Informatyczny kącik olimpijski (116): Ciągi i łańcuchy

Tym razem omówimy dwa zadania z IX International Autumn Tournament in Informatics, który odbył się w listopadzie 2017 roku w Szumem w Bułgarii.

Zadanie Ciągi: *Dane są trzy dodatnie liczby całkowite n, m i k. Należy obliczyć liczbę niemalejących ciągów długości n o wartościach będących liczbami całkowitymi z przedziału $[1;m]$, w których żadna wartość nie występuje więcej niż k razy. Przykład: dla $n=4$, $m=3$ i $k=2$ poprawną odpowiedzią jest 6. Poprawnymi ciągami są: $(1,1,2,2)$, $(1,1,2,3)$, $(1,1,3,3)$, $(1,2,2,3)$, $(1,2,3,3)$ oraz $(2,2,3,3)$.*

Rozwiązanie $O(n\cdot m\cdot\min(n,k))$
W rozwiązaniu skorzystamy z techniki programowania dynamicznego. Niech $DP[i][j]$ oznacza liczbę niemalejących ciągów długości i, złożonych z liczb całkowitych z przedziału $[1;j]$, w których żadna wartość nie występuje więcej niż k razy. Łatwo zauważyć, że:

- $DP[0][j]=1$ dla $j\in[1;m]$, jest tylko jeden pusty ciąg;
- $DP[i][1]=1$ dla $i\in[1;k]$;
- $DP[i][1]=0$ dla $i\in[k+1;n]$.

Zastanówmy się teraz, jak obliczyć wartość $DP[i][j]$ dla pozostałych par i,j. W ciągach niemalejących elementy o tych samych wartościach tworzą spójny przedział. W szczególności wartości j tworzą spójny przedział długości $l\in[0,\min(i,k)]$. Zatem, otrzymujemy ogólny wzór:

$$DP[i][j]=\sum_{l=0}^{\min(i,k)}DP[i-l][j-1].$$

Odpowiedzią w zadaniu jest wartość $DP[n][m]$. Rozwiązanie działa w czasie $O(n\cdot m\cdot\min(n,k))$.

Rozwiązanie $O(n\cdot m)$
Powyższy wzór możemy zapisać równoważnie jako:

- dla $i\leqslant k$,
  $DP[i][j]=\sum_{l=0}^{i}DP[i-l][j-1]=$
  $=DP[i][j-1]+\sum_{l=1}^{i}DP[i-l][j-1]=$
  $=DP[i][j-1]+\sum_{l=0}^{i-1}DP[i-1-l][j-1]=$
  $=DP[i][j-1]+DP[i-1][j]$
- dla $i>k$,
  $DP[i][j]=\sum_{l=0}^{k}DP[i-l][j-1]=$
  $=DP[i][j-1]+\sum_{l=1}^{k}DP[i-l][j-1]=$
  $=DP[i][j-1]+\sum_{l=0}^{k-1}DP[i-1-l][j-1]=$
  $=DP[i][j-1]+DP[i-1][j]-DP[i-1-k][j-1]$

Zauważmy, że obliczenie wartości $D[i][j]$ dla dowolnych i,j zajmuje czas stały. Zatem całe rozwiązanie działa w czasie $O(n\cdot m)$.

Zadanie Łańcuchy: *Dany jest ciąg liczb całkowitych $a=a_1,a_2,\ldots,a_n$. Łańcuch rozpoczynający się na k-tej pozycji powstaje w niżej opisany sposób. Znajdujemy pierwszy większy element na prawo od a_k i oznaczamy go przez a_{k_1}. Następnie znajdujemy pierwszy większy element na prawo od a_{k+1} i oznaczmy go przez a_{k_2}, itd. W ten sposób, dla ustalonego k otrzymujemy łańcuch $a_{k_1},a_{k_2},\ldots,a_{k_m}$. W zadaniu należy dla każdego $k\in\{1,2,\ldots,n\}$ znaleźć długość łańcucha rozpoczynającego się na k-tej pozycji. Przykład: dla $a=3,5,4,5,6$ wynikiem jest $(2,1,2,1,0)$.*

Rozwiązanie $O(n^2)$
Najprostsze rozwiązanie polega na wygenerowaniu dla każdego $k\in\{1,2,\ldots,n\}$ łańcucha rozpoczynającego się na k-tej pozycji (zgodnie z opisem w treści zadania) i wypisaniu jego długości. Wyznaczenie łańcucha dla każdej pozycji zajmuje $O(n)$ operacji (musimy przeiterować się po całym ciągu). Zatem całe rozwiązanie działa w czasie $O(n^2)$.

Rozwiązanie $O(n)$
Niech $F[k]$ oznacza długość łańcucha rozpoczynającego się na k-tej pozycji. W tym rozwiązaniu będziemy wyznaczali $F[k]$ dla kolejnych k od n do 1 (od prawej do lewej). Załóżmy, że obliczyliśmy już $F[k+1],F[k+2],\ldots,F[n]$ i chcemy obliczyć $F[k]$. Jeśli pierwszym większym elementem na prawo od a_k jest a_l, wtedy $F[k]=F[l]+1$ (korzystamy z wcześniej obliczonego wyniku dla l). Jeśli zaś wszystkie elementy na prawo są nie większe niż a_k, wtedy $F[k]=0$.

Zastanówmy się teraz, jak dla każdego a_k wyznaczyć pierwszy większy element na prawo od niego. Oczywiście, możemy to zrobić naiwnie (przeglądając kolejne elementy). Wówczas jednak otrzymamy rozwiązanie $O(n^2)$.

Zauważmy, że dla pary indeksów $1\leqslant l<m\leqslant n$, jeśli $a_l\geqslant a_m$, to a_m nie będzie kolejnym elementem w łańcuchu dla żadnego elementu na pozycji $[1;l]$ (możemy myśleć o tym w ten sposób, że a_l przysłania a_m). Zatem kandydaci na kolejny element w łańcuchu tworzą ciąg rosnący. Przeglądając ciąg a od prawej do lewej, przechowujemy kandydatów na stosie. Na szczycie stosu znajduje się najmniejszy element, na dole zaś największy. Kiedy chcemy znaleźć kolejny element w łańcuchu dla a_k, wówczas tak długo zdejmujemy elementy ze stosu, aż na szczycie stosu pojawi się element większy niż a_k. Jeśli taki element nie pojawi się, oznacza to, że taki element nie istnieje. Po wyznaczeniu kolejnego elementu w łańcuchu odkładamy a_k na szczyt stosu.

Powyższy algorytm działa w zamortyzowanym czasie liniowym. Wynika to bezpośrednio z własności stosu (każdy z n elementów został dokładnie raz odłożony na stos i raz z niego zdjęty). Powyższe rozwiązanie działa w czasie $O(n)$.

Bartosz ŁUKASIEWICZ