Funkcja nieobliczalna

Programowanie to czynność dość czasochłonna, a także wymagająca pewnej wprawy i podstawowej wiedzy o technikach programowania. Stąd większość komputerów jest fabrycznie wyposażana w zestaw podstawowych programów, a wiele firm tworzy i sprzedaje nowe dodatkowe programy dla chcących poszerzyć możliwości swojego komputera bez konieczności samodzielnego programowania. Niemniej należy sobie zdawać sprawę, że za każdą aplikacją stoi program komputerowy, który ktoś kiedyś napisał w jednym ze specjalistycznych języków programowania.

Komputery to urządzenia bardzo nietypowe. Same jako takie nie mają sprecyzowanego, jasno zdefiniowanego zadania, do wykonywania którego zostały zaprojektowane. Zadania, które wykonuje komputer, mogą się w czasie zmieniać, a ich określenie odbywa się poprzez pisanie *programów komputerowych*. Owe programy formułuje się w specjalnym języku (Pascal, C/C++, Java, Python itp.), a następnie zleca komputerowi do wykonania.

Skoro definiujemy komputer jako urządzenie elastyczne (zdolne do przedefiniowania się), naturalne jest pytanie o zakres jego wszechstronności. To znaczy: jak bardzo skomplikowane problemy daje się zapisać jako odpowiedni program komputerowy? Wydawałoby się, że odpowiedź mocno zależy od typu komputera oraz wyboru języka programowania. Być może zaskakująco, okazuje się, że wcale nie. To znaczy: dane zadanie zwykle albo da się rozwiązać przy użyciu niemal dowolnego komputera i języka programowania (niezależnie, czy dysponujemy starym Atari i językiem BASIC, czy najnowszym PC-tem i Pythonem), albo nie da się po prostu wcale. Wynika to z tego, że generalnie wszystkie komputery i języki programowania realizują ten sam model, tak zwaną (teoretyczną) maszyną Turinga. Innymi słowy: różnią się efektywnością (czasem) wykonania, a nie zbiorem problemów, które *da się* przy ich użyciu rozwiązać.

Więcej o maszynie Turinga poczytać można w Δ^{11}_{13}. Przykład funkcji nieobliczalnej (problem stopu) pokazaliśmy w Δ^{4}_{17}.

Co więcej, panuje powszechne przekonanie, że same prawa fizyki sprawiają, że żadna maszyna nigdy nie będzie w stanie realizować modelu silniejszego niż model Turinga. Czyli że każde zagadnienie nierozwiązywalne przez maszynę Turinga (tytułowa *funkcja nieobliczalna*) jest po prostu zadaniem niemożliwym do rozwiązania przez żaden komputer: ani obecnie istniejący, ani żaden z przyszłości.

No, chyba że odkryjemy i ujarzmimy jakieś nowe cudowne zjawisko fizyczne. W to jednak naprawdę nikt nie wierzy, bo nawet komputer kwantowy może być symulowany (bardzo nieefektywnie) przez maszynę Turinga.

Tomasz KAZANA

Złożoność obliczeniowa

Przykładami takich problemów są: „czy dana liczba naturalna jest liczbą pierwszą?”, „czy dany graf nieskierowany, ma cykl Hamiltona?”, „czy z danych dwóch słów pierwsze jest podsłowem drugiego?”.

Jak mierzyć trudność problemów? Trudność albo, inaczej mówiąc, ich skomplikowanie, złożoność. To nie jest łatwe pytanie. Aby móc na nie chociaż nieco sensownie odpowiedzieć, skupimy się tu na tzw. *problemach decyzyjnych*, czyli takich, na które odpowiedź zawsze brzmi „tak” lub „nie”. Żeby określić, jak złożone są te problemy, przyjmuje się zasadę, że problem jest tak trudny, jak jego najlepsze rozwiązanie. Innymi słowy mówimy, że złożoność problemu jest równa złożoności najlepszego algorytmu, który go rozwiązuje.

Jak jednak określić złożoność algorytmu? Pierwszy sposób mierzenia, który przychodzi nam do głowy, a mianowicie czas działania, ma duże wady. Zależy on bardzo od tego, na jak szybkim komputerze działa algorytm i od tego, jak duże są dane wejściowe (np. czy liczba, którą badamy, ma 10 cyfr, czy też 1 000 000). Rozwiązaniem pierwszej z tych wad jest mierzenie czasu nie w sekundach, ale w liczbie operacji wykonanych przez dany algorytm – ona nie zależy już od tego, na jakim sprzęcie wykonujemy algorytm. Z drugą wadą trudniej sobie poradzić. Najbardziej popularnym rozwiązaniem jest tzw. *pesymistyczna złożoność czasowa*. Mówimy, że dany algorytm ma pesymistyczną złożoność czasową $f : \mathbb{N} \to \mathbb{N}$, jeśli maksymalna liczba operacji, którą wykonuje dla danych wejściowych o rozmiarze n, to $f(n)$. Zazwyczaj nie interesuje nas dokładna wartość f, wiedza, że przykładowo $f(n) = 3n^2 + 5n + 17$, tylko raczej to, jak f zachowuje się dla dużych danych, czyli dużych n. W naszym przypadku powiemy, że f jest proporcjonalne do n^2 dla dużych n, i zapiszemy $f(n) = \Theta(n^2)$. Często przy badaniu złożoności nie umiemy poznać dokładnego zachowania algorytmu, a jesteśmy w stanie jedynie ustalić pewne górne oszacowanie na liczbę operacji, które ten algorytm wykonuje. Wówczas używamy tak zwanej notacji O, piszemy na przykład, że $f(n) = O(n^3)$, co oznacza, że dla dużych n funkcja f jest proporcjonalna do n^3 lub od niej mniejsza.

Precyzyjnie, $g(n) = \Theta(n)$, jeśli istnieją $C, D \in \mathbb{N}$ oraz $K \in \mathbb{N}$ takie, że dla wszystkich $n \geqslant K$ zachodzi $Cn \leqslant g(n) \leqslant Dn$.

Precyzyjnie, $g(n) = O(n)$, jeśli istnieją $C \in \mathbb{N}$ oraz $K \in \mathbb{N}$ takie, że dla wszystkich $n \geqslant K$ zachodzi $g(n) \leqslant Cn$.

Dość egzotyczny, ale bardzo ciekawy sposób mierzenia trudności to *złożoność wygładzona*, o której pisaliśmy w Δ^{1}_{18}.

Wypada przy okazji wspomnieć, że pesymistyczna złożoność czasowa to najpopularniejszy sposób mierzenia złożoności programów, ale dalece nie jedyny. Dość często używa się *średniej złożoności czasowej*, która to dla $n \in \mathbb{N}$ zwraca średnią liczbę operacji, jakie algorytm wykonuje dla danych wejściowych rozmiaru n. Nieraz bada się też inne parametry algorytmu, często zamiast liczby wykonywanych operacji rozważa się rozmiar pamięci, której używa dany algorytm. Wówczas mówimy o *złożoności pamięciowej*; znów może być ona pesymistyczna, średnia albo jeszcze inna. Można też iść dużo dalej, istnieje cały dział informatyki teoretycznej, **złożoność obliczeniowa**, który zajmuje się próbą zrozumienia, jak złożone są różne problemy. Jest tam wiele fascynujących tematów i pytań, które wciąż czekają na rozwiązanie.

Wojciech CZERWIŃSKI