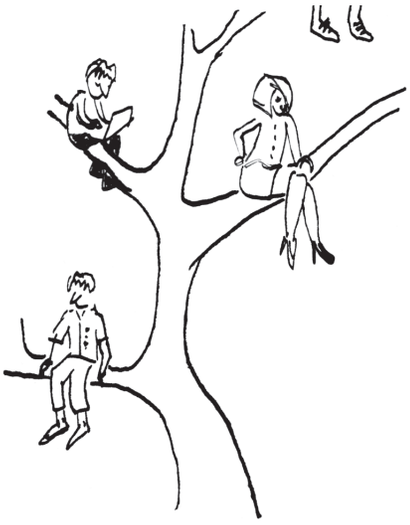

Informatyczny kącik olimpijski (27): Regiony

Tym razem weźmiemy pod lupę zadanie *Regions* z Międzynarodowej Olimpiady Informatycznej 2009.

Mamy N pracowników pochodzących z R regionów (każdy z dokładnie jednego). Każdy pracownik, poza pierwszym, ma swojego bezpośredniego przełożonego; zależności te nie tworzą, oczywiście, cykli. Mówimy, że A jest przełożonym B, jeśli A jest bezpośrednim przełożonym B lub, rekurencyjnie, gdy A jest przełożonym bezpośredniego przełożonego B. Znamy hierarchię pracowników oraz wiemy, skąd który pracownik pochodzi. Chcemy odpowiadać, dla różnych r_1 i r_2, na zapytania postaci: „ile jest par pracowników e_1, e_2, takich że e_1 pochodzi z regionu r_1, e_2 pochodzi z regionu r_2 oraz e_1 jest przełożonym e_2?". Pytań takich zostanie nam zadanych Q. Dla porządku należy dodać, że przedstawione w treści zadania górne ograniczenia na wartości N, R i Q były tego samego rzędu.

Problem nie wydaje się łatwy, warto więc zacząć od najprostszych poprawnych rozwiązań. Zauważmy, że hierarchia pracowników ma strukturę drzewa. Możemy zatem na każde zapytanie odpowiedzieć, wykonując przeszukiwanie tegoż drzewa, w którym zliczamy pracowników z regionu r_1 na ścieżce z korzenia do aktualnego wierzchołka, i zsumować te wartości z wierzchołków będących pracownikami z r_2. Daje to rozwiązanie w czasie $O(Q \cdot N)$.

Postępując jak wyżej, dla większości zapytań niepotrzebnie przeglądamy całe drzewo, gdy tymczasem istotne wierzchołki (te z regionów r_1 i r_2) stanowią tylko niewielką jego część. Aby temu zaradzić, znajdziemy sposób na szybkie sprawdzanie, czy jeden pracownik jest przełożonym drugiego. W terminach drzew chodzi o sprawdzenie, czy dany wierzchołek jest przodkiem innego wierzchołka. Odpowiedź na to pytanie przynosi algorytm przeszukiwania w głąb (DFS). Jak opisano w książce T. Cormena, C. Leisersona, R. Rivesta i C. Steina *Wprowadzenie do algorytmów*, wystarczy w tym celu obliczyć czasy wejścia (d) oraz wyjścia (f) dla poszczególnych wierzchołków w przeszukiwaniu – wówczas wierzchołek u jest przodkiem v, jeżeli $[d_u, f_u] \supset [d_v, f_v]$ (a każde dwa przedziały tej postaci są albo rozłączne, albo jeden jest zawarty w drugim). Jeśli teraz dla każdego regionu zapamiętamy na wstępie listę pracowników, którzy z niego pochodzą, otrzymamy rozwiązanie o złożoności $O(N + Q \cdot M^2)$, przy czym M jest maksymalną liczbą pracowników z jednego regionu. Celowo wprowadziliśmy takie oznaczenie, mimo że póki co wiemy jedynie, że $M \leqslant N$.

Aby jeszcze zmniejszyć czas odpowiedzi na pojedyncze zapytanie, możemy z pracowników z regionów r_1 i r_2 zbudować w czasie $O(M)$ nowe drzewo (zawierające tylko tych pracowników), w którym zależności bycia przełożonym pozostaną niezmienione w stosunku do oryginalnego drzewa (może zaistnieć potrzeba dodania sztucznego korzenia). W tym celu zaczynamy od scalenia list pracowników (przedziałów $[d_v, f_v]$) obu regionów uporządkowanych względem d_v. Następnie zauważamy, że rodzicem pracownika $[d_v, f_v]$ w nowym drzewie będzie najbliższy mu element występujący na scalonej liście przed nim, którego wartość f jest większa niż f_v. W ten sposób sprowadziliśmy problem budowy drzewa do klasycznego problemu typu „najbliższy większy położony z lewej", który można rozwiązać w czasie liniowym na wiele sposobów (np. za pomocą stosu). Z kolei w nowym drzewie wiemy już, jak rozwiązać nasz problem w czasie $O(M)$. Otrzymaliśmy zatem rozwiązanie o złożoności czasowej $O(N + Q \cdot M)$.

Ten algorytm działa szybko, gdy M jest małe, tzn. gdy regiony są małe. Zastanówmy się więc, co dzieje się, gdy są one duże (nie ma ich wówczas zbyt wiele, bo przecież suma ich rozmiarów to N). W takim przypadku najlepszym pomysłem wydaje się wstępne obliczenie wszystkich R^2 wyników i zapamiętanie ich. Dla każdego regionu r_1 wystarczy jedno przeszukiwanie całego drzewa, aby ustalić wyniki dla wszystkich możliwych r_2 (znów przeszukujemy drzewo, pamiętając w każdym wierzchołku liczbę pracowników z r_1 na ścieżce z korzenia, i w każdym wierzchołku uaktualniamy wynik całkowity dla jego regionu). Takie rozwiązanie ma złożoność czasową $O(R \cdot N + Q)$ i zużywa $O(N + R^2)$ pamięci.

Uzyskaliśmy dwa rozwiązania radzące sobie dobrze w różnych skrajnych przypadkach. Teraz chcielibyśmy je połączyć w jedno, działające szybko dla każdych danych. W tym celu ustalmy pewną granicę rozmiaru regionu, x. Jeśli oba regiony z danego zapytania mają rozmiar nie większy niż x, używamy pierwszego algorytmu. Jeśli zaś któryś z nich jest większy, używamy drugiego. „Dużych" regionów jest, oczywiście, nie więcej niż $\frac{N}{x}$. Musimy zatem zmodyfikować drugi algorytm: będzie on teraz w czasie $O\left(\frac{N^2}{x}\right)$ i pamięci $O\left(\frac{N \cdot R}{x}\right)$ obliczał wyniki dla zapytań, w których jeden z regionów jest duży, a drugi dowolny. Polecamy Czytelnikowi zastanowienie się, jak to zrobić (potrzeba dwóch przeszukiwań drzewa dla każdego dużego regionu). Stąd, łączny czas wyniesie $O\left(\frac{N^2}{x} + Q \cdot x\right)$ przy jednoczesnym zużyciu $O\left(N + \frac{N \cdot R}{x}\right)$ pamięci. Jeśli ustalimy $x = \sqrt{N}$, otrzymujemy złożoność czasową $O((N + Q)\sqrt{N})$ i pamięciową $O(N + R\sqrt{N})$.

Na koniec pozostawiamy Czytelnikowi pytanie dodatkowe: jak zmodyfikować powyższe rozwiązanie, tak aby zmniejszyć jego złożoność pamięciową do $O(N)$, kosztem zwiększenia złożoności czasowej o czynnik $\log N$?

Tomasz KULCZYŃSKI