Informatyczny kącik olimpijski (21): Naszyjniki

W tej edycji kącika – zadanie *Naszyjniki* z Olimpiady Europy Środkowej, CEOI 2007, Brno. Mamy napisać program symulujący zbiór ciągów liczbowych numerowanych kolejnymi liczbami naturalnymi, który będzie mógł wykonywać dwie operacje:

- utworzenie nowego ciągu (i nadanie mu pierwszego wolnego numeru) z ciągu i-tego poprzez dodanie elementu x na lewy lub prawy koniec (operacja dodaj(i, x, k), gdzie $k \in \{l, p\}$) albo usunięcie elementu z lewego lub prawego końca (operacja usuń(i, k));
- sprawdzenie, jaki element znajduje się na lewym lub prawym końcu i-tego ciągu.

Początkowo w zbiorze znajduje się jeden element, ciąg pusty o numerze 1.

Zadanie można rozwiązać, po prostu wykonując n opisanych czynności, w złożoności czasowej i pamięciowej $O(n^2)$, pamiętając bezpośrednio wszystkie ciągi ($O(n)$ ciągów, każdy długości mniejszej niż n). Nie jest to jednak rozwiązanie optymalne. Lepszy okazuje się pomysł, aby z dodawanych elementów tworzyć graf, a dla każdego ciągu pamiętać jedynie, gdzie w tym grafie znajdują się jego lewy i prawy koniec. Gdy dodajemy element na prawym końcu i-tego ciągu, po prostu tworzymy nowy wierzchołek reprezentujący ten nowy element i łączymy go krawędzią z prawym końcem ciągu i. Lewy koniec powstałego ciągu będzie się wówczas znajdował tam, gdzie lewy koniec i; prawy zaś koniec w nowo utworzonym wierzchołku. Łatwo zobaczyć, że konstruowany graf będzie zawsze drzewem.

Sprawdzenie, co znajduje się na którymś końcu i-tego ciągu, także nie nastręcza problemów. Kłopotliwe jest natomiast tworzenie nowego ciągu przez usunięcie elementu z końca i-tego ciągu. W tym celu musimy zlokalizować i-ty ciąg w naszym grafie. Znane jest nam położenie jego końców. Ponieważ sąsiednie elementy ciągu są połączone krawędzią, więc cały ciąg jest w pełni reprezentowany przez jedyną (bo graf jest drzewem) ścieżkę z wierzchołka reprezentującego lewy koniec do wierzchołka reprezentującego prawy. Przy usuwaniu, powiedzmy, lewego końca ciągu, potrzebna jest nam informacja o tym, jaki jest następny po nim element na tej ścieżce – będzie on lewym końcem nowo tworzonego ciągu.

Niecierpliwy rozwiązujący (tak jak i wielu zawodników na Olimpiadzie) powie od razu, że bez trudu można wskazać sąsiada lewego końca ciągu na tej ścieżce i że jest nim po prostu wierzchołek, do którego lewy koniec został podczepiony krawędzią przy tworzeniu. Okazuje się jednak, że nie musi być to prawda! Zachęcam Czytelnika do samodzielnej konstrukcji kontrprzykładu; w razie niepowodzenia można go znaleźć na stronie 11.

Musimy zatem znaleźć jakieś inne wyjście z tej sytuacji. Na początku ukorzeńmy nasze drzewo gdziekolwiek, np. w pierwszym dodanym wierzchołku. Kluczowe jest spostrzeżenie, że ścieżka łącząca wybrane wierzchołki przechodzi przez ich najniższego wspólnego przodka w tym drzewie. Problem wyznaczenia tego wierzchołka, zwany LCA, pojawił się w *Delcie* 9/2007 (i pośrednio w 11/2007), my jednak potrzebować będziemy innego algorytmu niż tam opisany.

Na początku dla każdego wierzchołka v zapamiętujemy jego głębokość w drzewie oraz jego 2^i-tych przodków dla kolejnych wartości i (tzn. wierzchołki w odległościach 2^i na drodze z v do korzenia). Żeby teraz znaleźć LCA wierzchołków v i w, należy poruszać się po drzewie dwoma wskaźnikami, zaczynając od v i w. Najpierw sprowadzamy wskaźniki na tę samą głębokość w drzewie, przesuwając głębszy w górę o odpowiednie potęgi dwójki. Jeśli po tym wskazują na to samo, to znaleźliśmy najniższego wspólnego przodka. Jeśli nie, poruszamy się jednocześnie oboma o potęgi dwójki tak, żeby zachować własność, że są na tej samej głębokości i wskazują na różne wierzchołki. Robiąc to tak długo, jak długo jest to możliwe, otrzymamy w końcu wskaźniki w dwóch różnych synach tego samego wierzchołka, a ów wierzchołek będzie szukanym LCA. Poprzez dobór odpowiednich (tzn. coraz mniejszych) potęg dwójki do naszych ruchów zapewniamy, że złożoność pojedynczego zapytania o LCA wynosi $O(\log n)$.

Jeśli teraz chcemy się dowiedzieć, który sąsiad wierzchołka reprezentującego lewy koniec jest kolejny na naszej ścieżce, sprawdźmy najpierw, gdzie jest najniższy wspólny przodek wierzchołków reprezentujących lewy i prawy koniec ciągu. Jeśli teraz LCA jest różny od lewego końca, to musimy z owego końca pójść ku górze, a więc do jego ojca. Jeśli natomiast LCA jest właśnie lewym końcem, nasza ścieżka wiedzie w dół ku jednemu z dzieci. Żeby dowiedzieć się ku któremu, należy wyszukać przodka prawego końca na głębokości o jeden większej niż głębokość lewego końca (binarnie, podobnie jak wcześniej). W ten sposób, jeśli przy wstawianiu nowych wierzchołków liczymy (na podstawie wcześniejszych wartości) i zapamiętujemy ich 2^i-tych przodków, to otrzymujemy rozwiązanie, w którym utworzenie nowego ciągu (obojętnie czy przez usunięcie czy dodanie elementu) zajmuje czas $O(\log n)$, a sprawdzenie wartości na końcu – czas $O(1)$. Sumaryczna złożoność tego rozwiązania, zarówno czasowa, jak i pamięciowa, wynosi $O(n \log n)$. Zachęcam do zastanowienia się, czy można ten problem rozwiązać jeszcze szybciej, szczególnie że problem znajdowania LCA jest rozwiązywalny liniowo (algorytm opisany we wspomnianych wcześniej numerach *Delty*).

Tomasz KULCZYŃSKI