O algorytmie

Prof. dr Władysław M. TURSKI

Już starożytni Grecy, a przed nimi Egipcjanie, Babilończycy i inni zajmowali się praktycznie stosowaniem reguł rachunkowych, używali algorytmów. Najsłynniejszy algorytm nosi imię Euklidesa, a samo pojęcie tak mocno zrosło się ze swą egzemplifikacją w „algorytmie Euklidesa", że przez pewien czas uważano, iż słowo „algorytm" ma grecki źródłosłów: wywodzono je od algiros (= bolesny, trudny) i arithmos (= liczba). Prawda jest jednak inna, słowo „algorytm", pierwotnie oznaczające sztukę wykonywania obliczeń przy użyciu cyfr „arabskich", pochodzi z arabskiego, dokładniej – od nazwiska Abu Dżafara Mohammeda ibn Musy al Horizmiego, w którym to kwiecistym nazwisku obok imion przodka i potomka występuje imię własne (Mohammed) i miasto (Horizm, dzisiejszy Chorezm w radzieckiej Azji Środkowej), z którego pochodził słynny matematyk bagdadzki, autor sławnej książki Kitab al dżabr łal mukabala (z tego tytułu wzięła się „algebra"!), z której to właśnie książki Europa nauczyła się liczyć używając cyfr „arabskich". Jednym słowem, zupełna arabszczyzna, wyjąwszy, naturalnie, same cyfry, które pochodzą z Indii.

Zanim Europejczycy posiedli trudną sztukę wykonywania obliczeń w systemie dziesiętnym, posługiwali się rzymskimi abakami, czyli tabliczkami podzielonymi na przegródki, do których wkładano kamyki, czyli calculi (stąd „kalkulować"!). Eksperci od takich rachunków nosili miano abacystów, uczniowie (pośredni!) Horezmijczyka — algorytmistów. Zwycięstwo algorytmistów nie przyszło szybko, nawet w drukowanych już książkach można znaleźć drzeworyty przedstawiające konkursy sprawnościowe z udziałem reprezentantów obydwu szkół. A więc wdrażanie algorytmów arytmetycznych trwało w Europie ponad 600 lat! (Książka Kitab ... powstała około roku 820). Warto o tym pamiętać, gdy sarka się na powolne tempo wdrażania nowych metod postępowania. Ma się te tradycje!

Dziś słowo „algorytm" ma tak wiele znaczeń, zależnych od okoliczności, że trudno byłoby dać uniwersalną, a jednocześnie precyzyjną wykładnię jego treści. Najogólniej (a przeto niezbyt precyzyjnie) powiedziawszy, algorytm to autonomicznie wyrażony przepis na osiąganie zamierzonego celu.

Zauważmy, że nie każdy przepis na osiąganie zamierzonego celu jest algorytmem: musi on być autonomicznie wyrażony, czyli sformułowany tak, aby mógł być stosowany niezależnie. To, od czego ma zachodzić owa niezależność, jest sprawą dość niejasną, a uściślanie tego aspektu pojmowania „algorytmu" prowadzi do wielu ciekawych, aczkolwiek trudnych do ujęcia we wspólne ramy, rozważań.

Łatwiej będzie zilustrować o co nam chodzi na przykładzie: nie jest algorytmem wewnętrznym „przepis postępowania" myszy, która znajduje drogę do kawałka sera ukrytego w głębi labiryntu. Istnieją natomiast dziesiątki algorytmów ustalających lepsze czy gorsze postępowanie prowadzące do znalezienia drogi w labiryncie. (Wiele z tych algorytmów zostało zrealizowanych w postaci programów sterujących różnymi mechanicznymi „myszkami", które np. co roku współzawodniczą o tytuł najlepszej sztucznej myszy). Różnica polega na owym „autonomicznym wyrażeniu". Nie znamy pełnego opisu postępowania myszy naturalnej, nie wiemy nawet, jakie czynniki na nie wpływają, co mysz uwzględnia decydując się na taki czy inny ruch mięśni, nie wiemy nawet, w jakim stopniu jej postępowanie zależy od odbieranych bodźców zewnętrznych (i jakich), a w jakim stopniu od pamięci, „świadomej" i „podświadomej", a także od „wbudowanych" odruchów ścięgien, a nawet poszczególnych komórek. Postępowanie sztucznej myszy jest natomiast całkowicie opisane jawnymi formułami, nawet jeśli opisują one czasem zachowanie niedeterministyczne, bo i wtedy zakres tudzież mechanizm owego niedeterminizmu są wyraźnie opisane (podobnie możemy powiedzieć, że algorytm gry w „chińczyka" jest niedeterministyczny — pewne czynności gracza zależą od wyniku rzutu kostką).

Aby autonomiczność wyrażenia przepisu miała jakikolwiek sens, musi być określony — lub dany domyślnie — pewien układ odniesienia, względem którego formułuje się ów przepis. Taki układ odniesienia nazywa się często dziedziną algorytmiczną, a w jego skład wchodzą obiekty, być może pogrupowane w kategorie, różniące się pod względem tych czy innych cech, ustalone relacje

między obiektami, być może uwzględniające podział obiektów na kategorie, a także pewien repertuar operacji określonych na obiektach, znów być może z uwzględnieniem kategorii.

Dziedziną algorytmu Euklidesa jest zbiór liczb całkowitych nieujemnych, ze zwykłymi relacjami równości, mniejszości itp., oraz z operacjami na liczbach całkowitych.

Jeśli wśród tych operacji dopuścimy występowanie operacji dzielenia z resztą, algorytm Euklidesa może mieć postać:
Dane są dwie liczby całkowite dodatnie X i Y.
Niech R będzie resztą z dzielenia X przez Y.
Jeśli $R = 0$, to szukanym wynikiem (największym wspólnym dzielnikiem) jest liczba Y i kończymy wykonywanie algorytmu.
Jeśli $R \neq 0$, to przyjmujemy za nową wartość X uprzednią wartość Y, a za nową wartość Y otrzymaną wartość R i powtarzamy całe postępowanie od początku.

Jeśli natomiast operacja dzielenia z resztą nie występuje w naszej dziedzinie, algorytm Euklidesa przyjmie postać:
Dane są dwie liczby całkowite dodatnie X i Y.
Jeśli $X = Y$, to szukanym wynikiem jest liczba X i kończymy wykonywanie algorytmu.
Jeśli $X \neq Y$, to niech M będzie większą z liczb X, Y, a m — mniejszą.
Obliczamy $R = M - m$.
Za nową wartość X przyjmujemy obliczoną wartość R, a za nową wartość Y — wartość m i powtarzamy całe postępowanie od początku.

Bez trudu można zauważyć, że w większości przypadków pierwsza wersja algorytmu wymaga mniejszej liczby kroków niż druga. Wynika to stąd, że podstawowa robocza operacja wersji pierwszej — dzielenie z resztą — jest „mocniejsza" niż robocza operacja wersji drugiej — odejmowanie.

Liczba operacji, które należy wykonać, aby osiągnąć cel algorytmu, jest jedną z ważniejszych jego własności. Nie jest to jednak własność najważniejsza. Podstawowe pytanie dotyczące każdego algorytmu odnosi się do jego poprawności: czy można zagwarantować, że wiernie przestrzegając przepisu wyrażonego w algorytmie zawsze osiągniemy żądany cel. Praktycznie, oczywiście, interesują nas tylko takie algorytmy, co do których gwarancja jest jeszcze mocniejsza, a mianowicie, że cel osiągniemy w skończonej liczbie kroków. Pytanie to można zresztą postawić inaczej: czy można zagwarantować, że jeśli wykonanie algorytmu kończy się (po skończonej liczbie kroków przepis dyktuje zaprzestanie wykonywania dalszych działań), to cel zostaje osiągnięty. Jeśli odpowiedź na to pytanie jest twierdząca, algorytm nazywamy częściowo poprawnym. Jeśli ponadto można zagwarantować, że wykonanie algorytmu zawsze się kończy, to nazywamy go po prostu poprawnym albo całkowicie poprawnym.

Dowód tego, że wykonywanie algorytmu zawsze się kończy, jest często niezależny od dowodu, iż kończąc się osiąga pożądany cel. Na przykład wykonanie drugiej wersji algorytmu Euklidesa musi się zawsze zakończyć, gdyż w każdym kroku bieżąca dodatnia wartość M jest zmniejszana o dodatnią liczbę całkowitą, mniejszą od M. Ponieważ proces zaczyna się ze skończoną wartością M, na pewno nie może liczyć więcej niż owo początkowe M kroków. Dowód tego, że wykonując algorytm do końca osiągamy zamierzony cel, wymaga spostrzeżenia dwu faktów:

1. z tego, że wykonanie algorytmu dobiegło końca, wynika, iż bieżące wartości X i Y spełniają $X = Y$,
2. z własności największego wspólnego dzielnika (NWD) dwu liczb całkowitych wynika, dla dowolnych dodatnich $A > B$, że NWD (A, B) = NWD $(A-B, B)$.

Pierwszy z tych faktów jest konsekwencją samego algorytmu, drugi jest własnością dziedziny, której algorytm dotyczy. Mamy tu do czynienia z bardzo typowym zjawiskiem: dowód poprawności algorytmu zależy zarówno od własności jego konstrukcji, jak i od własności dziedziny algorytmicznej.

Podobnie, na konstrukcję — projektowanie — algorytmu wpływają dwa, na ogół rozłączne, zespoły wiedzy: jeden, dotyczący zasad budowy poprawnych i sprawnych algorytmów i drugi, ściśle związany z dziedziną, dla której pisze się ów algorytm. Czasem owa dziedzina ma charakter formalny, jak w przypadku algorytmu Euklidesa, jednakże w wielu praktycznie interesujących przypadkach jest nią pewna dziedzina wiedzy albo zgoła umiejętność, nie mająca adekwatnej reprezentacji formalnej. Sytuacja taka występuje na przykład wtedy, gdy chcemy ułożyć algorytmy dla banku, biura podróży czy towarzystwa ubezpieczeniowego, nie mówiąc już o takich zastosowaniach, jak przekład z jednego języka potocznego na inny albo zarządzanie gospodarcze.

Rozwiązanie zadania M 385.
Jeżeli $(a+b\sqrt{3})^2 = c+d\sqrt{3}$, to $(a-b\sqrt{3})^2 = c-d\sqrt{3}$ i $(a+b\sqrt{3})^2(a-b\sqrt{3})^2 = (c+d\sqrt{3})(c-d\sqrt{3}) = c^2-3d^2$. Gdyby więc istniały a i b żądane w treści zadania, to $(99999)^2-3(100000)^2 < 0$ byłoby kwadratem liczby a^2-3b^2.

Inne rozwiązanie: Gdyby było $99999+100000\sqrt{3} = (a+b\sqrt{3})^2$, mielibyśmy $a^2+3b^2 = 99999$ i $2ab = 100000$.

Z pierwszej z tych równości wynika, że a dzieli się przez 3, z drugiej zaś, że się nie dzieli, co daje sprzeczność.

Użyteczność i przydatność stosowania komputerów polega na tym, że są one urządzeniami w zasadzie bezbłędnie wykonującymi algorytmy. Naturalnie, po to, aby algorytm mógł być wykonany przez komputer, należy ten algorytm przedstawić w pewien określony sposób, wyrazić w języku programowania. Kwestia wyrażenia algorytmu w języku programowania, aczkolwiek niezmiernie ważna z „wewnątrz—informatycznego" punktu widzenia, jest jednak czysto techniczna w tym sensie, że nie stanowi ona żadnej istotnej bariery przed możliwością komputerowego rozwiązania problemu. Barierę taką stanowi natomiast brak odpowiednich algorytmów, a także, niekiedy, nadmierna złożoność obliczeniowa istniejących algorytmów (przez „złożoność obliczeniową" algorytmu rozumie się oszacowanie kosztu jego realizacji, mierzonego liczbą wykonywanych operacji).

Informatyka bada metody budowy, analizy i przekształcania algorytmów dla dziedzin, które są na tyle dobrze opisane sformalizowanymi związkami, że stosując do nich (tj. dziedzin) zasady obliczalnych wnioskowań, nie należy obawiać się popełnienia grubego błędu. Stosowanie algorytmicznych metod postępowania w dziedzinach nie spełniających tego warunku jest błędem metodologicznym, nie dającym się zazwyczaj wykryć w żaden sposób przed tragiczną nieraz konfrontacją wyników takiego postępowania z rzeczywistością. Niestety, pokusa zastosowania komputerów w dziedzinach pozornie tylko albo połowicznie sformalizowanych bywa zbyt wielka.

Oprócz konstrukcji, analizy i przekształcania algorytmów, informatyka zajmuje się także algorytmami jako tworami abstrakcyjnymi. Badania takie mają fundamentalne znaczenie poznawcze. Próbuje się znaleźć odpowiedzi na takie na przykład pytania: Czy zawsze wtedy, gdy dany problem dopuszcza rozwiązanie algorytmiczne, istnieje algorytm optymalny dla tego zadania (np. algorytm najszybszy, o najmniejszej złożoności obliczeniowej)? Jaką klasę funkcji można obliczyć za pomocą algorytmów zbudowanych z danego zestawu operacji? Czy zmiana modelu realizacji algorytmu (np. wzbogacenie go o obliczenia niedeterministyczne) zmienia w istotnym stopniu zakres zadań, które można rozwiązać algorytmicznie?

Widzimy więc, że problematyka algorytmów — autonomicznych przepisów efektywnego postępowania — obejmuje całe spektrum zainteresowań matematyki: od najbardziej podstawowych, filozoficznych niemal, do całkowicie praktycznych, by nie rzec: utylitarnych. Niektórzy powiadają, że prawdziwa matematyka to właśnie studium algorytmów. Jak zwykle przy takich mocnych uogólnieniach nie wszyscy zechcą je podzielić. W każdym razie trudno nie zgodzić się z tym, że studium algorytmów to prawdziwie pasjonujące zajęcie i kawał pięknej matematyki.

Rozwiązanie zadania M 383.
Przypuśćmy, że ta sama liczba pojawia się dwukrotnie po raz pierwszy w wierszu $n+1$. Oznaczmy ją przez r. Ponieważ wszystkie liczby w n-tym wierszu są z założenia różne, więc $r = u^2 = v+1$ dla pewnej pary u, v elementów n-tego wiersza. Mamy: $v = u^2 - 1$ i ponieważ największym kwadratem liczby naturalnej nie przewyższającym $u^2 - 1$ jest $(u-1)^2 = (u^2-1)-(2u-2)$, więc dla otrzymania v potrzebujemy co najmniej $2u-1$ operacji (podniesienie do kwadratu i $2u-2$ dodawania jedynki). Wynika stąd, że $n \geqslant 2u$ i $u - a \geqslant 2u-1$, co jest niemożliwe, ponieważ $u > a > 1$.

Rozwiązanie zadania M 384.
Łatwo zauważyć, że części całkowite liczb $x_1, \ldots, x_n$ nie są w zadaniu istotne i możemy rozpatrywać jedynie części ułamkowe $y_1 = x_1 - [x_1], \ldots, y_n = x_n - [x_n]$, przy czym możemy założyć, że $0 \leqslant y_1 \leqslant y_2 \leqslant \ldots \leqslant y_n < 1$.

Wybierzmy teraz takie k, że $ky_k \leqslant \frac{n+1}{4}$ i $(k+1)y_{k+1} > \frac{n+1}{4}$ (ewentualnie $k = 0$).

Ponieważ $y_{k+1} > \frac{n+1}{4(k+1)}$, więc

$$(n-k)(1-y_{k+1}) < (n-k)\left(1-\frac{n+1}{4(k+1)}\right).$$

Nierówność $(n-k)\left(1-\frac{n+1}{4(k+1)}\right) \leqslant \frac{n+1}{4}$ jest równoważna $(n-k)(4k-n+3) \leqslant (n+1)(k+1)$, a ta jest równoważna nierówności $(n-(2k+1))^2 \geqslant 0$ spełnionej dla dowolnych k, n.

Jeśli teraz $y_1, \ldots, y_k$ zaokrąglimy do 0, a $y_{k+1}, \ldots, y_n$ do 1, to największy błąd przy zastępowaniu składników sumy ich zaokrągleniami możemy popełnić rozpatrując sumy $y_1 + \ldots + y_k$ i $y_{k+1} + \ldots + y_n$. W pierwszym przypadku błąd wyniesie najwyżej ky_k, a w drugim najwyżej $(n-k)(1-y_{k+1})$.

Wynika stąd, że dla podanego wyżej wyboru k zarówno $ky_k \leqslant \frac{n+1}{4}$, jak i $(n-k)(1-y_{k+1}) \leqslant \frac{n+1}{4}$.