Rozwiązanie zadania M 1323.
Podzielmy nasze 100 monet na trzy rozłączne podzbiory A, B i C, mające odpowiednio 33, 33 i 34 elementy. Najpierw ważymy zbiory A i B. Jeśli jeden, powiedzmy A, jest cięższy, to wiemy, że zawiera co najwyżej jedną fałszywą monetę. Wybieramy wtedy z A dwie monety i ważymy je. Wiadomo, że ta, która nie jest lżejsza, jest prawdziwa. Jeśli jednak A i B mają tę samą masę, to mamy trzy przypadki: (a) A i B nie zawierają fałszywej monety, (b) A i B zawierają po jednej fałszywej monecie, (c) A i B zawierają po dwie fałszywe monety. Wybieramy dowolną monetę $x \in A$ i ważymy zbiory $B \cup \{x\}$ i C. Jeśli mają równe masy, to musiał zajść przypadek (b), więc każda moneta z $A \setminus \{x\}$ jest prawdziwa. Jeśli $B \cup \{x\}$ jest cięższy, to nie mógł zajść przypadek (c), więc moneta x jest prawdziwa. Wreszcie, jeśli C jest cięższy, to mamy przypadek (c), więc każda moneta z C jest prawdziwa.

	A	B	C
(a)	0	0	4
(b)	1	1	2
(c)	2	2	0

Tabelka pokazuje liczbę fałszywych monet w każdym ze zbiorów w odpowiednim przypadku.

Informatyczny kącik olimpijski (44): Neon

Tytułowe zadanie pojawiło się na zeszłorocznym Obozie Naukowo-Treningowym im. A. Kreczmara: *Neon składa się z dwóch słów. Na ile sposobów można zgasić niektóre litery w neonie, tak by litery, które pozostaną zapalone, w obu słowach układały się w ten sam niepusty podciąg? Ile różnych podciągów da się w ten sposób uzyskać?* Na przykład dla neonu `supermarket stokrotka` możemy uzyskać 17 różnych niepustych podciągów (`skt`, `sra`, `srk`, `srt` i ich krótsze podciągi) na 27 sposobów (np. `srk` możemy uzyskać na 2 sposoby). Wynik należy obliczyć modulo pewna liczba całkowita.

Zadanie rozwiążemy, używając metody programowania dynamicznego. Niech $a[1..n]$ i $b[1..m]$ będą słowami neonu. Będziemy wypełniać prostokątną tablicę $w[0..n, 0..m]$. Dla $0 \leqslant i \leqslant n$, $0 \leqslant j \leqslant m$ oznaczmy przez $w[i, j]$ liczbę sposobów, na jakie możemy zgasić litery w słowach $a[1..i]$ oraz $b[1..j]$, tak by pozostałe litery tworzyły ten sam (być może pusty) podciąg. Odpowiedzią do pierwszej części zadania jest, oczywiście, $w[n, m] - 1$.

Jeśli $i = 0$ lub $j = 0$ (tzn. gdy jedno ze słów jest puste), to możemy utworzyć tylko pusty podciąg, zatem $w[i, j] = 1$.

Załóżmy zatem, że $i, j > 0$, i spójrzmy na ostatnie litery słów $a[1..i]$ i $b[1..j]$. Załóżmy na początek, że są one równe, tzn. $a[i] = b[j]$. Rozważmy cztery przypadki w zależności od tego, czy zostawiamy te litery zapalone, czy też nie. Jeśli obie będą zapalone, to znaczy, że uzyskane podciągi powstają z rozszerzenia jednego z podciągów dla słów $a[1..i-1]$ i $b[1..j-1]$ o dodatkową literę. To daje $w[i-1, j-1]$ możliwości. Liczba możliwych podciągów w przypadku, gdy litera $b[j]$ zostanie zgaszona, wynosi $w[i, j-1]$. Analogicznie liczba podciągów, gdy litera $a[i]$ zostanie zgaszona, wynosi $w[i-1, j]$. Zauważmy, że te podciągi, które wystąpią w przypadku zgaszenia obu tych liter, uwzględniliśmy już w wyniku, z tym że policzyliśmy je dwukrotnie, należy zatem odjąć ich liczbę (jest ich $w[i-1, j-1]$). Ostatecznie dostajemy prostą zależność $w[i, j] = w[i, j-1] + w[i-1, j]$.

Przypadek $a[i] \neq b[j]$ jest trochę łatwiejszy: nie możemy naraz zapalić liter $a[i]$ i $b[j]$, zatem $w[i, j] = w[i, j-1] + w[i-1, j] - w[i-1, j-1]$.

Zajmiemy się teraz drugą częścią zadania: obliczeniem liczby różnych wspólnych podciągów. Na pierwszy rzut oka może się wydawać, że w tym przypadku będziemy musieli zastosować zupełnie inną strategię i potrzebna będzie jakaś struktura danych, która umożliwi nam wyłapywanie duplikatów. Okazuje się jednak, że i ten problem można rozwiązać za pomocą zaskakująco prostej zależności rekurencyjnej.

Niech $p[i, j]$ oznacza liczbę różnych wspólnych podciągów, które możemy utworzyć w słowach $a[1..i]$, $b[1..j]$. Dla $i = 0$ lub $j = 0$ mamy, oczywiście, $p[i, j] = 1$. Znów patrzymy na ostatnie litery słów. Niech $a[i] \neq b[j]$. Okazuje się, że w tym przypadku mamy po prostu $p[i, j] = p[i, j-1] + p[i-1, j] - p[i-1, j-1]$, a rozumowanie jest analogiczne jak dla tablicy w.

Niech teraz $a[i] = b[j]$. Każdy wspólny podciąg słów $a[1..i]$ i $b[1..j]$ należy do co najmniej jednego z dwóch zbiorów: zbioru A tych różnych podciągów, które mogą powstać, gdy obie litery $a[i]$, $b[j]$ są zapalone, oraz zbioru B tych różnych podciągów, które mogą powstać, gdy obie te litery są zgaszone. Zatem $p[i, j] = |A| + |B| - |A \cap B|$. Zbiory A i B mają po $p[i-1, j-1]$ elementów, pozostaje zatem obliczyć liczność ich części wspólnej $A \cap B$. Niech $\alpha(i)$ oznacza największą liczbę dodatnią mniejszą niż i, dla której $a[\alpha(i)] = a[i]$; analogicznie niech $\beta(j)$ będzie największą liczbą $0 < \beta(j) < j$, dla której $b[\beta(j)] = b[j]$. Jeśli choć jedna z tych liczb nie istnieje, to $|A \cap B| = 0$. W przeciwnym przypadku mamy $|A \cap B| = p[\alpha(i) - 1, \beta(j) - 1]$ i wówczas $p[i, j] = 2 \cdot p[i-1, j-1] - p[\alpha(i) - 1, \beta(j) - 1]$.

Pozostaje oszacować złożoność naszego algorytmu. Tablice α i β możemy obliczyć w czasie $O(n + m)$ i takiej pamięci. Czas wypełniania tablic w i p to $O(nm)$, bezpośrednia realizacja da też taką złożoność pamięciową. Zauważmy jednak, że w przypadku tablicy w możemy zastosować standardową sztuczkę: jeśli wypełniamy tablicę wierszami, wystarczy trzymać w pamięci jedynie aktualny (i-ty) i ostatnio wypełniony ($(i-1)$-szy) wiersz – nigdy bowiem nie odwołujemy się do komórek z wierszy o numerach mniejszych niż $i - 1$. Problem powstaje w przypadku tablicy p, w której korzystamy z wiersza o numerze $\alpha(i) - 1$. Tu jednak radzimy sobie następująco: oprócz ostatnich dwóch wierszy trzymamy dodatkowo tablicę $q[1..n]$. W przypadku, gdy $a[i] = b[j]$, uaktualniamy ją według wzoru $q[j] = p[i-1, j-1]$. Dzięki temu mamy $p[i, j] = 2 \cdot p[i-1, j-1] - q[\beta(j)]$. Ostatecznie złożoność pamięciowa algorytmu wyniesie $O(n + m)$.

Tomasz IDZIASZEK