11

Trudność i zupełność problemów obliczeniowych

W czasie poprzednich wykładów zapoznaliśmy się z pojęciem klasy złożoności obliczeniowej. Poznaliśmy również kilka ważniejszych klas złożoności czasowej i pamięciowej wraz z przykładami problemów do nich należących. Podczas dzisiejszego wykładu zajmiemy się analizą problemów, których złożoność obliczeniowa jest ściśle związana ze złożonością obliczeniową całej klasy. Problemy takie nazywać będziemy trudnymi oraz zupełnymi.

Intuicyjnie problemami *trudnymi* dla ustalonej klasy złożoności (czasowej lub pamięciowej) będziemy nazywać problemy, których rozwiązanie jest co najmniej tak trudne, jak rozwiązanie każdego problemu z tej klasy. Jeśli dodatkowo problem taki należeć będzie do rozważanej klasy złożoności, będziemy nazywać go problemem *zupełnym* dla tej klasy.

Efektywne redukcje problemów obliczeniowych

W celu porównywania trudności obliczeniowej problemów będziemy potrzebować narzędzi pozwalających zamieniać instancję jednego problemu na instancję innego problemu. Procedurę taką nazywać będziemy *redukcją*. Istotne jest jednak, aby używana redukcja nie wpływała na zmianę rozpatrywanej klasy złożoności obliczeniowej. Dlatego też złożoność samej redukcji musi być „łatwa” w porównaniu ze złożonością typowego problemu rozważanej klasy złożoności.

Przypomnijmy, że wszystkie poznane przez nas deterministyczne modele obliczeń są wielomianowo równoważne, natomiast złożoność czasowa problemów mierzona na maszynach deterministycznych i niedeterministycznych może się

różnić wykładniczo. Traktowanie wielomianowych różnic w złożoności jako nieistotnych i pomijanie ich pozwala budować teorię niezależną od wyboru modelu obliczeń. Z tego właśnie powodu w większości zastosowań przydatne są redukcje w czasie wielomianowym.

Definicja 11.1

Powiemy, że funkcja $f : \Sigma^* \to \Sigma^*$ jest obliczalna w czasie wielomianowym, jeśli istnieje maszyna Turinga M działająca w czasie wielomianowym, która dla dowolnego słowa $w \in \Sigma^*$ zatrzymuje się zwracając wartość $f(w)$.

Przykładem takiej funkcji może być dowolny wielomian. Obliczenie jego wartości przez maszynę Turinga będzie wymagało co najwyżej wielomianowej liczby kroków.

Definicja 11.2

Język $A \subseteq \Sigma^*$ jest wielomianowo redukowalny do języka $B \subseteq \Sigma^*$, jeśli istnieje funkcja $f : \Sigma^* \longrightarrow \Sigma^*$ obliczalna w czasie wielomianowym taka, że dla każdego $w \in \Sigma^*$

$$w \in A \iff f(w) \in B.$$

Funkcję f nazywamy wielomianową redukcją A do B.

Redukcja w czasie wielomianowym pozwala zastosować efektywne rozwiązanie jednego problemu do rozwiązania innych problemów w następujący sposób. Przypuśćmy, że $f : \Sigma^* \longrightarrow \Sigma^*$ jest redukcją w czasie wielomianowym języka A do języka B, oraz potrafimy efektywnie (np. w czasie wielomianowym) rozwiązać problem „$w \in B$”. Wówczas, aby rozstrzygnąć problem „$w \in A$”, wyznaczamy wartość $f(w)$, a następnie rozstrzygamy problem „$f(w) \in B$”. Zauważmy, że czasowa złożoność obliczeniowa tak uzyskanego rozwiązania problemu „$w \in A$” będzie co najwyżej wielomianowo większa od złożoności rozwiązania problemu „$w \in B$”.

Wiemy, że maszyna Turinga działająca w czasie wielomianowym może wykorzystać wielomianową liczbę komórek pamięci. Zatem redukcja w czasie wielomianowym może nie być wystarczająca dla problemów należących do klas LOGSPACE oraz NLOGSPACE. W tym przypadku musimy rozważać redukcję w pamięci logarytmicznej.

Definicja 11.3

Powiemy, że funkcja $f : \Sigma^* \longrightarrow \Sigma^*$ jest obliczalna w pamięci logarytmicznej jeśli istnieje maszyna Turinga M wykorzystująca co najwyżej logarytmiczną liczbę komórek na taśmach roboczych, która dla dowolnego słowa $w \in \Sigma^*$ zatrzymuje się zwracając wartość $f(w)$.

Przykładem takiej funkcji może być $f(n) = n^2$. Jeśli maszyna Turinga licząca wartość funkcji f będzie przetwarzała liczby w zapisie binarnym użyje co najwyżej $O(\log n)$ komórek na taśmach roboczych.

Definicja 11.4

Język $A \subseteq \Sigma^*$ jest redukowalny do języka $B \subseteq \Sigma^*$ w pamięci logarytmicznej, jeśli istnieje funkcja $f : \Sigma^* \longrightarrow \Sigma^*$ obliczalna w pamięci logarytmicznej taka, że dla każdego $w \in \Sigma^*$

$$w \in A \iff f(w) \in B.$$

Funkcję f nazywamy redukcją A do B w pamięci logarytmicznej.

Zauważmy, że maszyna Turinga używająca pamięci rozmiaru $O(\log n)$ może mieć co najwyżej $n \cdot c^{O(\log n)}$ różnych konfiguracji, dla pewnego $c \in \mathbb{N}$. Zatem redukcja w pamięci logarytmicznej jest również redukcją w czasie wielomianowym. Implikacja w drugą stronę nie jest niestety prawdziwa.

Problemy trudne i zupełne

Możemy teraz podać formalną definicję problemu trudnego dla ustalonej klasy złożoności obliczeniowej.

Definicja 11.5

Niech $\mathcal{C}$ będzie klasą złożoności obliczeniowej (czasowej lub pamięciowej). Język A nazywamy $\mathcal{C}$-trudnym jeśli dla dowolnego języka $B \in \mathcal{C}$ istnieje *efektywna* redukcja (w czasie wielomianowym lub pamięci logarytmicznej) do języka A.

Jeśli dodamy wymaganie, aby rozważany problem należał do rozpatrywanej klasy złożoności, otrzymamy formalną definicję problemu zupełnego dla klasy złożoności obliczeniowej.

Definicja 11.6

Niech $\mathcal{C}$ będzie klasą złożoności obliczeniowej (czasowej lub pamięciowej). Język A nazywamy $\mathcal{C}$-zupełnym jeśli:

1. $A \in \mathcal{C}$
2. A jest $\mathcal{C}$-trudny

Dowodzenie $\mathcal{C}$-zupełności języka A polega zatem na wykazaniu, że należy on do klasy $\mathcal{C}$ oraz każdy inny problem z tej klasy może być sprowadzony do A za pomocą efektywnej redukcji w czasie wielomianowym lub pamięci logarytmicznej.

Przykład problemu zupełnego

Przypomnijmy rozważany w czasie poprzedniego wykładu problem istnienia ścieżki między dwoma wyróżnionymi wierzchołkami w grafie skierowanym

$$\text{PATH} = \Big\{(G, v_1, v_2) : \text{ w grafie skierowanym } G \text{ istnieje ścieżka } v_1 \rightsquigarrow v_2\Big\}.$$

Pokazaliśmy, że PATH $\in$ NL. Okazuje się, że problem jest również NL-trudny. Jest zatem przykładem problemu NL-zupełnego.

Twierdzenie 11.1

Problem PATH jest NL-zupełny.

Dowód

Przypomnijmy, że problem PATH $\in$ NL. Niedeterministyczna maszyna Turinga może rozstrzygnąć istnienie ścieżki między wierzchołkami v_1 oraz v_2 grafu G dokonując w każdym kroku obliczeń niedeterministycznego wyboru kolejnego wierzchołka ją tworzącego. Zapamiętując na taśmie roboczej wyłącznie binarną reprezentację numeru aktualnie przetwarzanego wierzchołka, może rozstrzygnąć ten problem w pamięci logarytmicznej. Aby udowodnić jego NL-zupełność wystarczy więc pokazać, że jest on NL-trudny, a więc dla dowolnego języka $A \in$ NL istnieje redukcja w pamięci logarytmicznej do problemu PATH.

Niech $A \in$ NL. Język A jest więc rozstrzygalny przez niedeterministyczną maszynę Turinga M używającą pamięci logarytmicznej. Redukcja języka A

do problemu PATH polega na skonstruowaniu grafu skierowanego G reprezentującego drzewo możliwych obliczeń maszyny M na słowie wejściowym w. Jego wierzchołkami są poszczególne konfiguracje rozważanej maszyny. W grafie G istnieje krawędź $c_1 \to c_2$, jeśli z konfiguracji c_1 maszyna M może przejść bezpośrednio do konfiguracji c_2.

Niech v_S oraz v_{ACC} oznaczają wierzchołki odpowiadające konfiguracji początkowej oraz konfiguracji akceptującej maszyny M. Jeśli maszyna M akceptuje słowo w, w drzewie jej obliczeń istnieje ścieżka prowadząca z konfiguracji początkowej do konfiguracji akceptującej. Z drugiej strony, jeśli w grafie$G$ istnieje ścieżka $v_S \rightsquigarrow v_{ACC}$, istnieje ciąg konfiguracji maszyny M prowadzący z konfiguracji początkowej do konfiguracji akceptującej. Zatem problem rozstrzygnięcia problemu „$w \in A$" jest równoważny rozstrzygnięciu problemu istnienia ścieżki $v_S \rightsquigarrow v_{ACC}$ w grafie G.

Pozostaje tylko uzasadnić, że opisana powyżej redukcja jest możliwa w pamięci logarytmicznej. Opiszemy maszynę Turinga M_G generującą graf obliczeń rozważanej maszyny M w pamięci logarytmicznej. Kodowanie grafu składa się z listy wierzchołków oraz listy krawędzi. Przypomnijmy, że dla maszyn Turinga z wyróżnionymi taśmami wejściową oraz wyjściową, konfiguracja opisana jest za pomocą zawartości taśm roboczych oraz pozycji jej głowic na wszystkich taśmach. Ponieważ maszyna M działa w pamięci rozmiaru $O(\log n)$, każda jej konfiguracja może być opisana za pomocą słowa rozmiaru $c \log n$. Maszyna M_G musi więc przejrzeć wszystkie słowa rozmiaru $c \log n$ i umieścić na taśmie wynikowej wyłącznie poprawne konfiguracje maszyny M. W podobny sposób maszyna M_G generuje zbiór krawędzi grafu G. Aby rozstrzygnąć istnienie krawędzi $c_1 \to c_2$ w grafie G, maszyna M_G sprawdza, czy zgodnie z funkcją przejścia maszyny M możliwe jest bezpośrednie przejście do konfiguracji c_2. ■

Problem P i NP

Przypomnijmy, że P jest klasą języków rozstrzygalnych w czasie wielomianowym przez *deterministyczne* maszyny Turinga, natomiast NP jest klasą języków rozstrzyganych w czasie wielomianowym przez *niedeterministyczne* maszyny Turinga. Rozwiązanie problemów z klasy NP na maszynach deterministycznych zazwyczaj powoduje wykładniczy wzrost czasu obliczeń. Problemy należące do klasy NP mają jednak bardzo ważną własność nazywaną własnością wielomianowej weryfikacji. Jeśli znane jest rozwiązanie takiego problemu, jego poprawność może być zweryfikowana na deterministycznej maszynie Turinga w czasie wielomianowym.

Dla przykładu przypomnijmy problem polegający na rozstrzygnięciu czy w grafie skierowanym G istnieje ścieżka Hamiltona łącząca dwa ustalone wierzchołki v_1 oraz v_2.

$$\text{HAMPATH} = \Big\{(G, v_1, v_2) : G \text{ zawiera ścieżkę Hamiltona } v_1 \rightsquigarrow v_2\Big\}.$$

Na maszynie niedeterministycznej problem ten może być rozwiązany w czasie wielomianowym, natomiast maszyna deterministyczna będzie potrzebowała czasu wykładniczego. Jeśli jednak dostaniemy opis ścieżki $v_1 \rightsquigarrow v_2$ (wyznaczonej w dowolny sposób), maszyna deterministyczna będzie w stanie zweryfikować w czasie wielomianowym czy jest ona ścieżką Hamiltona.

Zauważmy jednak, że własność wielomianowej weryfikacji nie jest cechą wszystkich problemów rozstrzygalnych w czasie wykładniczym za pomocą deterministycznych maszyn Turinga. Rozważmy dopełnienie problemu HAMPATH:

$$\overline{\text{HAMPATH}} = \Big\{(G, v_1, v_2) : G \text{ nie zawiera ścieżki Hamiltona } v_1 \rightsquigarrow v_2\Big\}$$

Znając rozwiązanie tego problemu (wyznaczone w czasie wykładniczym), aby zweryfikować jego poprawność nadal musimy wykonać wykładniczą liczbę kroków.

Każda deterministyczna maszyna Turinga jest szczególnym przypadkiem maszyny niedeterministycznej, zatem zachodzi oczywista inkluzja $\text{P} \subseteq \text{NP}$. Nie wiemy jednak czy jest ona właściwa (tzn. czy klasy P oraz NP są istotnie różne). Dlatego też wśród problemów należących do klasy NP na szczególną uwagę zasługują problemy NP-zupełne. Ich czasowa złożoność obliczeniowa jest ściśle związana ze złożonością całej klasy NP.

Każdy problem z klasy NP (również NP-zupełny) może być w czasie wielomianowym zredukowany do dowolnego problemu NP-zupełnego. Zatem, z rozwiązywalności w deterministycznym czasie wielomianowym dowolnego problemu NP-zupełnego wynika rozwiązywalność w deterministycznym czasie wielomianowym dla dowolnego problemu z klasy NP. Podobnie, aby udowodnić, że żadnego problemu z klasy NP nie da się rozwiązać w deterministycznym czasie wielomianowym, wystarczy wykazać ten fakt dla dowolnie wybranego problemu NP-zupełnego.

W klasie NP znajduje się wiele problemów o znaczeniu praktycznym. Przykładem może być problem rozkładu liczby na czynniki pierwsze, Jego trudność obliczeniowa jest podstawą powszechnie używanego algorytmu szyfrującego RSA. Dlatego też problem rozstrzygnięcia równości $\text{P} = \text{NP}$ jest jednym z najważniejszych problemów współczesnej informatyki teoretycznej. Za jego

poprawne rozwiązanie ufundowano nawet nagrodę w wysokości miliona dolarów. Jednak mimo ogromnego wysiłku wielu badaczy status tego problemu nadal pozostaje nieznany. Z kilkoma przykładami problemów NP-zupełnych zapoznamy się w czasie kolejnych wykładów.

Relacje między klasami złożoności

Klasa złożoności obejmuje grupę problemów o podobnej trudności obliczeniowej (czasowej lub pamięciowej). Wiemy już, że klasy złożoności tworzą hierarchię: niektóre z nich pokrywają się ze sobą, niektóre są istotnie różne, dla jeszcze innych wzajemna relacja nie jest jeszcze znana. Intuicyjnie oczekujemy, że klasy z bardziej ograniczonymi zasobami będą zawierały mniej języków niż klasy z mniejszymi ograniczeniami. Poniżej spróbujemy podsumować znane fakty dotyczące relacji między klasami złożoności obliczeniowej.

Przypadki ogólne:

- $\mathrm{DTIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n))$ oraz $\mathrm{NTIME}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$
  Dowolna maszyna Turinga (deterministyczna lub niedeterministyczna) działająca w czasie $f(n)$ będzie w stanie wykorzystać co najwyżej $f(n)$ komórek pamięci.

- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{DTIME}(c^{f(n)})$, dla pewnego $c \in \mathbb{N}$
  Każda ścieżka obliczeń maszyny niedeterministycznej M_1 działającej w czasie $f(n)$ ma długość co najwyżej $f(n)$. Jeśli $c \in \mathbb{N}$ oznacza maksymalną liczbę niedeterministycznych wyborów w każdym kroku obliczeń maszyny M_1, to symulująca jej obliczenia maszyna deterministyczna wykona co najwyżej $c^{f(n)}$ kroków.

- $\mathrm{DSPACE}(f(n)) \subseteq \mathrm{DTIME}(c^{f(n)})$, dla pewnego $c \in \mathbb{N}$
  Maszyna Turinga, która dla słowa wejściowego długości n wykorzystuje $f(n)$ komórek taśmy, może mieć co najwyżej $c^{f(n)}$ różnych konfiguracji, gdzie $c \in \mathbb{N}$ zależy od słowa wejściowego oraz alfabetu taśmy. Ponieważ rozważana maszyna zatrzymuje się dla każdego wejścia, żadna jej konfiguracja nie może się powtórzyć.

- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{DSPACE}(f^2(n))$
  Wynika bezpośrednio z twierdzenia Savitcha.

Przypadki szczególne:

- $\mathrm{P} \subsetneq \mathrm{EXPTIME}$

  Każdy problem rozstrzygalny w czasie wielomianowym jest w oczywisty sposób rozstrzygalny również w czasie wykładniczym. Z drugiej strony, potrafimy wskazać problemy rozstrzygalne w czasie wykładniczym, których nie da się rozstrzygnąć w czasie wielomianowym. Przykładem takiego problemu może być $\mathrm{HALT}_{\mathrm{K}}$ – rozstrzygalność zatrzymywania się maszyny Turinga na dowolnych danych w co najwyżej k krokach. A zatem inkluzja ta jest właściwa.

  Podobnie możemy uzasadnić, że $\mathrm{NP} \subsetneq \mathrm{NEXPTIME}$.

- $\mathrm{PSPACE} = \mathrm{NPSPACE}$

  Wynika bezpośrednio z twierdzenia Savitcha.

- $\mathrm{P} \subseteq \mathrm{PSPACE}$ oraz $\mathrm{NP} \subseteq \mathrm{NPSPACE}$

  Dowolna maszyna Turinga działająca w czasie wielomianowym (deterministycznym lub niedeterministycznym) może wykorzystać co najwyżej wielomianową liczbę komórek taśmy. Zatem, na mocy twierdzenia Savitcha, $\mathrm{NP} \subseteq \mathrm{PSPACE}$.

- $\mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$

  Zauważmy, że dla $f(n) \geqslant n$, maszyna Turinga działająca w pamięci $f(n)$ może mieć co najwyżej $f(n) \cdot 2^{O(f(n))}$ różnych konfiguracji. Ponadto, jeśli maszyna ta zatrzymuje się dla każdego wejścia, żadna konfiguracja nie może wystąpić dwukrotnie.

- $\mathrm{L} \subseteq \mathrm{NL}$

  Dowolny problem rozstrzygalny w pamięci logarytmicznej przez maszynę deterministyczną jest również rozstrzygalny w pamięci logarytmicznej przez maszynę niedeterministyczną. Nie wiemy natomiast, czy inkluzja ta jest właściwa. Aby rozstrzygnąć ten problem trzeba rozstrzygnąć pozytywnie lub negatywnie problem należenia do klasy L dla dowolnego problemu NL-zupełnego.

- $\mathrm{P} \subseteq \mathrm{NP}$

  Dowolny problem rozstrzygalny w czasie wielomianowym przez maszynę deterministyczną jest również rozstrzygalny w czasie wielomianowym przez maszynę niedeterministyczną. Nie wiemy natomiast, czy inkluzja ta jest właściwa. Aby rozstrzygnąć ten problem trzeba rozstrzygnąć pozytywnie lub negatywnie problem należenia do klasy P dla dowolnego problemu NP-zupełnego.

- NL $\subseteq$ P

  Niedeterministyczna maszyna Turinga rozstrzygająca dowolny problem w pamięci logarytmicznej może mieć co najwyżej wielomianową liczbę konfiguracji. Ponieważ rozważana maszyna musi zatrzymywać się dla każdych danych wejściowych, żadna konfiguracja nie może się powtarzać. Zatem działa ona w czasie wielomianowym.

Na podstawie powyższych rozważań wiemy, że zachodzą następujące inkluzje

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{NEXPTIME}.$$

Wiemy jednak również, że klasy P oraz EXPTIME są istotnie różne. Zatem co najmniej jedna z inkluzji P $\subseteq$ NP lub NP $\subseteq$ EXPTIME jest właściwa (być może obie). Gdyby udało się udowodnić równość P = NP, oznaczałoby to również równość EXPTIME = NEXPTIME.