## Informatyczny kącik olimpijski (18) – *edycja specjalna*

W niniejszej, specjalnej edycji kącika zajmiemy się zadaniem „Ryby" z tej samej Międzynarodowej Olimpiady Informatycznej (rok 2008), co kilka odcinków temu. Zadanie to jest na tyle ciekawe, że poświęcimy mu wyjątkowo aż 3 strony kącika.

W sadzawce pływało kiedyś $N$ ryb o różnych rozmiarach, będących liczbami całkowitymi, i każda z nich miała w żołądku po jednym szlachetnym klejnocie spośród $M$ ich rodzajów (klejnoty tego samego rodzaju mogły się powtarzać). Z biegiem czasu ryby (prawdopodobnie) zaczęły się nawzajem zjadać – jedna ryba może zjeść drugą tylko wtedy, gdy jej rozmiar jest liczbą co najmniej dwa razy większą. Kiedy ryba zjada drugą, to jej rozmiar nie zmienia się, ale klejnot z żołądka ryby zjadanej pozostaje w żołądku ryby jedzącej (w dalszej części będziemy zatem mówić, że ryba zjada klejnot). Ile jest różnych zestawów klejnotów, jakie możemy znaleźć w żołądku dowolnej ryby z jeziora po dowolnym okresie żerowania (modulo zadana liczba $T$)?

Na początku chciałbym gorąco zachęcić Czytelnika do samodzielnego zmierzenia się z tym zadaniem. Okazało się ono jednym z dwóch najtrudniejszych na zawodach. Powodem jest to, że do rozwiązania można podchodzić na wiele całkiem oczywistych..., lecz zarazem błędnych sposobów. Poniższe rozumowanie będzie prowadziło, poprzez kolejne spostrzeżenia, od razu do poprawnego rozwiązania – przegląd „ślepych ścieżek" to materiał na znacznie dłuższy artykuł.

Ryba może w ostatecznym rozrachunku mieć klejnot z żołądka innej ryby tylko wtedy, gdy druga ryba jest co najmniej dwa razy mniejsza od pierwszej. Nie interesują nas zatem ani dokładny łańcuch pokarmowy, ani kolejność zjadania, a jedynie multizbiory (tj. zbiory z powtórzeniami) klejnotów „zjadalnych" przez poszczególne ryby.

Wybierzmy dowolną rybę. Multizbiór klejnotów zjadalnych przez nią (do tego multizbioru nie zaliczamy początkowego klejnotu z żołądka tej ryby) można opisać jako ciąg $k_1, k_2, \ldots, k_M$ – liczb klejnotów poszczególnych rodzajów, będących w zasięgu ryby. Wtedy liczba różnych zestawów klejnotów, które mogą znaleźć się w jej żołądku, opisuje się prostym wzorem na liczbę różnych podzbiorów zbioru z powtórzeniami: $(k_1 + 1) \cdot (k_2 + 1) \cdot \ldots \cdot (k_M + 1)$.

Sprawa komplikuje się, kiedy zastanawiamy się nad zbiorem możliwych zestawów klejnotów w żołądkach wszystkich ryb, ponieważ niektóre zestawy dla różnych ryb mogą się powtarzać.

Byłoby idealnie, gdybyśmy mieli jedną wielką rybę bez żadnego klejnotu w żołądku, która byłaby w stanie skonsumować każdą inną – wtedy wystarczyłoby zastosować dla niej powyższe rozumowanie (pytanie do zastanowienia: dlaczego?). Tak dobrze jednak nie jest. Możemy za to ograniczyć liczbę rozważanych ryb, a następnie wyeliminować powtórzenia już tylko między nimi.

Poczynimy w tym celu dwa spostrzeżenia:

- Weźmy dwie ryby $A$ i $B$, o tym samym początkowym klejnocie, i niech $A$ będzie nie mniejsza niż $B$. Wtedy zbiór możliwych zestawów klejnotów w żołądku $B$ jest podzbiorem zbioru możliwych zestawów dla $A$, ponieważ multizbiór klejnotów zjadalnych przez $B$ jest podzbiorem multizbioru klejnotów zjadalnych przez $A$.
- Weźmy pewien klejnot (rodzaju $i$), który pojawia się jako początkowy w żołądku co najmniej jednej ryby. Wtedy musimy rozważyć co najmniej jedną rybę o takim klejnocie początkowym, ponieważ tylko taka ryba może mieć w chwili połowu dokładnie jeden klejnot rodzaju $i$ w żołądku.

Wniosek: minimalny zbiór ryb, jakie musimy rozważyć, to zbiór największych ryb o poszczególnych rodzajach klejnotów w żołądkach. Wszystkie takie ryby nazwijmy rybami *istotnymi*.

Zacznijmy od największej ryby w ogóle i w znany nam już sposób obliczmy, ile jest różnych zestawów klejnotów, które może ona mieć w żołądku w chwili złowienia.

Odtąd będziemy rozważać ryby istotne, od największej do najmniejszej. Posuńmy się od razu kilka(naście) kroków do przodu, do rozważania pewnej ryby $X$. Oznaczmy rodzaj klejnotu w jej żołądku przez $x$. Rozważone dotychczas rodzaje klejnotów oznaczmy przez $a_1, a_2, \ldots, a_k$ (i odpowiadające im ryby przez $\{A_1, A_2, \ldots, A_k\} = A$), a pozostałe przez $b_1, b_2, \ldots, b_l$ (i podobnie, odpowiadające im ryby przez $\{B_1, B_2, \ldots, B_l\} = B$); $x$ do żadnej z tych grup nie należy. Dodatkowo (abstrahując od wyboru ryby $X$), niech $K(R, i)$ będzie liczbą klejnotów rodzaju $i$, które ryba $R$ jest w stanie zjeść (nie licząc klejnotu w jej żołądku).

Każdy dotychczas znaleziony zestaw klejnotów zawierał co najmniej jeden z klejnotów $a_1, a_2, \ldots, a_k$, a zatem każdy zestaw niezawierający żadnego z nich, który może wystąpić w żołądku $X$, należy policzyć jako nowy. Tych zestawów jest dokładnie

$$(K(X, b_1) + 1) \cdot (K(X, b_2) + 1) \cdot \ldots \cdot (K(X, b_l) + 1) \cdot (K(X, x) + 1).$$

Jakie inne zestawy jeszcze się nie pojawiły? Jeśli ryba $A_i$ jest w stanie zjeść co najmniej $K(X, x) + 1$ klejnotów rodzaju $x$, to na pewno w żołądku tej ryby może znaleźć się każdy zestaw klejnotów z co najmniej jednym klejnotem $a_i$, jaki może znaleźć się w żołądku $X$. W przeciwnym przypadku tamta ryba jest w stanie zjeść dokładnie $K(X, x)$ klejnotów $x$, skoro mniejsza od niej ryba $X$ tyle może zjeść. Wtedy oprócz już wspomnianej liczby nowych zestawów klejnotów bez żadnego z klejnotów rodzajów $a_1, \ldots, a_k$, należy jeszcze doliczyć te zestawy, które mają co najmniej jeden z klejnotów rodzaju $a_i$ oraz dokładnie $K(X, x) + 1$ klejnotów $x$ (to $+1$ pochodzi z żołądka samej ryby $X$).

Do dalszych rozważań, dla wygody, podzielmy ryby $A_i$ na dwa zbiory $C$ i $D$:

- $R \in C$, jeśli $R \in A$ oraz $K(R,x) = K(X,x)$,
- $R \in D$, jeśli $R \in A$ oraz $K(R,x) > K(X,x)$.

Przez $C_i$ ($D_i$) będziemy oznaczać ryby należące do $C$ ($D$), a przez $c_i$ ($d_i$) – klejnoty w ich żołądkach.

Jak już stwierdziliśmy, zbiór $D$ nas nie interesuje – w żołądku $X$ nie może znaleźć się żaden nowy zestaw, który zawierałby jakikolwiek z klejnotów $d_i$.

Co z kolei z liczbą nowych zestawów zawierających klejnoty $c_i$? Już wcześniej policzyliśmy wszystkie nowe zestawy, które nie zawierają żadnego z nich, a zatem pozostałe nowe zestawy muszą już zawierać co najmniej jeden z nich. Co więcej, liczba klejnotów $x$ w tych zestawach wynosi dokładnie $K(X,x)+1$. Tych zestawów, w takim razie, jest:

$$[(K(X,c_1)+1)\cdot(K(X,c_2)+1)\cdot \\ \cdot\ldots\cdot(K(X,c_{k_1})+1)-1]\cdot \\ \cdot(K(X,b_1)+1)\cdot(K(X,b_2)+1)\cdot\ldots\cdot(K(X,b_l)+1)$$

(przez $k_1$ oznaczyliśmy tutaj rozmiar zbioru $C$). Pytanie kontrolne: skąd się bierze $-1$ w powyższym wzorze?

Ten wzór jest dość podobny do poprzedniego wzoru na liczbę nowych zbiorów klejnotów bez klejnotów $a_i$. Możemy nawet oznaczyć sobie

$$G(X) = (K(X,b_1)+1)\cdot(K(X,b_2)+1)\cdot \\ \cdot\ldots\cdot(K(X,b_l)+1),$$
$$H(X) = (K(X,c_1)+1)\cdot(K(X,c_2)+1)\cdot \\ \cdot\ldots\cdot(K(X,c_{k_1})+1).$$

I tak otrzymujemy ostateczny wzór – w żołądku ryby $X$ może znaleźć się *dokładnie*

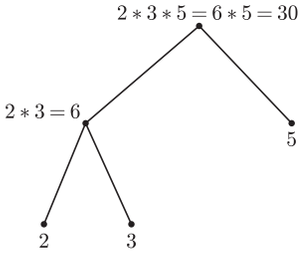
$$(H(X)-1+K(X,x)+1)\cdot G(X) = \\ = (H(X)+K(X,x))\cdot G(X)$$

nowych zestawów klejnotów. W takim razie, ostateczny czas działania naszego algorytmu będzie zależny od szybkości, z jaką możemy obliczać liczby $G(X)$ i $H(X)$ dla poszczególnych ryb $X$. Ale najpierw wspomnijmy jeszcze o jednym szczególe, a mianowicie, jak obliczać tablicę współczynników $K$.

Zauważmy, że podczas obliczeń dotyczących pojedynczej ryby $X$ korzystamy tylko z wartości $K(X,i)$, tj. druga współrzędna jest zmienna, ale pierwsza jest stała. Możemy w takim razie $K$ traktować tak naprawdę jak wektor. Jeśli na początku posortujemy ryby pod względem wielkości, i zaczniemy od pierwszej, największej, to w pierwszym kroku będziemy mogli obliczyć wektor $K$, przeglądając po prostu wszystkie inne ryby i szukając tych, które są co najmniej dwa razy mniejsze. Przy okazji możemy zapamiętać także taką największą wartość $z_X$, że $z_X$-ta ryba jeszcze jest zjadalna przez aktualną rybę, ale $(z_X+1)$-sza już nie. Kiedy teraz będziemy rozważać kolejną rybę $X'$, nieco mniejszą, to wskaźnik $z_{X'}$ może być mniejszy od $z_X$. Bezpośrednio ze zmniejszaniem wskaźnika $z$ możemy zmniejszać odpowiednie pozycje wektora $K(X)$ tak, żeby stał się aktualny dla kolejnej istotnej ryby. I chociaż każdy krok może wywołać przesunięcie $z_X$ nawet o blisko $N$ pozycji, to te kroki amortyzują się – czyli w pełnym wykonaniu algorytmu suma pozycji, o które przesunie się $z_X$, będzie ograniczona przez $N$.

Przejdźmy teraz do wyznaczania wartości $G(X)$. Obliczanie ich za każdym razem od nowa byłoby zbyt czasochłonne, a z drugiej strony wyznaczanie ich na podstawie poprzednich wartości mogłoby wymagać dzielenia, które przy obliczeniach modulo nie jest proste. Z tego względu do pomocy przy organizacji obliczeń zastosujemy drzewo przedziałowe.

Wyobraźmy sobie drzewo binarne, którego liście odpowiadają poszczególnym rodzajom klejnotów. W liściu przypisanym klejnotowi $i$ przechowywana jest wartość $K(X,i)+1$ dla aktualnego $X$. Z kolei w każdym węźle wewnętrznym znajduje się iloczyn wartości przechowywanych we wszystkich liściach pod nim – jest to równoznaczne z tym, że znajduje się w nim iloczyn wartości z jego lewego i prawego syna (patrz rysunek).




Jeśli następuje zmiana wartości w jednym z liści, to węzły wymagające poprawy to tylko te, które leżą na ścieżce od tego liścia do korzenia. Jest ich niewiele, jeśli drzewo jest zrównoważone. Ale to jest łatwo zapewnić – ponieważ nie będziemy zmieniać jego struktury, więc wystarczy zbudować je w sposób zrównoważony, np. tak:

**UTWÓRZ_DRZEWO**(początek_przedziału, koniec_przedziału)
  **jeśli** początek_przedziału = koniec_przedziału **to**
    korzeń.lewy := korzeń.prawy := **nil**;
    korzeń.ojciec := **nil**;
    korzeń.wartość := $K(X,$ początek_przedziału$)+1$;
  **a w przeciwnym przypadku**
    środek := $\lfloor$(początek_przedziału + koniec_przedziału)$/2\rfloor$;
    korzeń.lewy := **UTWÓRZ_DRZEWO**(początek_przedziału, środek);
    korzeń.prawy := **UTWÓRZ_DRZEWO**(środek + 1, koniec_przedziału);
    korzeń.lewy.ojciec := korzeń.prawy.ojciec := korzeń;
    korzeń.ojciec := **nil**;
    korzeń.wartość := (korzeń.lewy.wartość · korzeń.prawy.wartość) mod $T$;

Z kolei aktualizację pojedynczego liścia wykonujemy następująco:

**ZAKTUALIZUJ**$(w, \text{wartość})$
  $w$.wartość := wartość;
  $w := w$.ojciec;
  **dopóki** $w \neq$ **nil wykonuj**
    $w$.wartość := ($w$.lewy.wartość $\cdot$ $w$.prawy.wartość) mod $T$;
    $w := w$.ojciec;

Taka aktualizacja wymaga czasu rzędu $O(\log M)$; zauważmy, że zostanie ona wykonana w całym algorytmie co najwyżej $N$ razy.

Aby wyznaczyć szukany iloczyn, wystarczy zajrzeć do wartości znajdującej się w korzeniu drzewa – jest ona równa $\prod_i (K(X,i)+1)$. My jednak nie chcemy znać iloczynu wszystkich tych wartości, a jedynie tych, których istotne ryby należą do zbioru $B$. Ale do tego zbioru na początku należą wszystkie rodzaje klejnotów, a następnie są z niego, kolejno, bezpowrotnie usuwane. Konkretnie, wyrzucane są po kolei te klejnoty, które stają się klejnotem $x$. W takim razie wystarczy liczność każdego takiego klejnotu ustalać w drzewie na 1 (a później, oczywiście, już dalej nie zmniejszać).

A co z $H(X)$? Tutaj sprawa się trochę bardziej komplikuje, ale wciąż nie jest to nic, z czym byśmy sobie nie poradzili. Liczba $K(X,x)$ jest mniejsza od $K(A_i,x)$ wtedy, gdy wśród ryb na pozycjach $z_X+1,\ldots,z_{A_i}$ jest chociaż jedna z klejnotem rodzaju $x$ w żołądku. Niech w takim razie $Q_X$ będzie *najmniejszą rybą z klejnotem $x$ w żołądku, której $X$ nie jest w stanie zjeść*. W takim razie, do zbioru $C$ będą należeć wszystkie te ryby $A_i \in A$, które nie są w stanie zjeść $Q_X$. Jest tak dlatego, że

$$A_i \in C \Leftrightarrow K(X,x) = K(A_i,x).$$

Ryba $A_i$ jest nie mniejsza od $X$, a zatem może zjeść wszystkie ryby z klejnotem $x$, które $X$ jest w stanie zjeść. Ale skoro nie może zjeść najmniejszej z tych, których $X$ nie może zjeść, to nie może też zjeść żadnej innej. Za to jeśli $A_i$ może zjeść $Q_X$, to $K(A_i,x)$ jest na pewno większe niż $K(X,x)$, choćby ze względu na rybę $Q_X$ (jeśli $Q_X$ nie istnieje, to dla tej ryby $X$ mamy $C=\emptyset$).

Powiedzieliśmy już, że wszystkie ryby mamy zamiar uporządkować według wielkości. W takim razie, warunek „$A_i$ może zjeść $Q_X$” jest równoznaczny z „$Q_X \leqslant z_{A_i}$”. Wartości $Q_X$ dla poszczególnych istotnych ryb najprościej jest obliczyć na samym początku rozwiązania. Jak dokładnie – to pozostawiam Czytelnikowi jako małą łamigłówkę.

Możemy już w tym momencie sobie powiedzieć, jak efektywnie liczyć $H(X)$. Będziemy potrzebować struktury danych, za pomocą której można wykonywać następujące operacje:

- umieść w strukturze (istotną) rybę $X$ o zadanych wartościach $z_X$ oraz $w_X = K(X,x)+1$,
- zmniejsz o 1 parametr $w_Y$ dla ryby $Y$ odpowiadającej zadanemu klejnotowi $y$,
- podaj iloczyn aktualnych wartości $w_Y$ dla ryb $Y$ spełniających $z_Y < Q_X$ (tzn. $Y \in C$).

Ale to jest ta sama funkcjonalność, której potrzebowaliśmy w poprzednim przypadku! No, prawie ta sama. W każdym razie możemy do tego wykorzystać analogiczne drzewo przedziałowe. Tym razem jednak ryby istotne, odpowiadające klejnotom z liści, będą uporządkowane według niemalejących wartości $z_Y$. Wynika to z tej małej różnicy między podproblemami „$G$” i „$H$”: tym razem nie będzie nas interesował iloczyn wszystkich wartości, a jedynie tych o $z_Y$ mniejszym od $Q_X$ podanego w konkretnym zapytaniu. W tym celu uzupełnijmy węzeł w drzewie o wartość max_z – odpowiadającą maksymalnemu $z_Y$ w jego poddrzewie. To, jak ostatecznie takie zapytanie zrealizować (wykonanie zapytania zaczynamy od $w =$ korzeń), przedstawia program kończący tekst.

Łatwo zauważyć, że koszt czasowy pojedynczego zapytania o iloczyn zamknie się w $O(\log M)$. Pobieżna analiza całego algorytmu daje nam złożoność obliczeniową – ze względu na nierówność $M \leqslant N$, wynosi ona $O(N \log N)$.

*Filip WOLSKI*

**PODAJ_ILOCZYN**$(w, Q)$
  **jeśli** $w$.max_z $< Q$ **to**
    **return** $w$.wartość;
  **a w przeciwnym przypadku**
    **jeśli** $w$.lewy.max_z $< Q$ **to**
      **return** (**PODAJ_ILOCZYN**$(w.\text{prawy}, Q) \cdot w$.lewy.wartość) mod $T$;
    **a w przeciwnym przypadku**
      **return PODAJ_ILOCZYN**$(w.\text{lewy}, Q)$;

Autor zdaje sobie sprawę, że to zadanie nie jest zadaniem łatwym – wręcz przeciwnie, okazuje się bardzo wymagające. W razie problemów ze zrozumieniem poszczególnych części rozwiązania gorąco poleca próbę jego własnoręcznej implementacji. Dopiero kiedy jest się w stanie samemu zaimplementować rozwiązanie, można być pewnym, że się je całkowicie rozumie. A to zadanie jest tego warte, jako, zdaniem autora, jedno z najciekawszych i najbardziej satysfakcjonujących. Dodatkowo, przygotowany przez autora program rozwiązujący niniejsze zadanie można pobrać ze strony internetowej *Delty*.