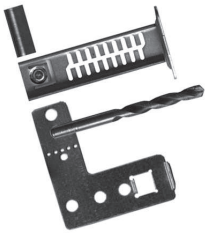
Automat do badania algorytmów

Przemysław KICIAK *



Żeby zrozumieć rekurencję,
należy najpierw zrozumieć rekurencję.

Są takie twierdzenia, z których mnóstwo innych twierdzeń wynika na tyle łatwo, że można je stosować niemal „mechanicznie”, na przykład w geometrii twierdzenie Cevy. Są też takie „mechaniczne” metody rozwiązywania różnych zadań. Zobaczymy, jak można zastosować równania różnicowe do badania złożoności różnych algorytmów iteracyjnych i rekurencyjnych.

Rozważmy nieskończony ciąg liczb $a_0, a_1, \dots$, którego pierwsze n wyrazów wybrano dowolnie, a pozostałe wyrazy (dla $k \geqslant n$) spełniają równanie

$$a_k = c_{n-1}a_{k-1} + \cdots + c_1 a_{k-n+1} + c_0 a_{k-n} + f(k),$$

w którym liczby $c_0, \dots, c_{n-1}$ oraz funkcja f są dane. Początkowe wyrazy ciągu (opisujące **warunek początkowy**) i powyższe równanie określają nasz ciąg jednoznacznie. Jeśli jakiś algorytm przetwarzania danych o rozmiarze $k \geqslant n$ polega na rozwiązaniu zadań dla mniejszych danych, a następnie wykonaniu $f(k)$ operacji w celu otrzymania wyniku, to koszt lub złożoność nieraz możemy opisać w ten sposób. Zwykle chcemy jednak mieć bardziej „jawne” wyrażenie opisujące a_k.

Zacznijmy od rozwiązania **równania jednorodnego**, tj. takiego, w którym funkcja f jest zerowa. Zbadamy hipotezę, że równanie takie jest spełnione przez ciąg geometryczny: $a_k = \lambda^k$, dla pewnej liczby $\lambda \neq 0$. Po wstawieniu mamy

$$\lambda^k = c_{n-1}\lambda^{k-1} + \cdots + c_1\lambda^{k-n+1} + c_0\lambda^{k-n},$$

skąd po uporządkowaniu otrzymamy tzw. **równanie charakterystyczne**:

$$\lambda^n - c_{n-1}\lambda^{n-1} - \cdots - c_1\lambda - c_0 = 0.$$

Ciąg geometryczny λ^k jest zatem rozwiązaniem równania jednorodnego, jeśli liczba λ jest rozwiązaniem równania charakterystycznego. Jeśli równanie to ma n różnych rozwiązań, to mamy n niezależnych liniowo ciągów geometrycznych spełniających równanie jednorodne i każda ich kombinacja liniowa jest rozwiązaniem. Dobierając jej współczynniki, możemy znaleźć ciąg spełniający warunki początkowe.

Zobaczmy przykład. Równanie

$$F_k = F_{k-1} + F_{k-2}$$

z warunkiem początkowym $F_0 = 0$, $F_1 = 1$ określa ciąg liczb, zgodnie z którym Leonardo z Pizy teoretycznie, a później Lejzorek Rojtszwaniec praktycznie, choć też na papierze, rozmnażali króliki. Rozwiązaniami równania charakterystycznego

$$\lambda^2 - \lambda - 1 = 0$$

są liczby $\lambda_1 = \frac{1}{2}(1 - \sqrt{5})$ i $\lambda_2 = \frac{1}{2}(1 + \sqrt{5})$. Mamy

$$F_k = b_1\lambda_1^k + b_2\lambda_2^k$$

dla pewnych liczb b_1, b_2. Na podstawie warunków początkowych,

$$F_0 = b_1 + b_2 = 0,$$
$$F_1 = b_1\lambda_1 + b_2\lambda_2 = 1,$$

obliczamy $b_1 = -\frac{1}{\sqrt{5}}$, $b_2 = \frac{1}{\sqrt{5}}$, skąd wynika, że po upływie k miesięcy od założenia hodowli możemy mieć nawet

$$F_k = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k\right)$$

par królików.

Co możemy zrobić, jeśli różnych rozwiązań równania charakterystycznego jest mniej niż n? Jeśli liczba λ jest rozwiązaniem o krotności $r > 1$, to równanie

*Instytut Matematyki Stosowanej
i Mechaniki, Uniwersytet Warszawski

jednorodne jest spełnione przez ciągi $\lambda^k, k\lambda^k, \ldots, k^{r-1}\lambda^k$, i znając wszystkie rozwiązania równania charakterystycznego i ich krotności, możemy znaleźć n niezależnych liniowo ciągów spełniających jednorodne równanie różnicowe. Zabierzmy się zatem za równania niejednorodne.

Jeśli mamy dowolne rozwiązanie równania niejednorodnego, to każde inne jego rozwiązanie możemy otrzymać, dodając pewne rozwiązanie równania otrzymanego przez „wyrzucenie" funkcji f (czyli równania jednorodnego). W badaniu złożoności algorytmów bardzo często $f(k) = p(k)\mu^k$, gdzie p jest wielomianem pewnego stopnia d. Jeśli liczba μ jest rozwiązaniem równania charakterystycznego (tak, tego rozpatrywanego wcześniej) o krotności r, to pewne rozwiązanie równania z taką funkcją f ma postać $k^r q(k)\mu^k$, gdzie q jest wielomianem stopnia d (to jest prawdą także dla $r = 0$). Współczynniki wielomianu q możemy obliczyć na podstawie równania. Zobaczmy to, badając konkretne algorytmy.

Maksymalna liczba porównań $T(k)$ potrzebna do posortowania k-elementowego ciągu **algorytmem sortowania przez wstawianie** spełnia równanie:

$$T(1) = 0,$$
$$T(k) = T(k-1) + k - 1 \quad \text{dla } k > 1.$$

Równanie charakterystyczne ma postać $\lambda - 1 = 0$, a $f(k) = (k-1) \cdot 1^k$. Ponieważ $\mu = 1$ jest rozwiązaniem równania charakterystycznego (o krotności 1), więc pewne rozwiązanie równania ma postać

$$a_k = k(bk + c) \cdot 1^k.$$

Po podstawieniu do równania mamy

$$bk^2 + ck = b(k-1)^2 + c(k-1) + k - 1.$$

Po uporządkowaniu otrzymujemy

$$ck = (-2b + c + 1)k + (b - c - 1).$$

Ponieważ ta równość zachodzi dla każdego k, więc $1 - 2b = 0$ i $b - c - 1 = 0$, a stąd $b = \frac{1}{2}$, $c = -\frac{1}{2}$. Mamy więc

$$T(k) = \frac{1}{2}(k^2 - k) + d \cdot 1^k.$$

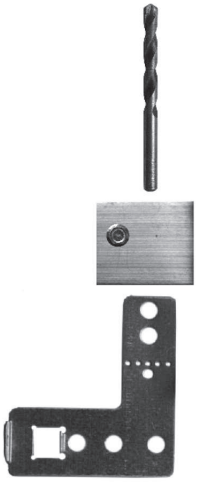
Liczbę d obliczymy na podstawie warunku początkowego; otrzymamy $d = 0$. Zatem, sortując ciąg k-elementowy algorytmem przez wstawianie, wykonamy co najwyżej $\frac{1}{2}(k^2 - k)$ porównań.

W **algorytmie sortowania przez scalanie** liczba porównań elementów potrzebna do posortowania ciągu o długości n nie przekracza

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n - 1.$$

Przyjmijmy, że $n = 2^k$ dla pewnej liczby naturalnej k i oznaczmy $a_k = T(2^k)$. Rozwiązując równanie $a_k = 2a_{k-1} + 2^k - 1$ z warunkiem początkowym $a_0 = 0$ (co Czytelnik powinien tu już umieć), otrzymamy $a_k = (k-1) \cdot 2^k + 1$, skąd dla $n = 2^k$, czyli $k = \log_2 n$, mamy

$$T(n) = (\log_2 n - 1) \cdot n + 1.$$



Rozważmy jeszcze dwa algorytmy przybliżonego rozwiązywania równania nieliniowego $f(x) = 0$: **metodę Newtona** i **metodę siecznych**. Znacie? To poczytajcie. W metodzie Newtona potrzebujemy jednego przybliżenia rozwiązania, x_0, a w metodzie siecznych – dwóch: x_0 i x_1, na podstawie których obliczamy kolejne przybliżenia. Jeśli liczba α jest rozwiązaniem, funkcja f jest klasy C^2 i jej pochodna w otoczeniu α jest różna od 0, to błędy $\varepsilon_k = x_k - \alpha$ kolejnych przybliżeń rozwiązania w tych dwóch metodach spełniają równości

$$\varepsilon_k = \frac{f''(\xi_{k-1})}{2f'(x_{k-1})}\varepsilon_{k-1}^2, \quad \text{oraz} \quad \varepsilon_k = \frac{f''(\xi_{k-1})}{2f'(x_{k-1})}\varepsilon_{k-1}\varepsilon_{k-2},$$

przy czym liczba ξ_{k-1} leży w przedziale zawierającym α i x_{k-1}, albo α, x_{k-1} i x_{k-2}, odpowiednio. Oznaczmy

$$a_k = \log|\varepsilon_k| \quad \text{oraz} \quad g(k) = \log\left|\frac{f''(\xi_{k-1})}{2f'(x_{k-1})}\right|$$

(podstawa logarytmu może być dowolna, większa niż 1). Dla metody Newtona dostaniemy równanie

$$a_k = 2a_{k-1} + g(k).$$

Zastąpmy $g(k)$ przez stałą G i załóżmy, że $a_0 < -G$. Rozwiązaniem tak zmienionego równania jest ciąg

$$\tilde{a}_k = (a_0 + G) \cdot 2^k - G.$$

Możemy dalej sprawdzić, że rozwiązanie równania z $g(k) < G$ spełnia warunek $a_k < \tilde{a_k}$. Wtedy ciąg a_k dąży wykładniczo do $-\infty$, a ponieważ mamy tu kolejne potęgi liczby 2, więc każde kolejne przybliżenie rozwiązania ma około 2 razy więcej cyfr dokładnych. Zbadanie metody siecznych zostawiam Czytelnikowi na deser.

Algorytm Strassena jest pierwszym poznanym algorytmem mnożenia macierzy o wymiarach $n \times n$, w którym trzeba wykonać mniej niż $C \cdot n^3$ działań arytmetycznych dla pewnej stałej C. Niech $n = 2^k > 1$. W algorytmie Strassena macierze do pomnożenia dzielimy na bloki o wymiarach $m \times m$, gdzie $m = n/2$. Aby otrzymać bloki, z których składa się wynik, trzeba wykonać 7 mnożeń oraz 18 dodawań i odejmowań macierzy $m \times m$. Każde dodawanie i odejmowanie to m^2 działań na współczynnikach. Mamy zatem

$$T(n) = 7T\left(\frac{n}{2}\right) + \frac{18}{4}n^2,$$

oraz $T(1) = 1$. Oznaczmy $a_k = T(2^k)$. Mamy równanie $a_k = 7a_{k-1} + 18 \cdot 4^{k-1}$ z warunkiem początkowym $a_0 = 1$. Rozwiązaniem spełniającym warunek początkowy jest ciąg $a_k = 7^{k+1} - 6 \cdot 4^k$, a zatem

$$T(n) = (2^{\log_2 7})^{k+1} - 6 \cdot 4^k = 7n^{\log_2 7} - 6n^2.$$

Ostatni przykład dotyczy **drzew AVL** (ich nazwa pochodzi od wynalazców, których było dwóch: Adelson-Velski i Landis). Koszt wstawiania, usuwania i dostępu do danych w drzewie binarnych wyszukiwań może być proporcjonalny do wysokości drzewa, która w drzewie o n wierzchołkach może wynosić nawet n. Jeśli jednak zażądamy, aby różnica wysokości poddrzew dowolnego wierzchołka była nie większa niż 1, to drzewo będzie istotnie niższe. Niech $C(k)$ oznacza minimalną liczbę wierzchołków drzewa AVL o wysokości k. Drzewo o wysokości 1 ma zawsze 1 wierzchołek (korzeń), a drzewo o wysokości 2 ma co najmniej 2 wierzchołki. Dla $k > 2$ mamy

$$C(k) = C(k-1) + C(k-2) + 1,$$

ponieważ drzewo o wysokości k ma najmniejszą możliwą liczbę wierzchołków wtedy, gdy oba poddrzewa korzenia mają różne wysokości i każde z nich ma najmniejszą dopuszczalną liczbę wierzchołków. Równanie $a_k = a_{k-1} + a_{k-2} + 1$ jest spełnione przez ciąg stały $a_k = -1$. Aby uzyskać rozwiązanie spełniające warunki początkowe, przyjmujemy

$$C(k) = e\lambda_1^k + f\lambda_2^k - 1$$

i obliczamy e i f, rozwiązując układ równań

$$C(1) = \lambda_1 e + \lambda_2 f - 1 = 1,$$
$$C(2) = \lambda_1^2 e + \lambda_2^2 f - 1 = 2,$$

w którym występują te same liczby λ_1 i λ_2, co w hodowli królików. Dostajemy stąd $e = \frac{1}{10}(5 - 3\sqrt{5})$ i $f = \frac{1}{10}(5 + \sqrt{5})$. Liczbę n wierzchołków w drzewie AVL możemy oszacować jak następuje:

$$n \geqslant C(k) = e\lambda_1^k + f\lambda_2^k - 1 > (e+f)\lambda_2^k - 1 = \lambda_2^k - 1,$$

ponieważ $f > -e > 0$, $e + f = 1$ oraz $\lambda_2 > -\lambda_1 > 0$. Zatem $\lambda_2^k < n + 1$, czyli wysokość drzewa

$$k < \log_{\lambda_2}(n+1).$$

Dalsze przykłady można mnożyć (prawie) jak króliki. Kto umie napisać i rozwiązać równanie różnicowe, temu analiza złożoności wielu pozornie całkiem różnych algorytmów niestraszna. Oczywiście, nie brakuje też takich algorytmów, których analiza jest straszna. Co kto lubi...