10

Pamięciowa złożoność obliczeniowa

Najważniejszymi parametrami branymi pod uwagę podczas analizy wydajności algorytmu są czas jego działania oraz rozmiar wykorzystywanej przez niego pamięci. W czasie poprzedniego wykładu poznaliśmy metody klasyfikacji problemów pod względem ich złożoności czasowej. Bieżący wykład poświęcony będzie badaniu złożoności pamięciowej.

Przypomnijmy, że jako modelu do badania złożoności czasowej używaliśmy $(k+2)$-taśmowej maszyny Turinga zbudowanej według schematu:

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

Ten sam model wykorzystamy do badania rozmiaru pamięci używanej w trakcie obliczeń. Podobnie jak w przypadku złożoności czasowej, konieczne jest rozdzielenie definicji złożoności pamięciowej maszyn deterministycznych oraz niedeterministycznych.

Definicja 10.1 (Złożoność deterministyczna)

Niech M będzie deterministyczną maszyną Turinga zatrzymującą się dla każdych danych wejściowych. *Złożonością pamięciową* maszyny M nazywamy funkcję $f : \mathbb{N} \longrightarrow \mathbb{N}$, gdzie $f(n)$ jest maksymalną liczbą komórek taśm roboczych wykorzystywanych (odczyt/zapis) przez maszynę M uruchomioną na dowolnym słowie długości n.

Definicja 10.2 (Złożoność niedeterministyczna)

Niech M będzie niedeterministyczna maszyna Turinga zatrzymującą się na wszystkich ścieżkach obliczeń. *Złożonością pamięciową* maszyny M nazywamy funkcję $f : \mathbb{N} \longrightarrow \mathbb{N}$, gdzie $f(n)$ jest maksymalną liczbą komórek taśmy wykorzystywanych (odczyt/zapis) przez maszynę M na dowolnej ścieżce obliczeń po uruchomieniu na dowolnym słowie długości n.

Pamięciową złożoność obliczeniową, podobnie jak złożoność czasową, określamy jako funkcję rozmiaru danych wejściowych. Zazwyczaj jesteśmy zainteresowani efektywnością działania algorytmu dla danych dużego rozmiaru. Przeprowadzamy więc analizę asymptotyczną i zapisujemy złożoność za pomocą notacji O.

Problemy obliczeniowe wygodnie jest klasyfikować pod względem zbliżonego rozmiaru wykorzystywanej pamięci. Definiując klasę złożoności pamięciowej konieczne jest określenie trybu obliczeń (deterministyczny lub niedeterministyczny) oraz funkcji złożoności ograniczającej rozmiar pamięci używanej w czasie obliczeń.

Definicja 10.3

Niech $f : \mathbb{N} \to \mathbb{R}^+$. Klasy złożoności pamięciowej definiujemy następująco:

$$\text{SPACE}(f(n)) = \Big\{\text{L: L – język rozstrzygany przez } \textit{deterministyczną} \text{ maszynę Turinga w pamięci } O(f(n))\Big\}$$

$$\text{NSPACE}(f(n)) = \Big\{\text{L: L – język rozstrzygany przez } \textit{niedeterministyczną} \text{ maszynę Turinga w pamięci } O(f(n))\Big\}$$

Przeanalizujemy teraz złożoność pamięciową przykładowego problemu obliczeniowego.

Przykład 10.1

Formułą logiczną nazywamy wyrażenie składające się ze zmiennych oraz operacji logicznych (alternatywa, koniunkcja oraz negacja), np.

$$\phi \;=\; (\neg x \vee y) \wedge (z \vee \neg y).$$

Powiemy, że formuła logiczna ϕ jest *spełnialna* jeśli istnieje takie wartościowanie zmiennych zawartych w ϕ, dla którego cała formuła jest prawdziwa.

Problem spełnialności polega na sprawdzeniu czy dana formuła logiczna jest spełnialna.

$$SAT = \{\phi : \phi \text{ jest spełnialna}\}.$$

Załóżmy, że formuła ϕ ma rozmiar n i zawiera m różnych zmiennych logicznych $x_1, \ldots, x_m$ ($m \leqslant n$). Maszyna Turinga rozstrzygająca problem SAT może działać według następującego schematu

- Generuj kolejno wszystkie wartościowania zmiennych $x_1, \ldots, x_m$
- Dla każdego wartościowania oblicz wartość formuły ϕ
- Zaakceptuj, jeśli ϕ ma wartość $TRUE$
- Odrzuć, jeśli dla żadnego wartościowania ϕ nie miała wartości $TRUE$

Zauważmy, że dla formuły zawierającej m różnych zmiennych logicznych konieczne może być sprawdzenie 2^m możliwych wartościowań. Maszyna deterministyczna rozstrzygająca problem SAT będzie więc działała w czasie $O(2^n)$ (gdzie n jest rozmiarem formuły). Z drugiej strony, do obliczenia wartości formuły ϕ maszyna M musi zapamiętać wyłącznie aktualne wartościowanie zmiennych. W każdej iteracji pętli może więc wykorzystać ten sam obszar taśmy. Liczba zmiennych jest ograniczona z góry przez długość formuły n. Maszyna M będzie więc działa w pamięci $O(n)$, czyli $SAT \in SPACE(n)$.

Obserwacja

Zauważmy, że w tym samym czasie maszyna Turinga może wykonać dokładnie jeden krok obliczeń. Z drugiej strony, każda komórka taśmy może być w czasie obliczeń używana wielokrotnie (zapis i odczyt). A zatem pamięć jest zasobem silniejszym niż czas.

Twierdzenie Savitcha

Przypomnijmy, że obliczenia dowolnej niedeterministycznej maszyny Turinga mogą być symulowane na maszynie deterministycznej. Symulacja taka powoduje jednak wykładniczy wzrost czasu obliczeń. Okazuje się jednak, że maszyna deterministyczna potrzebuje do jej przeprowadzenia zaskakująco mało pamięci.

Twierdzenie 10.1 (Savitch (1970))

Dla dowolnej funkcji $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ spełniającej warunek $f(n) \geqslant log(n)$ zachodzi inkluzja:

$$NSPACE\Big(f(n)\Big) \subseteq SPACE\Big(f^2(n)\Big).$$

Dowód

Niech M_1 będzie niedeterministyczną maszyną Turinga rozstrzygającą język A działającą w pamięci $f(n)$. Skonstruujemy deterministyczną maszynę Turinga M_2 rozstrzygającą język A. Bez straty ogólności możemy założyć, że maszyna M_1 ma dokładnie jedną konfigurację akceptującą c_{ACC} .

Zauważmy, że niedeterministyczna ścieżka w drzewie obliczeń maszyny M_1 wykorzystująca pamięć rozmiaru $f(n)$ może składać się z $2^{O(f(n))}$ kroków. Naiwne sekwencyjne przeglądanie ścieżki obliczeń wymagające zapisania wszystkich dokonywanych niedeterministycznych wyborów może wymagać pamięci rozmiaru wykładniczego. Do symulacji działania maszyny M_1 za pomocą M_2 wykorzystamy więc rozwiązanie problemu osiągalności – sprawdzenia czy z konfiguracji c_1 maszyny M_1 możemy dojść do konfiguracji c_2 w co najwyżej t krokach. Problem ten będzie rozwiązywany przez procedurę REACH(c_1,c_2,t). Symulacja maszyny M_1 będzie równoważna rozwiązaniu przez maszynę M_2 problemu osiągalności dla konfiguracji początkowej, konfiguracji końcowej oraz maksymalnej liczbie kroków obliczeń, które może wykonać maszyna M_1.

Procedura REACH(c_1, c_2, t) działa według następującego schematu:

- Jeśli $t = 0$, zwróć $TRUE$ jeśli $c_1 = c_2$ oraz $FALSE$ w przeciwnym przypadku.
- Jeśli $t = 1$, zwróć $TRUE$ jeśli $c_1 = c_2$ lub jeśli jest możliwe przejście w jednym kroku z konfiguracji c_1 do konfiguracji c_2 zgodnie z funkcją przejścia maszyny M_1.
- Jeśli $t > 1$, dla każdej konfiguracji c_m maszyny M_1 (wykorzystując pamięć rozmiaru $f(n)$):
  - Wykonaj REACH(c_1, c_m, $\lfloor \frac{t}{2} \rfloor$)
  - Wykonaj REACH(c_m, c_2, $\lceil \frac{t}{2} \rceil$)
  - Zwróć $TRUE$ jeśli oba powyższe wywołania zwróciły $TRUE$, oraz $FALSE$ w przeciwnym przypadku.
- Zwróć $FALSE$, jeśli do tej pory nie zostało zwrócone $TRUE$.

Niech c_0 będzie konfiguracją początkową maszyny M_1, oraz załóżmy że $2^{d \cdot f(n)}$, dla pewnego $d \in \mathbb{N}$, jest górnym ograniczeniem liczby konfiguracji maszyny M_1 (a zatem róznież długości ścieżki w drzewie obliczeń). Wówczas, działanie niedeterministycznej maszyny M_1 na słowie wejściowym w może być zasymulowane przez deterministyczną maszynę M_2 za pomocą wywołania procedury REACH(c_0, c_{ACC}, $2^{d \cdot f(n)}$).

Dla każdego wywołania procedury REACH(c_1, c_2, t) maszyna M_2 musi zapamiętać na taśmie roboczej c_1, c_2 oraz t, aby móc je odtworzyć po zakończeniu wywołania rekurencyjnego. Każde wywołanie rekurencyjne wymaga więc $O(f(n))$ dodatkowej pamięci. Ponieważ liczba konfiguracji maszyny M_1 jest ograniczona z góry przez $2^{d \cdot f(n)}$, zaś przy każdym wywołaniu rekurencyjnym dzielimy wartość t przez 2, głębokość rekursji wynosi $O(\log(2^{d \cdot f(n)})) = O(f(n))$. Zatem całkowity rozmiar pamięci używanej przez maszynę M_2 w czasie symulacji obliczeń maszyny M_1 jest rzędu $O(f^2(n))$. ■

Klasy złożoności pamięciowej

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

Klasy PSPACE oraz NPSPACE

Definicja 10.4

PSPACE jest klasą języków rozpoznawanych przez deterministyczne maszyny Turinga w pamięci wielomianowej

$$PSPACE = \bigcup_k SPACE(n^k).$$

Definicja 10.5

NPSPACE jest klasą języków rozpoznawanych przez niedeterministyczne maszyny Turinga w pamięci wielomianowej

$$NPSPACE = \bigcup_k NSPACE(n^k).$$

Przykład 10.2

W przykładzie 1 pokazaliśmy, że problem spełnialności formuł logicznych (SAT) może być rozwiązany w pamięci liniowej. A zatem: SAT $\in$ PSPACE.

Uwaga

Ponieważ wielomian podniesiony do dowolnej potęgi nadal pozostaje wielomianem, bezpośrednio z twierdzenia Savitcha wynika równość klas

$$PSPACE = NPSPACE.$$

Przykład 10.3

Formułą logiczną z kwantyfikatorami nazywamy formułę logiczną składającą się ze zmiennych, operatorów logicznych $\neg$, $\vee$ i $\wedge$, oraz kwantyfitatorów: uniwersalnego $\forall$ i egzystencjalnego $\exists$. Powiemy, że formuła ϕ nie ma zmiennych wolnych, jeśli każda zmienna jest w zasięgu pewnego kwantyfikatora. Powiemy, że formuła ϕ jest w *preneksowej postaci normalnej* jeśli wszystkie kwantyfikatory występują na początku ϕ (Dowolną formułę logiczną ϕ możemy przekształcić do równoważnej formuły ϕ' w preneksowej postaci normalnej). Przykładowo, formuła

$$\phi \;=\; \forall_x \exists_y \Big((x \vee y) \wedge (\neg x \vee \neg y) \Big)$$

jest formułą w preneksowej postaci normalnej bez zmiennych wolnych.

Rozważmy problem:

$TBQF = \{(\phi) : \phi \text{ jest prawdziwą formułą logiczną bez zmiennych wolnych}\}$.

Procedura $\mathcal{P}$ na maszynę Turinga rozstrzygająca problem $TBQF$ działa według następującego schematu:

- Jeśli formuła ϕ nie zawiera kwantyfikatorów, składa się wyłącznie ze stałych. Oblicz wartość ϕ i zaakceptuj jeśli jest ona równa TRUE oraz odrzuć w przeciwnym przypadku.
- Jeśli formuła ϕ jest postaci $\exists_x \psi$, wywołaj $\mathcal{P}$ rekurencyjnie na formule ψ podstawiając za y wartość 0 a następnie 1. Zaakceptuj, jeśli którekolwiek wywołanie $\mathcal{P}$ zwróciło wartość TRUE oraz odrzuć w przeciwnym przypadku.
- Jeśli formuła ϕ jest postaci $\forall_x \psi$, wywołaj $\mathcal{P}$ rekurencyjnie na formule ψ podstawiając za y wartość 0 a następnie 1. Zaakceptuj, jeśli oba wywołania $\mathcal{P}$ zwróciły wartość TRUE oraz odrzuć w przeciwnym przypadku.

Zauważmy, że głębokość wywołań rekurencyjnych jest ograniczona liczbą zmiennych formuły ϕ. Ponadto, dla każdego wywołania musimy zapamiętać wartość zmiennej. Rozmiar wykorzystanej pamięci nie przekroczy więc rozmiaru ϕ. A zatem $TBQF \in PSPACE$.

Klasa L (LSPACE)

Definicja 10.6

Klasa L to klasa języków rozpoznawanych przez deterministyczne maszyny Turinga w pamięci logarytmicznej:

$$L = SPACE(log(n)).$$

Zauważmy, że maszyna Turinga działająca w pamięci subliniowej może przeczytać całe słowo wejściowe, nie ma jednak miejsca, aby zapisać je na taśmie roboczej. Z drugiej strony, subliniowe ograniczenie czasu działania maszyny uniemożliwia jej przeczytanie słowa wejściowego w całości. Z tego powodu klasy o złożoności poniżej liniowej rozważa się wyłącznie dla złożoności pamięciowej. Ponieważ długość słowa wejściowego jest większa niż rozmiar dostępnej pamięci roboczej, konfigurację maszyny Turinga określamy w tym przypadku za pomocą zawartości taśm roboczych oraz pozycji jej głowic. Tak zdefiniowaną konfigurację możemy zapisać za pomocą słowa rozmiaru $c \log n$.

Przykład 10.4

Rozważmy język $A = \{a^k b^k : \ k \geqslant 0\}$. Rozstrzygająca go maszyna Turinga M liczy wystąpienia liter a oraz b zapisując reprezentacje binarne liczby wystąpień na taśmie roboczej. Ponieważ binarny zapis liczby całkowitej wymaga pamięci logarytmicznej, maszyna M działa w pamięci logarytmicznej. Wobec tego $A \in L$.

Klasa NL (NLSPACE)

Definicja 10.7

Klasa NL to klasa języków rozpoznawanych przez niedeterministyczne maszyny Turinga w pamięci logarytmicznej:

$$NL = NSPACE(\log(n)).$$

Przykład 10.5

Rozważmy problem istnienia ścieżki między dwoma wyróżnionymi wierzchołkami w grafie skierowanym.

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

Niedeterministyczna maszyna Turinga M rozstrzygająca problem PATH rozpoczyna przeszukiwanie grafu G od wierzchołka v_1. W każdym kroku wybiera w sposób niedeterministyczny kolejne wierzchołki tworzące ścieżkę $v_1 \rightsquigarrow v_2$. Maszyna M zatrzymuje się w stanie akceptującym po osiągnięciu wierzchołka v_2, lub w stanie odrzucającym po przejrzeniu wszystkich wierzchołków G osiągalnych z v_1. Jeśli maszyna M pamięta na taśmie roboczej wyłącznie numer aktualnie przetwarzanego wierzchołka, procedura ta może być zrealizowana w pamięci logarytmicznej. Zatem PATH $\in$ NL.

Klasy EXPSPACE oraz NEXPSPACE

Definicja 10.8

EXPSPACE jest klasą języków rozpoznawanych przez deterministyczne maszyny Turinga w pamięci wykładniczej

$$EXPSPACE = \bigcup_k SPACE\left(2^{n^k}\right).$$

Definicja 10.9

NEXPSPACE jest klasą języków rozpoznawanych przez niedeterministyczne maszyny Turinga w pamięci wykładniczej

$$NEXPSPACE = \bigcup_k NSPACE\left(2^{n^k}\right).$$

Przykład 10.6

Wyrażenia regularne konstruujemy za pomocą operacji konkatenacji, sumy oraz domknięcia (gwiazdki) Kleene'go. Jeśli rozszerzymy listę operacji o potęgowanie:

$$R^k = \underbrace{R \cdot R \cdot \ldots \cdot R}_{k}$$

otrzymamy *uogólnione* wyrażenia regularne. Zauważmy, że rozszerzenie to zmienia tylko zapis wyrażenia, ponieważ każde potęgowanie możemy zastąpić odpowiednią liczbą konkatenacji.

Rozważmy problem równoważności dwóch uogólnionych wyrażeń regularnych (równości języków przez nie opisywanych):

$$EQREX \uparrow = \{(R, S) : \ L(R) = L(S)\}$$

Równoważność dwóch wyrażeń regularnych możemy rozstrzygnąć w pamięci wielomianowej przez równoległą symulację odpowiadających im automatów skończonych. Aby zastosować powyższą procedurę do uogólnionych wyrażeń regularnych, musimy najpierw usunąć występujące w nich potęgi zastępując je wymaganą liczbą konkatenacji. Zauważmy jednak, że operacja ta może spowodować wykładniczy wzrost rozmiaru rozważanych wyrażeń regularnych. A zatem $EQREX \uparrow \in \ EXPSPACE$.

Uwaga

Ponieważ funkcja wykładnicza podniesiona do kwadratu nadal jest funkcją wykładniczą, na mocy twierdzenia Savitcha otrzymujemy równość:

$$EXPSPACE \ = \ NEXPSPACE.$$

Złożoność czasowa a złożoność pamięciowa

Zauważmy, że maszyna Turinga działająca w czasie $O(f(n))$ (zarówno deterministyczna jak i niedeterministyczna) wykonuje co najwyżej $O(f(n))$ kroków obliczeń. Nie może więc używać więcej niż $O(f(n))$ komórek pamięci. Możemy zatem sformułować wniosek:

$$TIME\Big(f(n)\Big) \ \subseteq \ SPACE\Big(f(n)\Big)$$

oraz

$$NTIME\Big(f(n)\Big) \ \subseteq \ NSPACE\Big(f(n)\Big).$$

Bardziej szczegółowa analiza relacji między znanymi klasami złożoności obliczeniowej (zarówno czasowej jak i pamięciowej) zostanie omówiona w czasie kolejnego wykładu.