Sortowanie z minimalną liczbą porównań

Marcin PECZARSKI

Sortowanie skończonego zbioru polega na ustawieniu go w ciąg według niemalejącej wartości jakiejś cechy. Na przykład zbiór znaczków pocztowych możemy ustawić według ich wartości. Jedną z metod sortowania jest wykonanie pewnej liczby porównań par elementów. Pojedyncze porównanie dostarcza nam informacji, który z dwóch porównywanych elementów powinien znaleźć się wcześniej w tworzonym ciągu. Gdyby zdarzyło się, że porównywane elementy mają tę samą wartość rozważanej cechy, to ustalamy ich kolejność w dowolny sposób. Z pewnością uda się posortować dany zbiór, gdy porównamy każdy element z każdym. Dla zbioru n-elementowego oznacza to konieczność wykonania

$$\frac{n(n-1)}{2}$$

porównań. Tyle w pesymistycznym przypadku potrzebuje algorytm *sortowania przez wstawianie*. Działa on w ten sposób, że kolejno dla $k = 1, 2, \ldots, n$ do posortowanego już ciągu $k-1$ elementów wstawiamy k-ty element, porównując go (w najgorszym przypadku) ze wszystkimi dotychczas rozważonymi elementami. Można lepiej. Aby wstawić element x do posortowanego ciągu $u_1, u_2, \ldots, u_j$, porównujemy go najpierw z elementem „środkowym” u_i, gdzie $i = \lceil j/2 \rceil$. Jeśli x poprzedza u_i, to z przechodniości relacji porządku wiemy, że poprzedza wszystkie elementy na prawo od niego (o większych indeksach) i musimy wstawić x do ciągu $u_1, u_2, \ldots, u_{i-1}$. Analogicznie, gdy u_i poprzedza x, to x powinien znaleźć się na prawo od u_i, czyli musimy go wstawić do ciągu $u_{i+1}, u_{i+2}, \ldots, u_j$. Powtarzamy powyższą procedurę, za każdym razem wybierając nowy element środkowy. Każde porównanie zmniejsza co najmniej dwukrotnie długość ciągu, do którego należy wstawić x. W pewnym momencie ciąg ten będzie miał długość 0. Oznacza to, że x znalazł się na właściwej pozycji. Dla wstawienia k-tego elementu potrzeba co najwyżej $\lceil \lg_2 k \rceil$ porównań. *Sortowanie ze wstawianiem binarnym* wymaga zatem

$$B_n = \sum_{k=1}^{n} \lceil \lg_2 k \rceil = n\lceil \lg_2 n \rceil - 2^{\lceil \lg_2 n \rceil} + 1$$

porównań. Dodajmy, że stosowany praktycznie algorytm *quicksort* dla dużych n wykonuje średnio około $1{,}4 \cdot n \lg_2 n$ porównań, choć sporadycznie w pesymistycznym przypadku potrzebuje ich około $n^2/2$.

Hugo Steinhaus postawił problem znalezienia minimalnej liczby porównań S_n zawsze wystarczającej do posortowania n elementów. Każde porównanie może dać dwa wyniki. Zatem algorytm wykonujący co najwyżej S_n porównań może zakończyć się co najwyżej 2^{S_n} różnymi wynikami. Z drugiej strony mamy $n!$ możliwych uporządkowań (permutacji) n elementów. Stąd $2^{S_n} \geqslant n!$, czyli $S_n \geqslant \lceil \lg_2 n! \rceil$. Liczbę $C_n = \lceil \lg_2 n! \rceil$ nazywamy *teorioinformacyjną dolną granicą sortowania*. Jak bardzo możemy zbliżyć się do tej granicy?

Wiadomo, że $B_n > C_n$ już dla $n > 4$. Lester Ford Jr i Selmer Johnson opublikowali w 1959 roku algorytm, który potrzebuje

$$F_n = \sum_{k=1}^{n} \lceil \lg_2 \tfrac{3}{4} k \rceil$$

porównań. Zatem $C_n \leqslant S_n \leqslant F_n$. Dla $n \leqslant 11$ i $n = 20, 21$ problem został rowiązany, gdyż wtedy $C_n = F_n$. Dla $12 \leqslant n \leqslant 19$ zachodzi $F_n = C_n + 1$. Jak wyznaczyć kolejne wartości S_n? Zaprzęgnijmy do pracy komputer. Aby formalnie opisać proces sortowania i stosowane metody, będziemy potrzebowali kilku pojęć.

Definicja 1. Porządkiem (częściowym) zbioru U nazywamy relację $r \subseteq U \times U$, która jest:
- zwrotna – $(u, u) \in r$ dla każdego $u \in U$;
- przechodnia – jeżeli $(u, v) \in r$ i $(v, w) \in r$, to $(u, w) \in r$ dla każdego $u, v, w \in U$;
- antysymetryczna – jeżeli $(u, v) \in r$ i $(v, u) \in r$, to $u = v$ dla każdego $u, v \in U$.

Definicja 2. Porządkiem liniowym zbioru U nazywamy relację $r \subseteq U \times U$, która jest porządkiem i jest spójna – $(u, v) \in r$ lub $(v, u) \in r$ dla każdego $u, v \in U$.

Definicja 3. Porządkiem dualnym do porządku r nazywamy relację $r^* = \{(u, v) : (v, u) \in r\}$.

Definicja 4. Porządek r_1 zbioru U_1 i porządek r_2 zbioru U_2 nazywamy izomorficznymi, jeśli istnieje funkcja różnowartościowa $f\colon U_1 \to U_2$ przekształcająca U_1 na U_2 spełniająca ponadto dla dowolnych $u, v \in U_1$ następujący warunek: $(u, v) \in r_1$ wtedy i tylko wtedy, gdy $(f(u), f(v)) \in r_2$.

Definicja 5. Jeżeli r jest porządkiem zbioru U oraz $(u, v) \notin r$ i $(v, u) \notin r$, to przez $r + uv$ oznaczamy porządek $r \cup \{(x, y) : (x, u) \in r \wedge (v, y) \in r\}$ zbioru U. Mówimy, że $r + uv$ jest domknięciem przechodnim zbioru $r \cup \{(u, v)\}$.

Niech $U = \{u_1, u_2, \ldots, u_n\}$ będzie sortowanym zbiorem. Sortowanie rozpoczyna się od totalnego nieporządku $r_0 = \{(u_1, u_1), (u_2, u_2), \ldots, (u_n, u_n)\}$, czyli w sytuacji, gdy nie znamy relacji pomiędzy żadną parą różnych elementów ze zbioru U. Niech stan sortowania po wykonaniu pewnej liczby porównań reprezentowany będzie przez porządek r. W następnym kroku wybieramy do porównania elementy u_j i u_k, których wzajemnego uporządkowania jeszcze nie znamy, czyli takie że $(u_j, u_k) \notin r$ i $(u_k, u_j) \notin r$. Powiedzmy, że jako wynik porównania uzyskujemy informację: element u_j poprzedza element u_k. Teraz stan sortowania reprezentowany jest przez nowy porządek $r + u_j u_k$. Sortowanie kończy się, gdy uzyskamy porządek liniowy, czyli gdy dla dowolnej pary (u, v) wiemy, czy jest ona w relacji, czy też w relacji jest (v, u).

Naiwna metoda jest następująca. Zaczynamy od totalnego nieporządku r_0. Sprawdzamy rekurencyjnie,

czy możemy go posortować za pomocą C_n porównań. Aktualnie rozważany porządek r można posortować za pomocą c porównań wtedy i tylko wtedy, gdy $c \geqslant 0$ i r jest już posortowany (innymi słowy r jest porządkiem liniowym) albo istnieje para (u_j, u_k), taka że $(u_j, u_k) \notin r$, $(u_k, u_j) \notin r$ i porządki $r + u_j u_k$, $r + u_k u_j$ można posortować za pomocą $c - 1$ porównań. Niestety liczba przypadków, które należy rozważyć jest tak ogromna, że dla $n = 12$ nie uda się zakończyć sprawdzania w rozsądnym czasie.

Można to zrobić lepiej. Mówimy, że porządki r_1 i r_2 są *przystające*, jeśli są izomorficzne lub porządek r_1 jest izomorficzny z porządkiem dualnym do porządku r_2. Zauważmy, że jeśli porządki r_1 i r_2 są przystające, to do posortowania wymagają tej samej liczby porównań. Eliminowanie przystających porządków pozwala istotnie zmniejszyć liczbę rozważanych przypadków. Liczbę sposobów, na jakie możemy posortować porządek r nazywamy jego *liczbą rozszerzeń liniowych* i oznaczamy przez $e(r)$. Jeżeli $e(r) > 2^c$, to analogiczne do użytego dla wyznaczenia teorio-informacyjnej granicy rozumowanie daje, że porządku r nie można posortować za pomocą c porównań. Zauważmy ponadto, że jeśli dla danej pary (u_j, u_k) jeden z porządków $r + u_j u_k$, $r + u_k u_j$ okaże się niesortowalny, to drugiego nie musimy już sprawdzać.

Powyższe spostrzeżenia wykorzystał Mark Wells, aby wyznaczyć S_{12}. Zaczynamy od zbioru P_0 zawierającego totalny nieporządek r_0. Następnie dla $c = 1, 2, \ldots, C_n$ tworzymy zbiory P_c porządków, które mogą powstać po c porównaniach. Robimy to tak, aby z niemożliwości posortowania wszystkich porządków z P_c za pomocą $C_n - c$ porównań wynikała niemożliwość posortowania wszystkich porządków z P_{c-1} za pomocą $C_n - c + 1$ porównań. Utożsamiamy przy tym porządki przystające. Dla każdego porządku $r \in P_{c-1}$ rozważamy wszystkie pary (u_j, u_k), których wzajemne uporządkowanie jest jeszcze nieznane, czyli $(u_j, u_k) \notin r$ i $(u_k, u_j) \notin r$. Niech

$$r_1 = r + u_j u_k, \qquad r_2 = r + u_k u_j.$$

Jeśli $r_1 \notin P_c$, $r_2 \notin P_c$, $e(r_1) \leqslant 2^{C_n - c}$ i $e(r_2) \leqslant 2^{C_n - c}$, to dodajemy do P_c ten z porządków r_1 lub r_2, który ma większą liczbę rozszerzeń liniowych. Jest to uzasadnione przypuszczeniem, że porządek o większej liczbie rozszerzeń liniowych jest trudniejszy do posortowania. Remisy rozstrzygamy dowolnie. Jeśli w wyniku takiego postępowania pewien zbiór P_c okaże się pusty, to żaden z porządków ze zbioru P_{c-1} nie może być posortowany za pomocą $C_n - c + 1$ porównań. W konsekwencji r_0 nie może być posortowany za pomocą C_n porównań, czyli $S_n > C_n$. Mark Wells przeprowadził stosowny eksperyment w 1965 roku i po około 60 godzinach obliczeń komputera Maniac II odkrył, że $S_{12} = F_{12} = 30$.

Wadą metody Wellsa jest to, że może ona wykazać tylko nieistnienie algorytmu sortującego. Dla każdego $r \in P_c$ zachodzi $2^{C_n - c - 1} < e(r) \leqslant 2^{C_n - c}$. Ponadto $e(r) = 1$ wtedy i tylko wtedy, gdy r jest porządkiem liniowym. Zatem jeśli $P_{C_n} \neq \emptyset$, to P_{C_n} zawiera jakiś porządek liniowy jako jedyny element (wszystkie porządki liniowe są izomorficzne). Niestety nie można na tej podstawie wnioskować o istnieniu algorytmu sortującego. Taka sytuacja ma miejsce dla $n = 13, 14$.

Zaproponowałem następujące rozwiązanie. Wyznaczamy podzbiory $P_c^* \subseteq P_c$ porządków sortowalnych za pomocą $C_n - c$ porównań. Zaczynamy od $P_{C_n}^* = P_{C_n}$. Następnie kolejno dla $c = C_n - 1, C_n - 2, \ldots, 0$ sprawdzamy wszystkie porządki ze zbioru P_c. Dla każdego $r \in P_c$ rozważamy wszystkie pary (u_j, u_k), dla których $(u_j, u_k) \notin r$ i $(u_k, u_j) \notin r$. Niech jak poprzednio $r_1 = r + u_j u_k$, $r_2 = r + u_k u_j$. Porządek r dodajemy do zbioru P_c^* wtedy i tylko wtedy, gdy istnieje taka para (u_j, u_k), że $r_1 \in P_{c+1}^*$ i dodatkowo $r_2 \in P_{c+1}^*$ lub $r_2 \notin P_{c+1}^*$ ale r_2 jest sortowalny za pomocą $C_n - c - 1$ porównań, co sprawdzamy rekurencyjnie. Jeśli któryś ze zbiorów P_c^* okaże się pusty, to algorytm sortujący nie istnieje. W przeciwnym przypadku poszukiwany algorytm istnieje i analiza zbiorów P_c^* umożliwi jego skonstruowanie.

Eksperymenty przeprowadzone dla $n = 13, 14$ w latach 2002–2003 wykazały, że w obu przypadkach $P_{15}^* = \emptyset$. Zatem $S_{13} > C_{13} = 33$ i $S_{14} > C_{14} = 37$, a ponieważ $F_{13} = 34$ i $F_{14} = 38$, więc $S_{13} = 34$ i $S_{14} = 38$. Ponadto dla $n = 22$ eksperyment pokazał, że $P_{40} = \emptyset$, czyli

$$C_{22} = 70 < S_{22} = F_{22} = 71.$$

Eksperymenty dla $n = 12, 13, 14, 22$ wykonywały się na moim komputerze odpowiednio 3 s, 41 min, 392 godz., 155 dni.

Na swoich odkrywców czekają S_{15}, S_{16}, …, S_{19}, S_{23}, S_{24}, … Otwartym problemem pozostaje też znalezienie najmniejszego n, dla którego istnieje algorytm wymagający mniej porównań niż algorytm Forda–Johnsona. Aktualny wynik pochodzi z 1981 roku i należy do Jürgena Schulte Möntinga, który znalazł algorytm sortujący 47 elementów za pomocą 200 porównań, podczas gdy $F_{47} = 201$ ($C_{47} = 198$). Istnieje uzasadnione przypuszczenie, że szukanym minimum może być 15 lub 16 ($C_{15} = 41$, $F_{15} = 42$, $C_{16} = 45$, $F_{16} = 46$).

Duże znaczenie ma efektywna implementacja przedstawionych metod. Szczególnie trudnymi problemami algorytmicznymi są rozpoznawanie izomorfizmu porządków i zliczanie rozszerzeń liniowych. Istotne przyspieszenie uzyskano stosując różne heurystyki, których nie sposób tu opisać. Zainteresowani mogą ściągnąć kod źródłowy ze strony autora `http://www.mimuw.edu.pl/~marpe` i sami poeksperymentować albo mogą też sięgnąć po książkę Donalda Knutha [Sztuka programowania, Tom 3, Sortowanie i wyszukiwanie, WNT 2002], gdzie znajdą wiele dodatkowych informacji i problemów dotyczących sortowania.