Wojciech Nadara
Treść zadania, Opracowanie

Paweł Czerwiński
Program

Dostępna pamięć: 128 MB. *OI, etap III, dzień pierwszy, 13.04.2016*

Pracowity Jaś

Jaś miał niedawno urodziny. Jak w każdym szanującym się zadaniu algorytmicznym, Jaś nie dostał w prezencie ani zabawek, ani gier, ani komputera, a jedynie długie tablice wypełnione liczbami, drzewa, mapy różnych krajów z drogami prowadzącymi tunelami i estakadami oraz długie taśmy z wypisanymi 1048576 początkowymi literami słów Fibonacciego i Thuego-Morse'a. Najbardziej z tych wszystkich prezentów spodobała mu się tablica z wypisaną permutacją[1] *pierwszych n liczb naturalnych. Zaczął się zastanawiać, jaka jest poprzednia permutacja w porządku leksykograficznym*[2]*. Po chwili udało mu się ją wymyślić i zapragnął zapisać ją na tej samej tablicy. Jaś potrafi w jednym kroku zamienić miejscami jedynie dwie z liczb zapisanych na tablicy (gdyby operował na większej liczbie liczb naraz, to by się pogubił). Jest jednak na tyle mądry, że przekształcił początkową permutację w poprzednią leksykograficznie, wykonując minimalną liczbę takich zamian. Gdy to zrobił, wpadł w permutacyjny szał i zaczął powtarzać tę operację w kółko, zapisując na tablicy kolejne wcześniejsze permutacje w porządku leksykograficznym!*

Niestety, po pewnym czasie Jaś będzie musiał przerwać swoją zabawę, gdyż dojdzie do permutacji $1, 2, \ldots, n$*, która jest najmniejsza w porządku leksykograficznym. Przyjaciele Jasia trochę się naśmiewają z jego monotonnej zabawy, jednak wiedzą, że nie mają szans go z niej wyrwać. Chcieliby się więc dowiedzieć, kiedy Jaś skończy. Pomóż kolegom Jasia i powiedz im, ile potrwa jego zabawa, jeśli każda zamiana zajmuje mu dokładnie sekundę. Jako że ta rozrywka może być dość długa (Jaś nie bez powodu ma przydomek Pracowity), wystarczy im reszta z dzielenia tej liczby przez* $10^9 + 7$*. W końcu* $10^9 + 7$ *sekund to na tyle długo, że są w stanie co tyle czasu sprawdzać, czy Jaś już skończył swoją zabawę.*

Wejście

W pierwszym wierszu standardowego wejścia znajduje się dodatnia liczba całkowita n oznaczająca długość permutacji, którą Jaś otrzymał na urodziny. W drugim wierszu znajduje się ciąg n parami różnych liczb naturalnych $p_1, p_2, \ldots, p_n$ $(1 \leqslant p_i \leqslant n)$ *pooddzielanych pojedynczymi odstępami, opisujący tę permutację.*

Wyjście

Twój program powinien wypisać na standardowe wyjście resztę z dzielenia przez $10^9 + 7$ *liczby zamian, które wykona Jaś, zanim jego zabawa się skończy.*

[1] Permutacja liczb od 1 do n to ciąg różnych liczb całkowitych $p_1, \ldots, p_n$ spełniających $1 \leqslant p_i \leqslant n$ (każda liczba całkowita od 1 do n występuje w takiej permutacji dokładnie raz).

[2] Permutacja $P = (p_1, \ldots, p_n)$ jest wcześniej w porządku leksykograficznym niż permutacja $Q = (q_1, \ldots, q_n)$ (co zapiszemy jako $P < Q$), jeżeli $p_j < q_j$, gdzie j jest najmniejszym takim indeksem, że $p_j \neq q_j$. Permutacja P poprzedza permutację Q w porządku leksykograficznym, jeżeli $P < Q$ oraz nie istnieje taka permutacja R, że $P < R < Q$.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`3` `3 1 2`	`6`

Wyjaśnienie do przykładu: *Na tablicy Jasia będą się po kolei pokazywały permutacje $(2,3,1)$, $(2,1,3)$, $(1,3,2)$, $(1,2,3)$. Aby je uzyskiwać, będzie musiał wykonać łącznie $2+1+2+1=6$ zamian.*

Testy „ocen":

`1ocen`: $1, 2, 3, \ldots, 10$

`2ocen`: *losowa 5-elementowa permutacja*

`3ocen`: $100, 99, 98, \ldots, 1$.

Ocenianie

Zestaw testów dzieli się na następujące podzadania. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

Podzadanie	Warunki	Liczba punktów
1	$n \leqslant 10$	*15*
2	$n \leqslant 5000$	*37*
3	$n \leqslant 1\,000\,000$, *permutacja to* $n, n-1, \ldots, 1$	*15*
4	$n \leqslant 1\,000\,000$	*33*

Rozwiązanie

Poprzednia leksykograficznie permutacja

Pierwszym pytaniem, jakie należy sobie postawić, jest „Dla danej permutacji $P = (p_1, \ldots, p_n)$, jak wygląda jej leksykograficzny poprzednik?". Nazwijmy go Q. Wśród wszystkich permutacji mniejszych leksykograficznie od P, Q jest ma maksymalny wspólny prefiks (początkowy kawałek) z P. Na początek chcielibyśmy zidentyfikować długość tego prefiksu. Podzielmy naszą permutację P na pewien prefiks oraz sufiks (końcowy kawałek) i oznaczmy je jako A i B. Leksykograficznie mniejsza od P permutacja o prefiksie równym A istnieje wtedy i tylko wtedy, gdy ciąg B nie jest posortowany rosnąco. Stąd możemy wysnuć wniosek, że P i Q będą miały wspólny prefiks o długości $k-1$, gdzie k to największy taki indeks, że $p_k > p_{k+1}$ (jeżeli P nie jest permutacją $(1, \ldots, n)$, która nie ma leksykograficznego poprzednika, to taki indeks istnieje). Sufiks $(p_k, \ldots, p_n)$ zaczynający się od k-tej pozycji nazwijmy *ogonem* permutacji P. Permutację Q możemy opisać w następujący sposób: ma ona wspólny prefiks z P o długości $k-1$ (tzn. $q_i = p_i$ dla $i = 1, \ldots, k-1$), q_k jest największą liczbą ze zbioru $\{p_{k+1}, \ldots, p_n\}$ mniejszą od p_k oraz $(q_{k+1}, \ldots, q_n)$ jest posortowanym

malejąco ciągiem złożonym z liczb ze zbioru $\{1, \ldots, n\}$, które nie występują w zbiorze $\{q_1, \ldots, q_k\}$.

Przykładowo dla permutacji $P = (2, 4, 1, 3, 5)$ mamy $k = 2$, zatem $q_1 = p_1 = 2$, największą liczbą z liczb $\{1, 3, 5\}$, która jest mniejsza od $p_2 = 4$, jest $q_2 = 3$, a $(q_3, q_4, q_5) = (5, 4, 1)$. Zatem bezpośrednim poprzednikiem leksykograficznym permutacji P jest permutacja $Q = (2, 3, 5, 4, 1)$. W dalszej części rozwiązania, bezpośredniego leksykograficznego poprzednika permutacji P będziemy oznaczać przez $prev(P)$.

Minimalna liczba zamian

Skoro już wiemy, jak wygląda leksykograficzny poprzednik permutacji P, to zastanówmy się, jaka jest minimalna liczba zamian par liczb, która pozwoli nam przejść od P do $prev(P)$.

Przyjmijmy, że ogon P to sufiks złożony z elementów o indeksach $[k, n]$. Przypomnijmy, że k możemy wyznaczyć jako największy taki indeks, że $p_k > p_{k+1}$. Na potrzeby tej sekcji dla prostoty załóżmy, że ogon ten jest permutacją liczb od 1 do t, i niech jego pierwszy element będzie równy r (mamy $r > 1$). Wtedy ogon wygląda następująco:

$$S_1 = (r, 1, \ldots, r-2, r-1, r+1, \ldots, t),$$

a jego poprzednik to

$$S_2 = (r-1, t, t-1, \ldots, r+1, r, r-2, \ldots, 1).$$

Zauważmy, że od S_1 do S_2 możemy przejść, najpierw zamieniając miejscami liczby r oraz $r-1$, a następnie odwracając sufiks $1, \ldots, r-2, r, r+1, \ldots, t$, otrzymując żądany $t, t-1, \ldots, r+1, r, r-2, \ldots, 1$. Odwrócenie ciągu c-elementowego można wykonać za pomocą $\lfloor \frac{c}{2} \rfloor$ operacji zamiany dwóch elementów[1]. W naszym przypadku $c = t - 1$, zatem jeżeli P ma ogon długości t, to potrafimy przejść z P do $prev(P)$ za pomocą $1 + \lfloor \frac{t-1}{2} \rfloor = \lfloor \frac{t+1}{2} \rfloor$ operacji.

Udowodnimy teraz, że jest to minimalna liczba takich operacji. Jest jasne, że jeżeli chcemy z pewnego ciągu A otrzymać ciąg B za pomocą operacji zamiany par elementów i te ciągi różnią się na d pozycjach, to potrzebujemy co najmniej $\lceil \frac{d}{2} \rceil$ operacji. Przekonamy się, że takie oszacowanie wystarczy nam do udowodnienia tezy.

Zauważmy na starcie, że ciągi S_1 i S_2 różnią się na pozycjach, na których w S_1 stoją liczby r i $r-1$. Dalej rozważymy dwa przypadki ze względu na parzystość t.

1. $t = 2l + 1$:

   Jeżeli ponumerujemy pozycje obu ciągów indeksami od 1 do t, to wtedy w S_1 liczby od 1 do $r-2$ stoją na pozycjach o parzystościach różnych od swoich wartości, a liczby od $r+1$ do t stoją na pozycjach o parzystościach takich samych jak swoje wartości. W przypadku S_2 mamy do czynienia z dokładnie odwrotną sytuacją. Stąd wniosek, że S_1 i S_2 różnią się na wszystkich pozycjach, zatem na otrzymanie S_2 z S_1 potrzeba co najmniej $\lceil \frac{t}{2} \rceil = l + 1 = \lfloor \frac{t+1}{2} \rfloor$ zamian.

[1] Przypomnijmy standardowe oznaczenia matematyczne: $\lfloor x \rfloor$ oraz $\lceil x \rceil$ oznaczają odpowiednio zaokrąglenia x do najbliższej liczby całkowitej w dół i w górę.

2. $t = 2l$:

   Po usunięciu pierwszych elementów z S_1 oraz S_2, pierwszy z nich staje się ciągiem rosnącym, a drugi malejącym. Ciąg rosnący i ciąg malejący mogą mieć co najwyżej jedną pozycję wspólną. Stąd wniosek, że S_1 i S_2 różnią się na co najmniej $t-1$ pozycjach, zatem do otrzymania S_2 z S_1 potrzeba co najmniej $\lceil \frac{t-1}{2} \rceil = l = \lfloor \frac{t+1}{2} \rfloor$ zamian.

W obu przypadkach oszacowaliśmy z dołu liczbę potrzebnych zamian przez liczbę, którą otrzymujemy we wcześniej opisanej metodzie, co dowodzi tego, że przedstawiony algorytm wykonuje minimalną liczbę zamian.

W szczególności dowiedzieliśmy się, że liczba zamian potrzebnych do przekształcenia S_1 w S_2 nie zależy od r, a jedynie od długości tych ciągów t. Pozwala nam to zdefiniować funkcję $g(t) = \lfloor \frac{t+1}{2} \rfloor$, która oznacza najmniejszą potrzebną liczbę zamian do przejścia od permutacji P do $prev(P)$, jeśli ogon permutacji P ma długość t.

Obliczenie sumarycznej długości zabawy

Mając do dyspozycji wniosek z poprzedniego akapitu, jesteśmy już w stanie napisać pewne (bardzo wolne) rozwiązanie. Dopóki nie otrzymamy permutacji identycznościowej, przechodzimy do poprzedniej leksykograficznie permutacji i zwiększamy liczbę potrzebnych zamian o odpowiednią wartość. Jednak takie rozwiązanie może działać bardzo długo, konkretnie w złożoności $O(n! \cdot n)$ lub $O(n!)$, jeżeli będziemy wyznaczali długość ogonu, idąc od końca permutacji. (Dociekliwemu Czytelnikowi pozostawiamy jako nieoczywiste zadanie udowodnienie, że średnią długość ogonu możemy ograniczyć przez stałą). Takie rozwiązanie wystarcza jedynie do rozwiązania pierwszego podzadania; w ogólności potrzebujemy czegoś zdecydowanie szybszego.

Na początku zauważmy, że analogiczny problem możemy zdefiniować dla dowolnych ciągów różnych liczb. Ciąg różnych liczb o długości m, tak samo jak ciąg wszystkich liczb od 1 do m, można spermutować na $m!$ sposobów i algorytm znajdowania leksykograficznego poprzednika wygląda w tym przypadku analogicznie. Dla ciągu $B = (b_1, \ldots, b_m)$ składającego się z różnych liczb możemy zdefiniować ciąg $compr(B) = C = (c_1, \ldots, c_m)$, gdzie c_i oznacza, ile liczb w ciągu B jest równych co najwyżej b_i. Ciąg C jest permutacją liczb od 1 do m. Na przykład $compr((7,3,5)) = (3,1,2)$. Jeżeli dla dowolnego ciągu różnych liczb $D = (d_1, \ldots, d_m)$ przez $ans(D)$ oznaczymy odpowiedź dla problemu analogicznego do problemu z treści, to jest jasne, że $ans(D) = ans(compr(D))$.

Przez $f(n)$ oznaczmy $ans((n, n-1, \ldots, 1))$. Zauważmy, że zaczynając od ciągu $(n, n-1, \ldots, 1)$, w trakcie zabawy Jaś napotka permutację $(n, 1, \ldots, n-1)$, a do tego momentu wykona $f(n-1)$ zamian. Ogon tej permutacji ma długość n, zatem w następnym kroku będzie musiał wykonać $g(n)$ zamian, otrzymując permutację $(n-1, n, n-2, n-3, \ldots, 1)$. Po pewnej liczbie kroków dojdzie do permutacji $(n-1, 1, \ldots, n-3, n-2, n)$, wykonując kolejne $f(n-1)$ zamian. Ogon tej permutacji ponownie ma długość n, zatem aby przejść do $(n-2, n, n-1, n-3, \ldots, 1)$, musi wykonać kolejne $g(n)$ zamian. Kontynuując to rozumowanie, dochodzimy do wniosku, że:

$$f(n) = n \cdot f(n-1) + (n-1) \cdot g(n),$$

co pozwala nam w czasie $O(n)$ wyznaczyć reszty z dzielenia $f(1), f(2), \ldots, f(n)$ przez zadaną stałą (wiedząc, że $f(1) = 0$). Otrzymujemy zarazem rozwiązanie podzadania 3.

Rozwiązanie ogólnego problemu wcale nie jest wiele trudniejsze. Jeżeli rozwiązujemy problem dla permutacji $P = (p_1, \ldots, p_n)$, to po pewnej liczbie zamian dojdziemy do ciągu $(p_1, 1, \ldots, p_1 - 1, p_1 + 1, \ldots, n)$ i od tego momentu wykonamy $(p_1 - 1) \cdot (f(n-1) + g(n))$ zamian, aby dojść do permutacji identycznościowej. Możemy zatem stwierdzić, że:

$$\begin{aligned} ans(P) &= ans((p_2, \ldots, p_n)) + (p_1 - 1) \cdot (f(n-1) + g(n)) \\ &= ans(compr((p_2, \ldots, p_n))) + (p_1 - 1) \cdot (f(n-1) + g(n)). \end{aligned}$$

Wartość $ans(compr((p_2, \ldots, p_n))$ możemy wyznaczyć, obliczając najpierw permutację $compr((p_2, \ldots, p_n))$ i wywołując się rekurencyjnie. Obliczenie permutacji $compr((p_2, \ldots, p_n))$ możemy łatwo wykonać w czasie $O(n)$ (pamiętajmy, że zbiór liczb $\{p_2, \ldots, p_n\}$ to zbiór liczb od 1 do n oprócz p_1), co daje nam rozwiązanie w czasie $O(n^2)$. Jest to zdecydowaną poprawą w stosunku do rozwiązania $O(n!)$ i pozwala zaliczyć podzadanie 2, ale wciąż nie wystarcza na zdobycie kompletu punktów.

Zauważmy jednak, że do obliczenia liczby zamian różniących $ans(P)$ oraz $ans((p_2, \ldots, p_n))$ potrzebujemy jedynie znać wartość elementu p_1. Jeżeli rozpatrujemy problem dla ogólnych ciągów różnych elementów, to wtedy analogiczna wersja wzoru z poprzedniego akapitu przedstawia się jako $ans(P) = ans((p_2, \ldots, p_n)) + s \cdot (f(n-1) + g(n))$, gdzie s to liczba elementów ciągu P mniejszych od p_1. Przewaga takiego zapisu polega na tym, że nie musimy wykonywać kosztownej kompresji przed wywołaniem rekurencyjnym. Niestety ma on też swoją wadę, mianowicie musimy znać wartość s. Zauważmy, że jeżeli oryginalną permutacją z zadania jest $(p_1, \ldots, p_n)$, to będziemy się wywoływać rekurencyjnie jedynie dla jej sufiksów. Po chwili zastanowienia możemy dojść do wniosku, że tak naprawdę wynik naszego zadania przedstawia się jako

$$s(1) \cdot (f(n-1) + g(n)) + s(2) \cdot (f(n-2) + g(n-1)) + \ldots + s(n-1) \cdot (f(1) + g(2)),$$

gdzie $s(i)$ to liczba liczb mniejszych od p_i spośród liczb $p_{i+1}, \ldots, p_n$. Jedyne, co musimy zatem zrobić, to efektywnie obliczyć wartości $s(i)$.

Będziemy potrzebowali struktury danych wspierającej zapytania $insert(i)$ oraz $less(i)$, gdzie $insert(i)$ dodaje element i do zbioru, a $less(i)$ odpowiada na zapytanie „Ile liczb mniejszych od i dodaliśmy już do zbioru?". Co więcej, w naszym zadaniu argumenty operacji są liczbami całkowitymi z przedziału $[1, n]$. Biorąc to wszystko pod uwagę, widzimy, że otrzymany problem to standardowy przykład na użycie drzew potęgowych lub przedziałowych (typu „dodaj punkt, sumuj na przedziale"), które to już wielokrotnie występowały w zadaniach z Olimpiady Informatycznej. Za pomocą podanych typów drzew możemy obsługiwać takie zapytania w złożoności czasowej $O(\log n)$. Do struktury danych będziemy dodawali elementy od prawej do lewej. Będziemy kolejno wykonywać polecenia $less(p_n), insert(p_n), less(p_{n-1}), insert(p_{n-1}), \ldots, less(p_1), insert(p_1)$. Odpowiedź na zapytanie $less(p_i)$ jest wartością $s(i)$.

Podsumowując, otrzymujemy rozwiązanie w złożoności czasowej $O(n \log n)$, które wystarczało do uzyskania kompletu punktów.