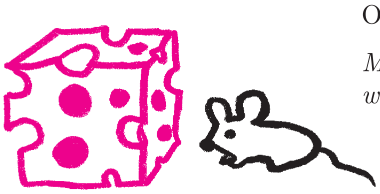
Informatyczny kącik olimpijski (2)



Oto zadanie z Bałtyckiej Olimpiady Informatycznej (2004):

Mamy dany ciąg liczb naturalnych a_i, $1 \leqslant i \leqslant N$. Należy znaleźć taki ciąg b_i, żeby wartość

$$\sum_{i=1}^{N} |a_i - b_i|$$

była najmniejsza możliwa oraz $b_i \leqslant b_{i+1}$ dla $1 \leqslant i < N$.

Na początek zajmijmy się trochę prostszym zadaniem. Oznaczmy $S = \sum_{i=1}^{N} |a_i - b_i|$ i spróbujmy obliczyć minimalną możliwą wartość S. Zadanie to ma proste rozwiązanie, korzystające z metod programowania dynamicznego. Niech liczba $A_k(v)$ oznacza minimalną wartość sumy $\sum_{i=1}^{k} |a_i - b_i|$, przy założeniu, że $b_k = v$, dla $1 \leqslant k \leqslant N$. Dla ułatwienia obliczeń przyjmijmy, że $A_0(v) = 0$ dla dowolnego v. Łatwo zauważyć, że optymalny ciąg b_i nie ma wartości mniejszych, niż najmniejsze a_i, ani większych, niż największe a_i. Oznaczmy te najmniejsze i największe a_i, odpowiednio, przez X i Y, zaś T niech oznacza liczbę możliwych wartości, jakie mogą przybierać elementy ciągu a_i ($T = Y - X + 1$). Zauważmy też, że optymalny ciąg b_i nie zawiera żadnych wartości poza wartościami występującymi w a_i – ta obserwacja przyda nam się później. Teraz $S = \min_{X \leqslant v \leqslant Y} A_N(v)$, zaś na $A_k(v)$, dla $1 \leqslant k$, mamy prosty wzór:

$$A_k(v) = |v - a_k| + \min_{X \leqslant v' \leqslant v} A_{k-1}(v').$$

Wzór ten daje nam algorytm, który w czasie $O(NT)$ oblicza minimalne możliwe S (jeśli tylko, dla $v > X$, będziemy pamiętać minimalne $A_{k-1}(v')$, co pozwoli obliczyć $A_k(v)$ na podstawie $A_k(v-1)$ w czasie stałym). Potrzebujemy tylko $O(T)$ pamięci, ponieważ do wyliczenia wartości A_k potrzebne są tylko wartości A_{k-1}, które możemy nadpisać przy zwiększaniu k. Jeśli jednak chcemy znaleźć sam ciąg b_i, musimy dla każdego $A_k(v)$ zapamiętać, jakie v' daje taką jego wartość. Liczby v i v' to nic innego, jak wartości b_k i b_{k-1}. W ten sposób złożoność pamięciowa rośnie do $O(NT)$, a czasowa pozostaje taka sama.

Dodatkowo, możemy obniżyć złożoność, ograniczając T. Jak zauważyliśmy, wyrazy ciągu b_i nie przyjmują innych wartości, jak te występujące w ciągu a_i. Tych wartości jest co najwyżej N. Możemy więc obliczać wartości $A_k(v)$ tylko dla $O(N)$ różnych wartości v. W ten sposób, po wstępnym „wyjęciu" i posortowaniu możliwych wartości elementów ciągu b_i w czasie $O(N \log N)$, główna część algorytmu będzie mieć złożoność czasową i pamięciową $O(N^2)$.

Ale istnieje też rozwiązanie o lepszej złożoności. Na początek dziedzinę A_k rozszerzmy na wszystkie liczby naturalne, nie tylko te, które leżą w przedziale $[X, Y]$. Wzór $A_k(v) = |v - a_k| + \min_{v' \leqslant v} A_{k-1}(v')$ wciąż ma sens, ponieważ dla $v < X$ liczba $A_k(v)$ jest coraz większa dla coraz mniejszych v (poza przypadkiem, gdy $k = 0$). Oznaczmy $f\diamondsuit(v) = f(v-1) + f(v+1) - 2f(v)$. Jest to podwójna pochodna dyskretna ciągu f, przesunięta o 1 pozycję. Ta „pochodna" ma własność:

$$f\diamondsuit + g\diamondsuit = (f+g)\diamondsuit.$$

co widać wprost z definicji $f\diamondsuit$. Zastanówmy się, jak – łatwo i szybko – znając $A_k\diamondsuit$, znaleźć $A_{k+1}\diamondsuit$? Przejście od A_k do A_{k+1} podzielimy na dwa etapy – najpierw wyliczymy $B_k(v) = \min_{v' \leqslant v} A_k(v')$, a następnie do tego ciągu dodamy ciąg $C_k(v) = |v - a_{k+1}|$, otrzymując A_{k+1}. Stąd $C_k\diamondsuit$ jest ciągiem samych zer, poza jedynym wyrazem $C_k\diamondsuit(a_{k+1}) = 2$. Ciąg B_0 składa się z samych zer, a więc $A_1 = C_0$. Zastanówmy się teraz, jak wyglądają ciągi A_k dla $k > 0$. Dla odpowiednio dużych v mamy $A_k(v+1) - A_k(v) = 1$, ponieważ ciąg b_i, dla $1 \leqslant i < k$ zachowuje optimum, i zmienia się tylko b_k. Co więcej, jak się za chwilę okaże, ciąg $A_k\diamondsuit$ ma wyrazy nieujemne. Niech t_k będzie największym indeksem takim, że $A_k\diamondsuit(t_k) > 0$. W takiej sytuacji $B_k(v) = A_k(v)$ dla każdego $v \leqslant t_k$, oraz $B_k(v) = A_k(t_k)$ dla pozostałych v. Podobnie, dla $v \neq t_k$ mamy $A_k\diamondsuit(v) = B_k\diamondsuit(v)$, a dla $v = t_k$ mamy $A_k\diamondsuit(v) = B_k\diamondsuit(v) + 1$. A skoro $A_{k+1}\diamondsuit = B_k\diamondsuit + C_k\diamondsuit$, to $A_{k+1}\diamondsuit$ ma wyrazy nieujemne. Warto zauważyć w tym momencie, że $A_k(t_k)$ jest najmniejszą wartością ciągu A_k.

Jak teraz opisać funkcję $A_k\diamondsuit$? Zapamiętajmy ją jako multizbiór – jeśli funkcja ma na pozycji 4 wartość 2, to w zbiorze znajdą się dwie czwórki itp. Przejście z $A_k\diamondsuit$ do $B_k\diamondsuit$ wymaga znalezienia największego elementu i usunięcia go (tylko raz, jeśli się powtarza). Przejście z $B_k\diamondsuit$ do $A_{k+1}\diamondsuit$ to dodanie do zbioru dwóch wartości a_{k+1}. W takim razie widać już, jak znaleźć opis funkcji $A_N\diamondsuit$ w czasie $O(N \log N)$ i pamięci $O(N)$. Ile wynosi zatem najmniejsze możliwe S? Oczywiście, jest to $A_N(t_N)$. Jak z kolei wyliczyć kolejne wartości $A_k(t_k)$? Wiemy na pewno, że

$$A_{k+1}(t_{k+1}) = B_k(t_{k+1}) + |t_{k+1} - a_{k+1}|.$$

Jeśli jeszcze zauważymy, że dla $k < N$ mamy $A_k(t_k) = B_k(t_k) = B_k(t_{k+1})$, to nie tracąc na złożoności możemy rozszerzyć algorytm obliczania $A_N\diamondsuit$ o obliczanie $A_k(t_k)$. Podobnie, nie tracąc nic na złożoności, możemy skorzystać ze spostrzeżenia, że $b_N = t_N$, a dla $k < N$ mamy $b_k = \min(t_k, b_{k+1})$, i obliczyć również optymalny ciąg b_k.

Filip WOLSKI