Informatyczny kącik olimpijski (72): Multizbiory

Kącik powstał na podstawie wpisu na blogu autora: `http://fajnezadania.wordpress.com`. Blog zawiera opisy rozwiązań ciekawych i często niełatwych zadań „z algorytmiki i okolic".

W tym kąciku proponuję zadanie polecane przez mojego korespondenta w Jekaterynburgu, mieście znanym również z turnieju Ural Sport Programming Championship, którego zeszłoroczną atrakcją był bezwzględny pojedynek pięciu najlepszych drużyn z Rosji z pięcioma najlepszymi drużynami z Chin. Popatrzmy na zadanie, którego nie udało się rozwiązać żadnej z nich!

Mamy dany ciąg liczb $S_1, S_2, \ldots, S_n$. Rozważamy jego wszystkie spójne podciągi, czyli fragmenty postaci $S_i, S_{i+1}, \ldots, S_j$, które sortujemy w dość dziwny sposób. Mianowicie, aby porównać dwa fragmenty X i Y, szukamy najmniejszej liczby a, która występuje różną liczbę razy w X i Y. Fragment X jest mniejszy niż Y (co zapisujemy jako $X \prec Y$) dokładnie wtedy, gdy owa liczba a występuje więcej razy w X niż w Y. Jeśli takiej liczby nie ma, to fragmenty są dla nas takie same; innymi słowy porównujemy *multizbiory* liczb występujące w obu fragmentach. Naszym zadaniem jest wyznaczenie multizbioru, który jest generowany przez k-ty fragment w tym dziwnym porządku.

Przykładowo $2, 2, 3, 1 \prec 1, 4, 2, 5, 2$, gdyż 3 (najmniejsza liczba występująca różną liczbę razy w tych fragmentach) występuje częściej w pierwszym z nich. Z kolei fragmenty $1, 3, 3, 2, 1$ i $2, 3, 1, 3, 1$ są dla nas takie same.

Pewnie warto przez chwilę zwątpić, czy ten porządek jest faktycznie porządkiem, czyli czy $X \prec Y \prec Z$ implikuje, że $X \prec Z$. W przeciwnym przypadku ciężko byłoby bowiem mówić o sortowaniu. Na szczęście, okazuje się, że tak jest w istocie ($\prec$ jest w rzeczywistości porządkiem leksykograficznym na fragmentach potraktowanych jako multizbiory). Ograniczenia podane w treści to $n \leqslant 150\,000$ i $k \leqslant \frac{n(n-1)}{2}$, czyli wygenerowanie i posortowanie wszystkich fragmentów nie jest najlepszym z możliwych pomysłów. Ba, problematyczne byłoby już nawet samo wygenerowanie fragmentów, nie mówiąc o tym, że porównanie dwóch fragmentów wydaje się wymagać czasu proporcjonalnego do ich długości. Zaczyna się robić ciekawie!

Co prawda, interesuje nas wyznaczenie k-tego fragmentu, ale może wypadałoby urealnić oczekiwania i zacząć od próby skonstruowania efektywnego sposobu zliczania fragmentów, które są ściśle mniejsze od danego? Jest to dość standardowe podejście: zamiast rozwiązywać problem „znajdź najmniejsze dobre rozwiązanie", zajmujemy się problemem „sprawdź, czy dane rozwiązanie jest dobre". Potem zwykle wystarczy tylko zastosować wyszukiwanie binarne, choć w naszym przypadku sytuacja okaże się odrobinę bardziej skomplikowana.

Mając dany fragment X, chcemy zliczyć fragmenty $Y = S_i, S_{i+1}, \ldots, S_j$, dla których $Y \prec X$. Dość naturalne jest ustalenie i i przyjrzenie się wszystkim j, które spełniają żądany warunek. Po chwili namysłu można dostrzec, że dodając kolejne elementy, możemy tylko zmniejszyć a, które ma więcej wystąpień, zatem $S_i, S_{i+1}, \ldots, S_{j+1} \preceq S_i, S_{i+1}, \ldots, S_j$. Wynika stąd, że j spełniające warunek tworzą spójny przedział $[f_i, n]$. Ale jak wyznaczyć to f_i? Pewnie moglibyśmy zacząć od $f_i = i$ i zwiększać je o jeden, dopóki $X \preceq S_i, S_{i+1}, \ldots, S_{f_i}$. Brzmi to całkiem rozsądnie, choć pojawiają się co najmniej dwa problemy. Przede wszystkim potrzebujemy efektywnej metody na sprawdzanie, czy aktualne f_i należy zwiększyć o jeden. Jest to jednak dość łatwe: trzeba tylko skonstruować strukturę danych, która umożliwi nam przechowywanie dla każdego a różnicy między liczbą jego wystąpień w X i liczbą jego wystąpień w aktualnym fragmencie $S_i, S_{i+1}, \ldots, S_{f_i}$, oraz znajdowanie najmniejszego a, dla którego ta różnica nie jest zerem. Musi również umożliwiać dodawanie nowych elementów S_j (co wiąże się ze zmniejszaniem odpowiadających im różnic). Taką strukturą może być zwykłe drzewo licznikowe, które umożliwia wykonywanie zarówno aktualizacji, jak i pytań w czasie $O(\log n)$. Dysponując takim narzędziem, będziemy w stanie wyznaczyć każde f_i w czasie $O(n \log n)$. Niby fajnie, ale w tym momencie pojawia się drugi problem: przecież mamy aż n różnych f_i, które wypadałoby znaleźć! Na szczęście można zauważyć, że $f_{i+1} \geqslant f_i$, czyli dla kolejnego i możemy zacząć od $f_{i+1} = f_i$. Jest to o tyle wygodne, że wspomniane przed chwilą drzewo licznikowe bez większych problemów pozwala także na zwiększenie i o jeden (czyli na usunięcie S_i z aktualnego fragmentu). Zatem dla kolejnych i zwiększamy f_i o jeden tak długo, jak aktualny fragment jest nie mniejszy niż X, a następnie usuwamy S_i ze struktury. Ponieważ $f_i \leqslant n$, wykonamy nie więcej niż $3n$ operacji na drzewie licznikowym, co daje sumaryczny czas $O(n \log n)$. Fantastycznie.

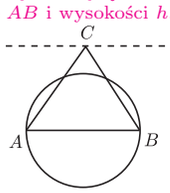
Rozwiązanie zadania M 1420. Rozważmy trójkąt ABC o wysokości opuszczonej z wierzchołka C długości h. Oznaczmy długości wysokości opuszczonych z wierzchołków B i C odpowiednio przez h_B i h_C, a jego pole przez S. Wiemy, że

$$2S = AC \cdot BC \cdot \sin C = \frac{2S}{h_B} \cdot \frac{2S}{h_A} \cdot \sin C,$$

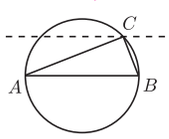
skąd

$$h_A h_B = 2S \sin C.$$

Ponieważ pole trójkąta $S = \frac{1}{2} \cdot AB \cdot h$ jest ustalone, to iloczyn $h_A h_B h$ jest maksymalny wtedy i tylko wtedy, gdy $\sin C$ ma maksymalną wartość. Oczywiście, kąt C ma maksymalną wartość (oznaczmy ją przez $\gamma_{\max}$), gdy C jest wierzchołkiem trójkąta równoramiennego o podstawie AB. Jeśli $h > \frac{AB}{2}$, to $\gamma_{\max} < 90^\circ$, więc rozwiązaniem jest trójkąt równoramienny o podstawie AB i wysokości h.



Jeśli natomiast $h \leqslant \frac{AB}{2}$, to $\gamma_{\max} \geqslant 90^\circ$, więc rozwiązaniem jest trójkąt prostokątny o przeciwprostokątnej AB (dla którego $\sin C = 1$).



Rozwiązanie zadania F 855.
Ponieważ masa elektronu jest około 40 tysięcy razy mniejsza od masy atomu neonu, to w warunkach równowagi termodynamicznej porusza się on około 200 razy szybciej niż atomy neonu, które wobec tego można w przybliżeniu uznać za nieruchome. Elektron „trafi" w atom, jeśli ten znajdzie się w odległości nie większej niż r od toru ruchu elektronu. Średnio nastąpi to, gdy w objętości $\pi r^2 l$ będzie znajdował się jeden atom. Oznacza to warunek

$$\frac{N}{V}\pi r^2 l_0 = 1,$$

gdzie N jest liczbą atomów, a V objętością naczynia. Dla gazu w warunkach normalnych mamy

$$\frac{N}{V} = \frac{p_0}{kT_0},$$

gdzie k jest stałą Boltzmanna. Ostatecznie:

$$l_0 = \frac{kT_0}{\pi r^2 p_0},$$

czyli $l_0 \approx 0{,}59$ µm.

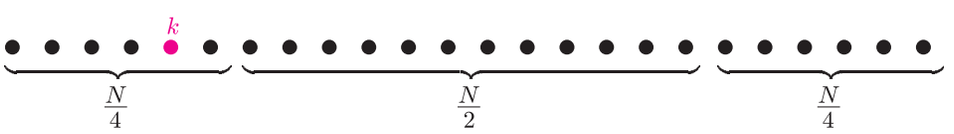
Umiemy więc obliczyć, ile fragmentów jest mniejszych od danego X. Używając bardzo podobnej metody, można też zliczyć fragmenty, które są równe X. O ile więc tylko ktoś podrzuciłby nam fragment X, umielibyśmy sprawdzić, czy faktycznie jest on k-tym w naszym porządku (jeśli mamy ℓ fragmentów, które są mniejsze od X, i e takich, które są równe X, wystarczy sprawdzić czy $\ell < k \leqslant \ell + e$). Niestety, nie mamy co liczyć na żadną życzliwą podpowiedź.

Spróbujemy zastosować strategię przypominającą wyszukiwanie binarne. Na dobry początek możemy stwierdzić, że szukany fragment na pewno znajduje się (w zdefiniowanym wyżej porządku) między $S_1, S_2, \ldots, S_n$ a (dowolnym) pustym fragmentem. Załóżmy więc, że wiemy już, iż szukany X leży między fragmentami A a B. Co dalej? Przydałby nam się fragment M, który leży mniej więcej w połowie drogi między A a B. Mając M, moglibyśmy szybko zliczyć fragmenty, które są od niego mniejsze, porównać tę liczbę z k i w zależności od wyniku zastąpić A lub B przez M (lub stwierdzić, że właśnie M jest szukanym fragmentem). Dobranie się do takiego M wydaje się jednak dość problematyczne, gdyż zakłada, że znamy cały porządek na fragmentach, a przecież tak nie jest.

Wyobraźmy sobie zbiór wszystkich fragmentów, które leżą (w naszym dziwnym porządku) między A a B. Dla każdego i prawe końce takich $S_i, S_{i+1}, \ldots, S_j$ tworzą spójny przedział $j \in [f_i^B, f_i^A)$, co więcej, używając opisanej wyżej metody, możemy efektywnie wyznaczyć wszystkie f_i^B i f_i^A (po prostu rozważamy wszystkie fragmenty mniejsze niż B i odrzucamy te, które są również mniejsze niż A). Można więc przedstawić interesujący nas zbiór jako sumę n mniejszych zbiorów $Z_1, \ldots, Z_n$ (każdy z nich jest tak naprawdę posortowany, choć nie będzie to dla nas istotne). Ale co z tego?

Skoro nie wiemy, co zrobić, może warto wpaść w panikę i zrobić coś losowego. Spróbujmy więc wybrać losowy spośród fragmentów, które leżą między A a B. Wystarczy tylko wylosować jeden ze zbiorów $Z_1, \ldots, Z_n$ (jako że ich rozmiary mogą być dość różne, to wybieramy Z_i z prawdopodobieństwem $\frac{|Z_i|}{|Z_1|+\ldots+|Z_n|}$), a następnie element w zbiorze. Wylosowawszy fragment M, możemy sprawdzić, czy szukany fragment jest mniejszy czy większy (a może równy) od M. A dlaczego to działa?

Wybierając losowy element, mamy sporą szansę, że liczba elementów między A a B istotnie się zmniejszy. Najłatwiej wyobrazić sobie sytuację, korzystając z poniższej ilustracji, na której kolejne kropki reprezentują elementy między A a B ułożone w kolejności zgodnej z naszym dziwnym porządkiem. Powiedzmy, że jest ich N, a szukany element to kolorowa kropka na pozycji k. Skoro wybieramy losowy element, to z prawdopodobieństwem $\frac{1}{2}$ będzie on w środkowym okienku długości $\frac{N}{2}$. Tak naprawdę sprawdzamy, czy wybrany element jest na prawo czy na lewo od szukanego, i w zależności od wyniku porównania pozbywamy się elementów na lewo lub na prawo od tego losowo wybranego. A to jest bardzo wygodne: zawsze pozbędziemy się przynajmniej lewego lub prawego kawałka długości $\frac{N}{4}$. Czyli mamy przynajmniej $\frac{1}{2}$ szansy na to, że liczba elementów między A a B spadnie przynajmniej o jedną czwartą.



Skoro zaczynamy z $\frac{n^2}{2}$ kandydatami, szansa na to, że konieczne okaże się dużo więcej niż, powiedzmy, $4 \log \frac{n^2}{2}$ iteracji, wydaje się niewielka. Darujemy sobie szczegółowe rachunki, wymagają one bowiem pewnej elementarnej wiedzy z rachunku prawdopodobieństwa: uwierzcie mi, że oczekiwana liczba rund to $O(\log n)$. Otrzymaliśmy więc rozwiązanie o oczekiwanym czasie działania $O(n \log^2 n)$, które w dodatku nie jest bardzo skomplikowane implementacyjnie.

Paweł GAWRYCHOWSKI
Instytut Informatyki, Uniwersytet Wrocławski