Nieformalny wstęp

Dodatkową trudnością w dzisiejszym konkursie było to, że zadania były sformułowane w języku angielskim, a do tego autorami zadań nie byli native speakers. W takich wypadkach nie warto się nadmiernie wgłębiać w dokładne znaczenie historyjki w zadaniu, ale najlepiej od razu starać się wyłuskać z niej właściwy problem algorytmiczny. Bardzo rzadko w treściach zadań występują trudne słowa, których zrozumienie jest bezwzględnie konieczne do rozwiązania zadania – często możemy posługiwać się nieznanymi słowami jako „black boksami". Np. ostatnio rozwiązywałem zadanie, w którym występowały niejakie *hurdles* – nie wiedząc, o co chodzi, po prostu rozwiązywałem zadanie o jakichś „hurdlesach".

A. Auxiliary Question of the Universe

Chcemy przerobić losowy napis nad alfabetem $\{+, (,), 0..9\}$ na poprawne wyrażenie z dodawaniem (zdefiniowane w treści zadania), dodając stosunkowo niewiele nowych znaków. Już na oko widać, że sposobów osiągnięcia celu może być mnóstwo; jako że nie musimy wypisać najkrótszego uzupełnionego wyrażenia, skupimy się na tym, jak dokonać uzupełnienia w najprostszy możliwy sposób.

Najłatwiej chyba skonstruować wyrażenie w postaci spłaszczonej, tzn. jeżeli spojrzymy na wynik jako na drzewo wyrażenia, to będzie miało stałą głębokość. W tym celu przechodzimy wyrażenie od lewej do prawej i dokonujemy następujących zamian:

$$\begin{aligned}) &\longrightarrow (0)+ \\ (&\longrightarrow (0)+ \\ + &\longrightarrow 0+0+ \\ d &\longrightarrow d+ \end{aligned}$$

przy czym d to dowolna cyfra. W ten sposób wychodzi nam długa suma bardzo prostych składników (albo pojedynczych cyfr, albo zer w nawiasach), na końcu której jest zbędny $+$. Usuwamy ten plus i mamy żądane wyrażenie (zauważmy, że w tym podejściu jest istotne, żeby to, na co zamieniamy plusa, miało w sobie co najmniej dwa plusy).

B. Circles on a Screen

Chcemy obliczyć pole sumy (teoriomnogościowej) niewielkiej ($n \leq 100$) liczby spikselowanych kółek. Istotne jest także to, że rozmiar planszy jest także nie za duży: co najwyżej 20000×20000.

Analizujemy kolejne kolumny planszy $i = 1, 2, \ldots, w$. Każda kolumna przecina się z każdym spikselowanym kółkiem, dając przedział albo zbiór pusty. W każdym razie, dla danego i oraz współrzędnych kółka możemy obliczyć postać tego przecięcia w czasie stałym. W ten sposób dla danego i musimy obliczyć moc zbioru będącego sumą co

najwyżej n przedziałów. To zadanie można rozwiązać na kilka sposobów, z których drugi wydaje mi się sympatyczniejszy.

- Rozbijamy każdy przedział $[a, b]$ na początek $(a, +1)$ i tuż-za-koniec $(b + 1, -1)$, sortujemy te pary leksykograficznie i idąc po kolei, obliczamy sumę przedziałów.
- Sortujemy przedziały jako pary i w tej kolejności scalamy kolejne przedziały w jeden, dopóki się przecinają.

Otrzymujemy rozwiązanie $O(w \cdot n \cdot \log n)$, ze względu na sortowanie. Gdyby chcieć być teoretycznie szybszym, można by te wszystkie sortowania wykonać kubełkowo w jednym dużym sortowaniu – to zmniejsza złożoność do $O(w \cdot n + h)$.

C. Homo or Hetero?

W zadaniu symulujemy dynamiczny (tzn. zmieniający się w czasie) multizbiór i mamy po każdej wykonanej na nim operacji odpowiadać na pytania o jakieś jego własności. Kluczowa obserwacja: zakres elementów jest od zera do miliona. W takim razie multizbiór symulujemy za pomocą tablicy zliczającej elementy. Dodatkowo pamiętamy dwa liczniki: ile jest *różnych* elementów w multizbiorze (to do pytań o hetero) oraz ile jest elementów mających w multizbiorze co najmniej dwa wystąpienia (pytania o homo). Teraz wystarczy pokazać, że te liczniki możemy aktualizować po kolejnych wstawieniach i usunięciach.

D. Least Common Multiple

To zadanie jest, po prawdzie, bardziej matematyczne niż informatyczne. Pytanie to: ile jest ciągów N-elementowych liczb naturalnych, których NWW elementów jest równe T?

Rozłóżmy T na czynniki pierwsze, $T = \prod p_i^{\alpha_i}$. Rozwiążemy nasze zadanie dla każdego czynnika pierwszego z osobna, po czym wymnożymy otrzymane liczby sposobów, co będzie odpowiadało przemnożeniu ciągów otrzymanych dla poszczególnych czynników pierwszych. Łatwo widać, że przy obliczaniu NWW czynniki pierwsze są od siebie niezależne.

Przy obliczaniu wyniku dla danego $p_i^{\alpha_i}$ istotne jest jedynie α_i. Wynik ten to

$$W(N, \alpha_i) - W(N, \alpha_i - 1),$$

przy czym $W(N, a)$ to liczba ciągów N-elementowych, których wyrazy są potęgami tej samej liczby pierwszej o wykładniku nieprzekraczającym a. Podkreślmy, że w tej różnicy chodzi o to, żeby choć jeden element ciągu był postaci p^a, czyli żeby NWW wyszło takie jak chcemy. Wreszcie $W(N, a) = (a + 1)^N$.

Dodajmy, że wszystkie obliczenia wykonujemy modulo żądane $10^9 + 7$ i nie ma w tym problemu, gdyż mamy tylko odejmowania i mnożenia.

E. Magic Power

W przeciwieństwie do poprzedniego zadania, wbrew pozorom, to zadanie jest mocno informatyczne. W rozwiązaniu musimy przede wszystkim zgadnąć, że dziesięciomilionowa potęga trójki, której zapis dziesiętny zaczyna się od 9, ma niezbyt wygórowany indeks – około $2 \cdot 10^8$. Pewnie można to jakoś formalnie udowodnić, ale nie będziemy tutaj wnikać – powiedzmy, że jeśli potraktujemy potęgu trójki jako losowo wyglądające liczby w układzie dziesiętnym, to zadziała *prawo Benforda* (więcej informacji na temat tego prawa na pewno jest gdzieś w internecie, a będzie też w *Delcie* 12/2010).

W rozwiązaniu rozpatrujemy kolejne potęgi trójki o wykładnikach $1, 2, 3, \ldots$, utrzymując równolegle największą potęgę dziesiątki nieprzekraczającą danego 3^i. Oczywiście, znając tę potęgę 10^j oraz wartość 3^i, łatwo stwierdzamy, czy 3^i zaczyna się od 9 – wystarczy sprawdzić, czy $3^i \geq 9 \cdot 10^j$. Z kolei same liczby 3^i oraz 10^j trzymamy w jakimś typie zmiennoprzecinkowym (`double`, lepiej `long double`) i mamy nadzieję, że nie będzie błędów obliczeń. Alternatywnie, żeby nie walczyć z ogromnymi liczbami, można operować na logarytmach z tychże. Znów pomijamy uzasadnienie tego, że raczej nie będzie problemów z dokładnością.

Niestety, tak czy siak możemy nie zmieścić się w czasie, skoro musimy rozpatrzyć $2 \cdot 10^8$ potęg. W takim razie potraktujemy to rozwiązanie jako preprocessing: wypiszemy indeks powiedzmy co M-tej znalezionej potęgi trójki zaczynającej się od cyfry 9 i wyniki te wpiszemy w kod wysyłanego rozwiązania w postaci stałej tablicy t, a kolejne zapytania k_i będziemy obsługiwać, rozpoczynając przeszukiwanie od liczby $t[\lfloor k_i/M \rfloor]$ i szukając od tego miejsca $(k_i - \lfloor k_i/M \rfloor \cdot M)$-tej potęgi trójki rozpoczynającej się od dziewiątki. Wartość M musimy dobrać tak, żeby była jak największa, ale także żeby tablica wpisana w kod wysyłanego programu zmieściła się w limicie 100 kB na kod źródłowy, w tym zadaniu np. $M = 3000$ jest dobre.

Dokładniejszy opis omawianego zastosowania metody preprocessingu/tablicowania można znaleźć w lekcji o sztuczkach programistycznych na MAIN-ie:
`http://main.edu.pl/user.phtml?op=lesson&n=30&page=algorytmika`

G. Round Table

Mamy dane dwie permutacje π_1 i π_2 zbioru $1..n$ i chcemy znaleźć dwa elementy występujące w obydwu permutacjach z takim samym odstępem (na razie nie definiujmy formalnie, co to znaczy). W przykładzie do zadania jedna permutacja jest identycznościowa; spróbujmy najpierw opisać rozwiązanie dla przypadku, gdy $\pi_1 = \pi$ (dowolne), $\pi_2 = id$.

W tym przypadku szukamy takich dwóch indeksów i, j, że

$$\pi[j] - \pi[i] \equiv j - i \pmod{n},$$

który to wzór polecam rozpisać i sprawdzić. Jest on równoważny następującemu:

$$\pi[j] - j \equiv \pi[i] - i.$$

W takim razie zadanie sprowadza się do obliczenia wartości $(\pi[i]-i) \bmod n$ dla wszystkich $i = 1, 2, \ldots, n$ i sprawdzenia, czy jakieś dwie są takie same, co można zrobić w czasie i pamięci $O(n)$ za pomocą tablicy zliczającej rozmiaru n.

W ogólnym przypadku może jednak zachodzić $\pi_2 \neq id$. Zauważmy, że wówczas nasze zadanie możemy zamienić na to samo zadanie dla permutacji $\pi_1' = \pi_1 \cdot \pi_2^{-1}$ oraz $\pi_2' = id$. Korzystamy tu z faktu, że po przyłożeniu do obu wyjściowych permutacji permutacji π_2^{-1}, położenia odpowiadających sobie elementów nie zmienią się. No to koniec.

H. Spaceship Connections

Jakoś tak często wychodzi, że zadania o przekątnych wielokąta sprowadzają się do drzew przedziałowych. Nie inaczej będzie tym razem.

Wykorzystamy dwa drzewa przedziałowe: *min* i *max*. Drzewo *min* działa na ciągu $(a_i)_{i=1}^n$, początkowo złożonym z samych nieskończoności, i pozwala na wykonywanie przypisań wartości poszczególnych a_i i pytanie o minimum na przedziale $[i, j]$, tzn.

$$\min\{a_i, a_{i+1}, \ldots, a_j\}.$$

Drzewo *max* podobnie, tylko pracuje na ciągu b_i, początkowo same minus nieskończoności. Takie drzewa można zaimplementować w $O(n)$ pamięci i z narzutem czasowym $O(\log n)$ przy każdej operacji, po szczegóły odsyłam do wykładu 2 na WAS-ie:
`http://was.zaa.mimuw.edu.pl/?q=node/8`

W naszym rozwiązaniu a_i będzie równe numerowi wierzchołka wielokąta połączonego przekątną z wierzchołkiem i-tym, a $+\infty$, jeżeli takiego wierzchołka nie ma. Ciąg b_i definiujemy symetrycznie, tylko z $-\infty$.

Zauważmy wreszcie, że do wielokąta możemy dołożyć przekątną (i, j), $i < j$, jeżeli wierzchołki i, j nie są jeszcze z niczym połączone oraz:

$$\min\{a_i, \ldots, a_j\} > i \quad \wedge \quad \max\{b_i, \ldots, b_j\} < j.$$

Intuicyjnie chodzi o to, że żaden wierzchołek ze zbioru $(i+1)..(j-1)$ nie może być połączony z żadnym spoza tego zbioru. Podane warunki sprawdzamy z użyciem wspomnianych drzew przedziałowych, całość działa w czasie $O(n \log n)$.

F. Recover Path

To zadanie zostawiłem na sam koniec, gdyż jest według mnie najtrudniejsze w zestawie. Z tego względu opis rozwiązania ograniczam do kilku wskazówek tudzież spostrzeżeń, niech P oznacza zbiór wierzchołków podanych na wejściu (ten do uzupełnienia).

1. Można założyć, że wynikowa ścieżka zaczyna się i kończy w jednym z wierzchołków z P, co łatwo pokazać.

2. Jeżeli puszczę algorytm Dijkstry z dowolnego wierzchołka z P, to najdalszy z pozostałych wierzchołków z P (oznaczmy go przez v_1) będzie jednym ze skrajnych na ścieżce.

3. Puszczamy drugi raz algorytm Dijkstry, tym razem z v_1, sortujemy wierzchołki z P po odległościach od niego – w ten sposób uzyskujemy kolejność wierzchołków $v_1, v_2, \ldots, v_k$ z P na ścieżce.

4. Teraz pozostaje tylko uzupełnić ścieżkę brakującymi wierzchołkami spoza zbioru P. To można zrobić na kilka sposobów. Mój ulubiony polega na puszczaniu DFS-ów z kolejnych v_i. W danym DFS-ie chodzimy tylko po krawędziach z drzewa najkrótszych ścieżek wyznaczonego w drugim algorytmie Dijkstry, no i interesujemy się jedynie wierzchołkami, których odległość od v_1 jest $\geq odl(v_1, v_i)$ oraz jest $\leq odl(v_1, v_{i+1})$. W ten sposób dochodzimy z v_i do v_{i+1}, po czym fragment ścieżki $v_i \longrightarrow v_{i+1}$, cofając się po tablicy ojców z DFS-a.