m} = \frac{\sum\limits_{i\in X} x_i + \sum\limits_{i\notin X} x_i}{m} \geq \frac{\sum\limits_{i\notin X} x_i}{m} \geq \frac{\sum\limits_{i\notin X} 2\cdot s}{m} > \frac{2\cdot s\cdot \frac{m}{2}}{m} = s.$$

Ta sprzeczność świadczy o prawdziwości tezy. $\diamond$

Niech k oznacza liczbę stanów maszyny M. Jest oczywiste, że można utworzyć mniej niż $k^{2\cdot p(i)}$ ciągów stanów długości mniejszej niż $2\cdot p(i)$. Zauważmy, że

$$\frac{2^n}{2} \leq \#\{w \in \{0,1\}^n : |\alpha_w(i)| < 2\cdot p(i)\} = \# \bigcup_{|\alpha|<2\cdot p(i)} \{w \in \{0,1\}^n : \alpha_w(i) = \alpha\} \leq$$

$$\sum_{|\alpha|<2\cdot p(i)} \#\{w \in \{0,1\}^n : \alpha_w(i) = \alpha\} \leq k^{2\cdot p(i)} \cdot 2^{n-i}.$$

Po uproszczeniu otrzymujemy stąd nierówność

$$2^{i-1} \leq k^{2\cdot p(i)},$$

z której – po przekształceniu – wynika, że

$$\frac{i-1}{2\cdot \log k} \leq p(i).$$

Aby zakończyć dowód zauważmy, że

$$\frac{n^2-n}{4\cdot\log k} = \sum_{i=1}^{n} \frac{i-1}{2\cdot\log k} \leq \frac{\sum\limits_{i=1}^{n} 2^n \cdot p(i)}{2^n} = \frac{\sum\limits_{i=1}^{n} \sum\limits_{w\in\{0,1\}^n} |\alpha_w(i)|}{2^n} = \frac{\sum\limits_{w\in\{0,1\}^n} \sum\limits_{i=1}^{n} |\alpha_w(i)|}{2^n} \leq$$

$$\frac{\sum\limits_{w\in\{0,1\}^n} f(2\cdot n)}{2^n} = f(2\cdot n). \ \Box$$

2.4 Twierdzenie o redukcji liczby taśm

Twierdzenie 2.4 (o redukcji liczby taśm) *Jeżeli język L jest akceptowany przez wielonapisową maszynę Turinga (deterministyczną lub niedeterministyczną) w czasie f, to L jest akceptowany przez jednonapisowaą maszynę Turinga (odpowiednio deterministyczną lub niedeterministyczną), działającą – dla pewnych stałych c_1, c_2 i c_3 – w czasie $c_1 \cdot f^2(n) + c_2 \cdot n + c_3$.*

Dowód. Niech $M = (K, \Sigma, \delta, s_M)$ będzie wielonapisową maszyną akceptującą L. Będziemy konstruować maszynę M_s symulującą maszynę M i posługującą się alfabetem $\Sigma \cup \{\bar{a} : a \in \Sigma\} \cup \{\triangleright', \triangleleft, \triangleleft'\}$. Maszyna M_s będzie symulować działanie M przy założeniu, że konfiguracja

$$(q, \triangleright u_1 a_1, v_1, \ldots, \triangleright u_k a_k, v_k)$$

$(a_i \in \Sigma,\ u_i, v_i \in \Sigma^*)$ maszyny M jest pamiętana przez M_s jako

$$(q, \triangleright, \triangleright' u_1 \bar{a}_1 v_1 \triangleleft \ldots \triangleright' u_k \bar{a}_k v_k \triangleleft \triangleleft).$$

Maszyna M_s najpierw przechodzi z konfiguracji początkowej $(s, \triangleright, x)$ do konfiguracji $(s_M, \triangleright, \bar{\triangleright'} x \triangleleft (\bar{\triangleright'} \triangleleft)^{k-1} \triangleleft$. Następnie symuluje pojedyncze ruchy maszyny M. Aby to zrobić, najpierw czyta napis i szuka w nim liter postaci $\bar{a}$. Po znalezieniu wszystkich takich liter zna litery obserwowane przez maszynę M w odpowiadającej konfiguracji i może rozpocząć symulowanie ruchu maszyny M. Robi to przeglądając ponownie cały napis i wprowadzając odpowiednie zmiany. Jeżeli konieczne jest wydłużenie fragmentu pamiętającego zapis pewnej taśmy maszyny M, to M_s zapamiętuje ten fakt zamieniając w odpowiednim miejscu $\triangleleft$ na $\triangleleft'$, a następnie (jednorazowo) wydłuża napis na swojej taśmie. Nietrudno zauważyć, że symulowanie pojedynczego ruchu wymaga czasu proporcjonalnego do długości napisu na taśmie maszyny symulującej.

Aby oszacować czas pracy M_s zauważamy, że przekształcenie konfiguracji początkowej wymaga czasu proporcjonalnego do długości danych, napisy pojawiające się na taśmie maszyny symukującej mają długość rzędu sumy długości napisów na taśmach maszyny M, a więc mają długość rzędu $\mathcal{O}(f)$, w konsekwencji symulacja pojedynczego ruchu wymaga czasu $\mathcal{O}(f)$, a symylacja całej pracy – czasu podanego w tezie twierdzenia. □

Twierdzenie 2.5 *Jeżeli język L jest akceptowany przez wielonapisową maszynę Turinga (deterministyczną lub niedeterministyczną) w czasie f $(f(n) \geq 2)$, to L jest akceptowany przez dwunapisowaą maszynę Turinga (odpowiednio deterministyczną lub niedeterministyczną), działającą – dla pewnych stałych c_1, c_2 i c_3 – w czasie $c_1 \cdot f(n) \cdot \log f(n) + c_2 \cdot n + c_3$.*

Dowód. W dowodzie jest wykorzystywana maszyna przesuwajaca zapis taśmy.
Idea maszyny przesuwającej zapis taśmy.

Symulacja maszyny wielotaśmowej na maszynie jednotaśmowej zwykle wymaga przeszukiwania całej taśmy po to, by ustalić znaki obserwowane przez poszczególne głowice maszyny symulowanej. Dzięki opisywanej konstrukcji, przeszukiwanie taśmy może zostać zastąpione przez odpowiednie przesuwanie zapisu tak, aby znaki obserwowane przez głowice maszyny symulowanej były zapamiętane w pojedyńczej komórce maszyny symulującej. Będziemy tym razem zakładać, że maszyny Turinga mają obustronie nieskończone taśmy. Ideę algorytmu przesuwającego zapis taśmy przedstawimy przy założeniu, że przesuwamy zapis pojedyńczej taśmy. Wymaga to konstruowania maszyny dwutaśmowej: druga taśma będzie odgrywać pomocniczą rolę i umożliwi szybkie wykonywanie pewnych operacji (będziemy więc symulować maszynę jednotaśmową na dwutaśmowej).

Przypuśćmy, że symulujemy maszynę M posługującą się alfabetem Σ. Maszyna symulująca M_s, oprócz znaków alfabetu Σ, będzie wykorzystywać znaki ze zbioru $(\Sigma \cup \{\sqcup, \triangle, \nabla\})^2$. Będziemy zakładać, że napis na pierwszej taśmie konstruowanej maszyny zawsze jest spójny (zapisane komórki nie są rozdzielone pustymi) i zawiera znaki podanego alfabetu. Przy tym założeniu zapis (pierwszej) taśmy maszyny M_s możemy uważać za słowo x nad alfabetem $(\Sigma \cup \{\sqcup, \triangle, \nabla\})^2$. Jeżeli słowo x potraktujemy jako konkatenację słów dwuliterowych, to możemy uważać je za słowo y nad alfabetem $\Sigma \cup \{\sqcup, \triangle, \nabla\}$. Usuńmy ze słowa y wszystkie znaki $\triangle$ i ∇. Otrzymane w ten sposób słowo uważamy za zapis taśmy maszyny M zakodowany na pierwszej taśmie maszyny M_s (znak $\sqcup$ odpowiada pustym komórkom).

Ustalmy numerację komórek taśmy maszyny M_s. Ponieważ taśma ta jest obustronnie nieskończona, jej komórki numerujemy liczbami całkowitymi. Oczywiście zakładamy, że sąsiednie komórki mają numery różniące się o jeden i numery komórek leżących na prawo od dowolnej komórki k są większe od numeru komórki k. Będziemy zakładać, że w komórce o numerze 0 zawsze znajduje się znak ze zbioru $\nabla(\Sigma \cup \{\sqcup\})$. Umawiamy się, że znak ∇ (jako część litery) może wystąpić tylko w komórce o numerze 0 (dzięki temu możemy znaleźć komórkę o numerze 0). Podana umowa pozwala też określić położenie głowicy maszyny symulowanej: głowica tej maszyny obserwuje komórkę, która zawiera drugą część znaku zapisanego w komórce o numerze 0.

Taśmę maszyny M_s dzielimy na bloki B_i indeksowane liczbami całkowitymi. Blok B_0 składa się tylko z komórki o numerze 0. Dla $i > 0$ blok B_i składa się z komórek o numerach j takich, że $2^{i-1} \leq j < 2^i$. Dla $i < 0$ blok B_i składa się z komórek o numerach j takich, że $-2^{-i} < j < 2^{-i+1}$. Tak więc bloki B_i i B_{-i} $(i > 0)$ są położone symetrycznie względem komórki o numerze 0 i składają się z 2^{i-1} komórek.

Maszyna M_s będzie posługiwała się blokami czterech rodzajów: blokami pustymi, zawierającymi wyłącznie puste komórki, blokami niewykorzystywanymi, we wszystkich komórkach tych bloków będzie zapisana litera $\triangle\triangle$, blokami pełnymi, z komórkami zapisanymi literami ze zbioru $(\Sigma \cup \{\sqcup\})^2$, oraz blokami częściowo zapełnionymi, z komórkami zapisanymi znakami ze zbioru $\triangle(\Sigma \cup \{\sqcup\})$. Łatwo ustalić rodzaj bloku badając znak zapisany w dowolnej jego komórce. Bloki poszczególnych rodzajów będą rozmieszczane na taśmie tak, aby dla dowolnego $i \neq 0$ były spełnione następujące warunki:

- B_i jest pusty wtedy i tylko wtedy, gdy B_{-i} jest pusty,
- B_i jest częściowo zapełniony wtedy i tylko wtedy, gdy B_{-i} jest częściowo zapełniony,
- B_i jest pełny wtedy i tylko wtedy, gdy B_{-i} jest niewykorzystywany.

Przypuśćmy, że symulujemy wykonanie ruchu w lewo, a więc przesuwamy (w pewnym sensie) zapis taśmy maszyny symulującej w prawo. Niech $i > 0$ będzie najmniejszą liczbą taką, że blok B_i nie jest pełny. Takie przesuwanie będziemy nazywać przesuwaniem rzędu i. Załóżmy, że głowica obserwuje komórkę o numerze 0.

Najpierw na drugiej taśmie odtwarzamy fragment zapisu taśmy maszyny symulowanej zakodowany w blokach $B_0, \ldots, B_{i-1}$. Fragment ten ma długość $2^i - 1$. Jeżeli blok B_i jest częściowo zapełniony, to odtwarzamy także fragment (złożony z 2^{i-1} znaków) zakodowany w tym bloku (długość i-tego bloku możemy ustalić zapoznając się z łączną długością bloków $B_0, \ldots, B_{i-1}$). Odkodowany fragment zapisu taśmy maszyny M ponownie kodujemy zapisując go w blokach $B_1, \ldots, B_i$. W każdej komórce bloków

$B_1, \ldots, B_{i-1}$ umieszczamy po jednej literze kodowanego fragmentu (wraz ze znakiem $\triangle$). W ten sposób zapiszemy 2^{i-1} liter. Pozostałe litery umieszczamy w bloku B_i. Jeżeli blok ten był częściowo zapełniony, to w każdej jego komórce zapisujemy po dwie litery (łączenie 2^i liter). W przeciwnym razie w każdej jego komórce umieszczamy jedna literę wraz ze znakiem $\triangle$ (więc umieszczamy tam 2^{i-1} liter). Następnie odkodowujemy fragment zapisu taśmy maszyny M umieszczony w bloku B_{-i}. Znając typ bloku B_i, wiemy ile liter jest zapisanych w bloku B_{-i}. Szukając tego bloku przygotowujemy odpowiedniej wielkości fragment drugiej taśmy i wypełniamy go odczytanym zapisem. Zależnie od typu bloku B_{-i} w tym fragmencie mamy 2^i lub 2^{i-1} liter. Jeżeli mamy 2^i liter, to umieszczamy je po jednej we wszystkich komórkach bloków $B_{-i}, \ldots, B_0$. Jeżeli mamy rozmieścić 2^{i-1} liter, to blok B_{-i} pozostawiamy niewykorzystany (wypełniamy go znakami $\triangle\triangle$), a wszystkie litery umieszczamy w blokach $B_{-i+1}, \ldots, B_0$.

Ruch w prawo symulujemy w analogiczny sposób.

Szacowanie liczby ruchów maszyny M_s jest możliwe dzięki następującym spostrzeżeniom. Tuż przed wykonaniem przesunięcia rzędu i ($i > 0$) w blokach $B_0, \ldots, B_{i-1}$ znajduje się (jest zakodowane) $2^i - 1$ alfabetu $\Sigma \cup \{\sqcup\}$. Tuż po wykonaniu tego przesunięcia w tych blokach jest 2^{i-1} liter wspomnianego alfabetu. Przed wykonaniem kolejnego przesunięcia rzędu $j \geq i$ bloki $B_1, \ldots, B_{i-1}$ powinny zostać wypełnione i może to się stać w wyniku wykonania pewnej liczby przesunięć rzędu $< i$. W wyniku wykonania przesunięcia rzędu $k > 0$ liczba liter pamiętach w blokach $B_0, \ldots, B_k$ zwiększa się o 1, a w blokach o większych numerach – nie zmienia się. Natomiast przesunięcia rzędu ujemnego albo zmniejszają liczbę liter pamiętanych w blokach $B_0, \ldots, B_{i-1}$ o jeden, albo powodują, że liczba ta staje się równa 2^{i-1}. Przed wykonaniem kolejnego przesunięcia rzędu i musi więc zostać wykonanych przynajmniej $2^{i-1} - 1$ przesunięć rzędu mniejszego od i. $\Box$

2.5 Twierdzenie o przyśpieszaniu

Twierdzenie 2.6 (o przyśpieszaniu) *Niech $\varepsilon > 0$. Jeżeli $L \in TIME(f)$, to $L \in TIME(f')$ dla funkcji f' zdefiniowanej wzorem $f'(n) = \varepsilon \cdot f(n) + 2 \cdot n + 9$ (stałe zostały ustalone dla symulacji m ruchów za pomocą siedmiu).*

Dowód. Bierzemy maszynę M akceptującą język L w czasie f i konstruujemy maszynę M_m symulującą pracę maszyny M. Maszyna M_m nie symuluje wykonania poszczególnych ruchów M, a wykonanie fragmentów pracy maszyny M złożonych z

m ruchów. Jeżeli m zostanie odpowiednio dobrane, to maszyna M_m będzie działała w czasie podanym w tezie twierdzenia.

Maszyna M_m na podstawie własnej konfiguracji powinna odtworzyć odpowiednią konfigurację maszyny M. Zapis taśmy maszyny M będzie pamiętany na taśmie maszyny M_m za pomocą alfabetu pozwalającego zapisać w pojedyńczej komórce ciąg m znaków alfabetu maszyny M. Położenie głowicy maszyny M będzie pamiętane przez położenie głowicy maszyny M_m, oraz przez dodatkową informację pamiętaną w stanie maszyny M_m, pozwalającą wybrać obserwowany znak spośród m znaków obserwowanych przez głowicę maszyny M_m. Stan maszyny M będzie można ustalić znając stan maszyny M_m.

Maszyna M_m najpierw przekształca dane do odpowiedniej postaci, a następnie przystępuje do symulacji pracy maszyny M. Aby symulować m ruchów, musi zapoznać się z dostatecznie dużym fragmentem zapisu na taśmie maszyny M. W tym celu zapoznaje się zawartością obserwowanej komórki i dwóch sąsiednich. Nietrudno zauważyć, że maszyna M podczas wykonywania m ruchów zmieni najwyżej zawartość m komórek i komórki te mogą znajdować się lewej lub po prawej stronie komórki obserwowanej w sytuacji początkowej. Po zapoznaniu się z zawartością tych trzech komórek odpowiednio zmienia ich zawartość i – w razie potrzeby – ustawia głowicę. Najprostszy sposób realizacji tych czynności wymaga od maszyny M_m wykonania 7 ruchów. W ten sposób maszyna M_m symuluje wykonanie m ruchów maszyny M za pomocą siedmiu własnych.

Można skomplikować program maszyny M_m tak, aby symulowała m ruchów maszyny M za pomocą 3. Znając zawartość obserowowanej komórki programista piszący program maszyny M_m może przewidzieć, czy maszyna M opuści fragment taśmy zakodowany w tej komórce i w którą stronę. Mają te informacje wystarczy zapoznać się z zawartością komórki obserwowanej i jednej z sąsiednich, zmienić zawartość tych dwóch komorek i przesunąć głowice nad odpowiednią komórkę. Te czynności wymagają trzech ruchów. $\Box$

Lemat 2.7 *Niech m będzie ustaloną liczbą naturalną. Przypuśćmy, że mamy daną maszynę M działającą w czasie f i język L takie, że $x \in L$ wtedy i tylko wtedy, gdy $M(x) =$ 'yes' dla dowolnego słowa x długosci $> m$ nad alfabetem maszyny M. Wtedy $L \in TIME(f')$ dla funkcji f' zdefiniowanej wzorem*

$$f'(n) = \begin{cases} n+2 & \textit{jeżeli } n \le m \\ f(n) + 2 \cdot m + 2 & \textit{dla pozostałych } n. \end{cases} \Box$$

Wniosek 2.8 *Niech f będzie funkcją taką, że $f(n) \geq n+2$ dla dowolnego n oraz*

$$\lim_{n\to\infty} \frac{f(n)}{n} = \infty.$$

Wtedy TIME($c\cdot f$) $\subseteq$ TIME(f) dla dowolnego $c \geq 1$, i w konsekwencji TIME($c\cdot f$) $=$ TIME(f). □

Wniosek 2.9 *Niech f będzie funkcją taką, że $f(n) \geq n+2$ dla dowolnego n oraz*

$$\lim_{n\to\infty} \frac{f(n)}{n} = \infty.$$

Jeżeli różnica symetryczna $L_1 \dot{-} L_2$ jest skończona i $L_1 \in$ TIME(f), to $L_2 \in$ TIME(f). □

Wniosek 2.10 *Niech f_2 będzie funkcją taką, że $f_2(n) \geq n+2$ dla dowolnego n oraz*

$$\lim_{n\to\infty} \frac{f_2(n)}{n} = \infty.$$

Jeżeli $f_1(n) \leq f_2(n)$ dla prawie wszystkich liczb naturalnych n i $L \in$ TIME(f_1), to $L \in$ TIME(f_2). □

2.6 Uniwersalna maszyna Turinga

Przypuśćmy, że mamy (jednotaśmową) maszynę Turinga $M = (K, \Sigma, \delta, s)$. Ponumerujmy

- elementy zbioru Σ liczbami od 1 do $|\Sigma|$,
- elementy zbioru K liczbami od $|\Sigma|+1$ do $|\Sigma|+|K|$; zakładamy, że stan s w tej numeracji ma numer $|\Sigma|+1$,
- stany h, *'yes'*, *'no'* oraz znaki $\leftarrow$, $\rightarrow$, $-$ odpowiednimi liczbami z przedziału od $|\Sigma|+|K|+1$ do $|\Sigma|+|K|+6$.

Będziemy utożsamiać stany maszyny M i wymienione znaki z ich numerami. Przyjmujemy zasadę, że numery znaków i stanów maszyny M bedziemy zapisywać w systemie binarnym używając zawsze tyle samo znaków i tak, aby ich liczba była możliwie mała (a więc powinna być równa $\lfloor \log(|K|+|\Sigma|+6) \rfloor + 1$).

Niech $\Sigma_U = \{0, 1, (,), \mathit{przecinek}, ;, \sqcup, \triangleright\}$. Opisem maszyny M nazywamy słowo $\overline{M} \in \Sigma_U^*$ będące konkatenacją przedstawienia binarnego liczby $|K|$, przedstawienia binarnego liczby $|\Sigma|$, opisu funkcji przejścia δ i dwóch przecinków oddzielających wymienione fragmenty. Po utożsamieniu stanów i znaków z przedstawieniami binarnymi ich numerów (o podanej długości), opis funkcji przejścia jest konkatenacją słów $((q, \sigma), (p, \rho, D)) \in \{0, 1, (,), \mathit{przecinek}\}^*$ takich, że $\delta(q, \sigma) = (p, \rho, D)$, wraz z przecinkami oddzielającymi te słowa.

Słowo $x \in \Sigma^*$ utożsamiamy ze słowem $\widetilde{x} \in \{0, 1\}^*$ będącym konkatenacją przedstawień binarnych numerów kolejnych znaków danego słowa x zapisanych przy użyciu określonej liczby znaków.

Lemat 2.11 *Dane są maszyny M_1 i M_2 oraz słowa x_1 i x_2 zapisane przy użyciu liter z alfabetów odpowiednich maszyn. Dla każdej z tych maszyn wybieramy numerację alfabetu, zbioru stanów i kolejność informacji o funkcji przejścia, a więc wszystko to, co wpływa na sposób kodowania maszyn i danych dla tych maszyn. Jeżeli wtedy $\overline{M_1}; \widetilde{x_1} = \overline{M_2}; \widetilde{x_2}$, to $M_1(x_1) = M_2(x_2)$.*

Dowód. Jeżeli $\overline{M_1}; \widetilde{x_1} = \overline{M_2}; \widetilde{x_2}$, to także $\overline{M_1} = \overline{M_2}$ i $\widetilde{x_1} = \widetilde{x_2}$. Jeżeli $\overline{M_1} = \overline{M_2}$, to przedstawienia liczebności zbioru stanów i alfabetów maszyn M_1 i M_2 są identyczne, a więc te maszyny posługują się równolicznymi alfabetami i mają tyle samo stanów. Ponadto w obu przypadkach długości kodów oraz kody stanów początkowych są identyczne. Równe są także opisy funkcji przejścia. Równość funkcji przejścia implikuje, że że jeżeli maszyny M_1 i M_2 znajdują się w stanach i widzą litery o identycznych kodach, to zapisują litery o identycznych kodach, przechodzą do tak samo kodowanych stanów i wykonują te same ruchy. Ponieważ obie maszyny zostały uruchomione z analogicznymi danymi, po wykonaniu tej samej liczby ruchów znajdują się w stanach o tych samych kodach. Więc obie maszyny albo jednocześnie akceptują dane, albo jednocześnie je odrzucają. $\Box$

Zdefiniujemy wielotaśmową maszynę Turinga U posługującą się alfabetem Σ_U, która dla dowolnej jednotaśmowej maszyny Turinga $M = (K, \Sigma, \delta, s)$ rozwiązującej problem decyzyjny i dla dowolnego słowa $x \in (\Sigma \setminus \{\sqcup, \triangleright\})^*$ spełnia warunek

$$U(\overline{M}; \widetilde{x}) = M(x).$$

Lemat 2.12 *Istnieją stałe c_1, c_2, c_3 takie, że dla dowolnej maszyny Turinga M i dla dowolnego słowa x nad alfabetem tej maszyny, jeżeli M uruchomiona z daną x wykonuje nie więcej niż t ruchów, to U uruchomiona z daną $\overline{M}; \widetilde{x}$ wykonuje nie więcej niż $c_1 \cdot |\overline{M}| \cdot t + c_2 \cdot |\overline{M}| + c_3$ ruchów.*

Uwaga. Przedstawioną konstrukcję maszyny uniwersalnej można powtórzyć przy założeniu, że rozważamy maszyny o ustalonej liczb taśm. Podane lematy pozostaną prawdziwe. Można powtórzyć ją także dla dowolnych (wielonapisowych) maszyn Turinga, ale wtedy powyższy lemat przestanie być prawdziwy. W konstrukcji można wykorzystać sposób zapamiętywania wielu napisów w postaci jednego, wykorzystywany w dowodzie twierdzenia o redukcji liczby taśm.

Niech

$$H = \{\overline{M};\widetilde{x} : M(x) \neq\nearrow\}.$$

Zbiór H nazywamy problemem stopu. Podana definicja jest mało precyzyjna. Dokładniejsza definicja H ma postać

$$\begin{aligned} H = \{w : \exists M &- \text{jednotaśmowa maszyna Turinga } \exists x \in \Sigma_M^* \text{ i istnieje} \\ &\text{sposób kodowania taki, że } w = \overline{M};\widetilde{x} \wedge M(x) \neq\nearrow\}. \end{aligned}$$

Twierdzenie 2.13 *Zbiór H jest rekurencyjnie przeliczalny (rozpoznawalny).* □

Twierdzenie 2.14 *Zbiór H nie jest rozstrzygalny.*

Dowód. Załóżmy, że H jest rozstrzygalny. Wtedy istnieje maszyna Turinga M_H rozstrzygająca H. Ustalmy kolejność liter alfabetu Σ_U. Konstruujemy maszynę D działającą zgodnie z następującym opisem:

- sprawdza, czy dane słowo $x \in \Sigma_U^*$; jeżeli nie, to odrzuca dane,
- sprawdza, czy x ma prefiks postaci α, β, (w x po β musi być przecinek) taki, że α i $\beta \in \{0,1\}^*$; jeżeli nie, to odrzuca dane, w przeciwnym razie ustala α i β,
- oblicza $k = \alpha + \beta + 6$,
- koduje x zgodnie z ustaloną numeracją, kodując poszczególne znaki słowami długości k, czyli oblicza $\widetilde{x}$,
- uruchamia M_H z daną $x;\widetilde{x}$,
- jeżeli $M_H(x;x) =$ 'yes', to zapętla się, w przeciwnym razie akceptuje dane.

Niech $\overline{D}$ będzie kodem maszyny D wykorzystającym ustaloną numerację znaków alfabetu Σ_U. Zastanówmy się, czemu jest równa wartość $D(\overline{D})$. Zauważmy, że $D(\overline{D}) =\nearrow$ lub $D(\overline{D}) =$ 'yes'.

Załóżmy, że $D(\overline{D}) = \nearrow$. Najpierw pokażemy, że $\overline{D};\widetilde{\overline{D}} \notin H$. Gdyby jednak $\overline{D};\widetilde{\overline{D}} \in H$, to $\overline{D};\widetilde{\overline{D}} = \overline{M};\widetilde{x}$ dla pewnej maszyny M, dla pewnego słowa x i dla pewnego sposobu kodowania takich, że $M(x) \neq \nearrow$. Z lematu 2.11 wynika jednak, że $D(\overline{D}) = M(x)$, a to jest niemożliwe.

Mamy więc $\overline{D};\widetilde{\overline{D}} \notin H$. Oznacza to, że maszyna M_H odrzuca $\overline{D};\widetilde{\overline{D}}$. Wynika stąd, że maszyna D akceptuje słowo $\overline{D}$. To jednak przeczy założeniu.

Jeżeli założymy, że $D(\overline{D}) =$ *'yes'*, to na podstawie definicji H otrzymujemy, że $\overline{D};\widetilde{\overline{D}} \in H$. Z drugiej strony maszyna D uruchomiona z daną $\overline{D}$ znajduje słowo $\widetilde{\overline{D}}$ i oblicza $M_H(\overline{D};\widetilde{\overline{D}})$. W wyniku otrzymuje *'yes'* i zapętla się. Oznacza to, że $D(\overline{D}) = \nearrow$ wbrew założeniu. □

3 Hierarchia czasowa

3.1 Twierdzenie o hierarchii czasowej

Zmodyfikujmy nieco maszynę uniwersalną U. Będziemy zakładać, że maszyna U ma następującą własność:

$$U(\overline{M}y;\widetilde{x}) = M(x)$$

dla dowolnej maszyny Turinga, dla dowolnej danej x dla maszyny M i dla dowolnego słowa $y \in \{0,1\}^*$. Jest oczywiste, że słowo $\overline{M}y;\widetilde{x}$ o wymienionych własnościach można podzielić na $\overline{M}$, y i $\widetilde{x}$. Zakładamy, że maszyna U podczas przygotowywania się symulacji odpowiedniej maszyny Turinga podzieli dane słowo na te trzy części i wymaże y lub po prostu będzie ignorować ten fragment danego słowa.

Ustalmy numerację elementów alfabetu Σ_U. Niech

$$\begin{aligned} L_f = \{x \in \Sigma_U^* : (\exists y \exists \alpha, \beta \in \{0,1\}^*\ x = \alpha, \beta, y) \ \wedge \ U \text{ nie akceptuje } x;\widetilde{x} \\ \text{podczas pierwszych } f(|x|) \text{ ruchów}\}, \end{aligned}$$

gdzie $\widetilde{x}$ oznacza słowo x zakodowane przez zastąpienie jego liter ich numerami w przyjętej numeracji, przedstawianymi w systemie binarnym słowami długości $1 + \lfloor \log(\alpha + \beta + 6) \rfloor$.

Funkcja f jest czasowo konstruowalna, jeżeli istnieje maszyna Turinga, która dla dowolnych danych długości n wykonuje dokładnie $f(n)$ ruchów. Funkcje czasowo konstruowalne można też definiować jako takie funkcje f, dla których istnieje maszyna Turinga tworząca napis długości $f(n)$ po uruchomieniu z dowolnymi danymi długości n.

Lemat 3.1 *Jeżeli f jest funkcją czasowo konstruowalną taką, że $f(n) \geq n+2$ dla wszystkich $n \in N$ oraz*

$$\lim_{n\to\infty} \frac{f(n)}{n^2} = \infty,$$

to $L_f \in TIME(f)$.

Dowód. Definiujemy maszynę Turinga M działającą w następujący sposób (w tym opisie x oznacza dane słowo, a n – jego długość):

- M sprawdza prawdziwość pierwszego członu koniunkcji z definicji L_f; wymaga to wykonania $\mathcal{O}(n)$ kroków,
- oblicza długość kodów liter, koduje słowo x (a więc oblicza słowo $\widetilde{x}$ i znajduje słowo $x;\widetilde{x}$, wymaga to wykonania $\mathcal{O}(n^2)$ ruchów,
- kopiuje słowo x na pomocniczą taśmę; wymaga to $\mathcal{O}(n)$ ruchów,
- jednocześnie uruchamia maszyę U ze słowem $x;\widetilde{x}$ i maszynę wykonującą dokładnie $f(|x|) = f(n)$ ruchów, wymaga to $f(n)$ ruchów,
- odrzuca x, jeżeli maszyna U znalazła się w stanie akceptującym i akceptuje to słowo w przeciwnym przypadku; wymaga to stałej liczby kroków.

Oczywiście maszyna M akceptuje L_f i działa w czasie $f(n) + w(n)$ dla pewnego wielomianu drugiego stopnia w. Korzystając z założenia o rzędzie f, dla pewnego $n_0 \in N$ i dla danych długości $n \geq n_0$ czas pracy M szacujemy przez $c \cdot f(n)$. Z wniosków 2.8 i 2.10 wynika, że $L_f \in TIME(f)$. □

Załóżmy, że $L_f \in TIME(g)$ i weźmy jednonapisową maszynę akceptującą L_f. Z twierdzenia 2.4 wynika, że możemy wybrać taką maszynę M działającą w czasie $h(n) = c \cdot g^2(n) + c' \cdot n + c''$. (Jeżeli weźmiemy maszyną M zbudowaną w dowodzie twierdzenia o redukcji liczby taśm, to stałe we wzorze na h będą zależeć od liczby taśm maszyny o złożoności g, akceptującej L_f. Otrzymaną w ten sposób maszynę możemy poprawić stosują twierdzenie o przyśpieszaniu. W ten sposób otrzymujemy co prawda maszynę dwutaśmową z analogicznym ograniczeniem czasu pracy, ale ze ściśle określonymi stałymi c, c' i c'').

Dla rozważanej maszyny uniwersalnej lemat 2.12 przyjmuje następującą postać:

Fakt 3.1.1 *Jeżeli maszyna M z daną x działa w czasie t, to maszyna U z daną $\overline{M}y;\widetilde{x}$ działa w czasie $c_1 \cdot |\overline{M}| \cdot t + c_2 \cdot |\overline{M}y| + c_3$.*

Oznacza to, że maszyna U oblicza $U(\overline{M}y; \widetilde{\overline{M}y})$ wykonując

$$c_1 \cdot |\overline{M}| \cdot h(|\overline{M}y|) + c_2 \cdot |\overline{M}y| + c_3$$

ruchów.

Lemat 3.2 *Jeżeli*

$$c_1 \cdot |\overline{M}| \cdot h(|\overline{M}y|) + c_2 \cdot |\overline{M}y| + c_3 \leq f(|\overline{M}y|)$$

dla pewnego słowa y, to założenie $L_f \in TIME(g)$ prowadzi do sprzeczności. Tak więc z podanego założenia wynika, że $L_f \notin TIME(g)$.

Dowód. Załóżmy, że $L_f \in TIME(g)$ i weźmy słowo y spełniające podaną nierówność. W przedstawionym dowodzie zostały pominięte szczegóły związane z operacją $\widetilde{x}$.

Przypuśćmy, że $M(\overline{M}y) =$ *'yes'*. Wtedy $\overline{M}y \in L_f$. Wobec tego U nie akceptuje $\overline{M}y; \widetilde{\overline{M}y}$ podczas wykonywania $f(|\overline{M}y|)$ ruchów. Z przyjętej nierówności wynika jednak, że maszyna U uruchomiona ze słowem $\overline{M}y; \widetilde{\overline{M}y}$ wykonuje najwyżej $f(|\overline{M}y|)$ ruchów. Oznacza to, że U nie akceptuje słowa $\overline{M}y; \widetilde{\overline{M}y}$. Tak więc

$$\textit{'no'} = U(\overline{M}y; \widetilde{\overline{M}y}) = M(\overline{M}y).$$

Oczywiście, nie jest to możliwe.

Przypuśćmy więc, że $M(\overline{M}y) =$ *'no'*. Wtedy $\overline{M}y \notin L_f$. Wobec tego U akceptuje $\overline{M}y; \widetilde{\overline{M}y}$ podczas wykonywania $f(|\overline{M}y|)$ ruchów i – po prostu – akceptuje to słowo. Mamy więc

$$\textit{'yes'} = U(\overline{M}y; \widetilde{\overline{M}y}) = M(\overline{M}y).$$

Także i to nie jest możliwe. □

Twierdzenie 3.3 (o hierarchii czasowej) *Jeżeli funkcje f i g spełniają $f(n) \geq n+2$ dla wszystkich $n \in N$, f jest funkcją czasowo konstruowalną oraz*

$$\lim_{n\to\infty} \frac{g^2(n)}{f(n)} = 0, \tag{2}$$

to istnieje język należący do klasy $TIME(f)$ i nie należący do klasy $TIME(g)$.

Dowód. Pokażemy, że L_f spełnia warunki podane w tezie dowodzonego twierdzenia.

Z lematu 3.1 wynika, że $L_f \in TIME(f)$.

Wobec lematu 3.2, aby wykazać, że $L_f \notin TIME(g)$ wystarczy znaleźć y takie, że

$$c_1 \cdot |\overline{M}| \cdot h(|\overline{M}y|) + c_2 \cdot |\overline{M}y| + c_3 \leq f(|\overline{M}y|).$$

Korzystając z nierówności $g(n) \geq n + 2$ otrzymujemy, że

$$\begin{aligned} &c_1 \cdot |\overline{M}| \cdot h(|\overline{M}y|) + c_2 \cdot |\overline{M}y| + c_3 = \\ &\quad = c_1 \cdot |\overline{M}| \cdot (c \cdot g^2(|\overline{M}y|) + c' \cdot |\overline{M}y| + c'') + c_2 \cdot |\overline{M}y| + c_3 \leq C \cdot g^2(|\overline{M}y|) \end{aligned}$$

dla pewnej stałej C. Wystarczy więc znaleźć y takie, że

$$C \cdot g^2(|\overline{M}y|) \leq f(|\overline{M}y|).$$

Istnienie takiego y wynika z założenia (2). □

Twierdzenie 3.4 (o hierarchii czasowej) *Jeżeli funkcje f i g spełniają $f(n) \geq n + 2$ dla wszystkich $n \in N$, f jest funkcją czasowo konstruowalną oraz*

$$\lim_{n\to\infty} \frac{g(n) \cdot \log g(n)}{f(n)} = 0,$$

to istnieje język należący do klasy $TIME(f)$ i nie należący do klasy $TIME(g)$.

Dowód. Twierdzenie to dowodzimy prawie dokładnie tak, jak poprzednie. Teraz korzystamy z maszyny uniwersalnej pozwalającej symulować pracą maszyn dwunapisowych oraz twierdzenia mówiącego, że maszynę wielonapisowaą działającą w czasie g można symulować na dwunapisowej maszynie działającej w czasie $g \cdot \log g$. □

3.2 Twierdzenie o luce

Twierdzenie 3.5 (o luce) *Niech $a : N \to N$ będzie dowolną funkcją rekurencyjną, a $p : N \to N$ – funkcją rekurencyjną taką, że $p(n) > n$ dla dowolnego $n \in N$. Istnieje rekurencyjna funkcja f taka, że $f(n) \leq a(n)$ dla wszystkich $n \in N$ oraz $TIME(f) = TIME(p \circ f)$.*

Dowód. Niech

$$P(i,k) \iff \forall M\ \forall x \in \Sigma_M^* \ (|x| \leq i \wedge |\overline{M}| \leq i$$
$$\Rightarrow \text{maszyna } M \text{ z daną } x \text{ wykonuje } \leq k \text{ lub } > p(k) \text{ kroków}).$$

Nietrudno zauważyć, że P jest relacją rekurencyjną. Aby sprawdzić, czy $P(i,k)$, wystarczy dla skończenie wielu danych uruchomić maszynę uniwersalną i sprawdzić, czy wykonuje więcej niż k i najwyżej $p(k)$ ruchów.

Zdefiniujmy funkcję $g(i,j)$ następującymi wzorami przez rekursję:

$$\begin{aligned} g(i,0) &= \max(a(i), 2\cdot i^2+2), \\ g(i,j+1) &= p(g(i,j)). \end{aligned}$$

Oczywiście, jest to funkcja rekurencyjna, rosnąca ze względu na j. Dla ustalonego i będziemy rozważać zbiory

$$I_j = \{n \in N : g(i,j) < n \leq g(i,j+1)\}.$$

Zbiory tej postaci otrzymane dla różnych j są rozłączne.

Ustalmy liczbę i. Zbiór

$$r(M) = \{t \in N : \exists x \ \in \Sigma_M \ (|x| \leq i$$
$$\wedge\ t \text{ jest liczbą ruchów } M \text{ uruchomionej z daną } x)\}$$

jest oczywiście zbiorem skończonym. Także jest oczywiste, zbiór $r(M)$ jest wyznaczony przez kod maszyny M, a więc $r(M_1) = r(M_2)$ dla dowolnych maszyn takich, że $\overline{M_1} = \overline{M_2}$. Wobec tego zbiór

$$r(w) = \{t \in N : \exists M \exists x \in \Sigma_M \ (w = |\overline{M}| \ \wedge\ |x| \leq i$$
$$\wedge\ t \text{ jest liczbą ruchów } M \text{ uruchomionej z daną } x)\}$$

także jest zbiorem skończonym, jak również skończonym jest zbiór

$$\bigcup_{|w|\leq i} r(w) = \{t \in N : \exists M\ \exists x \in \Sigma_M \ (|\overline{M}| \leq i \ \wedge\ |x| \leq i$$
$$\wedge\ t \text{ jest liczbą ruchów } M \text{ uruchomionej z daną } x)\}.$$

Elementy tego zbioru mogą należeć tylko do skończonej liczby zbiorów I_j. Dla dowolnego i istnieje więc k takie, że

$$I_k \cap \{t \in N : \exists M\ \exists x \in \Sigma_M \ (|\overline{M}| \leq i \ \wedge\ |x| \leq i$$
$$\wedge\ t \text{ jest liczbą ruchów } M \text{ uruchomionej z daną } x)\} = \emptyset.$$

Zauważmy, że własność z poprzedniego zdania można wyrazić krótko stwierdzając, że

$$\forall i\, \exists k\; P(i, g(i,k)).$$

Weźmy funkcję

$$\varphi(n) = \min_l P(n, g(n,l))$$

i zdefiniujmy funkcję f wzorem

$$f(n) = g(n, \varphi(n)).$$

Tak zdefiniowana funkcja f jest całkowita i rekurencyjna, a także ma własność

$$\forall n \in N\; P(n, f(n)).$$

Udowodnimy jeszcze, że $\mathit{TIME}(f) = \mathit{TIME}(p \circ f)$. Zauważmy zawieranie $\subseteq$ w tym wzorze nie wymaga uzasadnienia.

Weźmy język L należący do $\mathit{TIME}(p \circ f)$. Niech M będzie maszyną akceptującą język L w czasie $p \circ f$ i niech $m = |\overline{M}|$. Zdefiniujmy maszynę N działającą zgodnie z programem, który najpierw sprawdza, czy dane słowo jest długości $\leq m$. Maszyna N akceptuje takie słowa bez jakichkolwiek obliczeń, na podstawie informacji zawartych jej programie, a dla pozostałych uruchamia maszynę M i przekazuje wynik jej obliczeń. Jest oczywiste, że maszyna N działa w czasie h danym wzorem

$$h(n) = \begin{cases} n+2 & \text{jeśli } n \leq m \\ 2 \cdot m + 2 + \text{czas działania } M & \text{jeśli } n > m. \end{cases}$$

Oszacujemy czas działania maszyny M dla długich danych. Jeżeli $|\overline{M}| = m < |x| = n$, to w tym celu skorzystamy z własności $P(n, f(n))$. Z tej własności wynika, że m z daną x wykonuje $\leq f(n)$ lub $> p(f(n))$ ruchów. Maszyna o złożoności $p \circ f$ z daną długości n nie może wykonać więcej niż $p(f(n))$ ruchów. Oznacza to, że M z daną x wykonuje $\leq f(n)$ ruchów. Wobec tego, korzystając z nierówności $2 \cdot n^2 + 2 \leq g(n, 0) < g(n, \varphi(n)) = f(n)$, otrzymujemy, że

$$h(n) \leq \begin{cases} n+2 & \text{jeśli } n \leq m \\ 2 \cdot n + 2 + f(n) & \text{jeśli } n > m. \end{cases} \leq 2 \cdot f(n).$$

Mamy więc, że $L \in \mathit{TIME}(2 \cdot f)$. Z przytoczonych nierówności wynika także, że f spełnia założenia wniosku 2.8. Tak więc $L \in \mathit{TIME}(f)$. $\Box$

Wniosek 3.6 *Istnieje rekurencyjna funkcja f taka, że $f(n) \geq n + 2$ dla dowolnego $n \in N$ oraz $\mathit{TIME}(f) = \mathit{TIME}(2^f)$.* $\Box$

4 Hierarchia pamięciowa

4.1 Klasy $SPACE(f)$

W definiując hierarchię pamięciową będziemy posługiwać się maszynami Turinga z taśmą wejściową, rozwiązującymi problemy decyzyjne. Taka maszyna po uruchomieniu ze słowem x wykorzystuje pamięć wielkości $\leq m$ wtedy i tylko wtedy, gdy dla dowolnej konfiguracji $(q, u_1, v_1, u_2, v_2, \ldots, u_k, v_k)$ takiej, że

$$(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{*} (q, u_1, v_1, u_2, v_2, \ldots, u_k, v_k)$$

spełniona jest następująca nierówność

$$\sum_{i=2}^{k} |u_i v_i| \leq m$$

(umawiamy się, że pierwsza taśma jest wejściową; jeżeli maszyna ma także taśmę wyjściową, to jest to ostatnia taśma i wtedy w powyższej definicji sumujemy do $k-1$, a więc nie uwzględniamy pamięci potrzebnej do zapamiętania wyniku obliczeń).

Lemat 4.1 *Dla każdej maszyny Turinga M z taśmą wejściową istnieje dwunapisowa maszyna Turinga M_1 z taśmą wejściową taka, że jeżeli M po uruchomieniu z daną x na każdej taśmie wykorzystuje pamięć wielkości $\leq m$ (jeżeli długość napisu na dowolnej taśmie roboczej nie przekracza m), to M_1 po uruchomieniu z tą samą daną wykorzystuje pamięć wielkości $\leq m$.*

Dowód. Aby to dowieść, wykorzystujemy konstrukcję z dowodu twierdzenia o redukcji liczby taśm polegającą na zastosowaniu maszyny, która symuluje maszynę wielonapisową dzięki temu, że w i-tej komórce ma zapisane wszystkie litery z i-tych komórek maszyny symulowanej i informacje, które z tych liter są obserwowane przez głowice odpowiednich taśm. □

Lemat 4.2 *Jeżeli język (rekurencyjnie przeliczalny) L jest akceptowany przez maszynę Turinga z taśmą wejściową, wykorzystującą dla danych długości n pamięć wielkości nie przekraczającej $f(n)$, to L jest rekurencyjny.*

Dowód. Weźmy dwunapisową maszynę M z taśmą wejściową wykorzystującą pamięć wielkości $\leq f$, akceptującą L. Taka maszyna istnieje na podstawie poprzedniego lematu.

Zdefiniujemy czteronapisową maszynę M_p, która oprócz taśmy wejściowej i taśmy roboczej maszyny M będzie miała taśmę kontrolną i taśmę licznika. Maszyna M_p będzie się posługiwała alfabetem maszyny M i specjalnym p-cyfrowym alfabetem. Litery tego specjalnego alfabetu będziemy uważać za cyfry. Liczba p jest parametrem konstrukcji i zostanie później zostanie odpowiednio dobrana.

Jeżeli maszyna M_p znajdzie się w stanie s, to będzie obserwować na wszystkich taśmach symbole $\triangleright$, będzie mieć na taśmie wejściowej zapisane dane słowo, będzie mieć pustą taśmę roboczą (z napisem $\triangleright\sqcup^k$ dla pewnego k), a napisy pozostałych taśm będą równe $\triangleright 0^k$ (będą tej samej długości, co napis taśmy roboczej). Przed pierwszym przejściem do stanu s maszyna M_p tworzy na taśmie licznika przedstawienie o podstawie p długości danego słowa powiększonej o 2 i zamienia wszystkie cyfry tego przedstawienia na 0 (a więc na taśmie licznika, po uruchomieniu z danymi długości n, przed przejściem do stanu s tworzy napis $\triangleright 0^k$, gdzie $k = 1 + \lfloor \log(n+2) \rfloor > \log(n+2)$).

Na taśmie wejściowej i taśmie roboczej maszyna M_p będzie wykonywać rozkazy maszyny M rozpoczynając od stanu początkowego. Ale będzie je wykonywać tylko wtedy, gdy na taśmie kontrolnej będzie obserwować zapisaną komórkę. Głowica taśmy kontrolnej będzie wykonywać dokładnie te same ruchy, co głowica taśmy roboczej. Po wykonaniu każdego rozkazu maszyny M, maszyna M_p zmieni napis na taśmie licznika wykonując algorytm dodawania jedynki (napis na tej taśmie interpretujemy jako przedstawienie o podstawie p pewnej liczby).

Mogą zdarzyć się następujące sytuacje:

- maszyna M znajdzie się w stanie końcowym; wtedy maszyna M_p zakończy pracę w tym samym stanie końcowym,
- głowica taśmy kontrolnej opuści zapisany obszar; wtedy maszyna M_p wymaże taśmę roboczą, wyzeruje taśmę licznika, wydłuży napis taśmy licznika i taśmy kontrolnej o jeden znak 0, cofnie głowice i przejdzie ponownie do stanu s,
- M_p stwierdzi na taśmie licznika słowo złożone tylko z największych cyfr; wtedy maszyna M_p zakończy pracę i odrzuci dane.

Nietrudno zauważyć, że na wszystkich taśmach roboczych maszyna M_p wykorzystuje tyle samo komórek i tyle komórek, ile jest potrzebne maszynie M. Jest to więc maszyna wykorzystująca dla danych długości n pamięć wielkości najwyżej $f(n)$.

Jest oczywiste, że maszyna M_p zatrzymuje się po uruchomieniu z dowolnymi danymi, a także, że akceptuje tylko takie dane, które zostały zakceptowane przez maszynę M. Wystarczy pokazać, że M_p akceptuje wszystkie dane akceptowane przez M.

Niech $p = 4 \cdot |\Sigma|^2 \cdot |Q|^2$, gdzie Σ oznacza alfabet, a Q – zbiór stanów maszyny M. Przypuśćmy, że M_p została uruchomiona ze słowem długości n i zakończyła pracę po stwierdzeniu, że na taśmie licznika ma ciąg k maksymalnych cyfr. Oznacza to, że maszyna M_p pozwoliła wykonać maszynie M $p^k - 1$ ruchów i podczas wykonywania tych ruchów maszyna M wykorzystywała pamięć wielkości $k+1$. Oczywiście $k > \log(n+2)$. Nietrudno zauważyć, że maszyna M może się znaleźć w najwyżej

$$(n+2)\cdot(k+1)\cdot|\Sigma|^k\cdot|Q| < 2^{\log(n+2)+k}\cdot|\Sigma|^k\cdot|Q| \leq (2\cdot|\Sigma|\cdot|Q|)^{k+\log(n+2)} < (2\cdot|\Sigma|\cdot|Q|)^{2\cdot k} = p^k$$

różnych konfiguracjach wykorzystujących pamięć $\leq k+1$. Jeżeli więc wykonała $p^k - 1$ ruchów korzystając z pamięci wielkości k, to dwukrotnie znalazła się w tej samej konfiguracji i podczas dalszej pracy będzie się cyklicznie znajdować w konfiguracjach, w których już była wcześniej. Wobec tego maszyna M po uruchomieniu z rozważanymi danymi nie zatrzymuje się, i dane te nie należa do języka akceptowanego przez maszynę M. Oznacza to, że maszyna M_p w rozważanym przypadku udzieliła poprawnej odpowiedzi na pytanie, czy M akceptuje dane słowo. W pozostałych przypadkach maszyna M_p również udziela poprawnej odpowiedzi na to pytanie, a więc akceptuje ten sam język. $\Box$

Przyjmijmy, że język $L \in \mathit{SPACE}(f)$ wtedy i tylko wtedy, gdy istnieje maszyna Turinga z taśmą wejściową, rozstrzygająca (akceptująca) język L i wykorzystująca dla danych długości n pamięć wielkości nie przekraczającej $f(n)$.

Fakt 4.3 *Jeżeli SPACE$(f) \neq \emptyset$, to $f(n) \geq 2$ dla dowolnego $n \in N$.* $\diamond$

Twierdzenie 4.4 (o kompresji) *Jeżeli $L \in \mathit{SPACE}(f)$ i $\varepsilon > 0$, to $L \in \mathit{SPACE}(f')$, gdzie f' jest zdefiniowana wzorem $f'(n) = \varepsilon \cdot f(n) + 2$.*

Dowód. Maszyna Turinga wykorzystująca podaną pamięć, symuluje maszynę Turinga akceptującą dany język i wykorzystującą pamięć wielkości f zapisując w pojedyńczych komórkach m-literowe słowa i zapamiętując w swoich stanach także stan maszyny symulowanej i numer tej litery obserwowanego m-literowego słowa, która jest obserwowana przez symulowaną maszynę. $\Box$

Wniosek 4.5 *Jeżeli $c > 1$, $f(n) \geq 2$ dla wszystkich $n \in N$ oraz $f(n) \geq 3$ dla prawie wszystkich $n \in N$, to SPACE$(c \cdot f) \subseteq$ SPACE(f) (i w konsekwencji SPACE$(c \cdot f) =$ SPACE(f)).*

Dowód. Przypuśćmy, że $L \in \mathit{SPACE}(c \cdot f)$. Jeżeli skorzystamy z twierdzenia o kompresji dla $\varepsilon = \frac{1}{3 \cdot c}$, to otrzymamy maszynę Turinga M akceptującą L i wykorzystującą dla danych długości n pamięć wielkości $\frac{1}{3} \cdot f(n) + 2$. Weźmy liczbę m taką, że $f(n) \geq 3$ dla wszystkich $n \geq m$. Język L jest akceptowany także przez maszynę Turinga, która najpierw czyta i zapamiętuje dane słowo x, i jeżeli ma ono długość mniejszą od m, to akceptuje je lub odrzuca, zależnie od tego, czy $x \in L$, a dla słów dłuższych niż m uruchamia maszynę M i kończy pracę w tym samym stanie, w którym zatrzymuje się maszyna M. Tak zdefiniowana maszyna uruchomiona ze słowem długości n wykorzystuje pamięć wielkości $g(n)$, gdzie

$$g(n) = \begin{cases} 2 & \text{jeśli } n < m \\ \frac{1}{3} \cdot f(n) + 2 & \text{jeśli } n \geq m \end{cases} \leq f(n). \;\square$$

4.2 Twierdzenie o hierarchii pamięciowej

Będziemy nam potrzebna maszyna uniwersalna U

- o dwóch taśmach wejściowych: pierwszej, przeznaczonej do zapisywania kodu maszyny symulowanej, i drugiej – do zapisywania danego słowa,
- posługująca się alfabetem $\Sigma_U = \{0, 1, (,), \mathit{przecinek}\}$,
- posługująca się zwykłym pojęciem kodu maszyny Turinga, ale dopuszczającej możliwość wydłużenia kodu o słowo dowolnej długości,
- symulująca (choć niekoniecznie dla wszystkich danych) pracę maszyn Turinga z taśmą wejściową i jedną taśmą roboczą, rozwiązujących problemy decyzyjne i posługujących się alfabetem zawierającym Σ_U; jeżeli $U(u; v)$ oznacza wynik pracy maszyny U uruchomionej ze słowem u na pierwszej taśmie wejściowej i słowem v na drugiej, to dwa ostatnie warunki można precyzyjniej sformułować w następujący sposób:
$$U(\overline{M}y; x) = M(x)$$
dla dowolnej maszyny Turinga o podanych własnościach, dla dowolnych słów $x \in \Sigma_U^*$ i $y \in \{0, 1\}^*$.
- taka, że jeżeli na (drugiej) taśmie wejściowej widzi i-tą literę alfabetu Σ_U w kolejności, w jakiej litery te zostały wyżej wymienione, to uważa ją za literę kodowaną za pomocą przedstawienia (odpowiedniej długości) liczby i.

Taką maszynę uniwesalną konstruujemy podobnie jak zwykłą maszynę uniwersalną, modyfikując konstrukcję tak, aby spełniała podane warunki.

Fakt 4.6 *Jeżeli* $\overline{M}$ *jest kodem maszyny* $M = (K, \Sigma, \delta, s)$, *to* $48 \leq 16 \cdot |K| \cdot |\Sigma| < |\overline{M}|$. *Wobec tego 5 i długość najkrótszego przestawienia liczby* $|K| + |\Sigma| + 6$ *są mniejsze od* $\log|\overline{M}| - 1$. *Stad wynika także, że wykorzystywane przez maszynę* U *kody stanów i liter po uzupełnieniu o symbol początku taśmy mają długości mniejszą od* $\log|\overline{M}|$.

Dowód. Zauważmy, że opis funkcji przejścia maszyny Turinga $M = (K, \Sigma, \delta, s)$ zawiera (jako podsłowo) konkatenację $|K| \cdot |\Sigma|$ słów postaci , $((a, b), (c, d, e))$ dla pewnych niepustych a, b, c, d, e, a kod maszyny jest dłuższy od opisu funkcji przejścia. Każda maszyna ma przynajmniej jeden stan i posługuje się alfabetem o przynajmniej trzech literach (przynajmniej jedna litera i dwie pomocnicze $\sqcup$ i $\triangleright$). Jeżeli m jest długością najkrótszego przedstawienia większej z liczb $|K|$ i $|\Sigma|$, to kody stanów i liter są zapisywane za pomocą najwyżej $m + 2$ znaków. Ponieważ długość najkrótszego przedstawienia liczby a jest równa $1 + \lfloor \log a \rfloor \leq 1 + \log a$, więc

$$m + 2 \leq 3 + \max(\log|K|, \log|\Sigma|) < \log|\overline{M}| - 1. \ \Box$$

Lemat 4.7 *Jeżeli maszyna* M *po uruchomieniu ze słowem* $x \in \Sigma_U^*$ *wykorzystuje pamięć wielkości* $\leq m$, *to dla dowolnego* $y \in \{0, 1\}^*$ *maszyna* U *o podanych własnościach po uruchomieniu ze słowami* $\overline{M}y$ *i* x *na jednej z taśm wykorzystuje pamięć wielkości* $\leq m \cdot |\overline{M}|$, *a na pozostałych taśmach roboczych – pamięć wielkości* $< \log|\overline{M}|$.$\Box$

Niech

$$L_f = \{x \in \Sigma_U^* : x \text{ jest kodem maszyny Turinga i albo } U(x; x) \neq \text{'yes', albo obliczenie } U(x; x) \text{ wymaga na pewnej taśmie roboczej pamięci większej niż } f(|x|)\}.$$

Pierwszy człon koniunkcji w definicji L_f wymaga sprecyzowania i ma zagwarantować odpowiednią złożoność pamięciową maszyny akceptującej L_f także po uruchomieniu jej ze 'złymi' danymi.

Funkcja $f : N \to N$ jest pamięciowo konstruowalna, jeżeli istnieje maszyna Turinga z taśmą wejściową i jedną taśmą roboczą, która dla dowolnych danych długości n wykorzystuje pamięć wielkości dokładnie $f(n)$.

Lemat 4.8 *Dla dowolnej funkcji rekurencyjnej* $g : N \to N$ *istnieje pamięciowo konstruowalna funkcja* f *taka, że* $g(n) \leq f(n)$ *dla wszystkich* $n \in N$.

Dowód. Każdą funkcję rekurencyjną można – korzystając z jednotaśmowej maszyny Turinga – obliczyć w tym sensie, że maszyna dla danego słowa 1^n utworzy słowo $1^{g(n)}$. Funkcję f definiujemy przyjmując, że $f(n)$ jest wielkością pamięci wykorzystywanej przez taką maszynę po uruchomieniu ze słowem długości n. $\Box$

Lemat 4.9 *Jeżeli f jest funkcją pamięciowo konstruowalną taką, że* $\max(2, \log n) \leq f(n)$ *dla wszystkich $n \in N$, to język L_f należy do klasy SPACE(f).*

Dowód. Sprawdzanie, czy dane słowo należy do języka L_f wymaga między innymi uruchomienia maszyny U z tym samym słowem na obu taśmach wejściowych. Maszyna akceptująca L_f powinna mieć jedną taśmę wejściową i nie powinna kopiować danego słowa na żadną taśmę, gdyż wymaga to zbyt dużej pamięci. Zmodyfikujemy więc maszynę U tak, aby

- korzystała z jednej taśmy wejściowej: wymaga to symulowania dwóch głowic za pomocą jednej,
- w zwykły sposób analizowała dane słowo w tych momentach, w których zapoznaje się z kodem maszyny symulowanej,
- jeżeli chce zapoznać się z literą danej dla maszyny symulowanej, to odtwarza właściwe położenie głowicy na podstawie numeru obserwowanej litery pamiętanego na specjalnej taśmie w postaci przedstawienia dwójkowego,
- nie przekraczała specjalnego znaku, jeżeli zostanie on zapisany na taśmie; próba przekroczenia tego znaku przez maszynę U powinna spowodować przejście zmodyfikowanej maszyny do stanu *'no'*.

Tak zmodyfikowaną maszynę U będziemy oznaczać symbolem U_1.

Niech M_f będzie maszyną Turinga, która działa w następujący sposób:

- sprawdza, czy dane słowo x jest postaci α, β, γ dla pewnych $\alpha, \beta \in \{0,1\}^*$ i pewnego $\gamma \in \Sigma_U^*$; jeżeli tak nie jest, to odrzuca dane,
- znajduje najkrótsze słowo x_1 takie, że $x = x_1 x_2$ dla pewnego $x_2 \in \{0,1\}^*$ (wystarczy, aby znalazła ostatnią literę x_1),
- sprawdza, czy x_1 ma przynajmniej 4 litery; jeżeli nie, to odrzuca dane,

- dzieli słowo x_1 na fragmenty po osiem liter i ustala liczbę tych fragmentów; w ten sposób znajduje pewną liczbę l o przedstawieniu dwójkowym długości

$$1 + \left\lfloor \log \left\lfloor \frac{|x_1|}{8} \right\rfloor \right\rfloor = \lfloor \log |x_1| \rfloor - 2,$$

- sprawdza, czy $\alpha + \beta + 6$ ma przedstawienie długości $\leq \lfloor \log |x_1| \rfloor - 2$, tutaj α i β oznaczają słowa takie, jak wyżej i dodatkowo przedstawienie liczby zostało utożsamione z liczbą; jeżeli nie jest to prawdą, to odrzuca dane;
- sprawdza, czy wszystkie – z wyjątkiem dwóch pierwszych – maksymalne podsłowa słowa x_1 zbudowane z liter 0 i 1 mają długość równą długości przedstawienia liczby $\alpha + \beta + 6$; w przeciwnym razie odrzuca dane,
- buduje napis długości $f(|x|)$ i jego ostatnią literę zamienia na ustalony specjalny znak, a pozostałe litery zastępuje znakiem $\sqcup$,
- uruchamia maszyną U_1 w taki sposób, aby korzystała z danych znajdujących się na taśmie wejściowej, a taśmę z napisem, o którym jest mowa w poprzednim punkcie, uważała za taśmę roboczą maszyny symulowanej,
- akceptuje dane, wtedy i tylko wtedy, gdy zostaną odrzucone przez U_1.

Jeżeli przyjmiemy, że sprawdzenie, czy x jest kodem maszyny Turinga wymaga wykonania czynności opisanych w podanych punktach z wyjątkiem trzech ostatnich, to oczywiście maszyna M_f akceptuje język L_f. Nietrudno zauważyć, że jeżeli M jest maszyną Turinga i $y \in \{0,1\}^*$, to maszyna M_f uzna słowo $\overline{M}y$ za kod maszyny Turinga i podczas pracy będzie także realizować trzy ostatnie punkty swojego programu. Maszyna M_f podczas sprawdzania, czy x jest kodem maszyny Turinga, tworzy napisy długości mniejszej niż $\log |x| - 1$ i odrzuca te wszystkie dane, które mogą spowodować utworzenie na taśmach pomocniczych maszyny U_1 napisów długości przynajmniej równej $\log |x| - 1$. Wobec założenia, że f jest pamięciowo konstruowalna, zaznaczenie litery o numerze $f(|x|)$ wymaga pamięci tej samej wielkości. Oznacza to, że M_f z danymi długości n wykorzystuje na poszczególnych taśmach pamięć wielkości $\leq \max(f(n), \log n - 2)$. Na podstawie twierdzenia o kompresji 4.4 (lub lematu 4.1) stwierdzamy, że $L_f \in SPACE(f)$. $\square$

Twierdzenie 4.10 (o hierarchi pamięciowej) *Jeżeli f jest funkcją pamięciowo konstruowalną taką, że $f(n) \geq \max(2, \log n)$ dla wszystkich $n \in N$ oraz*

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0,$$

to pewien język (np. język L_f) należy do SPACE(f) i nie należy do SPACE(g).

Dowód. Jeżeli język $L_f \in SPACE(g)$, to istnieje maszyna Turinga z jedną taśmą roboczą akceptująca L_f i wykorzystująca pamięć wielkości g. Niech M będzie taką maszyną. Pokażemy, że możemy otrzymać sprzeczność odpowiednio dobierając $y \in \{0,1\}^*$ i analizując pracę maszyny M z daną $\overline{M}y$.

Maszyna M po uruchomieniu ze słowem $\overline{M}y$ wykorzystuje pamięć wielkości $g(|\overline{M}y|)$. Stąd wynika, że maszyna U_1 (podobnie jak maszyna U) po uruchomieniu ze słowem $\overline{M}y$ wykorzystuje na taśmie odpowiadającej taśmie roboczej maszyny symulowanej pamięć wielkości $\log|\overline{M}| \cdot g(|\overline{M}y|)$. Załóżmy, że słowo y zostało dobrane tak, aby

$$\log|M| \cdot g(|\overline{M}y|) \leq f(|\overline{M}y|) - 2.$$

Istnienie takiego słowa y wynika z założenia o granicy $\frac{g}{f}$. Przy tym założeniu, maszyna U_1 po uruchomieniu ze słowem $\overline{M}y$ nie opuści obszaru zaznaczonego na taśmie odpowiadającej taśmie roboczej maszyny symulowanej. Zakończy więc pracę w takim stanie, jak maszyna U, a więc

$$U_1(\overline{M}y) = U(\overline{M}y; \overline{M}y) = M(\overline{M}y).$$

Z drugiej strony, maszyny M i M_f po uruchomieniu z dowolnym słowem zatrzymują się w tym samym stanie, a maszyna M_f uruchomiona ze słowem $\overline{M}y$ zatrzymuje się w stanie różnym od $U_1(\overline{M}y)$. Oznacza to, że

$$M(\overline{M}y) = M_f(\overline{M}y) \neq U_1(\overline{M}y).$$

Udowodnione równości nie mogą być jednocześnie prawdziwe. □

5 *TIME* versus *SPACE*

5.1 Twierdzenie o maszynach respektujących bloki

Przyjmijmy, że M jest k-napisową maszyną Turinga, która po uruchomieniu ze słowem x oraz pomocniczym słowem długości m, i po wykonaniu t rozkazów znajduje się konfiguracji

$$(q_t, u_{t,1}, v_{t,1}, \ldots, u_{t,k}, v_{t,k}).$$

Maszyna Turinga M respektuje bloki, jeżeli dla dowolnych danych działa tak, że dla wszystkich możliwych i, t spełnione są następujace warunki:

- jeżeli $|u_{t,i}| = m \bmod m$ oraz $t \neq m-1 \bmod m$, to $|u_{t+1,i}| \leq |u_{t,i}|$,
- jeżeli $|u_{t,i}| = 1 \bmod m$ oraz $t \neq m-1 \bmod m$, to $|u_{t+1,i}| \geq |u_{t,i}|$.

Jeżeli czas pracy maszyny M podzielimy na przedziały, w których maszyna znajduje się w m kolejnych konfiguracjach (w konfiguracjach o numerach od $i{\cdot}m$ do $(i{+}1){\cdot}m{-}1$ dla $i \in N$), a taśmy podzielimy na bloki po m liter (składające się z liter o numerach od $i{\cdot}m{+}1$ do $(i{+}1){\cdot}m$ dla $i \in N$), to podane warunki oznaczają, że wszystkie głowice przekraczają granice między blokami w chwilach podzielnych przez m (wyrażanych liczbami podzielnymi przez m, a więc następne przekraczanie granicy może wystąpić po kilku seriach złożonych z m rozkazów).

O maszynie respektującej bloki mówimy, że działa w czasie f, jeżeli dla dowolnego słowa x i dla dowolnej liczby m wykonuje nie więcej niż $f(|x|)$ ruchów (a więc oszacowanie liczby ruchów maszyny respektującej bloki nie zależy od m).

O maszynie respektującej bloki mówimy, że respektuje bloki długości m, jeżeli została uruchomiona z pewnym słowem i liczbą m.

Twierdzenie 5.1 *Jeżeli $L \in TIME(f)$, to L jest rozpoznawany przez maszynę Turinga o złożoności $\mathcal{O}(f)$, respektującą bloki.*

Dowód. O maszynie M mówimy, że jest świadoma istnienia bloków, jeżeli po otrzymaniu dowolnego słowa i liczby m, w komórkach o numerach przystających do $1 \bmod m$ oraz w komórkach o numerach przystających do $m \bmod m$ zapisuje litery, które m.in. pozwalają ustalić, że numery komórek mają podane własności. Komórki między kolejnymi o numerach przystających do 1 i do $m \bmod m$ nazywamy blokiem. Innymi słowy, maszyna świadoma istnienia bloków ma zaznaczone początki i końce bloków.

Maszyna M świadoma istnienia bloków i posługująca się blokami długości m długo przebywa w poszczególnych blokach, jeżeli po wejściu do dowolnego bloku wykonuje kolejne $\geq m-1$ ruchów nie opuszczając tego bloku.

Fakt 5.1.1 *Jeżeli $L \in TIME(f)$, to L jest rozpoznawany przez maszynę Turinga o złożoności $\mathcal{O}(f)$, długo przebywającą w poszczególnych blokach.*

Dowód. Bierzemy maszynę M rozpoznającą L. Rozszerzamy jej alfabet, tak aby w pojedyńczej komórce mogła m.in. pamiętać trzy litery, a także aby mogła zaznaczać początki i końce bloków. Posługując się tym alfabetem definiujemy maszynę M^*, która będzie maszyną świadomą istnienia bloków i w poszczegółnych blokach będzie pamiętać także zawartość bloków sąsiednich.

Jeżeli maszyna M pamięta w poszczególnych blokach słowa

$$\ldots, x, y, z, t, \ldots$$

to maszyna M^* będzie odpowiednich blokach pamiętać po trzy następujące słowa:

$$\ldots, (x, \ldots, y^R), (y, x^R, z^R), (z, y^R, t^R), (t, z^R, \ldots), \ldots$$

(będziemy sobie wyobrażać, że słowa te są pamiętane jedno nad drugim, ponadto opisaną sytuacją należy uważać za początkową, później zapis będzie podobny, z pewnymi odstępstwami od podanej struktury). W pojedyńczym bloku jest więc pamiętana także zawartość (w odwrotnym porządku) bloków sąsiednich. Dzięki temu M^* będzie mogła zmieniać zawartość pewnego bloku mimo, że głowica będzie obserwować inny blok. Aby napis miał podaną strukturę konieczna będzie aktualizacja zapisu poszczególnych bloków. Aktualizacja będzie następować w odpowiednim momencie, z pewnym opóźnieniem.

Przypuśćmy, że maszyna M^* obserwuje blok z zapisem (y, x^R, z^R). Słowo y nazywamy główną częścią napisu z tego bloku, pozostałe słowa – dodatkowymi, odpowiednio lewą i prawą. Jeżeli M^* rozpoczyna obserwowanie takiego bloku, to uważamy, że obserwuje słowo y i wtedy przekształca je tak, jak maszyna M. Jeżeli M opuszcza odpowiedni blok w prawo, to M^* nie opuści tego bloku, a zacznie przekształcać słowo z^R tak, jak maszyna M, ale będzie wykonywać ruchy odwrotne i w ten sposób będzie dokładnie symulować działanie M. Po pewnym czasie może znowu przekształcać słowo y, po opuszczeniu tego słowa w lewo będzie przekształcać słowo x^R podobnie jak było przekształcane słowo z^R. W ten sposób działanie pierwotnej maszyny będzie symulowane tak długo, aż zajdzie potrzeba opuszczenia słowa z^R w lewo, albo słowa x^R w prawo. Pierwsza z tych sytuacji odpowiada opuszczaniu przez M słowa z w prawo. Wtedy M^* opuści dotychczas modyfikowany blok b i przesunie się w prawo nad sąsiedni blok b'.

Przypuśćmy, że tuż przed opuszczeniem (w prawo) bloku b mamy na taśmie następujący zapis:

$$\ldots, (x, \ldots, y^R), (\dot{y}, \dot{x}^R, \dot{z}^R), (z, y^R, t^R), (t, z^R, \ldots), \ldots$$

Teraz, przed dalszą pracą maszyny M^* powinna zostać zaktualizowana zawartość poszczególnych napisów pamiętanych w bloku b'. Zostanie tam umieszczona aktualna zawartość głównej części napisu z bloku b (jako lewa, dodatkowa część napisu) i prawej, dodatkowej części napisu (jako część główna). Otrzymamy w ten sposób następujący zapis:

$$\ldots, (x, \ldots, y^R), (\dot{y}, \dot{x}^R, \dot{z}^R), (\dot{z}, \dot{y}^R, t^R), (t, z^R, \ldots), \ldots$$

Głowica zostanie umieszczona w bloku z zapisem $(\dot{z}, \dot{y}^R, t^R)$ i będzie obserwować ostatnią literę słowa $\dot{z}$. W tej sytuacji może symulować ruch w prawo maszyny M bez opuszczania bloku, odpowiednio przesuwając się nad znajdujące się w tym bloku słowo t^R. Nietrudno zauważyć, że wykonanie ruchu nad granicą bloku (zmiana obserwowanego bloku) wymaga czasu proporcjonalnego do długości bloku.

Dodajmy dalsze szczegóły pracy maszyny M^*. Jeżeli M^* widzi pewną komórkę po raz pierwszy, to najpierw obejmuje w posiadanie blok zawierający tę komórkę. Blok ten zaczyna się lub kończy (zależnie od ruchu wykonanego przez głowicę) komórką obserwowaną po raz pierwszy, i ma ustaloną długość. Objęcie w posiadanie polega na zapisaniu do pierwszej i ostatniej komórki litery pamiętającej, że jest to odpowiednio pierwszą lub ostatnią literą pewnego bloku i dodatkowo pamiętającej literę znajdującą się w tej komórce. Po objęciu bloku w posiadanie, maszyna M^* nadaje informacji zapisanej w tym bloku odpowiednią postać (definiuje część główną i dodatkowe), aktualizuje zawartość tego bloku, a także umieszcza w nim słowo odpowiedniej długości zapisane w komórkach sąsiednich, nie objętych jeszcze w posiadanie (bez obejmowania tych komórek w posiadanie).

Maszyna M^* wykonuje trzy rodzaje ruchów: symuluje pojedyncze ruchy maszyny M, obejmuje bloki w posiadanie, i aktualizuje zawartość bloków. Maszyna M^* symuluje $f(n)$ pojedynczych ruchów i wymaga to od niej wykonania $f(n)$ ruchów. Maszyna ta obejmuje w posiadanie najwyżej $\frac{f(n)}{m}$ bloków. Objęcie w posiadanie pojedynczego bloku wymaga $c \cdot m$ ruchów. Łącznie zajmuje jej to $\mathcal{O}(f(n))$ ruchów. Podobnie szacujemy czas aktualizacji bloków. Maszyna M^* między kolejnymi aktualizacjami zawartości symuluje przynajmniej m ruchów maszyny M. Aktualizacje są więc wykonywane nie więcej niż $\frac{f(n)}{m}$ razy i wymagają czasu proporcjonalnego do m. Łącznie wymagają więc $\mathcal{O}(f)$ ruchów. $\diamond$

Bierzemy k-napisową maszynę długo przybywającą w poszczególnych blokach. Jest to maszyna świadoma istnienia bloków. Dodajemy jej licznik ruchów zliczający liczbę ruchów modulo m. Jeżeli ta maszyna chce przekroczyć granicę bloków, to jest wstrzymywana tak długo, aż nastąpi odpowiedni moment. Taka modyfikacja maszyny długo przybywającej w poszczególnych blokach wymaga najwyżej przemnożenia czasu pracy przez $k+1$. □

5.2 Gry z kamykami

Przypuśćmy, że mamy dany acykliczny graf skierowany $G = (V, E)$, wierzchołek $v \in V$ oraz pewną liczbę kamyków. Graf G uważamy za planszę do gry. Podczas gry będziemy kłaść kamyki na wierzchołkach tego grafu lub usuwać je zgodnie z dwoma regułami:

- mamy prawo położyć kamyk na wierzchołku v pod warunkiem, że kamyki leżą na wszystkich poprzednikach v,
- oraz mamy prawo usunąć dowolny kamyk.

Gry tej postaci będziemy nazywać grami z kamykami. Układanie kamyka na wierzchołku i usuwanie go będziemy nazywać ruchami w grze z kamykami.

Będą interesować nas następujące pytania: czy mając dany graf, jego wierzchołek i pewną liczbę kamyków możemy położyć kamyk na danym wierzchołku, albo ile kamyków potrzeba, aby móc położyć kamyk na dowolnym wierzchołku danego grafu.

Fakt 5.2 *Jeżeli mamy acykliczny graf skierowany i tyle kamyków, co wierzchołków, to możemy położyć kamyk na dowolnym wierzchołku tego grafu.* □

Fakt 5.3 *Jeżeli w grze z kamykami na grafie o n wierzchołkach, posługując się pewną liczbą kamyków możemy położyć kamyk na pewnym wierzchołku, to możemy położyć kamyk na tym wierzchołku wykonując nie więcej niż 2^n ruchów.*

Dowód. Na grafie o n wierzchołkach możemy rozmieścić kamyki na 2^n sposobów. Jeżeli podczas gry dwa razy wystąpi pewien układ kamyków, to można grać krócej. Najkrótsza partia gry z kamykami wymaga więc najwyżej 2^n ruchów. □

Sytuacją w grze z kamykami nazywamy układ (G, k, X, t) czterech elementów takich, że

- G jest acyklicznym grafem skierowanym (niech n będzie liczbą wierzchołków G),
- k jest liczbą naturalną nie większą niż n,
- X jest zbiorem najwyżej k wierzchołków grafu G,

- t jest liczbą naturalną nie większą niż 2^n.

Niech G będzie ustalonym acyklicznym grafem skierowanym. Wierzchołek v grafu G w sytuacji (G, k, X, t) jest osiągalny, jeżeli po ułożeniu kamyków na wierzchołkach grafu G należących do zbioru X, posługując się łącznie k kamykami i wykonując najwyżej t ruchów w grze na grafie G możemy położyć kamyk na wierzchołku v. Mając pewną ustaloną liczbę k kamyków, o wierzchołku v mówimy, że jest osiągalny w grze na grafie G (o n wierzchołkach), jeżeli jest osiągalny w sytuacji $(G, k, \emptyset, 2^n)$.

Będziemy zakładać, że grafy są kodowane za pomocą słów w taki sposób, że każda litera słowa kodującego odpowiada wierchołkowi grafu i dla danych dwóch liter tego słowa, potrafimy stwierdzić za pomocą maszyny Turinga wykorzystującej pamięć logarytmiczną, czy wierzchołki odpowiadające tym literom są połączone krawędzią.

Niech

$$\mathcal{R} = \{(G, k, X, t, v) : (G, k, X, t) \text{ jest sytuacją w grze z kamykami i } v \text{ w tej sytuacji jest osiągalny}\}.$$

Fakt 5.4 *Istnieje niedeterministyczna maszyna Turinga, która rozstrzyga problem $\mathcal{R}$ i po uruchomieniu ze słowem dlugości n kodującym graf G i czterema innymi słowami wykorzystuje na każdej taśmie pamięć wielkości $n + 2$ (wliczając w to komórki z symbolem początku taśmy i pozwalające stwierdzić koniec danych).*

Dowód. Maszyna ta odgaduje ruchy w grze na grafie G, symuluje ich wykonanie i jednocześnie je złicza. Akceptuje dane, jeżeli podczas działania położy kamyk na wierzchołku danym wierzchołku. Jeżeli zasymuluje wskazaną liczbę ruchów i nie zaakceptuje danych, to odrzuca je. □

Fakt 5.5 *Istnieje deterministyczna maszyna Turinga, która rozstrzyga problem $\mathcal{R}$ i po uruchomieniu ze grafem o n wierzchołkach wykorzystuje pamięć wielkości $\mathcal{O}(n^2)$ i*
.

Dowód. Jest to wniosek z poprzedniego faktu i twierdzenia Savitcha. □

Niech $\mathcal{M}$ będzie maszyną, której istnienie wynika z powyższego faktu. Zdefiniujemy inną maszynę $\mathcal{M}_s$: dla danej sytuacji (G, k, X, t) i dla danego wierzchołka v zwróci informację, że v nie jest osiągalny w danej sytuacji lub zwróci ruch, który należy wykonać, aby osiągnąć v. Ruch możemy utożsamiać z wierzchołkiem: jeżeli zostanie wskazany wierzchołek, na którym znajduje się kamyk, to ruch będzie polegać na usunięciu tego kamyka, a w przeciwnym razie – na położeniu kamyka na wskazanym wierzchołku. Niech $X+r$ oznacza zbiór X zmieniony przez wykonanie ruchu r, a więc

$$X + r = \begin{cases} X \setminus \{r\} & \text{jeśli } r \in X \\ X \cup \{r\} & \text{jeśli } r \notin X. \end{cases}$$

Maszyna $\mathcal{M}_s$ będzie działać zgodnie z następującym programem:

dla każdego ruchu r

$Y := X + r$**;**

jeżeli $\mathcal{M}(G, k, Y, t-1, v) =$ *'yes'*, **to zwróć ruch** r;

zwróć *'wierzchołka v nie można osiągnąć'*

Fakt 5.6 *Maszyna $\mathcal{M}_s$ po uruchomieniu z grafem o n wierzchołkach wykorzystuje pamięć wielkości $\mathcal{O}(n^2)$.* □

Rozważmy maszynę, która dla danego grafu G o n wierzchołkach, danej liczby $k \leq n$ i danego wierzchołka v działa zgodnie z programem z następującymi instrukcjami:

jeżeli $\mathcal{M}(G, k, \emptyset, 2^n, v) =$ *'no'*, **to zwróć** *'wierzchołka v nie można osiągnąć'***;**

$X := \emptyset$**;**

dla t **od** 2^n **do** 1 **wykonaj**

{gdyby teraz uruchomić $\mathcal{M}$ z danymi (G, k, X, t, v), to zakończyłaby pracę w stanie *'yes'*, a gdyby uruchomić $\mathcal{M}_s$ z tymi samymi danymi, to zwróciłaby pewien ruch}

$r := \mathcal{M}_s(G, k, X, t, v)$**;**

$X := X + r$**;**

jeżeli $v \in X$, **to zwróć** *'v zostało osiągnięte'***;**

{gdyby teraz uruchomić $\mathcal{M}$ z danymi $(G, k, X, t-1, v)$, to zakończyłaby pracę w stanie *'yes'*}

zwróć *'błąd!'*

Fakt 5.7 *Podany algorytm nie kończy pracy stwierdzeniem, że wystąpił błąd, a więc wykonując ruchy wskazane przez $\mathcal{M}_s$ osiągamy dany wierzchołek, jeżeli tylko jest to możliwe.*

Dowód. Wystarczy sprawdzić, że podczas wykonywania podanego programu informacje podane w jego treści będą zawsze prawdziwe. □

Niech $\mathcal{G}_d$ oznacza rodzinę acyklicznych skierowanych grafów z wierzchołkami o stopniu wejściowym $\leq d(wszystkimi)$.

Twierdzenie 5.8 *Jeżeli $G = (V, E) \in \mathcal{G}_d$ i w grze z kamykami na grafie G osiągnięcie pewnego wierzchołka wymaga $\geq n$ kamyków, to $|E| \geq \frac{1}{16 \cdot d} \cdot n \cdot \log n$.*

Dowód. Niech $d \geq 2$ będzie ustaloną liczbą oraz niech

$$R(n) = \min\{|E| : \exists V\ \exists v \in V\ (V, E) \in \mathcal{G}_d \wedge \text{osiągniecie } v \text{ w grze na } (V, E) \text{ wymaga} \geq n \text{ kamyków}\}.$$

Fakt 5.8.1 *Funkcja R jest niemalejąca.* ◇

Fakt 5.8.2 *Dla każdej liczby $n \geq 1$ mamy $R(n) \geq n - 1$. Jeżeli $1 \leq n \leq d + 1$, to $R(n) = n - 1$.*

Dowód. Przypuśćmy, że $R(n) < n - 1$. Wtedy istnieje graf G o najwyżej $n - 2$ krawędziach i wierzchołek tego grafu, którego osiągnięcie w grze z kamykami na grafie G wymaga $\geq n$ kamyków. Dojdziemy do sprzeczności wykazując, że osiągniecie tego wierzchołka wymaga $\leq n - 1$ kamyków. Zdefiniujmy zbiory

$$A_i = \{v : \text{ wszystkie poprzedniki } v \text{ w grafie } G \text{ należą do } \bigcup_{j<i} A_j\}.$$

Kładziemy $n - 2$ kamyki na takie wierzchołki, z których wychodzą krawędzie, i które należą kolejno do zbiorów A_0, A_1, $A_2, \ldots$. Teraz $(n - 1)$-szy kamyk możemy położyć na dowolnym wierzchołku grafu G.

Jeżeli $n \leq d + 1$, to łatwo zbudować graf z wierzchołkami stopnia wejściowego $\leq d$, z $n - 1$ krawędziami i wierzchołkiem, którego osiągnięcie wymaga $\geq n$ kamyków. Bierzemy n elementów i każdy z $n - 1$ pierwszych elementów łączymy krawędzią z ostatnim. W ten sposób otrzymujemy graf o żądanych własnościach. ◇

Niech $n > 2 \cdot d$. Niech $G = (V, E) \in \mathcal{G}_d$ będzie grafem, w którym jest wierzchołek wymagający $\geq n$ kamyków, i takim, że $|E| = R(n)$.

Niech

$$V_1 = \{v \in V : \ v \text{ jest osiągalne w grze na grafie } G \text{ za pomocą } \leq \frac{n}{2} \text{ kamyków}\}.$$

Przyjmijmy, że $V_2 = V \setminus V_1$, $G_1 = (V_1, E \cap V_1^2)$ oraz $G_2 = (V_2, E \cap V_2^2)$.

Fakt 5.8.3 *W grze z kamykami na G_2 jest wierzchołek wymagający $\geq \left\lceil \frac{n}{2} \right\rceil - d$ kamyków.*

Dowód. Załóżmy, że każdy wierzchołek G_2 wymaga $\leq \left\lceil \frac{n}{2} \right\rceil - d - 1$ kamyków. Pokażemy, że każdy wierzchołek w grze na grafie G wymaga najwyżej $n - 1$ kamyków. Jest to oczywiste dla wierzchołków z V_1.

Weźmy więc wierzchołek v z V_2. Kamyki dzielimy na dwie częsci: pierwsza składa się z $\left\lfloor \frac{n}{2} \right\rfloor + d - 1$ kamyków, kamyki z tej części będziemy kłaść na wierzchołki z V_1, druga składa się z $\left\lceil \frac{n}{2} \right\rceil - d - 1$ kamyków przeznaczonych do układania na elementach V_2. Gramy strategią pozwalającą położyć kamyk na v w grze na grafie G_2. Ponieważ gramy w grze na grafie G, aby położyć kamyk na $v' \in V_2$ musimy mieć kamyki także na poprzednikach v' ze zbioru V_1. Tych poprzedników jest najwyżej d i dysponujemy dostatecznie dużą liczbą kamyków przeznaczonych do układania na V_1, aby móc ułożyć kamyki na dowolnych d elementach V_1. $\diamond$

Fakt 5.8.4 *Jeżeli pewien wierzchołek wymaga n kamyków i $n > 1$, to pewien jego poprzednik wymaga $\geq n - d$ kamyków.*

Dowód. Możemy założyć, że $n > d + 1$. Niech v będzie dowolnym wierzchołkiem. Gdyby wszystkie poprzedniki v wymagały $\leq n - d - 1$ kamyków, to mając $(n - d - 1) + (d - 1)$ kamyków możemy położyć je na wszystkich poprzednikach v. Wtedy, mając jeszcze jeden kamyk możemy położyć go na v. Więc aby położyć kamyk na v, wystarczy posiadać $(n - d - 1) + (d - 1) + 1 = n - 1$ kamyków. $\diamond$

Fakt 5.8.5 *W grze z kamykami na G_1 jest wierzchołek wymagający $\geq \left\lceil \frac{n}{2} \right\rceil - d$ kamyków.*

Dowód. Weźmy minimalny wierzchołek v w grafie G_2. v musi mieć poprzedniki w grafie G. Wszystkie poprzedniki v należą do V_1. Ponieważ v nie należy do V_1, więc wymaga $m > \frac{n}{2}$ kamyków. Wobec poprzedniego faktu, pewien poprzednik wymaga $m - d > \frac{n}{2} - d$ kamyków. $\diamond$

Niech $A = E \setminus (E_1 \cup E_2)$.

Fakt 5.8.6 *Jeżeli* $|A| \leq \left\lfloor \frac{n}{4} \right\rfloor$, *to w grze z kamykami na* G_2 *pewien wierzchołek wymaga* $\geq \left\lceil \frac{3 \cdot n}{4} \right\rceil$ *kamyków.*

Dowód. Załóżmy, że w grze na grafie G_2 każdy wierzchołek wymaga $\leq \left\lceil \frac{3 \cdot n}{4} \right\rceil - 1$ kamyków. Uzyskamy sprzeczność dowodząc, że wtedy w grze na grafie G każdy wierzchołek wymaga $\leq n-1$ kamyków.

Gramy w następujący sposób: wybieramy $\left\lfloor \frac{n}{4} \right\rfloor$ kamyków. Wybrane kamyki kładziemy na wierzchołkach, z których wychodzą krawędzie należące do A. Wierzchołki te należą do V_1 i kładąc na nich kamyki możemy stosować strategię używaną w grze na grafie G wymagającą $\leq \frac{n}{2}$ kamyków. Układając te kamyki być może będziemy korzystać z pozostałych: ale jest ich dostatecznie dużo i będziemy mogli je zabrać z grafu G_1. Dalsze układanie kamyków odbywa się dokładnie tak, jak w grze na grafie G_2 i jest potrzebne tyle kamyków, ile potrzebujemy w grze na tym grafie. Jeżeli wystarczy $\left\lceil \frac{3 \cdot n}{4} \right\rceil - 1$ kamyków, to łącznie wykorzystamy $\left\lfloor \frac{n}{4} \right\rfloor + \left\lceil \frac{3 \cdot n}{4} \right\rceil - 1 \leq n-1$ kamyków. ◇

Fakt 5.8.7 *Jeżeli* $|A| \leq \left\lfloor \frac{n}{4} \right\rfloor$, *to graf* G_2 *ma podgraf* $G_3 = (V_3, E_3)$ *taki, że* $|E_2 \setminus E_3| \geq \left\lfloor \frac{n}{4 \cdot d} \right\rfloor$ *i grze z kamykami na* G_3 *pewien wierzchołek wymaga* $\geq \left\lceil \frac{n}{2} \right\rceil$ *kamyków.*

Dowód. Weźmy wierzchołek grafu G_2 wymagający $\geq \left\lceil \frac{3 \cdot n}{4} \right\rceil$ kamyków i oznaczmy go v_0. Dla $m = \left\lfloor \frac{n}{4 \cdot d} \right\rfloor$ zdefiniujemy ciąg $v_0, v_1, \ldots, v_m$ tak, aby jeżeli v_i wymaga k kamyków, to v_{i+1} jest poprzednikiem v_i wymagającym $\geq k-d$ kamyków. Nietrudno zauważyć, że v_m wymaga

$$\geq \left\lceil \frac{3 \cdot n}{4} \right\rceil - d \cdot \left\lfloor \frac{n}{4 \cdot d} \right\rfloor \geq \left\lceil \frac{3 \cdot n}{4} \right\rceil - d \cdot \frac{n}{4 \cdot d} = \left\lceil \frac{3 \cdot n}{4} \right\rceil - \frac{n}{4} \geq \frac{n}{2}$$

kamyków.

Usuńmy z grafu G_2 wszystkie krawędzie zakończone w wierzchołkach $v_0, v_1, \ldots, v_{m-1}$. Otrzymamy w ten sposób pewien graf G_3. Jest oczywiste, że $|E_2 \setminus E_3| \geq \left\lfloor \frac{n}{4 \cdot d} \right\rfloor$.

Teraz wystarczy zauważyć, że v_m w grafach G_2 i G_3 wymaga tyle samo kamyków. ◇

Na podstawie przeprowadzonych rozważań możemy otrzymać oszacowanie liczby krawędzi grafu G. Przypomnijmy, że $R(n) = |E|$, a także $E = E_1 \cup E_2 \cup A$. Te trzy części zbioru E są rozłączne, wiec $|E| = |E_1| + |E_2| + |A|$. Z udowodnionych faktów wynika, że

$$|E_1|,\ |E_2| \geq R(\left\lceil \frac{n}{2} \right\rceil - d).$$

Jeżeli założymy, że $|A| > \left\lfloor \frac{n}{4} \right\rfloor$, to otrzymujemy, że

$$R(n) = |E| = |E_1| + |E_2| + |A| \geq 2 \cdot R(\left\lceil \frac{n}{2} \right\rceil - d) + \frac{n}{4} \geq 2 \cdot R(\left\lceil \frac{n}{2} \right\rceil - d) + \frac{n}{4 \cdot d}.$$

Jeżeli $|A| \leq \left\lfloor \frac{n}{4} \right\rfloor$, to będziemy inaczej szacować $R(n)$ korzystając z oczywistej nierówności $|A| \geq 1$ i wynikającej z podanych faktów nierówności

$$|E_2| \geq |E_3| + \left\lfloor \frac{n}{4 \cdot d} \right\rfloor \geq R(\left\lceil \frac{n}{2} \right\rceil) + \left\lfloor \frac{n}{4 \cdot d} \right\rfloor.$$

Stąd wynika, że

$$R(n) = |E_1| + |E_2| + |A| \geq R(\left\lceil \frac{n}{2} \right\rceil - d) + R(\left\lceil \frac{n}{2} \right\rceil) + \left\lfloor \frac{n}{4 \cdot d} \right\rfloor + 1 \geq 2 \cdot R(\left\lceil \frac{n}{2} \right\rceil - d) + \frac{n}{4 \cdot d}.$$

Nietrudno zauważyć, że funkcja R spełnia założenia następującego faktu dla $c = \frac{1}{4 \cdot d}$.

Fakt 5.8.8 *Niech d i c będą stałymi takimi, że $d \geq 2$ i $0 < c \leq 1$. Załóżmy, że $f(n) \geq n - 1$ dla wszystkich $n > 0$ oraz*

$$f(n) \geq 2 \cdot f(\left\lceil \frac{n}{2} \right\rceil - d) + c \cdot n$$

dla wszystkich $n > 2 \cdot d$. Wtedy

$$f(n) \geq \frac{c}{4} \cdot n \cdot \log n$$

dla wszystkich $n \geq 16 \cdot d^2$, a jeżeli dodatkowo założymy, że $c \leq \frac{1}{\log(4 \cdot d)}$, to podana nierówność jest prawdziwa dla wszystkich dodatnich liczb naturalnych n.

Dowód. Niech

$$n(k, i) = 2^i \cdot k + 2 \cdot d \cdot (2^i - 1).$$

Weźmy liczbę naturalną $n > 2 \cdot d$. Niech i będzie liczbą naturalną taką, że

$$2 \cdot d \cdot (2^i - 1) < n \leq 2 \cdot d \cdot (2^{i+1} - 1).$$

Nierówności te implikują, że $i > 0$,

$$\log \frac{n}{2 \cdot d} < i + 1 \quad \text{oraz} \quad \frac{1}{2} \cdot \frac{n}{2 \cdot d} < 2^i.$$

Niech teraz k będzie taką liczbą, że

$$2^i \cdot (k-1) < n - 2 \cdot d \cdot (2^i - 1) \le 2^i \cdot k.$$

i wprowadźmy oznaczenie

$$\varepsilon = 2^i \cdot k - (n - 2 \cdot d \cdot (2^i - 1)).$$

W ten sposób dla danej liczby n znaleźliśmy liczby i, k oraz ε takie, że

$$n = 2 \cdot d \cdot (2^i - 1) + 2^i \cdot k - \varepsilon = n(k, i) - \varepsilon$$

a także spełniające $0 < k \le 2 \cdot d$ oraz $0 \le \varepsilon < 2^i$.

Zauważmy, że

$$n(k, i) \ge n = n(k, i) - \varepsilon \ge n(k, i) - 2^i$$

oraz, że z tej nierówności wynika następująca:

$$n(k, i-1) \ge \left\lceil \frac{n}{2} \right\rceil - d \ge n(k, i-1) - 2^{i-1}.$$

Z powyższego spostrzeżenia przez łatwą indukcję ze względu na i dowodzimy, że jeżeli

$$n(k, i) \ge n \ge n(k, i) - 2^i$$

dla dowolnych liczb n, k, i, to także

$$f(n) \ge 2^i \cdot f(k) + c \cdot \sum_{j=1}^{i} 2^{i-j} n(k, j) - c \cdot i \cdot 2^i.$$

Wobec tego, dla dowolnej liczby $n > 2 \cdot d$ i wyżej wskazanych liczb k, i mamy

$$\begin{aligned}
f(n) &\ge 2^i \cdot f(k) + c \cdot \sum_{j=1}^{i} 2^{i-j} n(k, j) - c \cdot i \cdot 2^i \\
&= 2^i \cdot f(k) + c \cdot \sum_{j=1}^{i} [2^i \cdot k + 2 \cdot d \cdot 2^i + 2 \cdot d \cdot 2^{i-j}] - c \cdot i \cdot 2^i \\
&= 2^i \cdot f(k) + c \cdot (k + 2 \cdot d) \cdot i \cdot 2^i + 2 \cdot d \cdot c \cdot \sum_{j=1}^{i} 2^{i-j} - c \cdot i \cdot 2^i \\
&\ge 2^i \cdot (k-1) + c \cdot (k + 2 \cdot d - 1) \cdot i \cdot 2^i + 2 \cdot d \cdot c \cdot (2^i - 1)
\end{aligned}$$

$$
\begin{aligned}
&\geq c \cdot [2^i \cdot (k+2\cdot d-1) + i \cdot 2^i \cdot (k+2\cdot d-1)] - 2\cdot c\cdot d \\
&= c \cdot (i+1) \cdot 2^i \cdot (k+2\cdot d-1) - 2\cdot c\cdot d \\
&\geq \frac{c}{2} \cdot \log(\frac{n}{2\cdot d}) \cdot \frac{n}{2\cdot d} \cdot (k+2\cdot d-1) - 2\cdot c\cdot d \\
&\geq \frac{c}{2} \cdot n \cdot \log(\frac{n}{2\cdot d}) - 2\cdot c\cdot d
\end{aligned}
$$

Dalej zakładamy, że $4\cdot d < 16\cdot d^2 \leq n$. Pozwala to zastąpić w odpowiednich miejscach n przez $16 \cdot d^2$ oraz $4\cdot d$ przez n. Dalsze szacowanie może zostać przeprowadzone w następujący sposób:

$$
\begin{aligned}
&= \frac{c}{4} \cdot n \cdot \log n + \frac{c}{4} \cdot (n \cdot \log n - 2\cdot n \cdot \log(2\cdot d) - 8\cdot d) \\
&\geq \frac{c}{4} \cdot n \cdot \log n + \frac{c}{4} \cdot (n \cdot \log(16\cdot d^2) - 2\cdot n \cdot \log(2\cdot d) - 2\cdot n) \\
&\geq \frac{c}{4} \cdot n \cdot \log n + \frac{c}{4} \cdot 2\cdot n \cdot (\log(4\cdot d) - \log(2\cdot d) - 1) \\
&= \frac{c}{4} \cdot n \cdot \log n.
\end{aligned}
$$

Natomiast nierówność $c \leq \frac{1}{\log(4\cdot d)}$ implikuje dodatkowo dla $n \leq 16\cdot d^2$, że

$$\frac{c}{4} \cdot n \cdot \log n \leq \frac{c}{4} \cdot n \cdot 2 \cdot \log(4\cdot d) \leq \frac{n}{2} \leq n-1 \leq f(n). \ \diamond$$

Z powyższego faktu otrzymujemy następujący wniosek:

$$R(n) \geq \frac{1}{16\cdot d} \cdot n \cdot \log n. \ \square$$

Twierdzenie 5.9 *Jeżeli $G \in \mathcal{G}_d$ ma $k \geq 2$ wierzchołków, to dowolny wierzchołek jest osiągalny w grze na grafie G z $48 \cdot d^2 \cdot \frac{k}{\log k}$ kamykami.*

Dowód. Zauważmy najpierw oczywisty fakt:

Fakt 5.9.1 *Jeżeli $G = (V, E) \in \mathcal{G}_d$, to $|E| \leq |V| \cdot d$.* $\diamond$

Korzystając z powyższego, twierdzenia 5.8 i definicji funkcji R podanej na początku dowodu tego twierdzenia otrzymujemy, że

$$
\begin{aligned}
&\min\{|V| : \exists E\ \exists v \in V\ (V,E) \in \mathcal{G}_d \wedge \text{osiągniecie } v \\
&\qquad \text{w grze na } (V,E) \text{ wymaga} \geq n \text{ kamyków}\} \geq \frac{1}{16\cdot d^2} \cdot n \cdot \log n.
\end{aligned}
\tag{3}
$$

Niech $S(n)$ oznacza

$$\min\{k : \forall G = (V,E) \in \mathcal{G}_d\ (|V| = n \Longrightarrow \text{ w grze na } G \text{ wystarcza } k \text{ kamyków})\}.$$

Fakt 5.9.2 *Jeżeli $G = (V, E) \in \mathcal{G}_d$ i $|V| = n$, to w grze z kamykami na G wystarcza $S(n+1)$ kamyków.*

Dowód. Weźmy $a \notin V$ i zdefiniujmy graf $G' = (V \cup \{a\}, E)$. G' ma $n+1$ wierzchołków i w grze na G' wystarcza tyle samo kamyków, co w grze na G. W grze na G wystarcza więc $S(n+1)$ kamyków. $\diamond$

Fakt 5.9.3 *Funkcja S jest niemalejąca.*

Dowód. Jest to konsekwencja poprzedniego faktu. $\diamond$

Fakt 5.9.4 *Weźmy liczby naturalne n i m takie, że*

$$1 \leq m = \left\lceil \frac{1}{16 \cdot d^2} \cdot n \cdot \log n \right\rceil - 1.$$

Wtedy $S(m) \leq n$.

Dowód. Załóżmy, że $S(m) > n$. Wtedy $S(m) - 1$ nie należy do zbioru z definicji $S(m)$. Oznacza to, że istnieje graf $G = (V, E) \in \mathcal{G}_d$ taki, że $|V| = m$, oraz istnieje $v \in V$, który w grze z kamykami na G wymaga więcej niż $S(m) - 1$ kamyków. Tym bardziej wierzchołek ten wymaga więcej niż n kamyków. Wobec tego $|V|$ należy do zbioru X ze wzoru 3 oraz

$$m = |V| \geq \min X \geq \frac{1}{16 \cdot d^2} \cdot n \cdot \log n > \left\lceil \frac{1}{16 \cdot d^2} \cdot n \cdot \log n \right\rceil - 1 = m,$$

co jest niemożliwe. $\diamond$

Niech k będzie dodatnią liczbą naturalną. Weźmy liczbę n taką, że

$$(n-1) \cdot \log(n-1) \leq 16 \cdot d^2 \cdot k < n \log n.$$

Dla tej liczby n mamy

$$k \leq \left\lceil \frac{1}{16 \cdot d^2} \cdot n \cdot \log n \right\rceil - 1 = m$$

i – tym bardziej –

$$S(k) \leq S(m) \leq n.$$

Z drugiej strony, zachodzi następujący fakt:

Fakt 5.9.5 *Jeżeli* $n > 3$ *oraz* $(n-1) \cdot \log(n-1) \leq K < n \cdot \log n$, *to*

$$n \leq \frac{3 \cdot K}{\log K}.$$

Dowód. Dla $n > 3$ mamy $n \cdot \log n < (n-1)^2$, a więc także $\log K \leq 2 \cdot \log(n-1)$ oraz

$$\frac{K}{\log K} \geq \frac{(n-1) \cdot \log(n-1)}{2 \cdot \log(n-1)} = \frac{n-1}{2} \geq \frac{n}{3}. \diamond$$

Korzystając z podanego faktu dla $K = 16 \cdot d^2 \cdot k$ otrzymujemy, że

$$S(k) \leq n \leq \frac{48 \cdot d^2 \cdot k}{\log(48 \cdot d^2 \cdot k)} \leq 48 \cdot d^2 \cdot \frac{k}{\log k}. \square$$

5.3 Twierdzenie Hopcrofta-Paula-Valianta

Twierdzenie 5.10 *Jeżeli funkcja* $f(n) \geq 4$ *dla wszystkich* $n \in N$ *oraz* $f(n) \geq 10$ *dla prawie wszystkich* $n \in N$, *to*

$$TIME(f) \subseteq SPACE(\frac{f}{\log f}).$$

Dowód. Niech $m \in N$. Przyjmijmy, że k-napisowa maszyna Turinga M po uruchomieniu ze ustalonymi danymi i po wykonaniu t rozkazów znajduje się konfiguracji

$$(q_t, u_{t,1}, v_{t,1}, \ldots, u_{t,k}, v_{t,k}).$$

Opisem pracy maszyny M po uruchomieniu z tymi danymi nazywamy ciąg informacji

$$(q_0, a_{0,1}, \ldots, a_{0,k}), (q_m, a_{1,1}, \ldots, a_{1,k}), \ldots (q_{T \cdot m}, a_{T,1}, \ldots, a_{T,k}) \tag{4}$$

gdzie T jest ilorazem czasu pracy maszyny przez m, a liczby $a_{i,j} \in \{-1, 0, 1\}$ spełniają następujące warunki:

- $a_{i,j} = 1$ wtedy i tylko wtedy, gdy $|u_{i \cdot m-1,j}| = m \bmod m$ oraz $|u_{i \cdot m-1,j}| < |u_{i \cdot m,j}|$,
- $a_{i,j} = -1$ wtedy i tylko wtedy, gdy $|u_{i \cdot m-1,j}| = 1 \bmod m$ oraz $|u_{i \cdot m-1,j}| > |u_{i \cdot m,j}|$,
- oraz $a_{i,j} = 0$ w pozostałych przypadkach.

Taśmy maszyny M podzielmy na bloki długości m. Zauważmy, że $a_{i,j} = 1$ wtedy i tylko wtedy, gdy podczas wykonania $(i \cdot m)$-tego rozkazu głowica j-tej taśmy przekracza granicę bloku wykonując ruch w prawo. Warunek $a_{i,j} = -1$ wskazuje, że wtedy przekraczanie granicy miało miejsce podczas wykonywania ruchu w lewo, a $a_{i,j} = 0$ – że wtedy nie nastąpiło przekroczenie granicy. Będziemy wyrównywać opisy pracy przez dopisanie na końcu zbędnych informacji.

Niech M będzie maszyną respektującą bloki. Skonstruujemy maszynę M_1, która będziemy akceptować ten sam język, co maszyna M dzięki symulacji pracy maszyny M w określonych przedziałach czasowych. Na razie będziemy konstruować pewną maszynę M_2, która będzie działać podobnie do maszyny M_1, ale po otrzymaniu danej x i – dodatkowo – pomocniczej liczby D.

Maszyna M_2 będzie uruchamiać maszynę M_3. M_3 będzie sprawdzać, czy pewne słowo jest opisem pracy maszyny M i będzie posługiwać się taśmą, którą będziemy określać jako taśma z blokami. Będzie też symulować pracę M w pewnych przedziałach czasowych. Dzięki temu będzie mogła informować nas, że:

- dany ciąg nie jest opisem pracy maszyny M (zawiera nieprawdziwe informacje),
- dany ciąg jest fragmentem opisu pracy maszyny M, nie pozwala na pełne odtworzenie jej pracy,
- dany ciąg jest opisem pracy maszyny M i podczas opisanej pracy M akceptuje dane słowo, lub odrzuca je.

Maszyna M_2 po uruchomieniu ze słowem x i liczbą D działa zgodnie z następującym schematem:

$m := (D-1)^2$**;**

$B := \left\lceil \dfrac{C \cdot D}{\lfloor \log D \rfloor} \right\rceil$**;** gdzie $C = 48 \cdot (k+1)^2$

$p := 1 + \lfloor \log B \rfloor$**;**
(przyjmujemy, że te trzy liczby i liczba D są zapamiętywane jako słowa o odpowiedniej długości)

utwórz na taśmie z blokami napis $(\#\sqcup^m)^B\#\#$**;**
(a więc zostanie zaznaczone miejsce na B bloków po m komórek)

$\alpha :=$ **pierwszy ciąg znaków długości** D**;**

dopóki maszyna M_3 **zwraca, że** *'α nie jest opisem pracy M'* (a więc zawiera nieprawdziwe informacje)

$\alpha :=$ **następny w porządku leksykograficznym ciąg długości** D**;**

zwróć ostatnią informację przekazaną przez M_3**.**

Znaczenie stałych definiowanych na początku programu, taśmy z blokami i utworzonego na niej napisu zostanie wyjaśnione dalszej części rozumowania. Jest oczywiste, że maszyna M_2 zwróci drugą lub trzecią z wyżej wymienionych informacji, przekazywanych przez maszynę M_3. Wynika to stąd, że istnieje ciąg długości D zawierający prawdziwą informację o pracy maszyny M i ciąg ten będzie analizowany w pewnym momencie pracy M_2.

Maszyna M_3 będzie grać w grę z kamykami na grafie $G_{M,m}$. Wierzchołkami tego grafu będą przedziały długości m czasu pracy maszyny M jako respektującej bloki długości m. Dwa przedziały: i-ty i j-ty łaczymy krawędzią (od i-tego do j-tego), jeżeli

- $j = i + 1$
- lub $i = 0$ i w j-tym przedziale czasu pewna głowica obserwuje komórki pewnego bloku, który nie był obserwowany przez tę głowicę podczas wcześniejszej pracy maszyny M,
- lub też $i < j$ oraz w i-tym i j-tym przedziale czasu pewna głowica maszyny M obserwuje komórki tego samego bloku, a w międzyczasie głowica ta obserwuje komórki innych bloków.

Nietrudno zauważyć, że graf $G_{M,m}$ jest acykliczny i skierowany, a stopień wejściowy jego wierzchołków jest $\leq k + 1$, gdzie k jest liczbą taśm maszyny M.

Fakt 5.10.1 *Opis pracy maszyny M jako respektującej bloki długości m zawiera pełną informację o grafie $G_{M,m}$.*

Dowód. Jeżeli dany jest opis (4), to aby ustalić, czy w grafie $G_{M,m}$ jest krawędź od i do j wystarczy policzyć sumy

$$s^l_{h,j} = \sum_{t=h}^{j-1} a_{t,l}$$

dla $h = i, \ldots, j-1$ i $l = 1, \ldots, k$ (liczby $s^l_{h,j}$ to różnice numerów bloków obserwowanych przez l-tą głowicę w h-tym i j-tym przedziale czasowym) i zauważyć, że taka krawędź jest w grafie $G_{M,m}$ wtedy i tylko wtedy, gdy dla pewnego l ($l = 1, \ldots, k$) zachodzi warunek

$$\begin{array}{ll} & j = i+1 \\ \vee & i = 0 \wedge \forall h < j \; s^l_{h,j} \neq 0 \\ \vee & s^l_{i,j} = 0 \wedge \forall h < j \; (i \leq h \Longrightarrow s^l_{h,j} \neq 0). \ \diamond \end{array}$$

Teraz wprowadzimy oznaczenia, które będziemy stosować w definicji maszyny M_3. Symbolem α będziemy oznaczać słowo zapisane na taśmie przeznaczonej do przechowywania opisu maszyny M (w końcu powinien być to opis maszyny M). Symbolem G^α oznaczamy graf wyznaczony przez słowo α tak, jak na podstawie opisu maszyny M wyznaczaliśmy graf $G_{M,m}$ w dowodzie faktu 5.10.1. Wierzchołki grafu G^α utożsamiamy z liczbami $i < D$, czyli numerami poszczególnych liter słowa α i opisujących własności odpowiadających im wierzchołków. Przyjmijmy, że Δ to wierzchołek odpowiadający liczbie $D-1$.

Maszyna M_3 będzie posługiwała się taśmą z kamykami. Komórki tej taśmy odpowiadają wierzchołkom grafu G^α. Na taśmie z kamykami będą pojawiały się – oprócz znaku pustej komórki – inne znaki. Znaki te będą wskazywały wierzchołki, na których znajdują się kamyki. Symbolem X będziemy oznaczać zbiór wierzchołków, na których znajdują się (aktualnie) kamyki.

Przyjmujemy, że kamyki mają swoje nazwy (lub numery) będące słowami długości p nad alfabetem $\{0, 1\}$. Możemy się umówić, że nazwy kamyków są umieszczone na specjalnej taśmie w takiej kolejności, w jakiej kamyki (informacje o kamykach) są umieszczone na taśmie z kamykami. Kamyki oznaczamy literą κ. Symbol ten utożsamiamy z nazwą lub numerem kamyka. Wierzchołek, na którym znajduje się κ oznaczamy symbolem $i(\kappa)$. Kamyk znajdujący się wierzchołku $i(\kappa) - 1$ będziemy oznaczać symbolem κ_0, pod warunkiem, że rzeczywiście się tam znajduje. Pozostałe poprzedniki (w grafie G^α) wierzchołka $i(\kappa)$ odpowiadają taśmom maszyny M. Jeżeli na poprzedniku wierzchołka $i(\kappa)$ odpowiadającym l-tej taśmie znajduje się kamyk, to kamyk ten będzie oznaczany symbolem κ_l.

Każdy kamyk zwiążemy z pewnym fragmentem długości m taśmy z blokami, ograniczonym znakami #. Jeżeli nazwę kamyka zinterpretujemy jako liczbę b, to kamyk jest związany z fragmentem o numerze b. Fragmenty taśmy z blokami będą zawierać słowa opisujące zawartość bloków znajdujących się na taśmach maszyny M i obserwowanych przez głowice w ustalonym momencie czasu. Słowo zapisane we fragmencie odpowiadającym kamykowi κ będziemy oznaczać symbolem $c(\kappa)$.

Kamyki będą rozmieszczane i usuwane na podstawie informacji uzyskanych za pomocą maszyny $\mathcal{M}_s$. Maszyna ta jest opisana w rozdziale 5.2. Oblicza strategię w grze z kamykami, a więc w szczególności przestrzega zasad obowiązujących w tej grze.

Podczas pracy maszyny M_3 będzie spełniony następujący warunek:

♠ dla dowolnego i oraz kamyka κ, jeżeli $i = i(\kappa)$, to w $c(\kappa)$ znajduje się informacja o zapisach na poszczególnych taśmach maszyny M w blokach, w których znajdują się głowice podczas i-tego etapu pracy maszyny M, ustalona tuż przed rozpoczęciem obliczeń maszyny M w $(i+1)$-szym etapie, oraz informacja o położeniu głowic nie opuszczających swoich bloków ustalona dla tego samego momentu.

Maszyna M_3 działa zgodnie z następującym programem (komentarze zawierają niezmienniki i są prawdziwe wtedy, gdy są czytane podczas wykonania programu):

$t := 2^D$;
{ostatni wierzchołek Δ jest osiągalny w sytuacji $(G^\alpha, B, \emptyset, t)$ (patrz tw. 5.9), a ponieważ na żadnym wierzchołku nie ma kamyka, jest także prawdą, że jeżeli na i-tym wierzchołku leży kamyk, to $\alpha(1)\ldots\alpha(i)$ jest prefiksem opisu pracy M, oraz, że jest spełniony warunek ♠}

powtarzaj:

$r := \mathcal{M}_s(G^\alpha, B, X, t, \Delta)$;
{w sytuacji $(G^\alpha, B, X+r, t-1)$ wierzchołek Δ jest osiągalny}

jeżeli ruch r **polega na usunięciu kamyka** κ**, to zastąp** $c(\kappa)$ **przez** $\sqcup^m$**, usuń numer kamyka i sam kamyk;**

w przeciwnym razie:

ustal numer κ **pustego fragmentu na taśmie z blokami;**
{zauważmy, że taki fragment na pewno istnieje}

ustaw kamyk κ **na wskazanym wierzchołku;**

dla wszystkich l **od 1 do** k (k to liczba taśm maszyny M)

skopiuj ze słowa $c(\kappa_l)$ **do** $c(\kappa)$ **informacje o zapisie na** l**-tej taśmie lub wypełnij odpowiednią część** $c(\kappa)$ **znakami, które były na** l**-tej taśmie w momencie uruchomienia** M**;**

korzystając z informacji o zapisie taśm z $c(\kappa)$ **oraz z informacji o stanie** M **i ruchach głowic zapisanych w** $\alpha(i(\kappa)-1)$ **symuluj wykonanie** m **ruchów maszyny** M;

jeżeli podczas symulacji maszyna M **zatrzymała się, to przekaż tę informację wraz ze stanem, w którym** M **zakończyła pracę;**

jeżeli informacja $\alpha(i_\kappa)$ **różni się od tego, co zostało ustalone po zakończeniu symulacji, to przekaż** *'α nie jest opisem pracy M'*;

$t := t - 1$;

$\{\Delta$ jest osiągalny w sytuacji (G^α, B, X, t), że jeżeli na i-tym wierzchołku leży kamyk, to $\alpha(1)\ldots\alpha(i)$ jest prefiksem opisu pracy M, oraz że spełniony jest warunek $\spadesuit\}$

tak długo, aż $\Delta \in X$; (aż kamyk zostanie położony na ostatnim wierzchołku)

przekaż, że *'D jest za małą liczbą'*.

Maszyna M_1 działa zgodnie zgodnie z programem:

$D := 3$;

dopóki M_2 **stwierdza, że** *'D jest za małą liczbą'*

$D := D + 1$;

przekaż ostatnią informację otrzymaną od M_2.

Nietrudno sprawdzić, że maszyna M_1 posługuję się taśmami (długość bez symbolu początku taśmy):

- długości D, m, B, p, t,
- z opisem pracy M, długości D,
- z kamykami, długości D,
- z nazwami kamyków, długości $B \cdot p$,
- pomocniczymi maszyny $\mathcal{M}_s$, długości $\mathcal{O}(D^2)$,

- z blokami, długości $B \cdot (m+1)+2$.

O złożoności pamięciowej maszyny M_1 decyduje długość napisu na taśmie z blokami. Zauważmy, że

$$B \cdot (m+1)+3 \leq (\frac{C \cdot D}{\lfloor \log D \rfloor}+1) \cdot ((D-1)^2+1)+3 \leq \frac{C \cdot D^3}{\lfloor \log D \rfloor}+D^2 \leq \frac{(C+1) \cdot D^3}{\log(\frac{1}{2} \cdot D)}.$$

Fakt 5.10.2 *Funkcja*

$$h(x)=\frac{x}{\log(\frac{1}{2} \cdot x)}$$

jest rosnąca dla $x \geq 4$. $\diamond$

Załóżmy, że M jest maszyną o złożoności czasowej f. Przypuśćmy, że D jest najmniejszą liczbą taką, że M uruchomiona ze słowem x i liczbą $m=(D-1)^2$ ma opis pracy długości $\leq D$. Przyjmijmy, że $D>3$ oraz M z danymi x i $(D-2)^2$ wykonuje R ruchów. Opis pracy M z tymi danymi ma długość $\left\lceil \frac{R}{(D-2)^2} \right\rceil + 1$. Z definicji D wynika, że liczba ta nie jest $\leq D-1$. Wobec tego

$$D-2<\left\lceil \frac{R}{(D-2)^2} \right\rceil.$$

Stąd wynika, że

$$(D-2)^3<R \leq f(|x|)=f(n).$$

W końcu otrzymujemy, że

$$D<\sqrt[3]{f(n)}+2 \leq 3 \cdot \sqrt[3]{f(n)}.$$

Otrzymana nierówność i przytoczony fakt pozwalają oszacować pamięć wykorzystywaną przez maszynę M_1 w następujący sposób:

$$\leq \frac{(C+1) \cdot D^3}{\log(\frac{1}{2} \cdot D)} \leq \frac{27 \cdot (C+1) \cdot f(n)}{\log(\frac{3}{2} \cdot \sqrt[3]{f(n)})} \leq \frac{27 \cdot (C+1) \cdot f(n)}{\log(\sqrt[3]{f(n)})}=\frac{51 \cdot (C+1) \cdot f(n)}{\log f(n)}.$$

Przytoczone oszacowanie pozostaje słuszne także w przypadku $D=3$. $\square$