Informatyczny kącik olimpijski (5) – największy XOR podciągu

Zadanie kolejnej edycji kącika pochodzi z informatycznych zawodów czesko-polsko-słowackich w Ružinie, 2005:

Mając dany ciąg liczb całkowitych $a_1, a_2, \ldots, a_N$, należy znaleźć taki jego podciąg, którego suma XOR (czyli wartość $a_{i_1} \oplus a_{i_2} \oplus \ldots \oplus a_{i_N}$) jest największa.

Przypomnijmy: działanie XOR jest podobne do dodawania w systemie binarnym, z tą różnicą, że pomijamy przeniesienie. W takim razie i-ty bit wyniku działania XOR na kilku liczbach jest wyznaczony przez i-te bity argumentów tej operacji (XOR jest łączne i przemienne, więc możemy mówić o „XORowaniu" kilku liczb naraz). Na przykład $1011 \oplus 0110 \oplus 1010 = 0111$ (w systemie binarnym).

Zastanówmy się, jaki *co najmniej* jest wynik? Jeśli dla jakiegoś i mamy $a_i \geqslant 2^M$, to wynik również jest nie mniejszy niż 2^M. Niech M będzie największym takim wykładnikiem (czyli liczby a_i są co najwyżej $(M+1)$-bitowe). Żeby otrzymać wynik nie mniejszy niż 2^M, nasz podciąg musiałby mieć nieparzystą liczbę a_i nie mniejszych niż 2^M – inaczej mówiąc, niepatrzystą ilość takich a_i, w których M-ty bit jest równy 1. Czy podobne rozumowanie możemy przeprowadzić dla niższych bitów? Spójrzmy na $(M-1)$-szy bit – chcemy, żeby jednocześnie podciąg miał nieparzystą ilość tych a_i, których M-ty bit jest równy 1, oraz nieparzystą ilość tych a_i, których $(M-1)$-szy bit jest równy 1.

Zapiszmy to. Niech $b_{i,j}$ będzie j-tym bitem i-tej liczby. Chcemy więc, żeby dla $k = M$ oraz $k = M-1$ nieparzysta była liczba:

$$|\{1 \leqslant i \leqslant N : x_i = 1 \wedge b_{i,k} = 1\}|,$$

gdzie x_i jest równe 1, jeśli i-tą liczbę wybieramy do podciągu, lub 0, jeśli nie. Czyli, pisząc inaczej:

$$\sum_{i=1}^{N} x_i \cdot b_{i,k} \equiv 1 \pmod 2.$$

Ale to już wygląda na układ równań liniowych (modulo 2) z niewiadomymi x_i, który potrafimy rozwiązywać! Jeśli oznaczymy przez R_k powyższe równanie, to rozwiązanie wyglądałoby tak:

- Weźmy $\mathcal{R} := \emptyset$.
- Dla kolejnych bitów liczb z wejścia (od pozycji $k = M$ do $k = 0$):
  - Piszemy równanie R_k odpowiadające bitom z pozycji k.
  - Sprawdzamy, czy spełnialny jest układ równań $\mathcal{R} \cup \{R_k\}$.
  - Jeśli tak, podstawiamy $\mathcal{R} := \mathcal{R} \cup \{R_k\}$ i ustawiamy k-ty bit wyniku na 1.

Przypomnijmy jeszcze, jak rozwiązywać układy równań liniowych – w szczególności, często spotykane na zawodach programistycznych – modulo 2. Układ równań zapisuje się w macierzy – w i-tym wierszu znajdują się kolejne współczynniki i-tego równania (czyli u nas b_{j,k_i} dla $1 \leqslant j \leqslant N$), a po nich prawa strona i-tego równania (wyraz wolny 1). Nasza macierz będzie miała ostatecznie $N+1$ kolumn i co najwyżej $M+1$ wierszy. Służy ona jedynie do reprezentacji – operacje na niej (zamiana dwóch wierszy miejscami, i dodanie jednego wiersza do drugiego) mają swoje odpowiedniki w operacjach na równaniach (zamiana dwóch równań miejscami i dodanie jednego równania do drugiego nie zmieniają zbioru rozwiązań układu równań).

Metodą eliminacji Gaussa, korzystając z tych dwóch operacji, sprowadzamy macierz do znanej *postaci schodkowej*. Po tej operacji może się okazać, że układ równań był sprzeczny – tj. otrzymamy równanie $0 = 1$. Jeśli zaś tak nie jest, to już w tym momencie wiemy, że układ ma rozwiązanie. To pierwsze miejsce, gdzie możemy wykorzystać specyfikę zadania – zamiast od razu szukać rozwiązań x_i, pozostańmy przy postaci schodkowej i do takiej macierzy dodawajmy po jednym wierszu (równaniu). Przyniesie to istotne polepszenie złożoności obliczeniowej. Po przejrzeniu wszystkich bitów chcemy poznać rozwiązanie x_i otrzymanego układu schodkowego $\mathcal{R}$. W tym celu macierz trzeba dodatkowo sprowadzić do *zredukowanej postaci schodkowej*, w której pierwszy niezerowy wyraz w każdym wierszu jest jedynym niezerowym wyrazem w swojej kolumnie oraz jest równy 1.

Dodatkowe przyspieszenie działania już nie algorytmu, a samego programu, uzyskamy wykorzystując „sztuczkę implementacyjną": do zapamiętania liczby z $\mathbb{Z}_2$ wystarczy jeden bit, a dodawanie modulo 2 możemy zastąpić XORem – wtedy każdy wiersz macierzy można „upakować" w $\lceil \frac{N+1}{B} \rceil$ liczbach, gdzie B jest liczbą bitów w typie całkowitym danego komputera.

Ostateczny algorytm ma złożoność $O(NM^2)$ – dominującą operacją jest dodawanie dwóch wierszy, wymagające $\Theta(N)$ kroków (choć z bardzo małą stałą), którą to operację trzeba wykonać $O(M^2)$ razy.

Filip WOLSKI