Informatyczny kącik olimpijski (108): *Hacker Cups and Balls*

W tym miesiącu omówimy zadanie o zagadkowym tytule *Hacker Cups and Balls*. Pojawiło się ono na obozie w Petrozawodsku, przygotowującym do finałów zawodów ACM ICPC. Treść opowiada o kubkach i kulkach. Jednak tak naprawdę mamy daną nieparzystą liczbę n oraz ciąg A będący permutacją liczb od 1 do n. Na wejściowym ciągu A wykonano w podanej kolejności m operacji sortujących pewne przedziały jego elementów. Dla każdej z operacji określone jest, w jakim przedziale indeksów zostały posortowane elementy oraz czy zostały posortowane rosnąco, czy też malejąco. Musimy stwierdzić, jaka wartość znajduje się na środkowej pozycji w ciągu po wykonaniu wszystkich operacji.

Początkowo chciałoby się symulować cały proces, nie zważając na fakt, iż wystarczy odzyskać tylko jeden element ostatecznego ciągu. Jednak nie tędy droga, jeśli nie chcemy stracić mnóstwo czasu na zaimplementowanie rozwiązania.

Weźmy dowolną liczbę naturalną x i zastanówmy się, czy potrafimy sprawdzić czy szukany środkowy element ciągu po m operacjach jest większy niż x. Jeśli uda nam się na pytania tego typu odpowiedzieć w czasie $\mathcal{O}((n+m)\log n)$, to za pomocą wyszukiwania binarnego rozwiążemy całe zadanie w czasie $\mathcal{O}((n+m)\log^2 n)$.

Mając dany x, zapomnijmy o dokładnych wartościach elementów ciągu A. Będziemy po kolejnych operacjach pamiętać tylko, które elementy ciągu A są większe niż x. Po pierwsze, zauważmy, że tę informację istotnie *da się* utrzymywać. Istotne jest tylko to, ile w sortowanym przedziale jest elementów większych niż x: tyle ostatnich (pierwszych) elementów przedziału po posortowaniu rosnąco (malejąco) będzie większych niż x, pozostałe natomiast będą mniejsze bądź równe x. Zakodujemy tę informację jako ciąg zer i jedynek – jedynka oznacza, że dany element jest większy niż x. Nasze operacje wyglądają więc następująco: sumujemy jedynki w przedziale, a następnie podstawiamy na odpowiednim jego podprzedziale jedynki, a na pozostałej części zera.

Istnieją różne struktury danych, które pozwalają na zaimplementowanie takich operacji w czasie $\mathcal{O}((n+m)\log n)$.

Pierwszym przykładem takiej struktury jest drzewo przedziałowe. W danym wierzchołku drzewa pamiętamy liczbę jedynek w odpowiednim przedziale bazowym. Wystarczy ona do określania w czasie $\mathcal{O}(\log n)$ liczby jedynek w sortowanym przedziale. Potrzebujemy ponadto operacji podstawienia konkretnej stałej na wszystkich elementach przedziału. Przy jej wykonywaniu, po zaktualizowaniu wartości w odpowiednich przedziałach bazowych, informacje w ich potomkach przestają być aktualne. Jednak zawsze na ścieżce od korzenia do przedziału bazowego z nieaktualną wartością jest przedział cały wypełniony zerami bądź cały wypełniony jedynkami.

Wystarczy więc, że zaimplementujemy drzewo w sposób rekurencyjny i zawsze, gdy napotkamy przedział bazowy, który zawiera same zera bądź same jedynki, aktualizujemy wartość w jego dzieciach. W przypadku tego konkretnego zadania implementacja tego typu drzewa okazuje się jeszcze prostsza niż zwykle. Nie musimy bowiem pamiętać żadnych dodatkowych informacji. To, czy przedział jest cały wypełniony zerami bądź jedynkami, możemy sami wydedukować z liczby jedynek oraz długości przedziału.

Drugim podejściem, być może prostszym ideowo, ale wymagającym więcej ostrożności przy szczegółach implementacyjnych, jest utrzymywanie w strukturze `set` z języka C++ ciągu przedziałów zer oraz jedynek. Jedna operacja sortowania agreguje i usuwa wszystkie przedziały, które przecinają się z sortowanym. Następnie wstawia do struktury co najwyżej cztery nowe przedziały. Pierwszy i ostatni zawierają odpowiednio prefiks pierwszego i sufiks ostatniego agregowanego przedziału niemodyfikowany przez sortowanie. Dwa pozostałe zawierają odpowiedniej długości przedziały zer oraz jedynek powstałe przez sortowanie. Aby przeanalizować czas działania, musimy zamortyzować sumaryczny czas usuwania elementów. Zauważmy, że w trakcie działania algorytmu dodamy do struktury $\mathcal{O}(n+m)$ przedziałów, więc w sumie również co najwyżej tyle usuniemy i ostatecznie otrzymujemy złożoność $\mathcal{O}((n+m)\log n)$ na sprawdzenie jednej wartości x.

Te dwa rozwiązania wystarczają już do zaliczenia zadania w trakcie konkursu. Jednak czysto teoretycznie, w celach edukacyjnych, zarysujmy jeszcze rozwiązanie odzyskujące cały ostateczny ciąg i to w lepszej złożoności czasowej $\mathcal{O}((n+m)\log n)$. Pomysł będzie podobny do drugiej z naszych implementacji, ciąg A będzie podzielony na przedziały, z których każdy będzie jednym elementem struktury `set`. Każdy z tych przedziałów będzie odpowiadał monotonicznemu fragmentowi ciągu A. Oprócz tego, jaką monotoniczność ma dany przedział, będziemy również utrzymywać dokładny opis zbioru jego elementów na dynamicznym drzewie przedziałowym. Okazuję się, że łączenie, a w przypadku tego zadania i dzielenie takich dynamicznych drzew przedziałowych, nawet w dość bezpośredni sposób, niespodziewanie amortyzuje się do zadowalającego czasu. Zachęcamy Najwytrwalszych Czytelników do przeanalizowania tego fenomenu. Właściwym potencjałem powinna okazać się sumaryczna liczba wierzchołków we wszystkich dynamicznych drzewach przedziałowych.

Marcin SMULEWICZ