Szyfr

IX Olimpiada Informatyczna (2001/2002), etap III
Autor zadania i opisu rozwiązania: Wojciech Guzicki
Limit pamięci: 32 MB
https://oi.edu.pl/pl/archive/oi/9/szy

Dany jest ciąg n dodatnich liczb całkowitych $a_1, a_2, \ldots, a_n$. Ciąg ten jest używany do szyfrowania n-bitowych wiadomości. Jeśli mamy wiadomość, której kolejne bity tworzą ciąg $t_1, t_2, \ldots, t_n$ ($t_i \in \{0, 1\}$), to po zaszyfrowaniu ma ona postać liczby

$$S = t_1 a_1 + t_2 a_2 + \ldots + t_n a_n.$$

Masz daną zaszyfrowaną wiadomość oraz ciąg liczb (a_i), którego użyto do jej zaszyfrowania. Twoje zadanie polega na odkodowaniu zaszyfrowanej wiadomości.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita n ($5 \leqslant n \leqslant 40$). W kolejnych n wierszach jest zapisany ciąg liczb $a_1, a_2, \ldots, a_n$. Suma liczb a_i nie przekracza 2 000 000 000. W kolejnym wierszu zapisana jest jedna liczba całkowita S – zaszyfrowana wiadomość ($0 \leqslant S \leqslant a_1 + a_2 + \ldots + a_n$).

Wyjście

W jedynym wierszu wyjścia należy zapisać kolejne liczby t_i, bez żadnych odstępów między nimi. Dane testowe zostały dobrane tak, że zaszyfrowane wiadomości są określone jednoznacznie.

Przykład

Dla danych wejściowych:

```
24
19226985
123697
67356296
19721773
1113273
69335448
23680077
9029881
85168664
93676782
5253843
77616588
78572630
13375812
17199980
101508862
59248276
3505733
35790095
62028546
85726819
56462819
103373994
91757169
667509506
```

poprawnym wynikiem jest:

```
110001000101101100010101
```

Rozwiązanie

Problem, który mamy rozwiązać, jest szczególnym przypadkiem tzw. problemu pakowania plecaka. W całej ogólności został on bardzo dobrze opisany w książce Macieja M. Sysły *Algorytmy* [13], gdzie w szczególności można znaleźć dość skuteczne algorytmy oparte na metodzie programowania dynamicznego. Tego typu rozwiązanie problemu plecakowego występuje także w opracowaniu zadania *Szatnia* na stronie 256. Wspomniane rozwiązania działają dobrze wtedy, gdy suma liczb $W = a_1+a_2+\ldots+a_n$ jest dość mała (złożoność wynosi $O(nW)$). W naszym przypadku, gdy liczba W może wynosić około $2 \cdot 10^9$, te algorytmy są nieprzydatne.

Zadanie nie ma dobrego rozwiązania. Mianowicie jeśli dane są dowolne liczby całkowite $a_1, a_2, \ldots, a_n$ i liczba całkowita S, to zadanie polegające na znalezieniu ciągu $t_1, t_2, \ldots, t_n$ o wyrazach ze zbioru $\{0, 1\}$, takiego że

$$t_1a_1 + t_2a_2 + \ldots + t_na_n = S,$$

jest problemem NP-zupełnym. Nie możemy więc spodziewać się rozwiązania działającego w czasie wielomianowym. Jedyne znane rozwiązania tego problemu w ogólności działają w czasie wykładniczym. Przyjrzyjmy się zatem takim rozwiązaniom.

Najbardziej narzuca się metoda przeszukania wszystkich ciągów zero-jedynkowych; też trzeba to zrobić zręcznie, by wielokrotnie nie dodawać do siebie tych samych liczb. Taki program działa w czasie $O(2^n)$.

Skuteczniejsza jest metoda przeszukiwania z nawrotami. Polega ona na przeglądaniu wszystkich przypadków, które mogą prowadzić do rozwiązania. Przypuśćmy, że w danym momencie zdecydowaliśmy się wybrać liczby $t_1, t_2, \ldots, t_k$, gdzie $k < n$. Niech

$$Suma = t_1a_1 + t_2a_2 + \ldots + t_ka_k.$$

Mamy teraz następujące możliwości:

- Jeśli $Suma > S$, to dalsze poszukiwania nie mają sensu i musimy cofnąć się do ciągu liczb $t_1, t_2, \ldots, t_{k-1}$.
- Jeśli $Suma + a_{k+1} + \ldots + a_n < S$, to w takim przypadku też możemy zakończyć poszukiwania i cofnąć się do ciągu liczb $t_1, t_2, \ldots, t_{k-1}$.
- Jeśli nie zachodzi żadne z powyższych, to rozważamy dwa przypadki: $t_{k+1} = 0$ oraz $t_{k+1} = 1$.

Tak może wyglądać zapis tego pomysłu w pseudokodzie:

```
1:  function Rek(k, Suma)
2:  begin
3:    if Suma > S or Suma + p_{k+1} < S then return;
4:    else if k = n then
5:      wypisz ciąg t_1, ..., t_n jako rozwiązanie;
6:    else begin
7:      t_{k+1} := 0; Rek(k + 1, Suma);
8:      t_{k+1} := 1; Rek(k + 1, Suma + a_{k+1});
9:    end
10: end
```