Informatyczny kącik olimpijski (52): Robot sortujący

Tym razem omówimy zadanie *Robot sortujący* (ang. *Robotic Sort*) z Mistrzostw Europy Środkowej w Programowaniu Zespołowym 2007 (CERC 2007). W zadaniu występuje nietypowy algorytm sortowania n-elementowego ciągu liczb całkowitych. W i-tym kroku (dla $i = 1, \ldots, n$) znajdujemy w ciągu i-ty najmniejszy element; załóżmy, że znajduje się on na pozycji j_i. Następnie odwracamy fragment ciągu znajdujący się między pozycjami i-tą a j_i-tą i w ten sposób rozważany element trafia na docelową pozycję w posortowanym ciągu. Naszym zadaniem jest efektywnie zasymulować ten algorytm, a dokładniej, wyznaczyć ciąg pozycji $j_1, \ldots, j_n$. Przykładowo, jeśli ciągiem do posortowania jest 3, 4, 5, 1, 6, 2, to wynikowy ciąg pozycji odpowiadających prawym końcom kolejnych odwróceń to 4, 6, 4, 5, 6, 6. Dla uproszczenia założymy, że wyjściowy ciąg jest permutacją liczb $1, \ldots, n$. W oryginalnym zadaniu ciąg nie musiał być różnowartościowy, ale sprowadzenie ogólnego przypadku do przypadku permutacji jest całkiem proste.

Bezpośrednia implementacja algorytmu sortowania przez odwracanie ma złożoność czasową $\Theta(n^2)$. Można się o tym przekonać, sprawdzając działanie algorytmu dla ciągu 2, 4, 6, 8, ..., 7, 5, 3, 1 – właśnie w tym przypadku wykonujemy największą liczbę operacji.

W takim razie spróbujemy zastosować jakieś inne podejście. Ponumerujmy elementy ciągu po kolei, od lewej do prawej, od 1 do n. Załóżmy, że najmniejszy element ciągu znajduje się na pozycji a. Wówczas po pierwszym odwróceniu elementy ciągu będą ustawione w następującej kolejności:

$$a,\ a-1,\ \ldots,\ 1,\ a+1,\ a+2,\ \ldots,\ n.$$

Widzimy, że tak naprawdę struktura ciągu zmieniła się dosyć nieznacznie. Jeśli teraz drugi najmniejszy element znajdował się, przykładowo, na pozycji $b < a$, to po drugim odwróceniu ciąg będzie miał postać:

$$a,\ b,\ b+1,\ \ldots,\ a-1,\ b-1,\ b-2,\ \ldots,\ 1,\ a+1,\ a+2,\ \ldots,\ n.$$

Domyślamy się już, co będzie dalej. Po i-tym odwróceniu ciąg będzie miał postać

$$s_1,\ s_2,\ \ldots,\ s_i,\ (a_1, b_1, r_1),\ (a_2, b_2, r_2),\ \ldots,\ (a_m, b_m, r_m),$$

dla pewnego $m \leqslant i$, przy czym $s_1, \ldots, s_i$ to numery i najmniejszych elementów ciągu, zaś (a_j, b_j, r_j) oznacza ciąg liczb a_j, $a_j + 1$, ..., b_j ustawionych w takim właśnie porządku (jeśli $r_j = 0$) lub odwrotnie, malejąco (jeśli $r_j = 1$).

Zauważmy, że używając takiej reprezentacji, kolejne odwrócenie w ciągu możemy wykonać w czasie $O(m)$, o ile tylko będziemy dla każdego elementu $1, \ldots, n$ pamiętać, na której pozycji wyjściowego ciągu się znajdował (oznaczenie: $p[i]$). Załóżmy, że $p[i+1] = k$. Znajdujemy teraz taką grupę (a_j, b_j, r_j), że $k \in [a_j, b_j]$, rozbijamy ją na dwie, a następnie między te dwie części wstawiamy, odwrócone, wszystkie wcześniejsze grupy. Przykładowo, jeśli $r_j = 0$, to wynikowy ciąg ma postać:

$$s_1,\ s_2,\ \ldots,\ s_i,\ k,\ (a_j, k-1, 1),\ (a_{j-1}, b_{j-1}, 1-r_{j-1}),\ \ldots,\ (a_1, b_1, 1-r_1),$$
$$(k+1, b_j, 0),\ (a_{j+1}, b_{j+1}, r_{j+1}),\ \ldots,\ (a_m, b_m, r_m).$$

Przypadek $r_j = 1$ rozpatrujemy analogicznie. Po każdej operacji przybywa co najwyżej jedna grupa – być może zero, jeśli k znajdowało się na skraju grupy (a_j, b_j, r_j). To jednak oznacza, że liczba grup może rosnąć liniowo, więc koszt czasowy całego algorytmu może być rzędu $\Theta(n^2)$. Pozostawiamy Czytelnikowi znalezienie złośliwego przykładu, wymuszającego wykonanie takiej liczby operacji – co ciekawe, poprzednio podany złośliwy przykład nie ma tej własności.

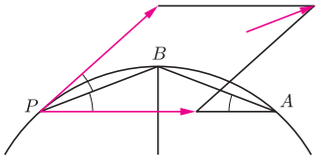
Mogłoby się wydawać, że ponieśliśmy porażkę – tyle wysiłku, a na końcu i tak algorytm jest kwadratowy. Możemy jednak osiągnąć lepszą złożoność czasową, jeśli tylko dostrzeżemy mocną stronę naszego algorytmu: otóż na samym początku jest on bardzo szybki. Koszt pierwszych k kroków algorytmu to $O(k^2)$. Przyjmijmy $k = \lfloor \sqrt{n} \rfloor$; wówczas podane kroki wykonujemy w czasie $O(n)$. Kolejne kroki byłyby bardziej kosztowne, dlatego zamiast nich zrobimy... porządek. Możemy mianowicie wypisać *explicite* wszystkie wyrazy ciągu powstałego po k odwróceniach, ponumerować je kolejno, wyznaczyć od nowa tablicę indeksów poszczególnych elementów w ciągu (p) i kolejne odwrócenia wykonywać już za pomocą nowego ciągu.

Uporządkowanie ciągu po $\sqrt{n}$ odwróceniach zajmuje czas $O(n)$, taki sam jak symulacja tych $\sqrt{n}$ odwróceń. W trakcie całego algorytmu takich większych faz musimy wykonać mniej więcej $\sqrt{n}$, co daje łączną złożoność czasową $O(n\sqrt{n})$.

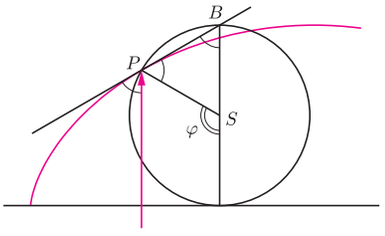
Na koniec warto dodać, że całe zadanie można rozwiązać także w czasie $O(n \log n)$, używając struktur danych opartych na drzewach zrównoważonych. Jednak takie rozwiązanie nie było nigdy autorowi kącika do szczęścia potrzebne.

Jakub RADOSZEWSKI

Rozwiązanie zadania F 813.
Styczna do cykloidy w każdym punkcie P przechodzi przez najwyższy punkt wyznaczającego ją okręgu. Istotnie, ponieważ nie ma poślizgu, więc prędkości ruchu „obrotowego" i postępowego są równej długości, a ich wypadkowa (czyli wektor styczny) jest dwusieczną kąta między nimi.



Prosta pozioma przechodząca przez punkt P przecina okrąg w punkcie A. Pionowa średnica okręgu przecina go u góry w punkcie B. Z symetrii mamy $\sphericalangle PAB = \sphericalangle APB$, ale kąt PAB jako wpisany równy jest kątowi dopisanemu między cięciwą PB i styczną do okręgu w P. Zatem PB jest dwusieczną kąta między wektorami.



Rozpatrzmy punkt P powstały po obrocie okręgu o kąt φ. Niech S będzie środkiem okręgu. Wówczas $\sphericalangle SPB = \sphericalangle SBP = \varphi/2$. Pionowy promień jest równoległy do SB i tworzy ze styczną ten sam kąt co SB, a więc ten sam co SP, co kończy dowód.