9 Czasowa złożoność obliczeniowa

Przy założeniu nieograniczoności dostępnych zasobów (czas oraz pamięć) wszystkie poznane do tej pory modele obliczeń są równoważne. Jednak w zastosowaniach praktycznych komputery dysponują pamięcią ograniczonych rozmiarów. Również akceptowalny czas działania algorytmu jest ograniczony.

Jeśli problem jest rozstrzygalny, możliwe jest jego rozwiązanie w sensie obliczeniowym. Może ono jednak wymagać niezwykle długiego czasu lub niezwykle dużego rozmiaru pamięci czyniąc problem nierozwiązywalnym w praktyce. W trakcie wykładu zapoznamy się z metodami analizy czasu potrzebnego do rozwiązania określonych problemów obliczeniowych. Pokażemy również jak klasyfikować problemy obliczeniowe w zależności od czasu wymaganego do ich rozwiązania. Ponieważ czas działania algorytmu zależy od wielu parametrów, dla uproszczenia będziemy wyznaczać go jako funkcję długości danych wejściowych.

Definicja 9.1

Niech M będzie *rozstrzygającą* deterministyczną maszyną Turinga. Złożonością czasową (czasem działania) maszyny M nazywamy funkcję $f : \mathbb{N} \to \mathbb{N}$, gdzie $f(n)$ jest równa *największej* liczbie kroków wykonywanych przez maszynę M dla dowolnego słowa długości n.

Zauważmy, że badanie maksymalnej liczby kroków obliczeń maszyny Turinga ma sens wyłącznie dla maszyn, które zatrzymują się dla każdych danych wejściowych. W analizie złożoności obliczeniowej algorytmów zazwyczaj nie rozpatrujemy dokładnego czasu działania algorytmu lecz pewne jego przybliżenie. *Analiza asymptotyczna* polega na zrozumieniu, jak działa algorytm na bardzo dużych danych wejściowych. W tym celu rozpatrujemy najbardziej znaczące składniki w wyrażeniu opisującym czas działania algorytmu.

W szczególności pomijamy stałe oraz składniki mniej znaczące.

Definicja 9.2

Niech $f, g : \mathbb{N} \longrightarrow \mathbb{R}^+$. Powiemy, że $f(n) = O\Big(g(n)\Big)$ jeśli istnieją dodatnie stałe c oraz n_0 takie, że

$$\forall_{n \geqslant n_0} f(n) \leqslant c \cdot g(n).$$

Funkcję g nazywamy *asymptotycznym ograniczeniem górnym* funkcji f.

Przykład 9.1

- $f(n) = 2n^2 + 2n - 1$ jest $O(n^2)$ ale nie jest $O(n)$
- $f(n) = 2n - 1$ jest $O(n^2)$, $O(n)$ ale nie $O(\log n)$
- $f(n) = \log_2 n$ jest $O(n)$, $O(\log n)$
- $f(n) = 4$ jest $O(1)$

Uwaga

Z twierdzenia o zmianie podstawy logarytmu mamy

$$\log_a x = \frac{\log_b x}{\log_b a}.$$

Ponieważ w notacji asymptotycznej pomijamy stałe współczynniki $\left(\frac{1}{\log_b a}\right)$, w przypadku złożoności logarytmicznej $O(\log n)$ nie ma potrzeby określania podstawy logarytmu.

Notacja $f = O(g)$ oznacza, że funkcja f jest asymptotycznie *nie większa* od funkcji g. Aby określić, że funkcja f jest asymptotycznie *mniejsza* od funkcji g, używamy notacji o.

Definicja 9.3

Niech $f, g : \mathbb{N} \longrightarrow \mathbb{R}^+$. Powiemy, że $f(n) = o\Big(g(n)\Big)$ jeśli

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

Innymi słowy, dla każdej liczby rzeczywistej $c > 0$ istnieje taka liczba naturalna n_0, że

$$\forall_{n \geqslant n_0} f(n) \leqslant c \cdot g(n).$$

Przykład 9.2

- $f(n) = n^2$ jest $o(n^3)$
- $f(n) = \sqrt{n}$ jest $o(n)$
- $f(n) = 3n$ nie jest $o(n)$

Spróbujmy teraz zbadać złożoność maszyny Turinga rozstrzygającej przykładowy problem obliczeniowy.

Problem

Rozważmy język $L = \{a^k b^k : \ k \geqslant 0\}$. Nie jest trudno skonstruować maszynę Turinga M zatrzymującą się dla każdego słowa $w \in \{a, b\}^*$ i rozstrzygającą problem „$w \in L$". Zastanówmy się więc, ile operacji musi wykonać maszyna M aby dać prawidłową odpowiedź.

Rozwiązanie 1

Rozważmy jednotaśmową maszynę Turinga M_1 działającą według następującego schematu:

1. Przeczytaj dane na taśmie i odrzuć jeśli nie są postaci $a \ldots ab \ldots b$
2. Dopóki na taśmie znajdują się znaki a i b, usuwaj naprzemiennie a z początku oraz b z końca słowa.
3. Zaakceptuj jeśli taśma jest pusta
4. Odrzuć jeśli na taśmie zostały wyłącznie litery a lub wyłącznie litery b

Policzmy ile operacji wykonuje maszyna M_1 dla słowa długości n:

- Pierwsze przeczytanie taśmy: $n+1$ kroków
- Usunięcie skrajnych liter słowa długości n: $n+2$ kroki
- Usuwanie skrajnych liter wykonywane $\frac{n}{2}$ razy

Otrzymujemy zatem złożoność $O(n) + O(n^2) = O(n^2)$.

Rozwiązanie 2

Rozważmy jednotaśmową maszynę Turinga M_2 działającą według następującego schematu:

1. Przeczytaj dane na taśmie i odrzuć jeśli nie są postaci $a \dots ab \dots b$
2. Dopóki taśma nie jest pusta:
   - Przeczytaj dane na taśmie sprawdzając czy liczba liter jest parzysta - odrzuć jeśli jest nieparzysta
   - Usuń z taśmy co drugie wystąpienie a oraz co drugie wystąpienie b
3. Zaakceptuj jeśli taśma jest pusta

Zanim oszacujemy złożoność obliczeniową maszyny M_2 musimy uzasadnić poprawność jej działania. Przy każdym przejściu przez taśmę liczba wystąpień a oraz liczba wystąpień b zmniejsza się o połowę (pomijamy w tym miejscu resztę z dzielenia). Sprawdzanie czy liczba wszystkich liter na taśmie jest parzysta jest równoważne sprawdzeniu czy liczba wystąpień a oraz liczba wystąpień b mają tę samą parzystość. Jeśli przy każdym dzieleniu przez 2 liczba wystąpień a oraz liczba wystąpień b mają tę samą parzystość, to są one równe.

Oszacujmy teraz czas działania maszyny M_2 dla słowa długości n:

- Pierwsze przeczytanie taśmy: $O(n)$ kroków
- Przeczytanie taśmy ze sprawdzeniem parzystości liczby liter: $O(n)$
- Przeglądanie taśmy z usuwaniem co drugiej litery: $O(n)$
- W każdym kroku usuwamy co najmniej połowę liter – maksymalnie $\log_2 n$ powtórzeń

Otrzymujemy zatem złożoność $O(n) + O(n \log n) = O(n \log n)$.

Rozwiązanie 3

Rozważmy dwutaśmową maszynę Turinga M_3 działającą według następującego schematu:

1. Czytaj słowo na pierwszej taśmie kopiując litery a na drugą taśmę (do napotkania pierwszego b).
2. Kontynuuj czytanie pierwszej taśmy dla każdego b usuwając kolejne a na drugiej taśmie.
3. Zaakceptuj jeśli po osiągnięciu prawego końca słowa wszystkie a z pierwszej taśmy zostały usunięte.
4. Odrzuć jeśli:
   - po osiągnięciu końca słowa na pierwszej taśmie na drugiej taśmie znajdują się jeszcze litery a
   - ostatnie a zostało usunięte z drugiej taśmy przed osiągnięciem końca słowa na pierwszej taśmie
   - za ostatnim b znajduje się a

Maszyna M_3 przegląda taśmę wejściową dokładnie raz i kończy działanie w stanie akceptującym lub odrzucającym. Zatem całkowity czas jej działania jest rzędu $O(n)$.

Zauważmy, że chociaż w teorii obliczalności z tezy Churcha-Turinga wynika, że wszystkie sensowne modele obliczalności są sobie równoważne, w teorii złożoności obliczeniowej wybór modelu obliczeń ma istotny wpływ na złożoność języka. Skonstruowaliśmy trzy maszyny Turinga rozwiązujące problem „$w \in L$” znacząco różniące się czasem działania. Okazuje się jednak, że zmiana liczby taśm maszyny Turinga powoduje co najwyżej wielomianową zmianę złożoności obliczeniowej.

Twierdzenie 9.1

Niech $f(n)$ będzie funkcją spełniającą warunek $f(n) \geqslant n$. Wówczas dla dowolnej k-taśmowej maszyny Turinga M_1 o złożoności czasowej $O(f(n))$ istnieje *równoważna jej* jednotaśmowa maszyna M_2 działająca w czasie $O(f^2(n))$.

Dowód

Przypomnijmy, że obliczenia k-taśmowej maszyny Turniga M_1 mogą być symulowane na maszynie M_2 z jedną taśmą (patrz Twierdzenie 5.3 Wykład 5). Zawartość k taśm maszyny M_1 zapisujemy na taśmie maszyny M_2 w postaci taśm wirtualnych, natomiast położenie głowic czytająco/piszących M_1 kodujemy za pomocą wyróżnionych symboli dodanych do alfabetu. Maszyna M_2 zmienia zawartość taśm wirtualnych oraz uaktualnia pozycje głowic wirtualnych zgodnie ze schematem działania maszyny M_1.

Załóżmy, że dane wejściowe mają rozmiar n, zaś maszyna M_1 ma złożoność obliczeniową $O(f(n))$. Oszacujmy liczbę operacji wykonywanych przez maszynę M_2.

- Symulacja pojedynczego kroku maszyny M_1 polega dwukrotnym przejrzeniu taśmy oraz co najwyżej k-krotnym skopiowaniu jej zawartości.
- Maszyna M_1 działając w czasie $O(f(n))$ może zapisać co najwyżej $O(f(n))$ różnych komórek na każdej taśmie, zatem symulacja pojedynczego kroku M_1 wykonywana jest w czasie $O(f(n))$.
- Maszyna M_2 musi zasymulować wykonanie $O(f(n))$ kroków obliczeń maszyny M_1.

Podsumowując, złożoność obliczeniowa maszyny M_2 wynosi $O(f^2(n))$. ■

Ponieważ obliczenia k-taśmowej maszyny Turinga możemy symulować na maszynie jednotaśmowej w czasie jedynie wielomianowo większym, modele te nazywamy *wielomianowo równoważnymi*. Podobna własność zachodzi dla dowolnych sensownych *deterministycznych* modeli obliczeń. Z tego powodu najwygodniejszym modelem używanym do badania złożoności obliczeniowej jest $(k+2)$-taśmowa maszyna Turinga zbudowana według schematu:

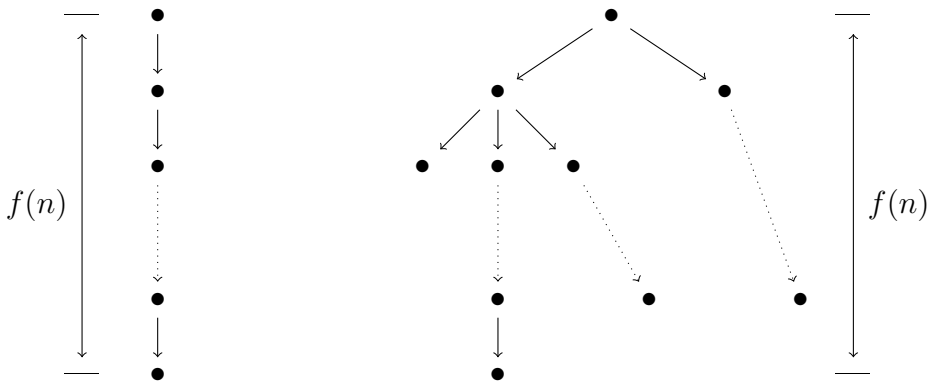
- pierwsza taśma zawiera dane wejściowe (tylko do odczytu)
- ostatnia taśma zawiera wynik (tylko do zapisu)
- k taśm roboczych wykorzystywanych do obliczeń

Ćwiczenie 9.1

Przypomnij dowód równoważności maszyn Turinga oraz maszyn licznikowych. Uzasadnij, że modele te są wielomianowo równoważne.

Niedeterministyczne modele obliczeń

Każdy krok obliczeń deterministycznej maszyny Turinga jest jednoznacznie wyznaczony przez jej aktualną konfigurację. Maszyna niedeterministyczna dopuszcza kilka możliwości wyboru kolejnego kroku. Obliczenia tworzą więc drzewo, którego gałęzie odpowiadają różnym sposobom obliczeń. Definicja złożoności obliczeniowej maszyny niedeterministycznej musi ten fakt uwzględniać.



Definicja 9.4

Niech M będzie niedeterministyczną maszyną Turinga rozstrzygającą pewien język. Złożonością czasową (czasem działania) maszyny M nazywamy funkcję $f : \mathbb{N} \longrightarrow \mathbb{N}$, gdzie $f(n)$ jest równa *największej* liczbie kroków wykonywanych przez maszynę M na *dowolnej* ścieżce obliczeń dla *dowolnego* słowa długości n.

Przypomnijmy, że obliczenia niedeterministycznej maszyny Turinga mogą być symulowane przez maszynę deterministyczną. Istotnym pytaniem jest złożoność obliczeniowa takiej symulacji.

Twierdzenie 9.2

Niech $f(n)$ będzie funkcją spełniającą warunek $f(n) \geqslant n$. Wówczas dla każdej jednotaśmowej niedeterministycznej maszyny Turinga M_1 o złożoności czasowej $O(f(n))$ istnieje równoważna jej jednotaśmowa deterministyczna maszyna Turinga M_2 działająca w czasie $2^{O(f(n))}$.

Dowód

Symulacja działania maszyny M_1 polega na przeglądaniu (metodą wszerz) jej drzewa obliczeń. Dla słowa wejściowego długości n każda niedeterministyczna ścieżka obliczeń maszyny M_1 ma długość co najwyżej $O(f(n))$, zatem dojście od korzenia do dowolnego węzła wymaga maksymalnie czasu rzędu $O(f(n))$. Ponadto, każdy węzeł drzewa obliczeń M_1 ma co najwyżej b potomków, gdzie b jest maksymalną liczbą wyborów dopuszczanych przez funkcję przejścia M_1. Maksymalna liczba węzłów w drzewie obliczeń M_1 (nie większa niż podwojona liczba liści) jest rzędu $O(b^{f(n)})$. Maszyna M_2 przeglądając wszerz drzewo obliczeń maszyny M_1 wykonuje więc co najwyżej $O(f(n)) \cdot O(b^{f(n)}) = 2^{O(f(n))}$ operacji. ■

Uwaga

W dowodzie Twierdzenia 5.4 (Wykład 5) do symulacji obliczeń niedeterministycznej maszyny Turinga wykorzystaliśmy trzytaśmową maszynę deterministyczną. Na mocy Twierdzenia 9.1 użycie maszyny jednotaśmowej zwiększy czas jej działania co najwyżej kwadratowo. Nie zmieni to jednak wykładniczej złożoności symulacji maszyny niedeterministycznej.

Klasy czasowej złożoności obliczeniowej

Badając problemy obliczeniowe wygodnie jest klasyfikować je pod względem trudności obliczeniowej. Zbiór problemów o podobnej złożoności nazywamy *klasą złożoności obliczeniowej*. W definicji klasy złożoności konieczne jest określenie trybu obliczeń (deterministyczny lub niedeterministyczny) oraz funkcji złożoności ograniczającą liczbę możliwych kroków obliczeń.

Definicja 9.5

Klasę złożoności $DTIME(f(n))$ definiujemy jako zbiór wszystkich języków (problemów), które są rozstrzygane w czasie $O(f(n))$ przez pewną deterministyczną maszynę Turinga.

Podobnie określamy klasy złożoności dla modeli niedeterministycznych.

Definicja 9.6

Klasę złożoności $NTIME(f(n))$ definiujemy jako zbiór wszystkich języków (problemów), które są rozstrzygane w czasie $O(f(n))$ przez pewną niedeterministyczną maszynę Turinga.

Zdefiniujemy teraz kilka najważniejszych klas złożoności obliczeniowej oraz podamy przykłady problemów należących do każdej z nich.

Klasa PTIME

Definicja 9.7

$PTIME$ (*PolynomialTIME*, w skrócie P) jest klasą języków rozstrzygalnych w czasie wielomianowym przez deterministyczne maszyny Turinga:

$$P = \bigcup_k DTIME(n^k)$$

Klasa P określa mniej więcej zakres problemów, które mogą być rozwiązane (w sensownym czasie) na rzeczywistych komputerach. Zauważmy ponadto, że definicja klasy P nie zależy od wyboru modelu obliczeniowego, jeśli tylko jest on wielomianowo równoważny deterministycznej maszynie Turinga.

Przykład 9.3

Rozważmy problem znalezienia w grafie skierowanym G ścieżki łączącej dwa ustalone wierzchołki v_1 i v_2. Problem ten możemy rozwiązać za pomocą algorytmów przeglądania grafu wszerz (BFS) oraz wgłąb (DFS) działających w czasie liniowo (a więc wielomianowo) zależnym od sumarycznej liczby wierzchołków oraz krawędzi grafu G.

Klasa EXPTIME

Definicja 9.8

$EXPTIME$ jest klasą języków rozstrzygalnych w czasie wykładniczym przez deterministyczne maszyny Turinga:

$$EXPTIME = \bigcup_k DTIME(2^{n^k})$$

Przykład 9.4

Przypomnijmy, że problem stopu dla maszyn Turinga

$$HALT = \Big\{(M, w) : \ M \text{ zatrzymuje się na } w\Big\}$$

jest nierozstrzygalny. Rozważmy zatem jego zmodyfikowaną wersję:

$$HALT_K = \Big\{(M,k) : \forall_w \ M \text{ zatrzymuje się po co najwyżej } k \text{ krokach}\Big\}$$

Zauważmy, że jeśli maszyna M zatrzymuje się po co najwyżej k krokach, może mieć na to wpływ wyłącznie zawartość pierwszych k komórek taśmy. Aby rozstrzygnąć problem $HALT_K$ wystarczy zasymulować działanie maszyny M na każdym możliwym wejściu długości k.

Klasa $EXPTIME$ zawiera problemy trudne obliczeniowo. Istnieją jednak problemy jeszcze trudniejsze o ponadwykładniczej złożoności obliczeniowej. Przykładem takiego problemu jest algorytm obliczający wartość funkcji Ackermana.

Klasa NPTIME

Definicja 9.9

$NPTIME$ (*NondeterministicPolynomialTIME*, w skrócie NP) jest klasą języków rozstrzygalnych w czasie wielomianowym przez niedeterministyczne maszyny Turinga:

$$NP = \bigcup_k NTIME(n^k)$$

Przykład 9.5

Ścieżką Hamiltona w grafie skierowanym G nazywamy ścieżkę skierowaną przechodzącą przez każdy jego wierzchołek dokładnie raz. Rozważmy problem rozstrzygnięcia czy w grafie skierowanym G istnieje ścieżka Hamiltona łącząca dwa ustalone wierzchołki v_1 oraz v_2.

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

Zauważmy, że nie jest trudno znaleźć deterministyczny algorytm rozwiązujący ten problem działający w czasie wykładniczym. Z drugiej strony, gdyby obliczenia wykonywała maszyna niedeterministyczna i w każdym kroku dokonywała wyboru właściwej krawędzi, rozwiązałaby problem w czasie wielomianowym. Zatem problem $HAMPATH$ ma niedeterministyczną złożoność wielomianową.

Problem $HAMPATH$ ma jeszcze jedną bardzo istotną własność. Jeśli znamy opis ścieżki $v_1 \rightsquigarrow v_2$ w grafie skierowanym G, weryfikacja czy jest ona ścieżką Hamiltona jest możliwa w czasie wielomianowym przez deterministyczną maszynę Turinga. Co ciekawe, własność tę posiadają wszystkie problemy z klasy NP. Możemy zatem sformułować alternatywną definicję tej klasy:

Definicja 9.10

NP ($NPTIME$) jest klasą języków posiadających wielomianowe algorytmy weryfikujące.

Uwaga

Zauważmy, że każdy język rozstrzygalny w czasie wielomianowym przez deterministyczną maszynę Turinga, jest również rozstrzygalny w czasie wielomianowym przez maszynę niedeterministyczną. Otrzymujemy zatem inkluzję $P \subseteq NP$. Odpowiedź na pytanie, czy jest to inkluzja właściwa jest jednym z najważniejszych problemów informatyki teoretycznej. Bardziej szczegółową analizą tego zagadnienia zajmiemy się w czasie kolejnych wykładów.