Szkice rozwiązań

XVIII OIJ, zawody I stopnia
25.09.2023 – 8.01.2024

Olimpiada Informatyczna Juniorów

Dziękujemy Wam za udział w zawodach I stopnia XVIII Olimpiady Informatycznej Juniorów. Mamy nadzieję, że dobrze bawiliście się rozwiązując zadania olimpijskie i że udało się Wam zdobyć jak najwięcej punktów. Poniżej znajduje się opis proponowanych rozwiązań.

Artykuły prawne

Zastanówmy się przez chwilę, jak będą wyglądały wszystkie dopiski literowe. Warto sobie rozpisać kilka przykładów, co zrobimy w poniższej tabeli.

N	wynik	N	wynik	N	wynik	N	wynik
1	a	27	za	53	zza	79	zzza
2	b	28	zb	54	zzb	80	zzzb
3	c	29	zc	55	zzc	81	zzzc
			...				
25	y	51	zy	77	zzy	103	zzzy
26	z	52	zz	78	zzz	104	zzzz

Można zauważyć, że zawsze zacznie się od pewnej liczby liter z (być może zera), po których może wystąpić dodatkowa litera (być może także równa z). Dla ustalenia uwagi, załóżmy że ostatnia litera nigdy nie jest częścią bloku złożonego z z, nawet jeżeli jest z.

Mamy zatem dwa podproblemy: ile z należy wypisać i jaką literę należy wypisać na końcu. Podproblemy te rozwiążemy osobno.

Po pierwsze zastanowimy się jak wiele z musi się znaleźć na początku. Z powyższej tabeli można zaobserwować, że punkty od 1 do 26 będą zawierały zero liter z. Z kolei wszystkie punkty od 27 do 52 zawierają jedno z, a punkty od 53 do 78 dwa z, i tak dalej. Innymi słowy, dopóki N jest większe od 26, możemy zawsze wypisać pojedyncze z i zredukować problem, do prostszego problemu z $N - 26$. De facto oznacza to przejście w naszej tabeli do kolumny po lewej stronie. Możemy zatem w pojedynczej pętli `while`, dopóki N większe niż 26, wypisywać jedno z i odejmować 26 od N.

Teraz pochylmy się nad drugim podproblemem. Zauważmy, że metodyczne odejmowanie 26, które zastosowaliśmy w poprzedniej części, sprawi, że znajdziemy się zawsze w pierwszej kolumnie tej tabeli (od 1 do 26). Wystarczy zatem, że będziemy znali odpowiedzi dla pierwszej kolumny tej tabeli. To okazuje się proste, bo widzimy, że wystarczy wypisać jedynie N-tą kolejną literę alfabetu angielskiego (poczynając od a).

Warto tutaj wspomnieć, że litery (i znaki) w komputerze są reprezentowane przez ich kody ASCII (skrót z angielskiego *American Standard Code for Information Interchange*). Zobacz `https://pl.wikipedia.org/wiki/ASCII`, aby dowiedzieć się więcej. Nam wystarczy wiedzieć, że każda litera ma jakiś kod i litery te są w kolejności alfabetycznej w tym kodzie, tj. a ma kod 97, b ma kod 98, c ma kod 99, i tak dalej aż do z, które ma kod 122. Zatem, aby dostać kolejną literę, wystarczy zwiększyć jej kod o 1.

W C++ możemy to zrobić dość prosto:

```cpp
char litera = 'a';
litera++;
cout << litera;  // Wypisze 'b'.
```

W Pythonie nie możemy bezpośrednio użyć kodu ASCII. Do konwersji pomiędzy kodem a znakiem, używamy funkcji `ord` i `chr`:

```python
litera = 'a'
# Funkcja ord() zwraca wartość ASCII znaku, np.  ord('a') zwróci 97.
kod = ord(litera)
kolejna_litera = kod + 1
```


© Olimpiada Informatyczna Juniorów, 2024

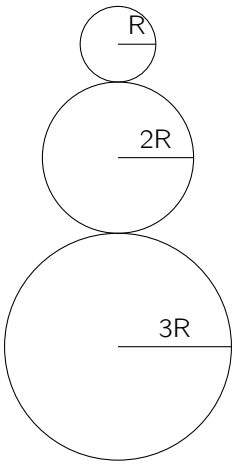
Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

CC BY NC SA

```
# Funkcja chr() przekształca wartość ASCII na odpowiadający jej znak,
# np.  chr(97) zwróci 'a'.
print(chr(kolejna_litera))  # Wypisze 'b'.
```

Idealny bałwan

Oznaczmy przez R promień najmniejszej kuli. Wtedy promienie wszystkich trzech kul to R, $2R$ i $3R$.



Aby zbudować te kule potrzebujemy R^2, $4R^2$, $9R^2$ wiaderek śniegu odpowiednio, co sumarycznie daje nam $R^2 + 4R^2 + 9R^2 = 14R^2$. Z kolei wysokość tego bałwana to $2R + 4R + 6R = 12R$.

Zadanie zatem sprowadza się do znalezienia największego naturalnego R, takiego że liczba potrzebnych wiaderek ($14R^2$) jest nie większa niż N, czyli:

$$14R^2 \leq N$$

Lub równoważnie:

$$R \leq \sqrt{\frac{N}{14}}$$

Możemy użyć standardowych funkcji z bibliotek matematycznych, aby znaleźć pierwiastek z danej liczby (funkcja `sqrt`), oraz obliczyć największą liczbę całkowitą nieprzekraczającą danej liczby (funkcja `floor` (podłoga)).

Wtedy wynikiem (szukaną wysokością) jest $12 \cdot R$.

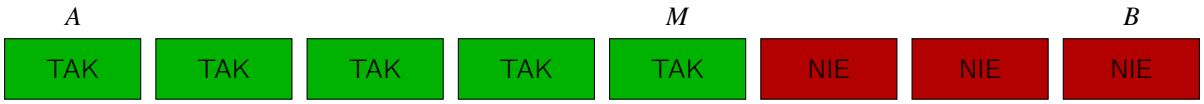
Wyszukiwanie binarne

Istnieje jednak inne, dużo ciekawsze rozwiązanie, które można zbudować na podstawie algorytmu wyszukiwania binarnego. Zmieńmy trochę pytanie postawione w zadaniu, i spróbujmy odpowiedzieć, czy dla danej liczby M, czy jest możliwe zbudowanie bałwana, dla którego M będzie stanowiło promień najmniejszej kuli. Zgodnie ze wcześniejszymi rozważeniami, wystarczy sprawdzić, czy zużyjemy nie więcej niż $14M^2$ wiaderek ze śniegiem. Możemy zatem sprawdzić dla kolejnych wartości M od 1 do nieskończoności, kiedy przekroczymy liczbę N wiaderek ze śniegiem – odpowiedzią będzie ostatnie M dla którego było to możliwe.

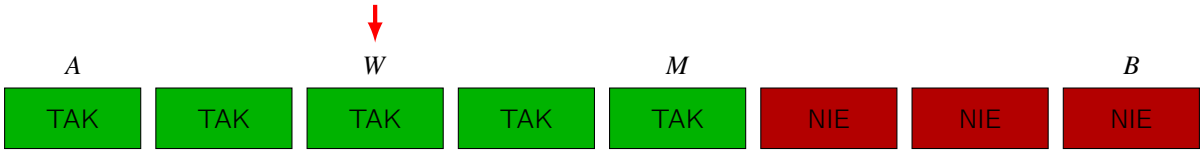
Oczywiście wynik może być bardzo duży (rzędu kilkuset milionów). Skupmy się zatem na szybszym wyznaczeniu maksymalnego M, dla którego jest możliwe ulepienie bałwana. De facto M jest czymś co możemy nazwać "punktem granicznym" – dla każdej wartości niewiększej od M (w tym M) jest możliwe zbudowanie bałwana, natomiast dla każdej wartości powyżej M nie jest to możliwe.

Niech A będzie liczbą, dla której na pewno wiemy, że możemy zbudować bałwana, gdzie najmniejszy promień to A, natomiast niech B – pewną liczbą dla której nie możemy już zbudować bałwana, bo nie mamy wystarczająco wiaderek ze śniegiem. Wiemy, że punkt graniczny M znajduje się pomiędzy A i B.

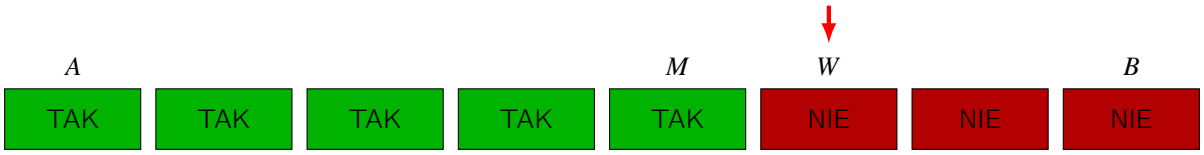
Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".



Teraz zamiast sprawdzać wszystkie liczby pomiędzy A i B po kolei. Możemy sprawdzić odpowiedź dla dowolnego punktu pomiędzy A i B, powiedzmy W. Jeżeli odpowiedź będzie pozytywna – oznacza to, że punkt M musi znajdować się pomiędzy W i B.



W przeciwnym wypadku, jeżeli odpowiedź będzie negatywna, punkt M musi znajdować się pomiędzy A i W.



Teraz jeżeli wybierzemy nasz punkt W sprytnie, to możemy pozbyć się sporej części liczb, które musimy sprawdzić. Konkretniej, zawsze opłaca się wybrać punkt pośrodku, pomiędzy A i B. Wtedy zawsze liczbę liczb, które musimy sprawdzić, zmniejszamy o połowę.

Algorytm ten kończy się, kiedy znaleźliśmy nasz punkt graniczny, czyli M. Oznacza, to że musimy mieć takie A i B, dla których $A+1=B$, jako że A zawsze będzie liczbą, dla której odpowiedzią jest TAK, a B zawsze będzie liczbą dla której odpowiedzią jest NIE.

To jest idea całego algorytmu. Opowiemy teraz jak go zaimplementować.

Zaczynamy od ustalenia wartości A i B, tak aby wynik zawsze mieścił się pomiędzy tymi wartościami, a także żebyśmy byli pewni że dla wartości A możemy zawsze wybudować bałwana, a dla wartości B nie było to możliwe. Warto także zwrócić uwagę, żeby B nie było zbyt duże. Po pierwsze, im mniejsze B, tym mniej wartości musimy przejrzeć. Po drugie zbyt duża wartość B może spowodować, że operacje arytmetyczne mogą spowodować wyjście poza typ liczb, które zastosowaliśmy. W tym zadaniu proponujemy wybranie np. $A=1$ oraz $B=10^9$.

Kiedy wybraliśmy już te wartości, w pętli `while` sprawdzamy czy różnica pomiędzy A i B wynosi więcej niż 1. Jeżeli tak, to wybieramy wartość pomiędzy A i B, powiedzmy $M=\frac{A+B}{2}$. Dla danego M sprawdzamy, czy mamy wystarczająco dużo śniegu, aby zbudować tego bałwana, czyli sprawdzamy czy $14M^2 \leq N$. Jeżeli ten warunek jest spełniony, to ustalamy nowe A jako M. Jeżeli nie – $B=M$. Kiedy finalnie $A+1=B$ i wyjdziemy z pętli, wiemy że A jest maksymalnym możliwym promieniem najmniejszej kuli, zatem możemy po prostu wypisać $12 \cdot A$.

Notka od autora: Warto zauważyć, że w naszym świecie, aby zbudować kulę śniegu, potrzebujemy zwykle ilości śniegu, której wartość zależy sześciennie od promienia kuli. Czy oznacza to, że Bajtocja jest światem 2D? A może autor zadania nie chciał wymagać od zawodników obliczania pierwiastka sześciennego? Pewnie nigdy się nie dowiemy...

Różnorodny ciąg

Zastanówmy się najpierw, jakie operacje musimy wykonać. Następnie pokażemy jak te operacje wykonać szybko.

Skupmy się najpierw na największej wartości w ciągu. Jeżeli jest to 0, to nie musimy nic robić. Zakładamy dalej, że ta największa wartość jest większa niż 0. Jeżeli mamy dokładnie jeden element o maksymalnej wartości (oznaczmy tą wartość przez W), to ponownie nie musimy nic robić – możemy nawet usunąć ten element z ciągu. Jeżeli mamy więcej niż jeden element o tej wartości to musimy użyć po jednej operacji na wszystkich elementach poza jednym, tak aby pozostał tylko jeden element o wartości W.

Zwróćmy uwagę, że może to nam spowodować kolejne problemy (teraz możemy mieć za dużo elementów o wartości $W-1$). Możemy jednak zastosować ten sam algorytm do kolejnej wartości i kontynuować, póki nie dojdziemy do 0. Można

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

pokazać, że ten algorytm prowadzi do minimalnej liczby operacji – intuicyjnie widać, że wszystkie operacje są "wymuszone" i nie wykonujemy żadnych niepotrzebnych operacji.

Teraz pomyślmy jak zaimplementować ten algorytm. Oczywiście możemy za każdym razem znaleźć maksymalną wartość i zmniejszać o jeden wszystkie elementy o tej wartości poza jednym, jednak możemy pokazać, że zajmie to nie tylko dużo czasu, ale także operacji. Wyobraźmy sobie, że zaczynamy do $N = 100\,000$ elementów, wszystkie równe 10^5. Wtedy musimy użyć $99\,999$ operacji i będziemy mieli $99\,999$ elementów o wartości $99\,999$. W kolejnym kroku użyjemy $99\,998$ operacji i zostaniemy z $99\,998$ elementami o wartości $99\,998$, i tak dalej...

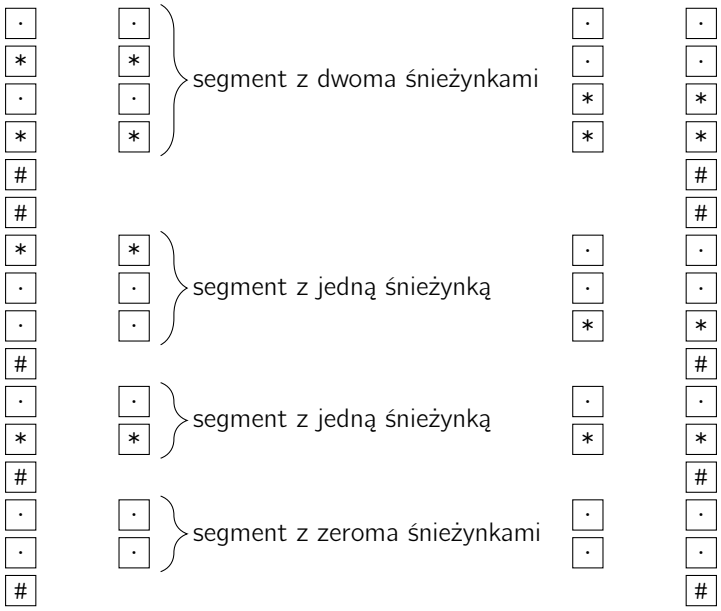
Zastosujemy zatem inną strategię. Będziemy utrzymywali tablicę $wystapienia$, w której indeksami będą wartości ciągu, a jej wartościami – ile razy mamy tą wartość w naszym ciągu. Teraz możemy łatwiej zaimplementować nasz algorytm. Przeglądamy tę tablicę od prawej do lewej (od największej wartości do najmniejszej). Kiedy widzimy, że wartość $wystapienia[i]$ w tej tablicy jest większa niż 1 (co oznacza to że mamy więcej niż jeden element o wartości i), wykonujemy jedną operację na $wystapienia[i] - 1$ elementach równych 1, co powoduje, że zmniejszamy $wystapienia[i]$ do 1, a do $wystapienia[i-1]$ dodajemy $wystapienia[i] - 1$. Liczba operacji wzrasta o $wystapienia[i] - 1$. Kontynuujemy ten algorytm dopóki nie dojdziemy do 0.

Domek

Jako że śnieżynki mogą spadać wyłącznie w dół, możemy zauważyć, że każdą kolumnę możemy rozważać osobno.

W każdej kolumnie, znaki # stanowią przeszkody, których śnieżynki nie mogą przekraczać. Znaki te dzielą zatem nasze kolumny na segmenty, w których śnieżynki mogą się przemieszczać (wyłącznie w dół).

Możemy przesuwać śnieżynki w tych segmentach pojedynczo, jednak może się to okazać czasochłonne. Dużo łatwiej jest zauważyć, że wszystkie śnieżynki finalnie spadną na sam dół tego segmentu, zatem możemy jedynie zliczyć ile śnieżynek mamy w tym segmencie i wypełnić dół tego segmentu tyloma śnieżynkami, ile mamy w danym segmencie, a pozostałą część segmentu pozostawić pustą.



Wykonujemy ten algorytm dla każdej kolumny naszego rysunku, a następnie wypisujemy cały rysunek.

OIJ

Zacznijmy od kilku pomysłów, które nie doprowadzają co prawda bezpośrednio do poprawnego rozwiązania zadania, ale budują intuicję, jak można myśleć o zadaniu i prowadzą do ostatecznej konstrukcji, której użyjemy.

Załóżmy, że wartość N (oczekiwana liczba podciągów `OIJ` napisu, który mamy wypisać) jest niewielka. Moglibyśmy wypisać napis postaci:

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

$$\texttt{OI}\underbrace{\texttt{JJ}\ldots\texttt{J}}_{N}$$

Istotnie, każdy podciąg `OIJ` składa się wtedy pierwszej i drugiej literki wypisanego napisu oraz jednej z N dostępnych liter `J`. Jedynym problemem z tym podejściem jest to, że każdy dodatkowy podciąg `OIJ` wymaga dodatkowej literki `J` w wypisanym napisie, a warunki zadania wymuszają oszczędne gospodarowanie wypisywanymi literkami (długość wypisanego napisu nie może przekroczyć $6\,000$ pomimo tego, że $N \leq 10^9$).

Pewną optymalizacją powyższego pomysłu byłoby zastosowanie napisu postaci:

$$\underbrace{\texttt{OO}\ldots\texttt{O}}_{a}\underbrace{\texttt{II}\ldots\texttt{I}}_{b}\underbrace{\texttt{JJ}\ldots\texttt{J}}_{c}$$

W ten sposób moglibyśmy poradzić sobie z każdym N, które dałoby się zapisać jako iloczyn trzech stosunkowo małych czynników (o łącznej sumie nie przekraczającej $6\,000$). Przykładowo, jeżeli $N = 8\,765\,432 = 2 \cdot 2 \cdot 2 \cdot 13 \cdot 89 \cdot 947$, moglibyśmy przyjąć $a = 2 \cdot 2 \cdot 2 \cdot 13 = 104, b = 89, c = 947$ i wtedy każda ze 104 pierwszych literek `O` z każdą z kolejnych 89 literek `I` z każdą z kolejnych 947 literek `J` tworzy podciąg `OIJ`. Uzyskaliśmy więc żądaną liczbę podciągów $abc = N$.

Niestety, nie każda liczba może być przedstawiona jako iloczyn trzech małych czynników. Wystarczy, aby w rozkładzie na czynniki pierwsze liczby N znajdował się czynnik większy niż $6\,000$ i takie rozwiązanie nie może działać. Powodem jest to, że w ten sposób potrafimy robić jedynie redukcję multiplikatywną problemu: mając za cel uzyskanie N podciągów `OIJ`, możemy to sprowadzić do problemu uzyskania $\frac{N}{a}$ podciągów `IJ` dla dowolnego a (przy założeniu, że N jest wielokrotnością a). Przydałoby się również stworzyć sobie sposób na redukcję addytywną, tzn. na przykład zamieniającą problem uzyskania N podciągów `OIJ` na $N-a$ podciągów `IJ`. Jest to niestety trochę „myślenie życzeniowe", ponieważ dokładnie takiej redukcji nie da się stworzyć, ale możliwe jest uzyskanie niewiele gorszego rezultatu.

Będziemy analizować ciągi postaci:

$$\texttt{OI}\underbrace{\texttt{JJ}\ldots\texttt{J}}_{a}\underbrace{\texttt{II}\ldots\texttt{I}}_{b}\underbrace{\texttt{JJ}\ldots\texttt{J}}_{c}\underbrace{\texttt{OO}\ldots\texttt{O}}_{d}\underbrace{\texttt{II}\ldots\texttt{I}}_{e}\underbrace{\texttt{JJ}\ldots\texttt{J}}_{f}$$

Wszystkie podciągi `OIJ` są wtedy postaci (małe literki oznaczają odpowiedni znak z grupy powyżej):

- `OI`a (jest ich a),
- `OI`c (jest ich c),
- `OI`f (jest ich f),
- `O`bc (jest ich bc),
- `O`bf (jest ich bf),
- `O`ef (jest ich ef),
- def (jest ich def).

Łącznie więc, słowem tej postaci, uzyskamy $a + c + f + bc + bf + ef + def$ podciągów `OIJ`. Naszym celem jest teraz dobrać wartości zmiennych $a, b, \ldots, f$, żeby wartość tego wyrażenia była równa liczbie N wczytanej z wejścia.

Zwiększenie a o 1 pozwala precyzyjnie zwiększyć liczbę podciągów `OIJ` o 1, jak w początkowym pomyśle.

Zwiększenie c o 1 pozwala zwiększyć liczbę podciągów `OIJ` o $b + 1$. Gdyby dobrać $b = 999$, możliwe jest precyzyjne zwiększanie liczby podciągów o $1\,000$, dodając jedynie jedną literkę (zwiększając c o 1).

Zwiększenie e o 1 pozwala zwiększyć liczbę podciągów `OIJ` o $f + df$. Gdyby dobrać $d = 999$ oraz $f = 1\,000$, możliwe jest precyzyjne zwiększanie liczby podciągów o $1\,000\,000$, dodając jedynie jedną literkę (zwiększając e o 1).

Możemy więc przedstawić liczbę N w systemie o podstawie $1\,000$ i „cyfry" (piszemy w cudzysłowie, bo w tym systemie cyfry są od 0 do 999 włącznie) jedności, tysięcy i milionów podstawić do zmiennych a, c oraz e.

Bardziej precyzyjnie, wystarczy przyjąć:

- $e \leftarrow \lfloor (N - 1\,000\,000)/1\,000\,000 \rfloor$,

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

- $c \leftarrow \lfloor (N - 1\,000\,000)/1\,000 \rfloor \bmod 1\,000$,
- $a \leftarrow (N - 1\,000\,000) \bmod 1\,000$.

To bardzo korzystne, bo każda grupa literek będzie miała co najwyżej 999 znaków, poza grupą f, która będzie miała $1\,000$. A zatem, w każdym przypadku, dla $N \leq 10^9$, użyjemy mniej niż zakładany limit $6\,000$ znaków.

To byłby już koniec, gdyby nie fakt, że, niejako „przez przypadek", utworzyliśmy jeszcze podciągi `OIJ`, które nie zawierają znaków z grup a, c oraz e. Jest ich dokładnie $f + bf = 1\,000\,000$, a więc obecne rozwiązanie działa jedynie dla $N \geq 10^6$, jeżeli przyjąć e o 1 mniejsze niż liczba pełnych milionów w N (jak zrobiliśmy to nie komentując tego we wzorach powyżej).

Dla mniejszych N możliwe jest jednak analogiczne działanie, przyjmując mniejsze podstawy systemów pozycyjnych. Przykładowo:

- dla $N \in [10\,000, 999\,999]$ możemy przyjąć $b = 99, d = 99, f = 100$ (system pozycyjny o podstawie 100),
- dla $N \in [100, 9\,999]$ możemy przyjąć $b = 9, d = 9, f = 10$ (system pozycyjny o podstawie 10),
- dla $N \leq 99$ możemy po prostu wypisać napis `OIJ..J` jak w początkowym pomyśle, zawierający tyle samo `J` ile jest równe N.

Podzielność

Spróbujemy rozwiązać to zadanie używając informacji o podzadaniach.

Rozwiązanie dla $K \in \{3, 9\}$

Reguła podzielności przez 3 i 9 dotyczy sumy cyfr liczby. Liczba jest podzielna przez 3 lub 9 wtedy i tylko wtedy, gdy jej suma cyfr jest podzielna przez, odpowiednio, 3 lub 9.

Jeżeli zatem suma cyfr podanych na wejściu jest niepodzielna przez K, wypisujemy 0 i kończymy. W przeciwnym razie, dowolne ustawienie cyfr liczby N tworzy liczbę podzielną przez K i wystarczy policzyć ile jest takich ustawień.

Niech C_i oznacza liczbę dostępnych cyfr i w liczbie N, zaś $C = C_0 + C_1 + \ldots + C_9$ i wyobraźmy sobie, że każda cyfra jest pomalowana pewnym (innym dla każdej cyfry) kolorem. W ten sposób nawet te same cyfry stają się rozróżnialne. Wówczas możemy je ustawić na $C!$ sposobów. W ten sposób niepotrzebnie jednak osobno policzyliśmy ustawienia tych samych cyfr, ale inaczej pomalowanych. Przykładowo, jeżeli dostępne były trzy jedynki oraz dwie piątki, to każda liczba została policzona 12 razy: każde z sześciu ustawień jedynek w różnych kolorach z każdym z dwóch ustawień piątek w różnych kolorach zostało policzone osobno. W ogólności, każda liczba została policzona $R = C_0! \cdot C_1! \cdot \ldots \cdot C_9!$ razy. Wystarczy więc podzielić $C!$ przez R, aby każdą istotnie różną liczbę (już bez rozróżnienia kolorów cyfr) policzyć dokładnie raz.

W ten sposób uzyskaliśmy wzór na tzw. „kombinacje z powtórzeniami", który pozwala rozwiązać podzadanie dla $K \in \{3, 9\}$:

$$\frac{C!}{C_0! \cdot C_1! \ldots C_9!}$$

Należy być ostrożnym przy doborze zakresu zmiennych: dla danych z zadania, wartość $C!$ ledwie mieści się w zakresie zmiennej 64-bitowej `long long int` w C++.

Rozwiązanie dla $K \in \{2, 5, 10\}$

Reguły podzielności przez 2, 5, 10 dotyczą jedynie ostatniej cyfry, która musi być podzielna przez odpowiednio 2, 5 oraz 10, aby cała liczba była podzielna przez 2, 5 lub 10.

Zadanie sprowadza się więc do ustalenia na ile sposobów można przestawić cyfry liczby N, aby na końcu znalazła się jedna z odpowiednich cyfr. Tę ostatnią cyfrę można zgadnąć (przetestować wszystkie dostępne możliwości) i usunąć z multizbioru dostępnych cyfr (zmniejszyć odpowiednie C_i). Te pozostałe cyfry możemy ustawić w kolejności jak chcemy (a liczbę sposobów na jakie możemy tego dokonać ustalamy jak w rozwiązaniu dla $K \in \{3, 9\}$).

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".

Rozwiązanie dla $K = 6$

Liczba jest podzielna przez 6 wtedy i tylko wtedy, gdy jest podzielna przez 2 oraz 3.

W związku z tym wystarczy połączyć pomysł dla $K = 2$ oraz $K = 3$: sprawdzić czy suma cyfr jest podzielna przez 3 (i wypisać 0 jeżeli nie) i uruchomić rozwiązanie dla $K = 2$.

Rozwiązanie dla $K \in \{4, 8\}$

Ponieważ 4 jest dzielnikiem 100, zaś 8 jest dzielnikiem $1\,000$, aby stwierdzić czy liczba jest podzielna przez 4, wystarczy sprawdzić czy liczba uzyskana z dwóch ostatnich cyfr liczby N się dzieli przez 4. Analogicznie, podzielność N przez 8 wynika z podzielności liczby uzyskanej z trzech ostatnich cyfr liczby N przez 8.

Dla $K = 4$, możliwe jest zatem zgadnięcie każdej końcówki $\{00, 04, 08, \ldots, 96\}$, zmniejszenie odpowiednich $C_?$, na wzór rozwiązania dla $K \in \{2, 5, 10\}$ i zsumowanie wyników. Analogicznie, dla $K = 8$ wystarczy sprawdzić każdą końcówkę $\{000, 008, 016, \ldots, 992\}$.

Rozwiązanie dla $K = 7$

Skonstruujemy rozwiązanie ogólne, które będzie działało dla dowolnego K, również tych, które już rozważyliśmy, a nawet dla nieco większych K niż dozwolone w zadaniu i wyprowadzimy ogólną cechę podzielności przez dowolną liczbę.

Załóżmy, że konstruujemy liczbę podzielną przez K, stawiając cyfry od lewej do prawej (od najbardziej do najmniej znaczącej pozycji dziesiątkowej). Powiedzmy, że użyliśmy już kilku cyfr i ustaliliśmy początkowy fragment F liczby. Kluczowa obserwacja jest taka, że analizując jak zmienia się podzielność przez K, po dostawieniu do F cyfry c z prawej strony, nie potrzebujemy wcale dokładnie znać całego fragmentu F. Wystarczy znać resztę z dzielenia F przez K, czyli wartość $F \bmod K$. Istotnie, jeżeli F dawał resztę r z dzielenia przez K, to po dostawieniu cyfry c z prawej strony F, otrzymamy liczbę, która daje resztę $(10r + c) \bmod K$.

Spróbujemy skorzystać z techniki programowania dynamicznego. Będziemy pamiętać ciąg $(C_0, C_1, \ldots, C_9)$ liczb użytych cyfr każdego rodzaju oraz uzyskaną już resztę r z dzielenia przez K z już zapisanego fragmentu F liczby. Wynikiem takiego podproblemu (nazwijmy go $R[(C_0, C_1, \ldots, C_9)][r]$) jest liczba fragmentów F, które dają resztę r, które dają się skonstruować zużywając konkretny (multi)podzbiór cyfr: dokładnie C_0 cyfr zero, C_1 cyfr jeden, ..., C_9 cyfr dziewięć.

Przechodzenie między tymi podproblemami w zasadzie zostało już opisane: znając $v = R[(C_0, C_1, \ldots, C_9)][r]$ możemy powiększyć (dla $i \in \{0, 1, \ldots, 9\}$) o v każdy z wyników $R[(C_0, C_1, \ldots, C_i, \ldots, C_9)][(10r + i) \bmod K]$. Startujemy ze stanu $R[(0, 0, \ldots, 0)][0] = 1$ („pusta" liczba jest jedna, nie zużywa żadnych cyfr i ma resztę 0), a ostatecznie naszym celem jest obliczyć wartość stanu $R[(C_0, C_1, \ldots, C_9)][0]$, dla ciągu $(C_0, C_1, \ldots, C_9)$ odpowiadającego liczbie N z wejścia.

Takich stanów nie ma w przypadku naszego zadania zbyt wiele: jest ich na pewno nie więcej niż $2^{19} \cdot 7$ (liczba podzbiorów pozycji użytych cyfr w liczbie N razy liczba reszt z dzielenia przez 7). W praktyce znacznie mniej, bo naiwnie licząc (tak jak w analizie z poprzednich podzadań) niepotrzebnie rozróżniliśmy te same cyfry „w różnych kolorach".

Aby uniknąć potrzeby tworzenia tablicy jedenastowymiarowej (tworzenia osobnej indeksacji dla każdego C_i oraz r z osobna), możemy zakodować ciąg $(C_0, C_1, \ldots, C_9)$ w jednej liczbie 64-bitowej: wystarczy ostatnie pięć bitów poświęcić na zapisanie C_0 (pozwalających zakodować wartości od 0 do 31 włącznie, czyli więcej niż potrzebujemy), kolejne pięć bitów na zapisanie C_1 itd. Innymi słowy: ciąg $(C_0, C_1, \ldots, C_9)$ można zapisać jako liczbę w systemie pozycyjnym o podstawie 32: $B = C_0 + 32 \cdot C_1 + 32^2 \cdot C_2 + \ldots + 32^9 \cdot C_9$. Zaletą tego rozwiązania (przyjęcia konkretnie podstawy 32 zamiast na przykład 18, 19 lub 20) jest możliwość użycia masek bitowych do „pakowania" i „rozpakowywania" tej reprezentacji. Aby utworzyć tablicę stanów, które występują w zadaniu należy użyć słownika (`std::unordered_map` w C++ lub `dict` w Pythonie).

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025".