IKO na start – Optymalizacja mandatów, czyli zachłanność czasem popłaca

W tym odcinku omówimy zadanie **Optymalizacja mandatów** z pierwszego etapu XV Olimpiady Informatycznej Juniorów.

O co tu chodzi?

Zadanie *Optymalizacja mandatów* ma dość skomplikowaną treść, którą można podsumować w następujący sposób: dane są dwa ciągi po N liczb, ciąg $K = (K_1, K_2, \ldots, K_N)$ oraz $R = (R_1, R_2, \ldots, R_N)$. Liczby dobieramy w pary, po jednej z K oraz po jednej z R, po czym każdą parę „sklejamy" – dopisujemy liczbę z R na koniec liczby z K. Potem te wszystkie uzyskane sklejenia sumujemy. Na przykład, jeśli mamy ciągi $K = (13, 9, 37)$ oraz $R = (100, 21, 5)$ i sparujemy 13 ze 100, 9 z 21, a 37 z 5, wtedy wynik to $13100 + 921 + 375 = 14396$. Możemy też skleić 13 z 5, 9 ze 100, a 37 z 21, osiągając $135 + 9100 + 3721 = 12956$. Celem jest takie sparowanie, żeby powstała suma była jak najmniejsza.

Pełną treść zadania z przykładami można znaleźć na stronie https://sio2.mimuw.edu.pl/c/oij15-1/p/opt/.

Dwie pary

Zobaczmy, co się dzieje, jeśli $N = 2$, czyli do utworzenia są tylko dwie pary liczb. Spójrzmy na kilka przykładów, zaczynając od $K = (14, 9)$ oraz $R = (15, 33)$. Z pewnym zaskoczeniem możemy się przekonać, że tutaj nasz wybór nie ma żadnego znaczenia! Możliwe sparowania to $1415 + 933 = 2348$ albo $1433 + 915 = 2348$. Dzieje się tak dlatego, że obie liczby z R są takiej samej długości. Zatem do liczby 14 możemy dokleić 15 lub 33, otrzymując 1415 lub 1433, ale w obu wypadkach początkowa liczba 14 powiększy swoją wartość do 1400, czyli 100 razy większej. Podobnie liczba 9 będzie miała wartość 900 przy obu sparowaniach.

Mówiąc formalnie, liczba k sparowana z liczbą r zamienia się w liczbę $10^{d(r)} \cdot k + r$, gdzie $d(r)$ jest liczbą cyfr r. Jeśli zatem $K = (K_1, K_2)$, a $R = (R_1, R_2)$ i obie liczby z R mają tę samą długość, czyli $d(r_1) = d(r_2) = d$, to wynik będzie równy

$$10^d \cdot K_1 + R_1 + 10^d \cdot K_2 + R_2 = 10^d(K_1 + K_2) + R_1 + R_2$$

niezależnie od sparowania.

Co jednak, gdy długości liczb z R są różne? Weźmy na przykład $K = (16, 8)$ i $R = (23, 9)$. Do jednej z liczb z K doklei się dwucyfrowa liczba 23, zwiększając jej wartość 100 razy, do drugiej – jednocyfrowa liczba 9 i powiększy wartość dziesięciokrotnie. Aby więc otrzymać najmniejszą możliwą wartość, opłacalne jest bardziej zwiększyć mniejszą liczbę. Rzeczywiście, sparowanie $169 + 823 = 992$ jest lepsze niż $1623 + 89 = 1712$. Znów, przechodząc na zapis formalny, mamy ciąg (K_1, K_2), gdzie $K_1 > K_2$; ciąg (R_1, R_2), w którym długości liczb z R są odpowiednio d_1 i d_2; oraz niech $d_1 > d_2$, czyli liczba R_1 jest dłuższa. Wynik wynosi $10^{d_1} \cdot K_1 + R_1 + 10^{d_2} \cdot K_2 + R_2$ w jednym sparowaniu i $10^{d_2} \cdot K_1 + R_2 + 10^{d_1} \cdot K_2 + R_1$ w drugim. Różnica między tymi wynikami to:

$$10^{d_1} \cdot K_1 + R_1 + 10^{d_2} \cdot K_2 - 10^{d_2} \cdot K_1 + R_1 - 10^{d_1} \cdot K_2 = (10^{d_1} - 10^{d_2})(K_1 - K_2),$$

czyli zawsze jest dodatnia, jeśli tylko $K_1 > K_2$ i $d_1 > d_2$.

Mniejszy jest więc drugi rezultat, co potwierdza to, co zauważyliśmy – mniejszą liczbę K_2 trzeba skleić z dłuższą R_1.

Pełne rozwiązanie, czyli algorytm zachłanny

Mogłoby się wydawać, że do rozwiązania dla ciągów N-liczbowych jeszcze daleko, ale tak naprawdę wykonaliśmy już większość (koncepcyjnej) pracy. Jeśli dane są ciągi $K = (K_1, \ldots, K_N)$ i $R = (R_1, \ldots, R_N)$, to zastanówmy się, jak przypisać liczby z R do liczb z K. Intuicja (z poprzedniego akapitu) sugeruje, że i teraz najdłuższe liczby z R powinniśmy skleić z najmniejszymi liczbami z K, i odwrotnie. Spróbujmy potwierdzić ten pomysł: ustawmy liczby z K w kolejności niemalejącej, czyli załóżmy, że $K_1 \leqslant K_2 \leqslant \ldots \leqslant K_N$. W jakiej kolejności dobrać do nich liczby z ciągu R? Popatrzmy najpierw na K_1 i K_2: zauważmy, że jeśli do K_1 dokleimy jakąś liczbę R_i, a do K_2 inną liczbę R_j, to R_i powinna być krótsza, a w każdym razie nie dłuższa niż R_j. Inaczej bowiem moglibyśmy zamienić liczby miejscami, łącząc $K_1 \leftrightarrow R_j$ i $K_2 \leftrightarrow R_i$. A na podstawie tego, co mówiliśmy poprzednio o dwóch liczbach, to dałoby lepsze rozwiązanie! Zatem do K_2 powinna być doklejona dłuższa liczba niż do K_1. Ale to samo rozumowanie możemy powtórzyć teraz dla K_2 i K_3, a potem dla każdej pary kolejnych liczb z ciągu K i R – gdyby gdziekolwiek krótsza liczba z R stała przed dłuższą, to opłacałoby się zamienić je miejscami. Potwierdza to naszą intuicję, że do liczb z K ustawionych w kolejności rosnącej powinniśmy doklejać liczby z R w kolejności malejących długości. Jako że przy równych długościach każda kolejność jest dobra, najłatwiej po prostu posortować ciąg R malejąco (wtedy na pewno dłuższe liczby będą przed krótszymi) i skleić po kolei liczby z K, biorąc je od najmniejszych, z liczbami z R branymi od największych.

Kiedy już znamy strategię, implementacja jest prosta: ciągi K i R trzymamy w tablicach, sortujemy obydwie (np. funkcją `sort()` w C++), a następnie przechodzimy je pętlą – pierwszą tablicę od początku, drugą od końca – i sumujemy iloczyny.

Algorytmy, w których rozwiązanie konstruujemy od tego, co wydaje się najlepsze w danej chwili (takie jak użyte przed chwilą „posortuj i połącz największą z najmniejszą") nazywamy *algorytmami zachłannymi*. Czasem taka strategia działa i daje najlepsze możliwe rozwiązanie, czasem nie. W tym wypadku udowodniliśmy, że jak najbardziej działa. Zastosowaliśmy przy tym coś, co nazywa się *exchange argument*, czyli „dowód przez zamianę": pokazaliśmy, że każde ustawienie ciągów inne niż rosnące/malejące dałoby się poprawić, zamieniając miejscami dwa sąsiednie elementy. To jedna z typowych technik dowodzenia, że strategia zachłanna działa.

A Czytelnicy i Czytelniczki obeznani z technikami z olimpiad matematycznych z pewnością zauważą w tym dowodzie znajomy fakt zwany „twierdzeniem o ciągach jednomonotonicznych" – to jest właśnie informatyczna wersja tego twierdzenia.

Lech DURAJ