Informatyczny kącik olimpijski (101): Coś się popsuło

W noworocznym kąciku omówimy zadanie *Wykrywanie wrednej usterki* pochodzące z zeszłorocznej Międzynarodowej Olimpiady Informatycznej, która odbyła się w Kazaniu (Rosja). Autorzy zadania oczekują od nas, że pomożemy zdiagnozować usterkę, która wkradła się do bazy danych zaimplementowaną przez niefrasobliwego inżyniera Ilszata.

Baza danych inż. Ilszata miała działać bardzo prosto. Zaprojektowane były tylko dwie operacje. Pierwsza operacja `add(x)` miała za zadanie dodanie do bazy danych nowego elementu `x`, o którym zakładamy, że jest zawsze ciągiem n bitów (przy czym dodatkowo możemy przyjąć, że n jest potęgą dwójki). Druga operacja `check(x)` powinna sprawdzać, czy `x` znajduje się w bazie, czy też nie.

Jako się rzekło, nie wszystko poszło według z góry ustalonego planu.

Okazało się, że operacja `check(x)` działa zupełnie poprawnie, natomiast operacja `add(x)` ma w sobie dziwnego *buga* (oszibkę?). Otóż, po wywołaniu `add(x)`, zamiast ciągu `x`, do bazy danych dodawany jest ciąg π(`x`) dla pewnej (nieznanej nam) permutacji π. Celem zadania jest właśnie ustalenie tej permutacji, poprzez wykonanie pewnego ciągu operacji na (początkowo pustej) bazie danych. Dodatkowym utrudnieniem jest założenie, że najpierw możemy wykonywać wyłącznie operacje typu `add`, a później wyłącznie operacje typu `check`. Innymi słowy: nie możemy przeplatać różnych typów operacji.

Przykładowo, jeśli $n = 4$ oraz $\pi = [4, 2, 3, 1]$, to po próbie dodania do pustej bazy elementów 0000, 1100 oraz 0111, tak naprawdę znajdą się w niej elementy 0000, 0101 i 1110.

Zaprezentujemy dwa rozwiązania: pierwsze wymaga wykonania $O(n^2)$ operacji, drugie (maksymalnie punktowane): $O(n \log n)$ operacji.

W pierwszym podejściu zaczynamy od wykonania n operacji `add` na następujących elementach: 10^{n-1}, 110^{n-2}, 1110^{n-3}, …, 1^n. Następnie przechodzimy do fazy `check`-ów i zaczynamy od zapytania bazy danych o elementy: 10^{n-1}, 010^{n-2}, 0010^{n-3}, …, $0^{n-1}1$. Oczywiście dokładnie raz otrzymamy odpowiedź pozytywną, dzięki czemu ustalimy na co zamienił się element 10^{n-1} (jedyny z jedną jedynką), czyli poznamy $\pi(1)$. Czas na kolejne pytania. Tym razem pytamy (kolejno) o wszystkie ciągi zawierające dwie jedynki, z których jedna jest zawsze na pozycji $\pi(1)$. Podobnie jak poprzednio dokładnie raz otrzymamy odpowiedź pozytywną, co pozwoli na ustalenie obrazu 110^{n-2}, a ten określi wartość $\pi(2)$. Postępując dalej w ten sposób ustalamy w kolejnych krokach kolejne wartości $\pi(i)$ dla $3 \leqslant i \leqslant n$. Każdy krok nie wymaga więcej niż $O(n)$ zapytań, a więc łączny czas działania szacuje się z góry, zgodnie z obietnicą, przez n^2.

Tutaj wykładnik oznacza krotność powtórzenia ostatniego symbolu.

Rozwiązanie szybsze opiera się na zastosowaniu popularnej (znacznie dłużej niż teoria algorytmów) metody „dziel i zwyciężaj”. Idea polega na ustaleniu obrazów pierwszej i drugiej połówki indeksów permutacji (czyli zbiorów $\{\pi(1), \ldots, \pi(n/2)\}$ oraz $\{\pi(n/2+1), \ldots, \pi(n)\}$) a następnie przeanalizowanie (rekurencyjnie) każdej z połówek niezależnie od siebie. Aby ten pomysł zrealizować w ramach założeń zadania (nie możemy przeplatać `add`-ów i `check`-ów!) potrzeba pewnej zręczności.

Zacznijmy od definicji pewnej rodziny zbiorów. Niech $A_2 = \{10\}$ oraz niech $A_{2k} = A_k \cdot 1^k \cup 1^k \cdot A_k \cup \{10^{2k-1}, 010^{2k-2}, \ldots, 0^{k-1}10^k\}$ (patrz przykład na marginesie).

$A_4 = \{1011, 1110, 1000, 0100\}$
$A_8 = \{10111111, 11101111, 10001111, 01001111, 11111011, 11111110, 11111000, 11110100, 10000000, 01000000, 00100000, 00010000\}$
itd.

Właściwe rozwiązanie zaczyna się od wykonania operacji `add` na wszystkich elementach zbioru A_n. Odtąd możemy już wykonywać wyłącznie operacje `check`. Zaczynamy od zapytania o wszystkie elementy postaci 0^m10^{n-m-1}, czyli te z jedną jedynką. Widzimy, że w zbiorze A_n znajduje się dokładnie $n/2$ elementów z jedną jedynką, a więc i tyle dostaniemy pozytywnych odpowiedzi, przy okazji ustalając na co „przechodzą” obie połówki permutacji π. W tym miejscu chciałoby się po prostu napisać *i dalej rekurencyjnie*. Jest tak w istocie, ale trzeba mieć w głowie pewną subtelność. Otóż zbiór elementów, które włożyliśmy do bazy danych został już raz na zawsze ustalony. Trzeba się więc upewnić, czy w wywołaniach rekurencyjnych nie przeszkadzać nam będą jakieś inne elementy z innych faz oraz jak właściwie tłumaczyć zapytania z faz mniejszego rozmiaru na prawdziwe zapytania, gdzie mamy jedno n ustalone z góry.

Czas działania wyraża rekurencja $T(n) = 2T(n/2) + n$, co oznacza, że $T(n) = O(n \log n)$.

Zakończę sakramentalnym: da się, ale ze szczegółami pozostawiam Czytelnika samego.

Tomasz KAZANA