Omówienie zadań z Turnieju 38 (dla początkujących)

Jakub Nowak, Mateusz Feczan

1 Kaucja

W tym zadaniu mamy na wejściu dany ciąg N symboli (oznaczających typy opakowań) ze zbioru $\{\texttt{P}, \texttt{M}, \texttt{S}\}$, przy czym każdemu symbolowi odpowiada pewna wartość: `P` – 0,50 zł, `M` – 0,50 zł, `S` – 1,00 zł. Zadaniem uczestnika jest wypisanie sumarycznej wartości opakowań z wejścia z dokładnie dwoma miejscami po kropce dziesiętnej.

Jest to proste zadanie, którego rozwiązanie sprowadza się do napisania jednej pętli `for` wczytującej po kolei każdy symbol i aktualizującej zmienną pamiętającą sumaryczną wartość dotąd wczytanych symboli. Zmienną zliczającą inicjalizujemy wartością 0. W każdej iteracji pętli należy wczytać jeden symbol reprezentujący typ opakowania. Następnie można trzema poleceniami `if` sprawdzić, czy opakowanie jest typu `P`, `M` czy `S`, oraz odpowiednio zwiększyć zmienną zliczającą wartość.

Możemy to zaimplementować na przykład na dwa sposoby:

1. Można zliczać aktualną wartość wczytanych symboli w zmiennej typu `float` lub `double`, a następnie wypisać ją z określoną precyzją, np. przy pomocy polecenia:
   `cout << fixed << setprecision(2) << total;`
   w C++, gdzie `total` to zmienna typu `float` lub `double` zawierająca sumaryczną wartość opakowań z wejścia. W tym zadaniu dodajemy tylko 0,5 i 1, a suma nie przekracza 10^6, więc w standardowym typie `float` wszystkie możliwe sumy są reprezentowane dokładnie.

2. Można też przechowywać aktualną sumę w jednostkach po 50 groszy: za `P` i `M` dodajemy 1, a za `S` dodajemy 2. W tym przypadku wystarczy, by zmienna zliczająca była typu `int`.

   Żeby otrzymać oczekiwany format wyjścia, przy jej wypisywaniu można wypisać wynik dzielenia całkowitego tej liczby przez 2, a następnie `.00` lub `.50` zależnie od tego, czy liczba jest parzysta czy nieparzysta (co w C++ można łatwo sprawdzić poleceniem `if` i operatorem modulo `%`).

Złożoność obliczeniowa takiego rozwiązania to $\mathcal{O}(N)$. Złożoność pamięciowa wynosi $\mathcal{O}(1)$.

Autor zadania: Jakub Nowak

2 Kółko i Krzyżyk

W tym zadaniu na wejściu otrzymujemy opis planszy do gry w kółko i krzyżyk o stałych wymiarach 3×3, złożonej ze znaków ze zbioru $\{\texttt{X}, \texttt{O}, \texttt{.}\}$. Zadaniem uczestnika jest ustalenie, który z graczy (`X` czy `O`) ułożył wygrywającą linię, czy też żadnemu z nich się to nie udało (co oznacza remis).

Ponieważ rozmiar planszy jest stały i niewielki, zadanie sprowadza się do bezpośredniego sprawdzenia wszystkich możliwych linii wygrywających. Na planszy 3×3 istnieje dokładnie 8 takich linii: 3 wiersze, 3 kolumny oraz 2 przekątne. Puste pola (`.`) mogą występować również w zakończonej partii, jeśli któryś gracz wygrał przed zapełnieniem planszy. Nie zmienia to sposobu sprawdzania zwycięstwa – interesuje nas wyłącznie to, czy na którejkolwiek z ośmiu linii znajdują się trzy identyczne symbole któregoś z graczy.

Wygodnym sposobem implementacji jest wczytanie planszy jako tablicy trzech napisów typu `string` (lub dwuwymiarowej tablicy znaków `char`) i napisanie pomocniczej funkcji przyjmującej jako argument znak gracza (`'X'` lub `'O'`). Funkcja ta sprawdza za pomocą krótkiej pętli `for`, czy w

którymś z trzech wierszy lub trzech kolumn wszystkie trzy znaki są równe szukanemu symbolowi, a następnie dwoma instrukcjami `if` weryfikuje obie przekątne.

Mając taką funkcję, w programie głównym wystarczy sprawdzić, czy wywołanie jej dla znaku `'X'` zwraca prawdę (wtedy wypisujemy `X`), w przeciwnym razie sprawdzić, czy zwraca prawdę dla znaku `'O'` (wtedy wypisujemy `O`), a jeśli żaden z tych warunków nie zachodzi – wypisać `R`.

Z uwagi na stały rozmiar wejścia, zarówno złożoność obliczeniowa, jak i pamięciowa tego rozwiązania to $\mathcal{O}(1)$.

Autor zadania: Mateusz Feczan

3 Trójkąty

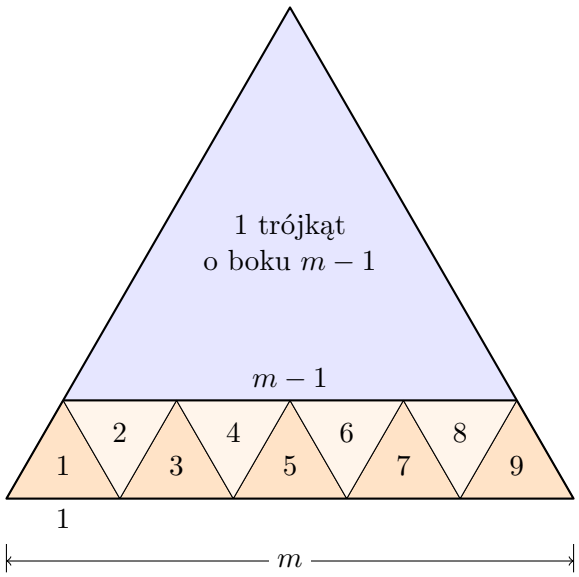
To zadanie polegało na określeniu, czy trójkąt równoboczny da się podzielić na dokładnie N rozłącznych (poza bokami i wierzchołkami) trójkątów równobocznych (niekoniecznie tego samego rozmiaru!).

Okazuje się, że jest to możliwe dla każdego dodatniego całkowitego N poza $N = 2, 3, 5$. Dowód tego faktu znajduje się poniżej. Znając ten fakt, można napisać proste rozwiązanie składające się z 3 `if`-ów sprawdzające, czy N jest jedną z powyższych liczb. Złożoność takiego rozwiązania (zarówno obliczeniowa, jak i pamięciowa) to $\mathcal{O}(1)$.

W dalszej części omówienia wykażemy, dlaczego odpowiedź to `NIE` tylko dla $N \in \{2, 3, 5\}$.

Po pierwsze, dla $N = 1$ oraz $N = 4$ da się dokonać takiego podziału: dla $N = 1$ pozostawiamy cały trójkąt, a dla $N = 4$ łączymy środki jego boków. Stąd, mając dowolny podział trójkąta równobocznego na K trójkątów równobocznych, jesteśmy w stanie podzielić dowolny z tych trójkątów na 4 mniejsze trójkąty, otrzymując podział na $K + 3$ trójkąty.

Po drugie, dla dowolnej liczby parzystej $K \geq 4$ da się dokonać takiego podziału na K trójkątów. Dla $K = 2m$, gdzie $m \geq 2$, rozważmy trójkąt o boku m: przy jednym wierzchołku pozostawiamy trójkąt o boku $m-1$, a pozostały pas dzielimy na $2m-1$ trójkątów równobocznych o boku 1. Razem otrzymujemy $2m = K$ trójkątów. Konstrukcję ilustruje poniższy rysunek. Łącząc to z obserwacją, że istnienie podziału dla K implikuje istnienie podziału dla $K + 3$, dochodzimy do wniosku, że podział jest możliwy dla wszystkich $K \geq 6$ (również nieparzystych).



Konstrukcja dla $K = 2m$ na przykładzie $m = 5$. Pas składa się z m trójkątów skierowanych w górę i $m - 1$ skierowanych w dół, każdego o boku 1. Wraz z trójkątem o boku $m - 1$ otrzymujemy $1 + m + (m - 1) = 2m$ trójkątów (na rysunku 10).

Pozostaje rozważyć przypadki $N \in \{2, 3, 5\}$. Pokażemy, że dla tych N podział jest niemożliwy. Oznaczmy wierzchołki oryginalnego trójkąta przez A, B i C, a długość jego boku przez s. Przy każdym z tych wierzchołków musi znajdować się dokładnie jeden trójkąt podziału: kąt dużego trójkąta ma 60°, tyle samo co kąt każdego z mniejszych trójkątów. Nazwijmy te trójkąty *wierzchołkowymi*.

Jeśli jeden z nich ma bok długości s, wypełnia cały trójkąt ABC i $N = 1$. W przeciwnym razie trójkąty przy A, B i C są różne, więc $N \geq 3$. W szczególności wyklucza to $N = 2$.

Niech długości boków tych trzech trójkątów wynoszą odpowiednio a, b i c. Ponieważ ich wnętrza są rozłączne, mamy

$$a + b \leq s, \qquad b + c \leq s, \qquad c + a \leq s.$$

Dwa trójkąty wierzchołkowe stykają się wtedy i tylko wtedy, gdy odpowiednia nierówność jest równością. Na przykład trójkąty przy A i B stykają się w jednym punkcie boku AB dokładnie wtedy, gdy $a + b = s$.

Po usunięciu trójkątów wierzchołkowych pozostaje wewnątrz niepusty obszar o dodatnim polu. Jego brzeg tworzą trzy boki trójkątów wierzchołkowych oraz te fragmenty boków ABC, na których trójkąty wierzchołkowe się nie zetknęły. Jest to wypukły sześciokąt, pięciokąt, czworokąt albo trójkąt, zależnie od tego, czy stykających się par jest odpowiednio 0, 1, 2 czy 3. Zatem same trzy trójkąty wierzchołkowe nigdy nie wystarczą do pokrycia ABC. Wyklucza to również $N = 3$.

Aby wykluczyć $N = 5$, wystarczy wykazać, że pozostałego obszaru nie da się pokryć dokładnie dwoma trójkątami równobocznymi. Przyda się następująca obserwacja: przy każdym wierzchołku tego obszaru o kącie 120° muszą spotykać się dokładnie dwa trójkąty podziału, każdy swoim kątem 60°. Taki wierzchołek nie może leżeć we wnętrzu boku trójkąta podziału, ponieważ wymagałoby to kąta co najmniej 180°. Rozważmy teraz wszystkie cztery przypadki.

1. Żadna para trójkątów wierzchołkowych się nie styka. Pozostały obszar jest sześciokątem o sześciu kątach 120°. Przy każdym z jego sześciu wierzchołków potrzebne są dwa wierzchołki trójkątów podziału, czyli łącznie 12 takich wystąpień. Każdy trójkąt ma tylko 3 wierzchołki, więc potrzeba co najmniej 4 trójkątów. W szczególności dwa nie wystarczą.

2. Styka się dokładnie jedna para trójkątów wierzchołkowych. Pozostały obszar jest pięciokątem o czterech kątach 120° i jednym kącie 60°. Same cztery kąty 120° wymagają łącznie 8 wierzchołków trójkątów podziału. Dwa trójkąty mają ich tylko 6, więc do pokrycia tego obszaru potrzeba co najmniej 3 trójkątów.

3. Stykają się dokładnie dwie pary trójkątów wierzchołkowych. Bez straty ogólności trójkąt przy A styka się z trójkątami przy B i C, ale te dwa ostatnie się nie stykają. Pozostały obszar jest czworokątem. Jego dwa kąty 120° znajdują się w punktach D i E na boku BC, będących końcami niepokrytego fragmentu tego boku. Pozostałe dwa kąty mają po 60°.

   Gdyby czworokąt był pokryty tylko dwoma trójkątami równobocznymi, oba musiałyby mieć wierzchołek w D i oba musiałyby mieć wierzchołek w E, aby wypełnić kąty 120°. Odcinek DE byłby więc bokiem obu trójkątów. Oba leżałyby po tej samej stronie prostej BC, wewnątrz ABC. Tymczasem po ustalonej stronie odcinka istnieje tylko jeden trójkąt równoboczny mający ten odcinek za bok. Trójkąty musiałyby się zatem pokrywać, co jest sprzeczne z warunkami podziału. Potrzeba więc co najmniej 3 trójkątów.

4. Stykają się wszystkie trzy pary trójkątów wierzchołkowych. Wtedy $a + b = b + c = c + a = s$, więc $a = b = c = s/2$, a pozostały obszar jest trójkątem równobocznym o boku $s/2$. Wcześniej wykazaliśmy już, że trójkąta równobocznego nie można podzielić na 2 ani na 3 trójkąty równoboczne. Pozostały obszar jest więc jednym trójkątem albo składa się z co najmniej 4 trójkątów. Łącznie otrzymujemy $N = 4$ albo $N \geq 7$.

We wszystkich przypadkach, jeśli $N \neq 1$ i $N \neq 4$, to $N \geq 6$. Wyklucza to $N = 5$ i kończy dowód: podział jest możliwy dokładnie dla dodatnich liczb całkowitych innych niż 2, 3 i 5.

Autor zadania: Jakub Nowak

4 Kolorowanie alfabetu

Niniejsze zadanie miało dwa podzadania. Zaczniemy od wyjaśnienia oczekiwanego rozwiązania pierwszego podzadania. Mając dane słowo składające się z liter z 12-elementowego zbioru i ustalone kolorowanie liter z tego zbioru, jesteśmy w stanie w czasie $\mathcal{O}(N)$ wyznaczyć *kolorowość* słowa. Wystarczy zacząć od jednego bloku, a następnie przejść po kolejnych parach sąsiednich liter i zwiększać wynik o 1, gdy ich kolory są różne.

Możliwych kolorowań jest $3^{12} = 531\,441$, ponieważ każdej z 12 liter przypisujemy jeden z 3 kolorów. Nie musimy wykorzystać wszystkich kolorów, więc uwzględniamy również kolorowania jednym lub dwoma kolorami. Możemy zatem sprawdzić wszystkie możliwe kolorowania, dla każdego wyznaczyć kolorowość słowa i wypisać największą znalezioną wartość. Takie rozwiązanie działa w czasie $\mathcal{O}(3^{12} \cdot N)$. Dla $N = 100$ iloczyn $3^{12} \cdot N$ wynosi około $5{,}3 \cdot 10^7$. Napisanie rozwiązania o tej złożoności pozwalało rozwiązać pierwsze podzadanie i zdobyć połowę punktów.

Pełne rozwiązanie również polega na rozważeniu wszystkich możliwych kolorowań, lecz szybciej wyznacza kolorowość. Wymaga ono obserwacji, że kolorowość słowa jest równa liczbie miejsc w słowie, w których sąsiadują litery różnych kolorów, zwiększonej o 1. Stąd możemy raz na początku programu zliczyć dla każdej uporządkowanej pary liter (a, b), ile razy litera a występuje bezpośrednio przed literą b. Oznaczmy tę liczbę przez $c_{a,b}$.

Następnie dla każdego kolorowania zaczynamy od wyniku 1, iterujemy po zbiorze par liter (zamiast po całym słowie) i dodajemy $c_{a,b}$ dla tych par, w których litery mają różne kolory. Każda para sąsiednich pozycji słowa została uwzględniona dokładnie raz, więc otrzymujemy właściwą kolorowość. Złożoność czasowa takiego rozwiązania to

$$\mathcal{O}\left(N + 3^{12} \cdot 12^2\right).$$

Dla $N = 10^6$ wyrażenie $N + 3^{12} \cdot 12^2$ wynosi około $7{,}8 \cdot 10^7$. Takie rozwiązanie otrzymywało maksymalną liczbę punktów. Tablica liczników zajmuje $\mathcal{O}(12^2)$ pamięci, a kolorowanie $\mathcal{O}(12)$. Słowo można przetwarzać podczas wczytywania, pamiętając tylko poprzednią literę; wówczas dla stałego alfabetu złożoność pamięciowa wynosi $\mathcal{O}(1)$. Jeśli przechowujemy cały napis, wynosi ona $\mathcal{O}(N)$.

Autor zadania: Jakub Nowak

5 Maratończyk

Zadanie polegało na wyznaczeniu minimalnej początkowej pojemności energii X, która pozwoli biegaczowi pokonać N kolejnych odcinków trasy o kosztach $A_1, A_2, \ldots, A_N$, przy możliwości co najwyżej K-krotnego skorzystania z taksówki (która pozwala ominąć dany odcinek i całkowicie odnawia energię biegacza do poziomu X). Zadanie składało się z 4 podzadań.

W podzadaniu 1 zachodzi $K = 1$. Biegacz może użyć taksówki na co najwyżej jednym odcinku. Jeśli skorzysta z niej na odcinku o indeksie i, to musi o własnych siłach przebiec wszystkie odcinki przed i oraz wszystkie odcinki po i. Wymagana początkowa energia wynosi wówczas $X_i = \max\left(\sum_{j=1}^{i-1} A_j, \sum_{j=i+1}^{N} A_j\right)$. Korzystając z tablicy sum prefiksowych, możemy w czasie $\mathcal{O}(N)$ przetestować każdy możliwy odcinek $i \in \{1, 2, \ldots, N\}$ i wybrać najmniejszy możliwy wynik.

W podzadaniu 2 mamy $N \leq 20$, co pozwala na wykładnicze sprawdzenie wszystkich podzbiorów odcinków, na których biegacz bierze taksówkę, lub zastosowanie prostego programowania dynamicznego. Z kolei w podzadaniu 3 zachodzi $N \leq 1000$, co umożliwia zastosowanie programowania dynamicznego w czasie $\mathcal{O}(N^2)$, gdzie stanem jest minimalna energia potrzebna na pokonanie prefiksu z użyciem zadanej liczby taksówek.

Aby rozwiązać zadanie w pełnej ogólności (podzadanie 4), należy zauważyć monotoniczność wyniku: jeśli biegacz jest w stanie ukończyć maraton z początkową energią X, to tym bardziej ukończy go z energią $X + 1$. Pozwala to zastosować wyszukiwanie binarne po wyniku.

Dla ustalonej w wyszukiwaniu binarnym wartości X możemy zachłannie w czasie $\mathcal{O}(N)$ sprawdzić, czy jest ona wystarczająca. Symulujemy bieg od pierwszego do ostatniego odcinka, pamiętając

aktualny zapas energii. Jeśli obecna energia pozwala na pokonanie kolejnego odcinka ($energia \geq A_i$), odejmujemy A_i od zapasu energii i biegniemy dalej. W przeciwnym razie biegacz musi wziąć na tym odcinku taksówkę – zwiększamy licznik użytych taksówek o 1 i ustawiamy aktualną energię z powrotem na X. Jeśli w którymś momencie liczba użytych taksówek przekroczy K, badana wartość X jest za mała.

Wyszukiwanie binarne prowadzimy na przedziale od 0 do $\sum_{i=1}^{N} A_i$ (ponieważ suma może osiągnąć 10^{14}, należy pamiętać o użyciu 64-bitowego typu całkowitego np. `long long` w C++). Złożoność obliczeniowa rozwiązania wzorcowego to $\mathcal{O}\left(N \log\left(\sum_{i=1}^{N} A_i\right)\right)$, natomiast złożoność pamięciowa to $\mathcal{O}(N)$.

Autor zadania: Mateusz Feczan

6 Zderzające Kulki

W zadaniu należało rozstrzygnąć, czy ciąg N kulek o początkowych masach $M_1, M_2, \ldots, M_N$ da się posortować niemalejąco za pomocą zamian sąsiednich elementów, wiedząc, że kulka przesuwana w prawo traci 1 jednostkę masy, a przesuwana w lewo zyskuje 1 jednostkę masy.

Dla małych wartości N (podzadanie 1, gdzie $N \leq 10$) można było próbować sprawdzić każdą możliwą kombinację ułożenia kulek, a w podzadaniu 2 ($N \leq 1000$) zastosować zmodyfikowane sortowanie bąbelkowe działające w czasie $\mathcal{O}(N^2)$. Pełne rozwiązanie wymaga jednak odkrycia kluczowego niezmiennika.

Zauważmy, co dzieje się z sumą masy kulki i jej aktualnej pozycji w ciągu: $S_i = M_i + i$. Kiedy zamieniamy miejscami dwie sąsiednie kulki na pozycjach i oraz $i+1$:

- kulka z pozycji i trafia na pozycję $i+1$ (jej indeks rośnie o 1), ale jej masa maleje o 1,
- kulka z pozycji $i+1$ trafia na pozycję i (jej indeks maleje o 1), ale jej masa rośnie o 1.

W obu przypadkach wartość wyrażenia $M_i + i$ dla każdej kulki pozostaje stała niezależnie od liczby i kolejności wykonywanych ruchów!

Załóżmy, że po pewnej serii ruchów kulki ustawiły się w pewnej kolejności, a ich niezmienniki na kolejnych pozycjach $1, 2, \ldots, N$ to odpowiednio $S'_1, S'_2, \ldots, S'_N$. Wówczas masa kulki stojącej docelowo na pozycji j wynosi $M'_j = S'_j - j$. Chcemy, aby dla każdego $1 \leq j < N$ zachodziła nierówność:

$$M'_j \leq M'_{j+1} \iff S'_j - j \leq S'_{j+1} - (j+1) \iff S'_j + 1 \leq S'_{j+1} \iff S'_j < S'_{j+1}.$$

Ponieważ niezmienniki są liczbami całkowitymi, nierówność $S'_j < S'_{j+1}$ dla każdego j możemy spełnić wtedy i tylko wtedy, gdy posortujemy kulki rosnąco względem ich niezmienników oraz wszystkie wartości $S_i = M_i + i$ będą parami różne. Jeśli dwie kulki mają identyczną wartość $M_i + i$, to po ustawieniu ich obok siebie zawsze ta stojąca bardziej z lewej będzie miała o 1 większą masę niż ta z prawej, więc ciągu nigdy nie uda się posortować.

Wzorcowy algorytm polega zatem na wczytaniu ciągu, wyznaczeniu dla każdego elementu wartości $M_i + i$, posortowaniu otrzymanej tablicy (np. funkcją `std::sort`) i sprawdzeniu w jednej pętli `for`, czy żadne dwa sąsiednie elementy po posortowaniu nie są sobie równe. Jeśli znajdziemy duplikat, wypisujemy `NIE`, a w przeciwnym wypadku `TAK`.

Złożoność obliczeniowa takiego rozwiązania to $\mathcal{O}(N \log N)$ ze względu na sortowanie, a pamięciowa to $\mathcal{O}(N)$.

Autor zadania: Mateusz Feczan

7 Liczby chropowate

To było najtrudniejsze zadanie konkursu. Polegało na policzeniu, ile liczb z zadanego przedziału $[A, B]$ jest zarówno pierwszych, jak i postaci $6p^k + 1$ (dla p pierwszego i całkowitego $k \geq 1$) albo nie

ma żadnej z tych dwóch własności. Zadanie ma 3 podzadania, a omówienie przejdzie przez nie po kolei.

W analizie oznaczymy liczbę elementów przedziału przez $L = B - A + 1$, a przez $\pi(t)$ liczbę liczb pierwszych nie większych niż t. Przyjmujemy, że operacje arytmetyczne na liczbach mieszczących się w typie całkowitym mają stały koszt. Wzory z logarytmami opisują zachowanie dla rosnących parametrów; małe przypadki obsługujemy bezpośrednio.

Podzadanie 1: jedna liczba

W podzadaniu 1 zachodzi $A = B$. Innymi słowy, sprawdzamy tylko jedną liczbę. W tym wypadku możemy po prostu sprawdzić, czy dana liczba jest pierwsza oraz czy jest postaci $6p^k+1$. Sprawdzenia pierwszości możemy dokonać w czasie $\mathcal{O}(\sqrt{A})$, iterując po liczbach całkowitych od 2 do $\lfloor\sqrt{A}\rfloor$ i sprawdzając, czy aktualna liczba jest dzielnikiem A (`A%i == 0`). Jeśli $A \geq 2$ i nie znajdziemy żadnego dzielnika w tym zakresie, oznacza to, że A jest pierwsza. Liczby mniejsze niż 2 nie są pierwsze.

Podobnie możemy sprawdzić, czy A jest postaci $6p^k + 1$. Najpierw sprawdzamy, czy $A - 1$ jest wielokrotnością 6. Jest to warunek konieczny. Jeśli tak, wystarczy sprawdzić, czy $A' = \frac{A-1}{6}$ jest postaci p^k. Dla $A' < 2$ odpowiedź jest od razu przecząca.

Zauważmy, że jeśli $A' = p^k$, to $k \leq \lfloor \log_2 A' \rfloor \leq 50$, ponieważ $p \geq 2$ oraz $A \leq 10^{16}$. Stąd możemy dla każdego k w tym zakresie sprawdzić, czy istnieje pierwsze p spełniające $p^k = A'$. Sprawdzenie sprowadza się do wyznaczenia pierwiastka k-tego stopnia z A' oraz określenia, czy jest on całkowity i pierwszy. Można to zrobić wyszukiwaniem binarnym: szukamy największej liczby całkowitej p, dla której $p^k \leq A'$. Dopiero gdy stwierdzimy, że $p^k = A'$, sprawdzamy pierwszość p metodą opisaną wcześniej. Dla $k = 1$ kandydatem jest po prostu A'.

Porównania potęg wykonujemy w arytmetyce całkowitej, aby uniknąć błędów zaokrągleń. Należy też zapobiec przepełnieniu: przed pomnożeniem dodatniej wartości v przez p sprawdzamy, czy $v > \lfloor A'/p \rfloor$. Jeśli tak, potęga przekroczyłaby A' i możemy zakończyć porównanie.

Koszt wszystkich testów pierwszości podstaw potęg jest ograniczony przez

$$\mathcal{O}\left(\sum_{k=1}^{\lfloor \log_2 A' \rfloor} (A')^{1/(2k)}\right) = \mathcal{O}\left(\sqrt{A'} + (A')^{1/4} \log A'\right) = \mathcal{O}(\sqrt{A'}).$$

Pierwszy składnik odpowiada $k = 1$, a każdy z pozostałych jest nie większy niż $(A')^{1/4}$. Samo wyszukiwanie pierwiastków również trzeba uwzględnić. Nawet jeśli dla każdego k szukamy w całym przedziale $[1, A']$ i porównujemy potęgę przez najwyżej k mnożeń, łączny koszt wynosi

$$\mathcal{O}\left(\sum_{k=1}^{\lfloor \log_2 A' \rfloor} k \log A'\right) = \mathcal{O}(\log^3 A').$$

Jest on asymptotycznie mniejszy niż $\sqrt{A'}$. Razem z testem pierwszości samego A otrzymujemy więc czas $\mathcal{O}(\sqrt{A})$ i pamięć $\mathcal{O}(1)$. Wynik wynosi 1, jeśli oba testy dały tę samą odpowiedź, i 0 w przeciwnym przypadku.

Podzadanie 2: krótki przedział

Możemy uruchomić procedurę z podzadania 1 dla każdej liczby w przedziale $[A, B]$, otrzymując złożoność czasową $\mathcal{O}(L\sqrt{B})$ i pamięciową $\mathcal{O}(1)$. W tym podzadaniu $L \leq 100$, lecz dla dużego B nadal wykonujemy bardzo wiele dzieleń.

Można przyspieszyć testy pierwszości, wyznaczając raz na początku zbiór liczb pierwszych nie większych niż $R = \lfloor\sqrt{B}\rfloor \leq 10^8$, np. sitem Eratostenesa. Następnie zamiast sprawdzać wszystkie możliwe dzielniki liczby x, sprawdzamy tylko dzielniki pierwsze nie większe niż $\sqrt{x}$. W najgorszym przypadku wykonujemy $\mathcal{O}(\pi(\sqrt{x}))$ dzieleń. Ponieważ $\pi(t) = \Theta(t/\log t)$, daje to dla dużych liczb przyspieszenie samych testów w najgorszym przypadku o czynnik rzędu $\log B$. Nie jest to jednak gwarantowane przyspieszenie całego programu, ponieważ trzeba jeszcze przygotować sito.

Koszt sprawdzania wszystkich podstaw potęg dla jednej liczby również mieści się w $\mathcal{O}(\pi(R))$: największy składnik pochodzi od $k = 1$, a pozostałe testy kosztują łącznie najwyżej $\mathcal{O}(B^{1/4} \log B)$. Koszt wyznaczania pierwiastków, $\mathcal{O}(\log^3 B)$, także jest asymptotycznie mniejszy niż $\pi(R)$. Sito bazowe kosztuje $\mathcal{O}(R \log\log R)$ czasu i $\mathcal{O}(R)$ pamięci. Łącznie otrzymujemy

$$\mathcal{O}\left(R \log\log R + L\pi(R)\right) = \mathcal{O}\left(\sqrt{B} \log\log B + L\frac{\sqrt{B}}{\log B}\right)$$

czasu i $\mathcal{O}(R)$ pamięci. Taki algorytm pozwalał rozwiązać drugie podzadanie.

Pełne rozwiązanie: sito przedziałowe

Żeby rozwiązać zadanie w pełnej ogólności, należy podejść do niego sprytniej. Będziemy chcieli jednocześnie wyznaczyć, które liczby z przedziału $[A, B]$ są pierwsze. Osiągamy to, wyznaczając liczby pierwsze do R sitem Eratostenesa, a następnie dla każdej takiej liczby pierwszej p zaznaczając jej złożone wielokrotności w przedziale $[A, B]$. Jest to *sito przedziałowe*.

Zaznaczanie zaczynamy od

$$\max\left(p^2,\ \left\lceil \frac{A}{p} \right\rceil p\right),$$

a następnie zwiększamy zaznaczaną liczbę o p, dopóki nie przekroczymy B. Dzięki rozpoczęciu od co najmniej p^2 nie zaznaczymy samej liczby pierwszej p. Mniejsze złożone wielokrotności p mają mniejszy dzielnik pierwszy i są zaznaczane przy jego przetwarzaniu. Każda liczba złożona nie większa niż B ma dzielnik pierwszy nie większy niż $\sqrt{B}$, więc zostanie zaznaczona. Liczbę 1, jeśli należy do przedziału, oznaczamy osobno jako niepierwszą. Pozostałe niezaznaczone liczby są pierwsze.

Jeśli lista liczb pierwszych jest już gotowa, koszt tego etapu wynosi

$$\mathcal{O}\left(L + \sum_{\substack{p \le R \\ p \text{ pierwsza}}} \left(\frac{L}{p} + 1\right)\right) = \mathcal{O}\left(\pi(R) + L \log\log R\right).$$

Korzystamy tu ze znanego oszacowania sumy odwrotności liczb pierwszych: $\sum_{p \le R,\ p \text{ pierwsza}} 1/p = \mathcal{O}(\log\log R)$. Składnik $\pi(R)$ pochodzi z przejścia po liście liczb pierwszych i wyznaczenia początków zaznaczania. Nie można go pominąć nawet wtedy, gdy przedział jest krótki.

W powyższy sposób można szybko wyznaczyć zbiór liczb pierwszych w $[A, B]$. Zbiór liczb postaci $6p^k + 1$ wyznaczymy podobnie. Zaznaczamy je w osobnej tablicy, niezależnie od pierwszości. Osobno rozważymy przypadek $k = 1$ i osobno $k \ge 2$.

Przypadek $k = 1$. Wyznaczenie takich liczb odpowiada wyznaczeniu liczb pierwszych w przedziale

$$\left[\max\left(2, \left\lceil \frac{A-1}{6} \right\rceil\right), \left\lfloor \frac{B-1}{6} \right\rfloor\right].$$

Jeśli lewy koniec jest większy od prawego, przedział jest pusty. W przeciwnym razie stosujemy tę samą metodę sita przedziałowego, co dla $[A, B]$, wykorzystując już przygotowaną listę liczb pierwszych. Dla każdej znalezionej liczby pierwszej p zaznaczamy liczbę $6p + 1$ w tablicy odpowiadającej $[A, B]$.

Oznaczmy $Q = \lfloor (B-1)/6 \rfloor$. Przedział kandydatów zawiera co najwyżej $\lceil L/6 \rceil$ liczb, a do jego przesiania wystarczą dzielniki pierwsze do $\sqrt{Q}$. Koszt tego etapu jest więc ograniczony przez $\mathcal{O}(\pi(R) + L \log\log B)$.

Przypadek $k \ge 2$. Widzimy, że liczb postaci $6p^k + 1$ nie większych niż B jest mało: musi zachodzić $p^k \le Q$, zatem $p \le \sqrt{Q} \le R$. Znając liczby pierwsze do R, możemy przejść po tych nie większych niż $\sqrt{Q}$. Dla każdej z nich zaczynamy od $v = p^2$ i kolejno mnożymy v przez p, dopóki $v \le Q$. Jeśli $6v + 1 \ge A$, zaznaczamy $6v + 1$ w tablicy. Jeśli $Q < 4$, nie ma żadnych kandydatów dla $k \ge 2$.

Przed obliczeniem następnej potęgi sprawdzamy, czy $v > \lfloor Q/p \rfloor$. Jeśli tak, kończymy przetwarzanie tej podstawy. Pozwala to uniknąć przepełnienia i zbędnego obliczania potęg przekraczających ograniczenie. Liczby wejściowe i wartości pośrednie należy przechowywać w typie 64-bitowym, np. `long long` w C++.

Liczba wygenerowanych potęg wynosi

$$E(Q) = \sum_{k=2}^{\lfloor \log_2 Q \rfloor} \pi\left(Q^{1/k}\right).$$

Składnik dla $k = 2$ jest równy $\pi(\sqrt{Q})$. Pozostałych składników jest $\mathcal{O}(\log Q)$, a każdy z nich jest nie większy niż $Q^{1/3}$. Zatem

$$\pi(\sqrt{Q}) \leq E(Q) \leq \pi(\sqrt{Q}) + \mathcal{O}(Q^{1/3} \log Q).$$

Ponieważ $\pi(\sqrt{Q}) = \Theta(\sqrt{Q}/\log Q)$, a $Q^{1/3} \log Q = o(\sqrt{Q}/\log Q)$, otrzymujemy

$$E(Q) = \Theta(\pi(\sqrt{Q})).$$

Każda kolejna potęga wymaga tylko stałej liczby operacji. Generowanie potęg kosztuje więc $\mathcal{O}(\pi(\sqrt{Q}))$, a nawet jeśli przejdziemy po całej liście liczb pierwszych do R, koszt tego etapu pozostaje $\mathcal{O}(\pi(R))$. Nie trzeba mnożyć tego oszacowania przez maksymalną liczbę wykładników: dla większości podstaw generujemy bardzo mało potęg.

Zliczenie odpowiedzi i łączna złożoność. Wiedząc, które liczby w $[A, B]$ są pierwsze, a które są postaci $6p^k+1$, jesteśmy w stanie łatwo policzyć, ile liczb należy do obu zbiorów albo do żadnego z nich. Przechodzimy po obu tablicach i zwiększamy wynik, gdy odpowiadające sobie wartości logiczne są równe. Ten etap zajmuje $\mathcal{O}(L)$ czasu.

Łącznie przygotowanie sita bazowego zajmuje $\mathcal{O}(R \log\log R)$ czasu, oba sita przedziałowe $\mathcal{O}(\pi(R) + L \log\log B)$, a generowanie wyższych potęg $\mathcal{O}(\pi(R))$. Otrzymujemy więc złożoność czasową

$$\boxed{\mathcal{O}\left((\sqrt{B} + L) \log\log B\right)}, \qquad L = B - A + 1,$$

oraz pamięciową $\mathcal{O}(\sqrt{B} + L)$. Pamięć służy do przechowania sita bazowego, listy liczb pierwszych i tablic przedziałowych. Takie rozwiązanie pozwala rozwiązać wszystkie podzadania.

Autor zadania: Jakub Nowak