Zadanie sortowania teoretycznego

Doc. dr Antoni KRECZMAR

Mamy danych n różnych liczb $x_1, \ldots, x_n$ i chcemy je uporządkować, tzn. ustawić w ciąg $y_1 < y_2 < \ldots < y_n$. Ponadto jedyną operacją, którą możemy na tych liczbach wykonywać, jest operacja porównania dwu liczb. Ile takich porównań trzeba wykonać, aby dane liczby uporządkować?

Jeden z najprostszych algorytmów polega na sukcesywnym wstawianiu do już uporządkowanego początkowego ciągu $y_1 < \ldots < y_k$ $(1 \leqslant k \leqslant n-1)$ elementu x_{k+1}, porównując go kolejno z $y_1, \ldots, y_k$, aż natrafimy na właściwe miejsce. Może się jednak zdarzyć, że takie wstawienie będzie wymagało porównania x_{k+1} ze wszystkimi elementami $y_1, \ldots, y_k$, gdy x_{k+1} będzie większe od y_k. Zatem liczba porównań przy takim postępowaniu będzie co najwyżej równa

$$\sum_{k=2}^{n} (k-1) = \frac{n(n-1)}{2}.$$

Sam proces wstawiania można jednak łatwo przyspieszyć. Zauważył to po raz pierwszy polski matematyk Hugo Steinhaus. Mianowicie zamiast porównywać x_{k+1} po kolei z $y_1, \ldots, y_k$ można porównać x_{k+1} z elementem środkowym ciągu $y_1, \ldots, y_k$, czyli z $y_{\lfloor (k+1)/2 \rfloor}$ (gdzie $\lfloor r \rfloor$ oznacza największą liczbę całkowitą j nie większą niż r). W zależności od wyniku porównania x_{k+1} z $y_{\lfloor (k+1)/2 \rfloor}$ wstawiamy x_{k+1} albo do ciągu $y_1, \ldots, y_{\lfloor (k+1)/2 \rfloor}$ albo do ciągu $y_{\lfloor (k+1)/2 \rfloor + 1}, \ldots, y_k$, ale każde takie wstawienie też wykonujemy metodą porównania z elementem środkowym. Metodę tę nazywa się bisekcją (metodą dzielenia na połowy). Ile porównań wystarczy do wstawienia x_{k+1} w ciąg $y_1, \ldots, y_k$ tą metodą? Łatwo zauważyć, że gdy $k+1$ jest potęgą dwójki, tzn. $k+1 = 2^i$, wówczas wstawianie bisekcją kosztuje i porównań. Na przykład dla $k = 3$ mamy dwa porównania (najpierw z y_2, potem z y_1 lub y_3), a dla $k = 7$ mamy trzy porównania (najpierw z y_4, a potem wstawiamy w y_1, y_2, y_3 lub y_5, y_6, y_7). Dla $k+1 = 2^i$ bisekcja za każdym razem dzieli dany ciąg na dwie równe części, dzięki czemu liczba porównań zawsze wynosi i. Jeżeli jednak $k+1$ nie jest potęgą dwójki, to takie dzielenie na dwie części nie zawsze da dwa podciągi o tej samej długości. Jaka będzie wówczas maksymalna liczba porównań? Weźmy najbliższą potęgę dwójki, większą niż $k+1$. Czyli bierzemy najmniejsze i, takie że $2^i > k+1$. Liczba porównań wykonywanych przy stosowaniu algorytmu bisekcji nie może być większa niż to i. Zatem wstawienie x_{k+1} metodą bisekcji w ciąg $y_1, \ldots, y_k$ kosztuje co najwyżej $\lceil \log_2(k+1) \rceil$ porównań, gdzie $\lceil r \rceil$ oznacza najmniejszą liczbę całkowitą j nie mniejszą niż r. Wstawiając więc w każdym kroku do ciągu $y_1 < \ldots < y_k$ element x_{k+1} metodą bisekcji otrzymamy algorytm sortowania o koszcie co najwyżej

$$\sum_{k=2}^{n} \lceil \log_2(k) \rceil$$

porównań. Czytelnik z łatwością sam udowodni, że powyższa suma równa się

$$n \cdot \lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1,$$

czyli że algorytm Steinhausa jest znacznie lepszy od algorytmu przedstawionego na samym początku.

W latach pięćdziesiątych trwało poszukiwanie jeszcze lepszego algorytmu. Odkryli taki algorytm Lester Ford i Selmer Johnson. Ma on trzy fazy. W pierwszej fazie dzieli się ciąg na $\lfloor n/2 \rfloor$ par, zostawiając ewentualnie jeden element na boku i porządkuje się każdą parę. W fazie drugiej stosując rekurencyjnie algorytm Forda-Johnsona porządkuje się elementy większe par. Po tych dwu fazach otrzymamy wynik, który można zilustrować diagramem (gdzie strzałka oznacza $<$):

$$\begin{array}{ccccccccccccc} a_1 & \to & a_2 & \to & a_3 & \to & a_4 & \to & a_5 & \ldots\ldots & a_i & & i = \lfloor n/2 \rfloor \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \\ b_1 & \to & b_2 & \to & b_3 & \to & b_4 & \to & b_5 & \ldots\ldots & b_i & b_{i+1} & \end{array}$$

W fazie trzeciej musimy elementy $b_2, b_3, b_4, b_5, \ldots, b_i, b_{i+1}$ wstawić do uporządkowanego ciągu $b_1, a_1, a_2, a_3, a_4, a_5, \ldots, a_i$. Teraz następuje najważniejszy moment algorytmu. Przypomnijmy, że bisekcja działa najskuteczniej, gdy ciąg wraz ze wstawianym elementem ma długość, która jest potęgą dwójki. Wstawienie bisekcją b_2 w ciąg b_1, a_1 wymaga dwu porównań, ale wówczas wstawienie b_3 w tak otrzymany ciąg czteroelementowy wymagałoby już trzech porównań.

Zamieńmy tę kolejność wstawiania. Najpierw wstawiamy b_3 w ciąg b_1, a_1, a_2, co kosztuje dwa porównania, następnie wstawiamy b_2 w ciąg co najwyżej trzyelementowy (dojdzie, być może, b_3, ale a_2 i a_3 są większe od b_2), co też kosztuje tylko trzy porównania. Po wstawieniu bisekcją tych dwu elementów otrzymamy diagram:

$$\begin{array}{ccccccccccccc} c_1 & \to & c_2 & \to & c_3 & \to & c_4 & \to & c_5 & \to & c_6 & \to & a_4 & \to & a_5 & \ldots\ldots & a_i \\ & & & & & & & & & & & & \uparrow & & \uparrow & & \\ & & & & & & & & & & & & b_4 & & b_5 & \ldots\ldots & b_i \quad b_{i+1} \end{array}$$

gdzie $c_1, c_2, c_3, c_4, c_5, c_6$ jest uporządkowanym ciągiem elementów $a_1, a_2, a_3, b_1, b_2, b_3$. Następną potęgą dwójki jest $8 = 2^3$. Jakie elementy można wstawić za pomocą trzech porównań? Otóż wstawiając b_5 w ciąg $c_1, \ldots, c_6, a_4$ wykonujemy trzy porównania i wstawiając b_4 w $c_1, \ldots, c_6$ powiększony być może o b_5, też wykonujemy tylko trzy porównania.

Następną potęgą dwójki będzie $16 = 2^4$. Licząc skrupulatnie stwierdzimy, że elementy $b_{11}, b_{10}, b_9, b_8, b_7, b_6$ można wstawić bisekcją, każdy za pomocą czterech porównań, o ile będą one wstawiane w takiej odwrotnej kolejności. Albowiem wstawienie b_{11} przedłuży, być może, ciąg początkowy, ale przechodząc do wstawiania b_{10} wiemy, że $b_{10} < a_{10}$, co zapewnia, że bisekcja będzie w dalszym ciągu kosztować co najwyżej cztery porównania itd.

Poniżej przedstawiamy tabelkę kosztów trzech omówionych algorytmów dla początkowych wartości n.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
bezpośredni	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136	153
Steinhausa	3	5	8	11	14	17	21	25	29	33	37	41	45	49	54	59
Forda-Johnsona	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50	54

Wyznaczenie wzorem kosztu działania algorytmu Forda-Johnsona jest zadaniem trudnym. Podamy tylko, że koszt ten można wyrazić sumą:

$$\sum_{k=2}^{n} \lceil \log_2(3k/4) \rceil,$$

co po zwinięciu daje:

$$n \cdot \lceil \log_2(3n/4) \rceil - \lfloor 2^{\lfloor \log_2(6n) \rfloor}/3 \rfloor + \lfloor \log_2(6n) \rfloor/2 \rfloor.$$

Dopiero w roku 1981 wykazano, że istnieje algorytm jeszcze lepszy, jednak nawet dla wytrawnych informatyków zrozumienie jego struktury jest poważnym problemem.