Informatyczny kącik olimpijski (45): Wypisz napis

W tym kąciku omówimy pierwsze zadanie z finału konkursu Google Code Jam 2010. Mamy dany napis $w[1..n]$ złożony z n liter ze zbioru $\{\texttt{A}, \texttt{B}, \texttt{C}\}$. Naszym celem jest wypisanie kolejno liter tego napisu od lewej do prawej. Mamy do dyspozycji stempel, który możemy wyobrazić sobie jako stos, umożliwiający wykonywanie trzech typów operacji: doczepienie plakietki z literą `A`, `B` lub `C`, odczepienie plakietki znajdującej się na wierzchu oraz przyłożenie stempla do kartki, w wyniku czego nadrukowuje się na niej litera z wierzchniej plakietki. Chcemy wypisać w za pomocą najmniejszej możliwej liczby operacji, przy czym zaczynamy i kończymy z pustym stemplem (stosem).

Jednym z pierwszych pomysłów na rozwiązanie może być programowanie dynamiczne. Wyznaczamy w nim kolejne komórki tablicy $t[i][S]$, oznaczające, w jakiej minimalnej liczbie operacji można wypisać początkowy fragment napisu, $w[1..i]$, tak aby na końcu otrzymać stos reprezentowany przez napis S (nad alfabetem $\{\texttt{A}, \texttt{B}, \texttt{C}\}$). Wynikiem jest wówczas $t[n][\emptyset]$. Przy tym podejściu nieuchronnie napotykamy problem, że liczba możliwych stosów jest naprawdę duża. Zauważmy, że nigdy na stosie nie opłaca nam się trzymać więcej niż n liter, gdyż wówczas musielibyśmy łącznie wykonać ponad $3n$ operacji, a w $3n$ operacjach naprawdę łatwo wypisać cały napis w. Czyli mamy co najwyżej 3^n różnych stosów, ale do tego jeszcze mnóstwo możliwych przejść z każdego stanu $t[i][S]$ do stanów postaci $t[i+1][S']$.

Pokażemy teraz, jak trochę polepszyć to rozwiązanie. Przede wszystkim na stosie ewidentnie nie opłaca się nigdy trzymać dwóch takich samych liter pod rząd. To redukuje liczbę interesujących nas stosów do $O(2^n)$. Możemy też istotnie zmniejszyć liczbę rozważanych przejść, jeśli zauważymy, że w danym stanie wystarczy próbować dołożyć na stos co najwyżej jedną nową literkę. Skrótowe uzasadnienie na przykładzie: zamiast w danym stanie dokładać na stos literki `AB`, wystarczy dołożyć samą literkę `B`. Jeśli potem, po zdjęciu `B`, będę chciał użyć tego `A`, którego nie dołożyłem, to dokładam to `A` właśnie wtedy. Liczba operacji nie zwiększyła się, a postąpiłem jakby sprytniej. W ten sposób zmniejszyliśmy liczbę przejść z danego stanu do $O(n)$ (dodanie co najwyżej jednej literki i usunięcie dowolnie wielu literek), co daje już rozwiązanie o koszcie czasowym $O(2^n n^2)$. Wyraźnie lepiej niż poprzednio, ale wciąż jakoś wolno.

Spróbujmy więc zmienić podejście do rozwiązania. Przyjrzyjmy się sytuacji początkowej. Zaczynamy od pustego stosu, a chcemy wypisać literkę $w[1]$, więc musimy ją włożyć na stos. Oznaczmy przez i „moment", po którym $w[1]$ zostanie zdjęte ze stosu, tzn. załóżmy, że $w[1]$ znajduje się na spodzie stosu podczas wypisywania liter fragmentu $w[2..i]$, po czym zostaje zdjęte ze stosu ($1 \leqslant i \leqslant n$). Podzieliliśmy zatem wyjściowy problem na dwa niezależne podproblemy, polegające na wypisaniu każdego z fragmentów $w[2..i]$ oraz $w[i+1..n]$, tak aby na końcu zdjąć ze stosu wszystkie dołożone po drodze litery. Jedyna różnica między podproblemami polega na tym, że podczas wypisywania $w[2..i]$ możemy skorzystać z faktu, iż na stosie jest już litera $w[1]$.

To prowadzi nas do innego rozwiązania opartego na programowaniu dynamicznym. Przez $q[i][j][c]$ oznaczmy minimalną liczbę operacji potrzebnych do wypisania fragmentu $w[i..j]$, tak aby nie pozostawić na końcu żadnych dodatkowych liter na stosie. A to wszystko przy założeniu, że na szczycie stosu przed rozważeniem tego fragmentu znajduje się literka $c \in \{\texttt{A}, \texttt{B}, \texttt{C}, \emptyset\}$; $\emptyset$ oznacza, że przed rozpoczęciem wypisywania $w[i..j]$ stos był pusty. Dodajmy dla jasności, że literki c, ani niczego poniżej, nie możemy tu usunąć ze stosu. Przykładowo, wartości postaci $q[i][j][\texttt{A}]$ obliczamy następująco: jeśli $w[i] = \texttt{A}$, to $q[i][j][\texttt{A}] = q[i+1][j][\texttt{A}] + 1$, a w przeciwnym razie

$$q[i][j][\texttt{A}] = \min\{q[i+1][k][w[i]] + q[k+1][j][\texttt{A}] + 3 : k = i, \ldots, j\}.$$

Przy tym $q[i][j][c] = 0$ dla $i > j$. Takie rozwiązanie ma złożoność czasową $O(n^3)$ – pozwalało to zdobyć częściowe punkty na zawodach.

Istnieje zatem jeszcze lepsze rozwiązanie. Co ciekawe, aby je uzyskać, należy wrócić do poprzedniego, wykładniczego rozwiązania, i jeszcze dokładniej przyjrzeć się możliwym zawartościom stosu. Kluczowa obserwacja to: można nie rozważać stosów, w których występuje fragment `ABA`. Faktycznie, rozważmy rozwiązanie, w którym na stosie pojawia się fragment tej postaci, i przyjrzyjmy się chwili, w której umieszczamy na stosie drugą z literek `A`. Zróbmy jednak inaczej: zamiast umieszczać drugie `A`, usuńmy ze stosu literkę `B`. Liczba operacji nie zmieniła się, a na szczycie stosu wciąż mamy `A`. Dalej wykonujemy rozwiązanie bez zmian, aż do chwili, w której próbuje ono zdjąć ze stosu to drugie `A`, którego teraz nie ma. W zamian włóżmy na stos literkę `B`. Liczba operacji nie zmieniła się, układ liter na stosie również. W ten sposób pozbyliśmy się jednego wystąpienia fragmentu `ABA`; jeśli jest ich więcej, eliminujemy je w ten sam sposób.

Podobnie można uzasadnić, że możemy nie rozważać stosów zawierających fragment xyx dla dowolnych, różnych liter x i y. Stosów bez takich napisów jest już bardzo mało: są to tylko stosy postaci `ABCABC`..., `BACBAC`... i jeszcze czterech innych (dowolna permutacja liter `A`, `B`, `C` na początku). Tak oto tablica t skurczyła się nam do rozmiaru $O(n^2)$. Wreszcie liczbę przejść z każdego stanu możemy zmniejszyć do stałej, korzystając ze spostrzeżenia, że jeśli przy wypisywaniu $w[i]$ decydujemy się na usuwanie liter ze stosu, to możemy to robić do chwili, gdy po raz pierwszy na szczycie stosu znajdzie się litera $w[i]$ lub gdy stos stanie się pusty. To oznacza, że w danym stanie usuniemy ze stosu co najwyżej dwie literki (lub wszystkie) i dodamy na stos co najwyżej jedną. W ten sposób uzyskaliśmy rozwiązanie wzorcowe, o złożoności czasowej $O(n^2)$.

Jakub RADOSZEWSKI