Informatyczny kącik olimpijski (137): po prostu znajdź wzór

Wiele zadań konkursowych proponowanych podczas zawodów programistycznych wymaga od uczestników zakodowania zawiłych algorytmów czy skomplikowanych struktur danych. Właśnie takie zadania nie raz i nie dwa, ale wielokrotnie prezentujemy w niniejszej rubryce. Dziś jednak opowiemy o pewnym bardzo specyficznym typie zadań olimpijskich, które zwykle sprowadzają się do znalezienia zwartego wzoru opisującego odpowiedź na pytanie zawarte w zadaniu. Jako przykład niech posłuży nam zadanie „Count the Arrays", prezentowane na platformie *Codeforces*. Zadanie jest następujące:

Na wejściu otrzymujemy dwie liczby naturalne n oraz m. Interesować nas będą pewne bardzo specyficzne ciągi. Konkretnie powiemy, że ciąg jest elegancki, *jeśli zawiera n elementów o wartościach w zbiorze $\{1, 2, \ldots, m\}$ oraz:*
$(*)$ *istnieje dokładnie jedna para elementów ciągu, które są sobie równe;*
$(**)$ *istnieje indeks i (szczytowy) taki, że ciąg jest ściśle rosnący na pozycjach $1, \ldots, i$ oraz ściśle malejący na pozycjach $i, \ldots, n$.*
Niech teraz $A_{n,m}$ oznacza liczbę eleganckich ciągów dla pewnych ustalonych n i m. Celem zadania jest obliczenie $A_{n,m}$ (modulo 998244353).

Przykładowo wszystkie eleganckie ciągi dla $n = 4$, $m = 4$ to: $(1, 2, 3, 2)$, $(1, 2, 3, 1)$, $(1, 3, 2, 1)$, $(2, 3, 2, 1)$, $(2, 3, 4, 3)$, $(2, 3, 4, 2)$, $(3, 4, 3, 2)$, $(2, 4, 3, 2)$, $(1, 3, 4, 1)$, $(1, 3, 4, 3)$, $(1, 4, 3, 1)$, $(3, 4, 3, 1)$, $(1, 2, 4, 1)$, $(1, 2, 4, 2)$, $(1, 4, 2, 1)$, $(2, 4, 2, 1)$.

Rozwiązanie. Twierdzimy, że prawdziwy jest wzór:

$$A_{n,m} = \binom{m}{n-1} \cdot (n-2) \cdot 2^{n-3}.$$

Dlaczego? Aby go udowodnić, spróbujmy zdefiniować „procedurę" generowania wszystkich rozważanych $A_{n,m}$ ciągów. Otóż wyobraźmy sobie, iż w pierwszym kroku chcemy zadecydować, które w ogóle elementy choć raz pojawią się w naszym ciągu. Ponieważ ciąg ma długość n i tylko jeden element się powtarza – co więcej zawsze z krotnością 2 – to widać, że różnych elementów jest zawsze $(n-1)$. Możemy więc je wybrać ze zbioru wartości na dokładnie $\binom{m}{n-1}$ sposobów. Zauważmy, że gdy już ustalimy swój „alfabet", to element szczytowy został już wybrany. Tam, gdzie mamy jeszcze swobodę, to wybór elementu zduplikowanego – dokonujemy go na $(n-2)$ sposobów (możemy wybrać każdy, poza szczytowym). Dla wszystkich pozostałych $(n-3)$ (poza szczytowym i zduplikowanymi) elementów musimy jeszcze podjąć decyzję, czy dany element znajdzie się po lewej, czy po prawej stronie szczytu. Różnych możliwości jest więc 2^{n-3}, a ustalenie tego wyboru już determinuje cały ciąg – elementy z obu stron szczytu są posortowane, więc nie pozostała już żadna swoboda w generowaniu eleganckich ciągów. Ta obserwacja kończy dowód prawdziwości postulowanego wzoru.

Zadania takie jak wyżej mogą oczywiście sprawiać kłopot w trakcie analizy kombinatorycznej, ale gdy już znajdziemy stosowny wzór, to wydaje się, że wystarczy już tylko napisać króciutki programik, który oblicza i zwraca wynik. W zasadzie jest to prawda, ale i tu czyhają pułapki. Skupimy się na dwóch problemach:

1. Jak szybko obliczyć 2^{n-3}(modulo 998244353)?

Kolejne potęgowanie dwójki może okazać się za wolne dla dużych wartości n. Wówczas warto skorzystać z metody, która działa w czasie $O(\log n)$. Polega ona na tym, że najpierw liczymy (podnosząc do kwadratu kolejne elementy) wartości

$$2^1, 2^2, 2^4, 2^8, \ldots, 2^{\lfloor \log(n-3) \rfloor},$$

a następnie zapisujemy $(n-3)$ w systemie binarnym, dzięki czemu ustalamy, iloczyn których potęg dwójki da nam 2^{n-3}. Na przykład jeśli $(n-3) = 1011001_2$, to $2^{n-3} = 2^1 \cdot 2^8 \cdot 2^{16} \cdot 2^{64}$. *Uwaga:* wszystkie mnożenia wykonujemy oczywiście modulo 998244353.

2. Jak szybko obliczyć $\binom{m}{n-1}$(modulo 998244353)?

Dwumian Newtona można obliczać wprost z rekurencji $\binom{k+1}{l+1} = \binom{k}{l+1} + \binom{k}{l}$. Ta metoda, zastosowana jednak bezpośrednio, skończy się czasem wykładniczym. Ulepszenie, polegające na spamiętywaniu już obliczonych wartości (tak zwane programowanie dynamiczne), da złożoność kwadratową, która wciąż będzie za duża, aby rozwiązanie przeszło skutecznie testy sprawdzające.

Skorzystamy więc z wzoru $\binom{m}{n-1} = \frac{m!}{(n-1)!(m-n+1)!}$.

Ponownie, proponujemy aby najpierw wykonać obliczenia wstępne i obliczyć (żargonowo: *spamiętać*):

$$2!, 3!, \ldots m! \pmod{998244353},$$

ale także (pomijamy, jak dokładnie to zrobić)

$$(2!)^{-1}, (3!)^{-1}, \ldots, (m!)^{-1} \pmod{998244353}.$$

Uwaga: chodzi oczywiście o odwrotność w ciele $\mathbb{Z}_{998244353}$, czyli np. $(4!)^{-1} \bmod 998244353 = 291154603$.

Wyposażeni w powyższe dane już łatwo (w czasie $O(n+m)$) możemy obliczyć $\binom{m}{n-1} \bmod 998244353$.

Powyższe rozwiązanie już jest wystarczające, bo we wszystkich testach mamy $n, m \leqslant 2 \cdot 10^5$. Co jednak, gdybyśmy chcieli rozwiązać zadania dla parametrów rzędu nawet kilku miliardów? Tutaj możemy sobie poradzić za pomocą następującego *Olimpijskiego Triku*:

Zanim zaczniemy pisać program właściwy, możemy obliczyć np. co milionową silnię (i jej odwrotność) modulo 998244353, a wyniki (wygenerowane choćby i przez godzinę w trakcie zawodów) zapisać BEZPOŚREDNIO W KODZIE rozwiązania jako stałą tablicę!

Wówczas kod programu może być ogromny, osiągając nawet kilkadziesiąt kilobajtów tekstu (co wciąż mieści się w regulaminowym limicie!). Dzięki temu same obliczenia właściwe po odczycie danych wejściowych podczas testów będą znacznie szybsze, bo fazy obliczania kolejnych silni nie musimy zaczynać od 2 tylko od najbliższej wielokrotności miliona!

Tomasz KAZANA