Informatyczny kącik olimpijski (56): Melodia

Tym razem omówimy zadanie z pogranicza informatyki i muzykologii, które pojawiło się na Bałtyckiej Olimpiadzie Informatycznej w bieżącym roku. W zadaniu mamy do czynienia z pewnym instrumentem muzycznym, za pomocą którego można zagrać n rodzajów nut. Muzyk ma zadaną melodię, którą chciałby zagrać – jest to sekwencja m kolejnych nut. Niestety, nie jest to takie proste. Otóż zagranie każdej nuty wymaga zakrycia niektórych otworów instrumentu, a jeśli dwie nuty istotnie różnią się wymaganym sposobem zakrycia, to muzyk nie jest w stanie zagrać jednej z tych nut bezpośrednio po drugiej... Innymi słowy, dla każdej pary nut wiadomo z góry, czy można zagrać je pod rząd, czy też nie. Z tego względu zagranie zadanej melodii może nie być możliwe. Muzyk chciałby więc zagrać melodię możliwie najbliższą zadanej, to znaczy zagrać taką melodię złożoną z m nut, która będzie różniła się od zadanej na najmniejszej możliwej liczbie pozycji – i my mamy mu w tym pomóc. Dodajmy, że w naszym zadaniu liczba możliwych nut jest stosunkowo niewielka (rzędu 100), natomiast cała melodia może składać się nawet ze 100 000 nut.

Jednym z pierwszych pomysłów, jaki narzuca się po przeczytaniu tego zadania, jest programowanie dynamiczne. Ponumerujmy poszczególne rodzaje nut od 1 do n i oznaczmy kolejne nuty w zadanej melodii przez $a_1, a_2, \ldots, a_m$. Niech dalej tablica wartości logicznych $c[i,j]$ reprezentuje podane pary nut, które można zagrać bezpośrednio jedna po drugiej; zakładamy, że dla każdego $i = 1, \ldots, n$ zachodzi $c[i,i] = \textbf{true}$. Chcemy wypełnić dwuwymiarową tablicę t, w której pole $t[k,i]$ oznacza najmniejszą liczbę błędów, jakie może popełnić muzyk, grając – zamiast fragmentu melodii $a_1, \ldots, a_k$ – jakąś melodię długości k kończącą się nutą typu i. Pomijając warunki brzegowe, pola tablicy t możemy wyznaczać za pomocą prostej zależności rekurencyjnej, w której sprawdzamy wszystkie możliwe rodzaje nut, jakie mogły wystąpić w melodii bezpośrednio przed zadaną nutą:

$$(*) \qquad t[k,i] = (1 - \delta_{ia_k}) + \min_j \{t[k-1,j] : c[j,i] = \textbf{true}\}.$$

W powyższym wzorze δ_{xy} oznacza 1, jeśli $x = y$, a 0 w przeciwnym przypadku. To rozwiązanie działa w czasie $O(mn^2)$ i pamięci $O(mn)$.

Da się jednak lepiej. Naturalne wydaje się skorzystanie z faktu, że powyższe rozwiązanie oblicza jedynie $O(mn)$ różnych wartości. Gdybyśmy tylko byli w stanie jakoś sprytniej wyznaczać minimum we wzorze $(*)$, np. w czasie stałym, to uzyskalibyśmy rozwiązanie działające zadowalająco szybko. Pewną wskazówką jest tu pierwszy składnik występujący we wzorze $(*)$. Otóż, w przeciwieństwie do drugiego składnika, może on przyjmować tylko dwie różne wartości, które zależą jedynie od tego, czy w k-tym kroku decydujemy się zagrać poprawną nutę a_k, czy też nie.

Przyjrzyjmy się pierwszemu z tych przypadków i spróbujmy zmienić nieco nasze podejście. Gdyby udało nam się wyznaczyć jednowymiarową tablicę p, w której $p[k]$ to maksymalna liczba poprawnie zagranych nut podczas wykonywania początkowego fragmentu melodii długości k, przy założeniu, że k-tą nutę zagramy poprawnie, to moglibyśmy już łatwo podać wynik – byłoby to maksimum ze wszystkich elementów tablicy p (dlaczego?). Przy wypełnianiu tego typu tablic za pomocą programowania dynamicznego $p[k]$ oblicza się zazwyczaj jako maksimum z tych wartości $p[l] + 1$, dla $l < k$, które „pasują do faktu $p[k]$". W naszym przypadku przy obliczaniu $p[k]$ bierzemy pod uwagę te wartości $p[l] + 1$, dla których istnieje poprawna (czyli możliwa do zagrania) melodia długości k zawierająca na odpowiednich pozycjach nuty a_l i a_k.

Nie dosyć, że powyższe rozwiązanie ma na starcie złożoność czasową $\Omega(m^2)$, to jeszcze nie bardzo widać, jak takie $p[l]$ znajdować. Jest do tego potrzebny pewien pomysł: otóż tablicę c możemy potraktować jako macierz sąsiedztwa pewnego grafu. Wówczas wartość $p[l] + 1$ pasuje do $p[k]$, jeśli w tym grafie istnieje ścieżka długości $k - l$ łącząca nuty a_l oraz a_k. W tym celu wystarczy sprawdzić, czy *najkrótsza* ścieżka w grafie od a_l do a_k ma długość nieprzekraczającą $k - l$ (to przejście myślowe wynika akurat z faktu, że $c[i,i]$ jest zawsze prawdziwe). Najkrótsze ścieżki między wszystkimi parami wierzchołków (nut) możemy wyznaczyć w czasie $O(n^3)$, czy to za pomocą algorytmu Floyda–Warshalla, czy to wielokrotnie wykonując przeszukiwanie wszerz.

W ten sposób rozwiązaliśmy problem implementacji nowego rozwiązania, ale ciągle nie uporaliśmy się z jego paskudną złożonością czasową. Jedynym nowym pomysłem, jaki pojawił się po drodze, było wprowadzenie grafu nut. Jeśli zdecydujemy się poświęcić mu jeszcze trochę uwagi, to ta decyzja okaże się kluczowa dla skonstruowania efektywnego rozwiązania.

Potrzebne jest nam mianowicie już tylko następujące spostrzeżenie: graf nut ma zaledwie n wierzchołków, więc wszystkie najkrótsze ścieżki w tym grafie mają długości nieprzekraczające $n - 1$. To oznacza, że dla danego $p[k]$ *wszystkie* wartości $p[l] + 1$ dla $l \leqslant k - n + 1$ są z pewnością dobre, a zatem przy obliczaniu $p[k]$ wystarczy tylko rozważyć elementy tablicy położone co najwyżej n pól wstecz oraz maksimum ze wszystkich elementów położonych jeszcze wcześniej!

Zanim jednak zaczniemy świętować nasz sukces, musimy poczynić przykre spostrzeżenie, że graf nut może nie być spójny. To oznacza, że dla danego $p[k]$ niektóre nawet bardzo odległe wartości $p[l] + 1$ mogą nie być dobre. Przezwyciężenie tej trudności okazuje się już tylko kwestią czasu. Możemy, na przykład, pamiętać zawsze n ostatnich elementów tablicy p, a także maksima ze wszystkich wcześniejszych elementów pogrupowane po poszczególnych nutach (czyli $p[l] + 1$ grupujemy po a_l). Sprytniejsze rozwiązanie polega na rozważeniu każdej spójnej składowej grafu nut z osobna i zastosowaniu dla niej metody z jednym pamiętanym maksimum – wówczas nuty melodii występujące poza składową traktujemy po prostu jako niemożliwe do zagrania. W obu przypadkach otrzymujemy algorytm działający w czasie $O(mn)$ i pamięci $O(n)$. Do tego musimy jeszcze doliczyć koszt wyznaczenia najkrótszych ścieżek: czas $O(n^3)$ i pamięć $O(n^2)$.

Dodajmy na koniec, że nasz muzyk nie będzie specjalnie zadowolony, jeśli podamy mu jedynie minimalną liczbę błędów, jakie może popełnić – chciałby przecież wiedzieć, jaką melodię powinien zagrać! Polecamy zatem Czytelnikowi zastanowić się nad tym, jak odtworzyć wynik.

Jakub RADOSZEWSKI