Złożoność obliczeniowa algorytmów

Doc. dr Antoni KRECZMAR

Jednym z głównych nurtów rozwoju współczesnej informatyki jest dążenie do konstruowania coraz lepszych algorytmów. Kiedy mówimy, że jeden algorytm jest lepszy od drugiego, mamy na myśli pewną ich własność. Może to na przykład dotyczyć większej precyzji otrzymywanych wyników albo mniejszej liczby zmiennych pomocniczych potrzebnych do przechowywania wyników pośrednich, czyli mniejszego obciążenia pamięci komputera. A może uznamy, że dany algorytm jest lepszy, ponieważ jest bardziej uniwersalny lub dlatego, że jego sformułowanie jest bardziej zwięzłe i czytelne? Każda z powyżej wymienionych własności może być dokładnie sformułowana, a następnie używana jako miara jakości algorytmu. Niemniej jest pewna własność algorytmów, której światowa literatura informatyczna od ponad 10 lat poświęca najwięcej uwagi. Jest to mianowicie czas wykonania algorytmu.

W latach trzydziestych naszego stulecia matematycy zajmujący się podstawami matematyki rozwiązali wiele problemów dotyczących teorii algorytmów. Głównym pytaniem jakie sobie stawiali, a następnie starali się rozwiązać, było pytanie o istnienie algorytmu dla danego zadania. Klasyfikowano wówczas zadania na dwie grupy. Pierwszą tworzyły zadania rozstrzygalne, to znaczy te, dla których istnieje algorytm, drugą — zadania nierozstrzygalne, dla których algorytm nie istnieje.

Zadania rozstrzygalne nie wzbudzały wielkiego zainteresowania. Jeżeli stwierdzono, że jakiś problem jest rozstrzygalny, świadczyło to o jego małym stopniu skomplikowania. Cóż bowiem mogło być interesującego na przykład w rachunku zdań? Przecież rachunek zdań jest rozstrzygalny, co więcej algorytm sprawdzania, czy dana formuła rachunku zdań jest spełnialna, jest bardzo prosty. Wystarczy bowiem dla wszystkich zmiennych występujących w takiej formule podstawić na wszelkie możliwe sposoby wartości logiczne „prawda" i „fałsz" i sprawdzać, czy dla takich podstawień wartością formuły będzie „prawda". Jeżeli formuła jest spełnialna, to na takie podstawienie trafimy. Ponieważ zaś takich różnych podstawień jest skończona ilość, to po skończonej liczbie kroków poznamy odpowiedź na zadane pytanie. Nikt nie miał jednak wówczas zamiaru stosować takich algorytmów dla bardzo skomplikowanych formuł, na przykład o 100 zmiennych.

Burzliwy rozwój technologii w latach sześćdziesiątych i siedemdziesiątych pozwolił na zbudowanie komputerów o olbrzymich pamięciach i niesłychanie szybko liczących procesorach. To wspaniałe narzędzie daje możliwość wykonywania algorytmów dla bardzo dużych zadań. Ale kwestia wyboru algorytmu nie jest już teraz obojętna, tak jak to było wtedy, gdy stosowalność algorytmu miała sens li tylko teoretyczny. Chcemy bowiem, by nasz algorytm był jak najlepszy. I okazało się, że kluczową własnością, która ma wielki wpływ na zakres wykorzystania algorytmu, jest jego czas działania.

U zarania epoki komputerów wydawało się, że wszystko, co jest rozstrzygalne, będzie mógł w przyszłości wykonać komputer. Ale niestety praktyka ostatnich dwudziestu lat dowodzi czegoś innego. Są bowiem problemy rozstrzygalne, których nawet najszybszy komputer nie może w rozsądnym czasie rozwiązać. Dlaczego tak jest i na czym właściwie polega to ograniczenie? Przedstawmy sprawę bardziej precyzyjnie.

Możemy powiedzieć, że to, co chcemy liczyć, jest opisane pewną funkcją $f: D \to W$ ze zbioru danych D w zbiór wyników W. Na przykład dla rachunku zdań zbiorem danych jest zbiór wszystkich formuł zdaniowych, natomiast zbiorem wyników jest dwuelementowy zbiór wartości logicznych „*prawda*" i „*fałsz*". Dla danej formuły α, $f(\alpha) =$ „*prawda*" wtedy i tylko wtedy, gdy formuła α jest spełnialna.

Rozważmy jeszcze dla przykładu kluczowy problem algebry liniowej, tj. rozwiązywanie układu równań liniowych. Zadanie to polega na znalezieniu wektora rozwiązań $x = [x_j]$, $j = 1, \ldots, n$ układu postaci $Ax = b$, gdzie A jest macierzą kwadratową $A = [a_{ij}]$, $i, j = 1, \ldots, n$, a b jest wektorem wyrazów wolnych $b = [b_j]$, $j = 1, \ldots, n$. Zakładamy ponadto, że A jest macierzą nieosobliwą, zatem układ równań, zgodnie z twierdzeniem Cramera, ma jednoznaczne rozwiązanie. Przyglądając się uważnie temu zadaniu możemy stwierdzić, że dane dla takiego zadania stanowią: liczba równań, wartości elementów macierzy A oraz wartości elementów wektora b. Zbiorem wyników są wektory x. Funkcja, którą chcemy liczyć jest określona następująco: $f(n, A, b) = x$ wtedy i tylko wtedy, gdy $Ax = b$.

Jeżeli f jest funkcją, którą chcemy liczyć na komputerze, to musimy znaleźć algorytm Al ją liczący. Algorytm Al musi dla danego elementu d ze zbioru D policzyć wartość w ze zbioru W taką, że $f(d) = w$. Zakładamy przy tym, że algorytm ten jest jednorodny, to znaczy jeden dla całej funkcji f, a nie oddzielnie dobierany dla różnych danych d. Aby dla danej d algorytm Al wyliczył wynik w, musi on wykonać pewną liczbę operacji elementarnych. Sam rodzaj takich operacji może zależeć od tego, jakie f liczymy. Dla rachunku zdań chodzi tu o operacje logiczne, a dla układu równań liniowych będą to operacje arytmetyczne.

Tablicę współczynników układu równań

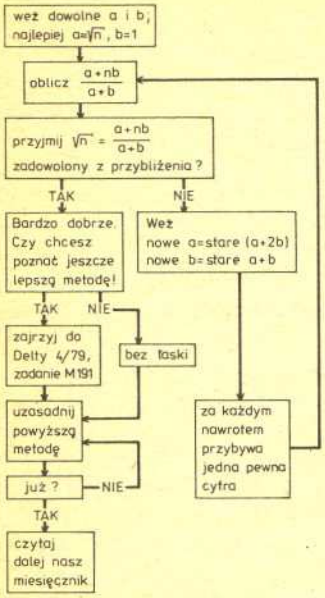
$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$,
$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$,
.....
.....
$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$

nazywamy jego macierzą, a układ taki zapisujemy skrótowo: $Ax = b$. W teorii układów równań liniowych dowodzi się, że warunkiem rozwiązalności powyższego układu jest, by wyznacznik jego macierzy był różny od zera.

Autor używa w artykule słowa „liczyć" w znaczeniu „obliczać". Choć mówią tak i piszą chyba wszyscy matematycy, jest to pewne wykroczenie przeciwko regułom polszczyzny.

Jak obliczyć $\sqrt{n}$?

Na przykład tak.



Ustaliwszy, jakie elementarne operacje wykonuje algorytm Al możemy określić koszt czasowy wykonania algorytmu Al dla danej d. Będzie to liczba wykonanych przez algorytm Al elementarnych operacji dla danej d. Tak rozumiany koszt oznaczymy przez $c(Al(d))$.
Określamy teraz rozmiar danej z rozważanego przez nas zbioru danych D. Na przykład dla rachunku zdań, za rozmiar danej możemy przyjąć długość formuły, a dla układu równań liniowych może to być liczba równań. Rozmiar danej jest zawsze liczbą naturalną. Będziemy ją oznaczać dla danej d w następujący sposób: $|d|$.
Definiujemy koszt czasowy działania algorytmu dla danych długości n jako

$$T(n) = max\{c(Al(d)) : d \in D \quad \text{i} \quad |d| = n\}.$$

A więc jako globalny koszt algorytmu dla danych rozmiaru n bierzemy maksimum kosztów wykonania algorytmu dla wszystkich danych tego rozmiaru.

Załóżmy, że funkcja f jest wszędzie określona i że algorytm Al dla każdych danych d wykonuje skończoną liczbę kroków. Wówczas $c(Al(d)) < +\infty$. Jeżeli ponadto dla rozmiaru n zbiór danych tego rozmiaru jest skończony albo $c(Al(d))$ zależy tylko od n, to $T(n) < +\infty$. Dla zadania spełnialności rachunku zdań mamy ten pierwszy przypadek, gdyż liczba różnych formuł długości n jest skończona. Dla układu równań liniowych możemy przyjąć, że czas działania algorytmu nie zależy od wielkości współczynników występujących w tych równaniach, tylko od liczby tych równań. Zatem zachodzi ten drugi przypadek.

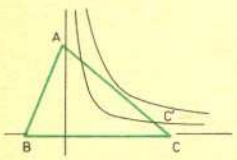
Jeżeli już wszystko mamy ustalone, to znaczy algorytm, operacje elementarne, które wliczamy do kosztu algorytmu i rozmiar danych, to możemy precyzyjnie badać własności funkcji kosztu T. Ale własnością, która nas najbardziej interesuje, jest szybkość wzrostu tej funkcji. Im bowiem szybciej ta funkcja rośnie, tym wolniej działa nasz algorytm. I tym, co gra tu najważniejszą rolę jest rząd wielkości tej funkcji, a nie dokładne współczynniki. Jeżeli T jest rzędu n, to znaczy, że od pewnego miejsca koszt algorytmu zachowuje się zawsze jak funkcja liniowa i bez względu na to, na jakim komputerze zaprogramujemy ten algorytm, to będzie się on w sensie czasu działania zachowywał jak funkcja liniowa. Natomiast jeżeli T jest rzędu 2^n, to czas ten będzie się zachowywał jak funkcja wykładnicza zmiennej n. Spójrzmy na poniższą tabelę:

Algorytm	rząd T	maksymalny rozmiar zadania, które komputer może policzyć w czasie 1 godziny	maksymalny rozmiar zadania po dziesięciokrotnym przyspieszeniu komputera
A_1	n	s_1	$10s_1$
A_2	$n\,lg_2 n$	s_2	$\approx 10s_2$
A_3	n^2	s_3	$\approx 3{,}16\,s_3$
A_4	n^3	s_4	$\approx 2{,}15s_4$
A_5	2	s_5	$\approx s_5+3{,}3$

Ostatnia kolumna tej tabeli wskazuje, jak wielki wpływ na rozmiar zadania, które w określonym czasie danym nam do dyspozycji chcemy rozwiązać, ma rząd funkcji T. Jeżeli uda się nam znaleźć algorytm, którego rząd kosztu będzie mniejszy, da to znacznie większe przyspieszenie obliczeń, niż wielokrotne nawet przyspieszenie działania komputera. Konstruowanie zatem coraz szybszych algorytmów jest bardzo ważnym zadaniem stojącym przed współczesną informatyką. Ponadto w sposób naturalny pojawiają się także ciekawe pytania teoretyczne. Czy dla danego zadania określonego funkcją f istnieje algorytm optymalny, to znaczy taki, którego koszt T ma najmniejszy rząd? Koszt tego optymalnego algorytmu, o ile istnieje, nazywamy kosztem zadania. Poszukiwanie kosztu zadania polega najczęściej na konstruowaniu coraz lepszych algorytmów, czyli na tak zwanym poprawianiu oszacowania górnego, oraz na wykazywaniu, że każdy algorytm rozwiązujący to zadanie musi mieć dostatecznie duży koszt, czyli na poprawieniu oszacowania dolnego. Jeżeli oszacowanie górne i dolne spotkają się, oznacza to, że znaleziony został koszt zadania, ale jak dotąd dla niewielu zadań udało się taki koszt znaleźć.

Przedstawimy teraz analizę kosztu zadania na przykładzie dwóch problemów, o których wspominaliśmy już poprzednio. Zajmiemy się najpierw zadaniem rozwiązywania układów równań liniowych. Ponieważ wykazano, że koszt tego zadania jest równy kosztowi mnożenia dwóch macierzy kwadratowych przedstawimy, jak wygląda stan badań nad tym drugim zadaniem. Mając dane dwie macierze $A = [a_{ij}]$, $i, j = 1, \ldots, n$ oraz $B = [b_{jk}]$, $j, k = 1, \ldots, n$ ich iloczyn określamy jako macierz $C = [c_{ik}]$ $(i, k = 1, \ldots, n)$ gdzie $c_{ik} = \sum_{j=1}^{n} a_{ij} \cdot b_{jk}$. Możemy zatem wyliczyć C wykonując (dla każdego $i, k = 1, \ldots, n$) n mnożeń oraz $n-1$ dodawań. Razem n^3 mnożeń oraz $n^3 - n^2$ dodawań. Możemy więc powiedzieć, że koszt bezpośredniego algorytmu jest rzędu n^3. Przez wiele lat sądzono, że jest to algorytm najszybszy, próbowano nawet tego dowodzić.

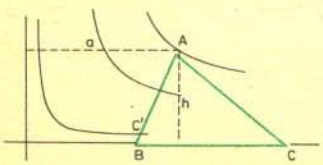
Rozwiązanie zadania M 228.
Przyłóżmy siatkę tak, jak na rys. 1, i oznaczmy na niej położenie wierzchołka C.



Rys. 1

Jeżeli teraz przesuniemy ją tak, jak to wskazuje rys. 2, to wierzchołek A znajdzie się na hiperboli o równaniu $xy = 2S_{ABC}$.
Tak więc $(2+\sqrt{3})\,\frac{l}{a} < n < (\sqrt{6}+\sqrt{2})\,\frac{l}{a}$,
czyli $3{,}73\,\frac{l}{a} < n < 3{,}87\,\frac{l}{a}$, $n \approx \frac{3{,}8}{a} \cdot l$.

Gdy teraz będziemy przybliżać krzywą łamanymi otrzymamy analogiczną zależność dla długości krzywej.
Uwaga: Omówiliśmy w tym zadaniu działanie tzw. longimetru Steinhausa opisanego w jego książce „Kalejdoskop Matematyczny".



Rys. 2

Eliminacja Gaussa to najbardziej popularny i znany każdemu algorytm rozwiązywania układów równań liniowych. Polega on na wyznaczaniu jednej z niewiadomych względem innych z któregokolwiek z równań, podstawieniu do pozostałych, wyznaczeniu następnej niewiadomej itd. Koszt tego algorytmu jest rzędu n^3.

W 1969 roku V. Strassen opublikował krótką pracę pod tytułem „Eliminacja Gaussa nie jest optymalna". V. Strassen zauważył, jak można pomnożyć dwie kwadratowe macierze 2 na 2 używając mniej niż 8 mnożeń. Wprowadźmy oznaczenia:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix} \quad A \cdot B = \begin{bmatrix} r & s \\ t & u \end{bmatrix}.$$

Stosując zwykłe mnożenie liczymy według wzorów:

$$r = a \cdot e + b \cdot g \quad s = a \cdot f + b \cdot h \quad t = c \cdot e + d \cdot g \quad u = c \cdot f + d \cdot h$$

co wymaga 8 mnożeń i 4 dodawań.

Wykonajmy teraz te obliczenia sposobem na pozór bardzo dziwacznym. Najpierw policzymy 7 następujących wyrażeń:

$$p_1 = (a+d)(e+h) \quad p_2 = (c+d)e \quad p_3 = a(f-h) \quad p_4 = d(g-e)$$
$$p_5 = (a+b)h \quad p_6 = (c-a)(e+f) \quad p_7 = (b-d)(g+h)$$

a następnie liczymy iloczyn według wzorów:

$$r = p_1 + p_4 - p_5 + p_7 \quad t = p_2 + p_4 \quad s = p_3 + p_5 \quad u = p_1 + p_3 - p_2 + p_7.$$

Łatwo sprawdzamy poprawność powyższych wzorów. Następnie zauważmy, że ten sposób liczenia wymaga tylko 7 mnożeń, choć aż 18 operacji addytywnych (tzn. dodawań lub odejmowań). Co więcej, przy wskazywaniu poprawności tych wzorów nie posługiwaliśmy się przemiennością mnożenia (co nasz uważny Czytelnik z łatwością dostrzeże). Zatem a, b, c, d, e, f, g, h, mogą być macierzami kwadratowymi tych samych wymiarów i powyższe wzory będą w dalszym ciągu poprawne. Jeżeli mnożenie kosztuje tyle samo co dodawanie, to oczywiście pomnożenie dwóch macierzy 2 na 2 metodą Strassena nie daje zysku, wręcz stratę. Ale przecież mnożenie dwóch macierzy jest znacznie bardziej kosztowne niż ich dodawanie lub odejmowanie. Załóżmy więc, że dwie dane macierze A i B są wymiaru 2^k na 2^k i potraktujemy je jako macierze 2 na 2, gdzie a, b, c, d, e, f, g, h są podmacierzami o wymiarach 2^{k-1} na 2^{k-1}. Sprowadzamy teraz rekurencyjnie zadanie pomnożenia A przez B do zadania pomnożenia 7 razy macierzy o wymiarach dwa razy mniejszych oraz do wykonania 18 operacji addytywnych na tych podmacierzach. Koszt takiego algorytmu będzie wyrażał się następującym wzorem rekurencyjnym:

$$T(1) = 1 \quad T(2^k) = 7 \cdot T(2^{k-1}) + 18 \cdot 2^{k-2}$$

Wykazujemy łatwo przez indukcję, że $T(2^k) = 7^{k+1} - 6 \cdot 4^k$. Zatem dla $n = 2^k$ $T(n) = 7 \times$ $\times 7^{\lg_2 n} - 6n^2$, czyli $T(n)$ jest rzędu $n^{\lg_2 7}$. Można tak zmodyfikować ten algorytm, aby w tym samym (co do rzędu) czasie wykonywał mnożenie macierzy kwadratowych dowolnych wymiarów, niekoniecznie 2^k. Ale $\lg_2 7$ równa się około 2,81 czyli algorytm Strassena wyprzedza zwykły algorytm, który jest rzędu n^3.

Zastosowanie chwytu Strassena z mnożeniem macierzy o małych wymiarach starano się w dalszym ciągu wykorzystać. Dla macierzy 2 na 2 mniej niż 7 mnożeń uzyskać się nie da (wykazano to na początku lat siedemdziesiątych). S. Winogradowi udało się natomiast poprawić liczbę operacji addytywnych, mianowicie zredukował ją do 15. Może któremuś Czytelnikowi uda się znaleźć rozwiązanie Winograda, ale uprzedzam, że zadanie jest trudne. Na tym sprawa utknęła i przez prawie 10 lat nie było nowych istotnych wyników. Wydawało się nawet, że algorytm Strassena jest optymalny i koszt zadania jest rzędu $n^{\lg_2 7}$. Dopiero w 1978 roku V. Pan opublikował pracę pod tytułem „Algorytm Strassena nie jest optymalny". V. Pan znalazł algorytm działający w czasie rzędu $n^{2,79}$. W rok później grupa czterech matematyków z Pizy zaatakowała problem w trochę inny sposób niż Strassen i Pan (praca V. Pan'a opierała się na podobnej zasadzie co algorytm Strassena) i stosując metody aproksymacyjne uzyskali rząd $n^{2,7799}$. Są także i nowsze wyniki, jeszcze nie opublikowane (podobno A. Schönhage skonstruował algorytm, którego koszt jest prawie rzędu $n^{2,5}$).

Ale jak dotąd nie znamy kosztu zadania mnożenia dwu macierzy. Jedyne znane dolne oszacowanie wynosi n^2, gdyż tyle informacji przynoszą same dane, zatem niemożliwe jest pomnożenie dwu macierzy w mniejszej liczbie operacji arytmetycznych.

Z problemem spełnialności formuł rachunku zdań sytuacja jest jeszcze bardziej skomplikowana. Bezpośredni algorytm, o którym wspomniałem na początku jest bardzo kosztowny. Wymaga on bowiem wykonania obliczeń w koszcie rzędu 2^n. Co więcej, nie znamy do dziś żadnego algorytmu rozwiązującego to zadanie w koszcie wielomianowym, to znaczy rzędu n^k dla pewnej stałej k. Znalezienie takiego algorytmu lub udowodnienie, że go nie ma jest centralnym otwartym problemem złożoności obliczeniowej. Mogłoby się wydawać, że jego znaczenie jest niewielkie, gdyż dotyczy dziedziny bardzo specyficznej, jaką jest rachunek zdań. Otóż, niestety, okazuje się, że olbrzymią większość zadań z zakresu programowania dyskretnego, kombinatoryki, optymalizacji, teorii grafów itp. może być sprowadzona do problemu spełnialności rachunku zdań. Dolne oszacowanie kosztu tego zadania wynosi n. Jak widać pomiędzy n i 2^n jest wielka luka, znacznie większa niż pomiędzy n^2 i $n^{2,5}$. Dla równań liniowych algorytm eliminacji Gaussa jest całkiem zadowalający, dla rachunku zdań nie znamy takiego zadowalającego algorytmu. Zatem i tego zadania, i do niego podobnych, wspomnianych powyżej, nie da się w ogóle rozwiązywać na komputerach, chyba że rozmiary danych są bardzo małe.

Rozwiązanie zadania F 77.
Ciśnienie w powietrzu oraz w cieczy na jej poziomej powierzchni jest równe ciśnieniu atmosferycznemu p. W związku z tym ciśnienie wywierane na zanurzoną część płyty przez ciecz tworzącą menisk wklęsły jest mniejsze od ciśnienia atmosferycznego, zaś przez ciecz tworzącą menisk wypukły musi być większe od ciśnienia atmosferycznego. Dzięki zjawisku włoskowatości nie ma równowagi ciśnień działających na zewnętrzne i wewnętrzne płaszczyzny płyt. Widać to łatwo na rysunku, którego analiza przekonuje, że płyty w przypadku (a) i (b) przyciągają się, podczas gdy w przypadku (c) odpychają.