Jakub Pawlewicz
Treść zadania, Opracowanie

Jakub Radoszewski
Program

OI, Etap III, dzień pierwszy, 29.03.2006

Kryształy

Bajtocy opracował proces tworzenia kryształów złożonych z określonej liczby atomów różnych pierwiastków. Po latach eksperymentów udało mu się znaleźć formułę, która opisuje, dla jakiej liczby atomów poszczególnych pierwiastków można wytworzyć kryształ składający się z tych atomów. Ciekawi go jak wiele różnych kryształów może otrzymać.

Niech x i y oznaczają nieujemne liczby całkowite. Przez $x \oplus y$ oznaczmy wynik wykonania operacji xor na odpowiadających sobie bitach liczb x i y. Wyniki działania operacji xor na bitach to: $1 \oplus 1 = 0 \oplus 0 = 0$, $1 \oplus 0 = 0 \oplus 1 = 1$.

Na przykład, $44 \oplus 6 = (101100)_2 \oplus (000110)_2 = (101010)_2 = 42$.

Bajtocy zna n różnych pierwiastków, ponumerowanych od 1 do n. Dla każdego pierwiastka i istnieje górne ograniczenie m_i na liczbę atomów tego pierwiastka, jaka może być użyta do stworzenia kryształu. Utworzenie kryształu, który składa się z a_i atomów i-tego pierwiastka (dla $i = 1, \ldots, n$) jest możliwe tylko, gdy:

- $0 \leqslant a_i \leqslant m_i$, *dla* $i = 1, 2, \ldots, n$,
- $a_1 \oplus \ldots \oplus a_n = 0$, *oraz*
- $a_1 + a_2 + \cdots + a_n \geqslant 1$.

Ostatni z powyższych warunków oznacza, że kryształ musi składać się z przynajmniej jednego atomu.

Zadanie

Napisz program, który:

- *Wczyta ze standardowego wejścia liczbę pierwiastków oraz ograniczenia na liczbę atomów danego pierwiastka w krysztale.*
- *Wyliczy liczbę różnych kryształów jakie można otrzymać.*
- *Wypisze wynik na standardowe wyjście.*

Wejście

Pierwszy wiersz standardowego wejścia zawiera liczbę pierwiastków n, $2 \leqslant n \leqslant 50$. W drugim i ostatnim wierszu znajduje się n liczb $m_1, \ldots, m_n$ pooddzielanych pojedynczymi odstępami, $1 \leqslant m_i < 2^{32} - 1$.

Wyjście

Twój program powinien wypisać na standardowe wyjście jedną liczbę — liczbę kryształów, które można otrzymać. Możesz założyć, że liczba ta jest mniejsza od $2^{64} - 1$.

Przykład

Dla danych wejściowych:

```
3
2 1 3
```

poprawnym wynikiem jest:

```
5
```

Możliwe ilości atomów poszczególnych pierwiastków to: $(0,1,1)$, $(1,0,1)$, $(1,1,0)$, $(2,0,2)$, $(2,1,3)$.

Rozwiązanie

XOR

Podstawą rozwiązania zadania jest znajomość własności operacji *różnicy symetrycznej* `xor`, dlatego zanim przejdziemy do poszukiwania rozwiązania, warto własności te przypomnieć. Operację $\oplus$ wykonujemy na bitach, czyli liczbach 0 i 1. Możemy ją zdefiniować jako

$$x \oplus y = x + y \bmod 2,$$

co można przedstawić także w formie tabeli:

$\oplus$	0	1
0	0	1
1	1	0

Następujące własności łatwo wynikają z definicji:

$$x \oplus x = 0 \tag{1}$$
$$x \oplus 0 = x \tag{2}$$
$$x \oplus y = y \oplus x \tag{3}$$
$$x \oplus (y \oplus z) = (x \oplus y) \oplus z \tag{4}$$

Własności (3) i (4) — *przemienność* i *łączność* działania — oznaczają, że wynik operacji dla większej liczby argumentów:

$$x_1 \oplus x_2 \oplus \cdots \oplus x_k$$

nie zależy od tego, w jakiej kolejności ustawimy argumenty i jak rozstawimy nawiasy w wyrażeniu.

Operację $\oplus$ uogólnia się na dowolne nieujemne liczby całkowite — w tym celu liczby zapisujemy *binarnie*, czyli w systemie dwójkowym:

$$x = (b_{k-1} \ldots b_1 b_0)_2 = b_{k-1} \cdot 2^{k-1} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0,$$

gdzie $b_0, \ldots, b_{k-1} \in \{0,1\}$ są cyframi, zwanymi też *bitami*. Bit b_0 nazywamy *najmniej znaczącym*, a bit b_{k-1} — *najbardziej znaczącym*. Dla wygody ponumerujmy jeszcze bity x w następujący sposób: b_0 będzie bitem pierwszym, b_1 bitem drugim, …, b_{k-1} bitem k-tym.

Zauważmy, że za pomocą k bitów możemy zapisać każdą liczbę x spełniającą nierówności $0 \leqslant x \leqslant 2^k - 1$ — zapis taki nazywamy *reprezentacją k-bitową* liczby. Jeżeli dodatkowo $2^{k-1} \leqslant x \leqslant 2^k - 1$, czyli $b_{k-1} = 1$, to x nazywamy *liczbą k-bitową*.

Wynik operacji `xor` dla dwóch liczb definiujemy jako liczbę złożoną z bitów, powstałych przez wykonanie operacji `xor` na odpowiadających sobie bitach tych liczb:

$$(a_{k-1} \ldots a_1 a_0)_2 \oplus (b_{k-1} \ldots b_1 b_0)_2 = \big((a_{k-1} \oplus b_{k-1}) \ldots (a_1 \oplus b_1)(a_0 \oplus b_0)\big)_2.$$

Na przykład:

$$\begin{aligned} 12 \oplus 5 &= (2^3 + 2^2) \oplus (2^2 + 2^0) = (1100)_2 \oplus (0101)_2 \\ &= \big((1 \oplus 0)(1 \oplus 1)(0 \oplus 0)(0 \oplus 1)\big)_2 = (1001)_2 = 2^3 + 2^0 = 9. \end{aligned}$$

Własności (1)-(4) pokazane dla bitów, zachodzą także dla nieujemnych liczb całkowitych.

Przypomnijmy, że naszym zadaniem jest znalezienie dla zadanego ciągu $m_1, \ldots, m_n$ liczby elementów zbioru:

$$K(m_1, \ldots, m_n) = \{(a_1, \ldots, a_n) : a_1 \oplus \cdots \oplus a_n = 0, 0 \leqslant a_i \leqslant m_i \text{ dla } i = 1, \ldots, n\}.$$

Szukana liczba kryształów będzie oczywiście o jeden mniejsza, ponieważ z rozwiązania wykluczamy element $(0, \ldots, 0)$.

Rozwiązanie wzorcowe

Przyjmijmy, że ograniczenia na liczbę atomów każdego z pierwiastków spełniają nierówności: $m_1 \geqslant m_2 \geqslant \ldots \geqslant m_n$. Jeśli tak nie jest, to oczywiście możemy na wstępie te liczby posortować.

Spróbujmy wyrazić w inny sposób równość $a_1 \oplus \cdots \oplus a_n = 0$. Zauważmy, że wynik `xor` dla ciągu bitów jest równy zero wtedy i tylko wtedy, gdy w tym ciągu występuje parzysta liczba jedynek. Można stąd wywnioskować, że:

Fakt 1 *Równość $a_1 \oplus \cdots \oplus a_n = 0$ zachodzi wtedy i tylko wtedy, gdy dla każdego j liczba jedynek w ciągu powstałym z wzięcia j-tych bitów liczb $a_1, \ldots, a_n$ jest parzysta.*

Zanotujmy też inną własność.

Fakt 2 *Równość $a_1 \oplus \cdots \oplus a_n = 0$ zachodzi wtedy i tylko wtedy, gdy $a_1 = a_2 \oplus \cdots \oplus a_n$.*

Dowód Następujące równości są równoważne:

$$\begin{aligned} a_1 \oplus a_2 \oplus \cdots \oplus a_n &= 0, \\ a_1 \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_n &= a_1 \oplus 0, \\ 0 \oplus a_2 \oplus \cdots \oplus a_n &= a_1, \\ a_2 \oplus \cdots \oplus a_n &= a_1. \end{aligned}$$

■

Łatwy przypadek

Fakt 2 oznacza, że jeśli mamy ciąg n argumentów i ustalimy wartości $n-1$ spośród nich, to tylko dla jednej wartości pozostałego argumentu wynik `xor` dla całego ciągu jest równy zero. Ze spostrzeżenia tego może zrodzić się następujący pomysł. Wybierzmy dowolnie ciąg liczb $a_2,\ldots,a_n$ spełniających ograniczenia $0 \leqslant a_i \leqslant m_i$ dla $i = 2,\ldots,n$ i wyznaczmy wartość a_1 ze wzoru z faktu 2. Ponieważ m_1 — ograniczenie na a_1, jest największe (czyli jest większe lub równe od każdej z liczb $a_2,\ldots,a_n$), więc może także $a_1 \leqslant m_1$.

Przy takich optymistycznych założeniach liczba szukanych wyborów $a_1,\ldots,a_n$, takich że $a_1 \oplus \cdots \oplus a_n = 0$, wynosiłaby po prostu $(m_2+1)\cdot\ldots\cdot(m_n+1)$. Niestety, nie zawsze jest tak dobrze. Na przykład, dla $a_2 = 1$ i $a_3 = 2$ oraz $m_1 = m_2 = m_3 = 2$, mamy $a_1 = 1 \oplus 2 = 3$, czyli liczbę większą od 2.

Zastanówmy się jednak, czy potrafimy określić, jakie warunki muszą spełniać wartości $m_1,\ldots,m_n$, żeby dla dowolnie wybranych $a_2 \leqslant m_2,\ldots,a_n \leqslant m_n$, zachodziła nierówność $a_2 \oplus \cdots \oplus a_n \leqslant m_1$? Otóż, jeśli $m_2,\ldots,m_n$ są co najwyżej k-bitowe, to wtedy także $a_2,\ldots,a_n$ są co najwyżej k-bitowe i $a_2 \oplus \cdots \oplus a_n \leqslant 2^k - 1$. Wystarczy więc, że $m_k \geqslant 2^k - 1$, by a_1 mieściło się w zadanym ograniczeniu. Podsumujmy powyższy wywód następującym faktem:

Fakt 3 *Jeśli istnieje k takie, że $m_1 \geqslant 2^k - 1 \geqslant m_2 \geqslant \ldots \geqslant m_n$, to liczba elementów zbioru $K(m_1,\ldots,m_n)$ jest równa $(m_2+1)\cdot\ldots\cdot(m_n+1)$.*

Redukcja

Niestety nie zawsze mamy do czynienia z łatwym przypadkiem. Spróbujmy więc podążać w stronę rozwiązania problemu redukując dany (niełatwy) przypadek do nieco mniejszego. Sposób redukcji będzie raczej nietypowy. Zamiast zmniejszać liczbę elementów ciągu $a_1,\ldots,a_n$ postaramy się go trochę „określić", czyli ustalić wartości wybranych bitów w liczbach $a_1,\ldots,a_n$. Aby było nam wygodniej odwoływać się do poszczególnych bitów tych liczb, wprowadźmy następujące oznaczenia. Załóżmy, że m_1 jest liczbą k-bitową, stąd wszystkie dopuszczalne wartości $a_1,\ldots,a_n$ są liczbami co najwyżej k-bitowymi i dla każdej z nich możemy przedstawić reprezentację k-bitową: $a_i = (b^i_{k-1}\ldots b^i_1 b^i_0)_2$ dla $1 \leqslant i \leqslant n$. Jeden krok redukcji będzie polegał na ustaleniu wartości wszystkich najbardziej znaczących bitów $b^1_{k-1}, b^2_{k-1},\ldots,b^n_{k-1}$. W ten sposób rzeczywiście zredukujemy rozmiar problemu nie zmniejszając co prawda długości zadanego ciągu, ale „skracając" jego elementy.

Załóżmy, że z ograniczeń wynika, iż tylko s początkowych liczb $a_1,\ldots,a_n$ może być k-bitowych, czyli:

$$2^k > m_1,\ldots,m_s \geqslant 2^{k-1} > m_{s+1},\ldots,m_n.$$

Oznacza to, że bity $b^{s+1}_{k-1}, b^{s+2}_{k-1},\ldots,b^n_{k-1}$ muszą być zerami, a wartości $b^1_{k-1}, b^2_{k-1},\ldots,b^s_{k-1}$ możemy wybrać. Oczywiście wybór nie może być całkowicie dowolny — zgodnie z faktem 1 musimy wybrać jedynkę dla parzystej liczby bitów. Niech $I \subseteq \{1,\ldots,s\}$ będzie zbiorem indeksów tych bitów, dla których wybraliśmy wartość 1.

Oznaczmy przez $a'_1,\ldots,a'_n$ pozostałe do ustalenia części elementów $a_1,\ldots,a_n$, czyli $a'_i = (b^i_{k-2}\ldots b^i_0)_2$. Jakie ograniczenia muszą spełniać ich wartości?

- Dla $i > s$ wartością a'_i może być dowolna wartość ze zbioru $\{0,\ldots,m_i\}$, a więc ograniczenie na wartości a'_i nadal wynosi m_i.

- Rozważmy $i \leqslant s$ oraz $i \notin I$, czyli element a_i, dla którego przyjęliśmy $b^i_{k-1} = 0$. Ponieważ jednocześnie $m_i \geqslant 2^{k-1}$, więc pozostałe bity a_i możemy wybierać całkowicie dowolnie — na pewno nie przekroczymy zadanego ograniczenia. Możemy więc przyjąć, że pozostaje nam do wyboru wartość a'_i ze zbioru $\{0,\ldots,2^{k-1}-1\}$.
- Pozostał przypadek $i \leqslant s$ oraz $i \in I$, czyli wartość a_i, dla której przyjęliśmy $b^i_{k-1} = 1$. Aby nie przekroczyć ograniczenia m_i wartość a'_i musimy wybrać ze zbioru $\{0,\ldots,m_i - 2^{k-1}\}$.

Podsumowując, po ustaleniu k-tych bitów w elementach $a_1,\ldots,a_n$ otrzymaliśmy ciąg $a'_1,\ldots,a'_n$, dla którego ograniczenia, oznaczmy je $m'_1,\ldots,m'_n$, wynoszą odpowiednio:

- $m'_i = m_i$ dla $i > s$,
- $m'_i = 2^{k-1} - 1$ dla $i \leqslant s$ i $i \notin I$,
- $m'_i = m_i - 2^{k-1}$ dla $i \leqslant s$ i $i \in I$.

Nowe ograniczenia (a tym samym wybierane elementy) są więc liczbami co najwyżej $(k-1)$-bitowymi.

Plon redukcji

Niestety plon zaproponowanej redukcji może być dość liczny. Z ciągu k-bitowych ograniczeń $m_1,\ldots,m_n$ powstaje tyle podproblemów, na ile sposobów możemy ustawić najbardziej znaczące bity w opisanym wyżej kroku redukcji. Ponieważ liczba różnych podzbiorów zbioru $\{1,\ldots,s\}$ parzystej mocy jest rzędu $O(2^s)$, więc otrzymamy właśnie tyle mniejszych problemów do rozwiązania.

Pesymizm jest jednak przedwczesny, gdyż wygenerowane w trakcie redukcji podproblemy nie są tak zupełnie przypadkowe. Właściwie jak się dobrze przyjrzeć, to można zauważyć, że większość z nich należy do przypadków łatwych. Otóż jeśli nie ustawiliśmy wszystkich bitów równych jeden, czyli istnieje $j \in \{1,\ldots,s\}$ takie że $b^j_{k-1} = 0$, to wtedy $m'_j = 2^{k-1} - 1$. W ciągu ograniczeń $m'_1,\ldots,m'_n$ występuje więc największa liczba $(k-1)$-bitowa — czyli mamy do czynienia z przypadkiem łatwym! Stąd liczbę elementów zbioru $K(m'_1,\ldots,m'_n)$ możemy wyznaczyć na podstawie faktu 3, jako:

$$|K(m'_1,\ldots,m'_n)| = (m'_1+1)\cdot\ldots\cdot(m'_{j-1}+1)\cdot(m'_{j+1}+1)\cdot\ldots\cdot(m'_n+1). \tag{5}$$

Pozostaje tylko jeden przypadek niełatwy — gdy nie istnieje $j \notin I$, czyli przyjęliśmy $b^1_{k-1} = b^2_{k-1} = \cdots = b^s_{k-1} = 1$ (oczywiście może się to zdarzyć tylko wówczas, gdy s jest parzyste). Jest więc *co najwyżej* jeden przypadek, gdy nie będzie można użyć wzoru (5) i trzeba będzie zastosować rekursję.

Została nam ostatnia kwestia, z którą musimy się uporać. Niestety przypadków łatwych jest bardzo dużo, więc choć mamy gotowy wzór, to policzenie sumarycznej liczby kryształów, w których przynajmniej na jednej pozycji i, takiej że $1 \leqslant i \leqslant s$, k-ty bit został ustawiony na zero nie jest zadaniem łatwym. Jest to wręcz najtrudniejsza część rozwiązania i pokażemy cztery metody jej rozwiązania.

Ostatnia kwestia

Metoda 1 — przeglądanie wszystkich podzbiorów

Najprostszą metodą jest przejrzenie wszystkich podzbiorów $I \subseteq \{1,\ldots,s\}$ o parzystej liczbie elementów i różnych od zbioru $\{1,\ldots,s\}$. Dla każdego takiego podzbioru wyznaczamy liczbę kombinacji używając wzoru (5), a następnie sumujemy wyniki częściowe. Podzbiorów takich jest rzędu $O(2^s)$, więc dla dużych s rozwiązanie to jest zupełnie niepraktyczne — przypomnijmy, że s może być równe nawet 50.

Metoda 2 — grupowanie podzbiorów

Możemy uzyskać duże przyspieszenie w stosunku do metody 1, jeśli na wybieranie podzbioru $I = \{i_1, i_2, \ldots\} \subseteq \{1,\ldots,s\}$ spojrzymy raczej jak na wybieranie podciągu $m_{i_1}, m_{i_2}, \ldots$ ciągu $m_1,\ldots,m_s$. Zauważmy, że jeśli elementy $m_1,\ldots,m_s$ nie są różne, to dwa różne podzbiory indeksów $I, I' \subset \{1,\ldots,s\}$ mogą dawać takie same zbiory (z powtórzeniami) wartości $\{m_i \mid i \in I\} = \{m_i \mid i \in I'\}$. Dla takich podzbiorów formuła (5) zwraca taką samą wartość, więc obliczenia wystarczy przeprowadzić raz. Jest to bardzo opłacalne wtedy, gdy wśród liczb $m_1,\ldots,m_s$ często pojawiają się takie same, co jest bardzo prawdopodobne dla małych k, gdyż wiemy, że $2^{k-1} \leqslant m_1,\ldots,m_s < 2^k$.

Wystarczy więc wygenerować wszystkie możliwe podzbiory wartości $\{m_1,\ldots,m_s\}$ i dla każdego z nich wyznaczyć liczbę podzbiorów indeksów $\{1,\ldots,s\}$, z których ten podciąg możemy otrzymać. Oznaczmy przez L_j, $0 \leqslant j < 2^{k-1}$, liczbę wystąpień wartości $2^{k-1}+j$ w ciągu $m_1,\ldots,m_s$. Wtedy jeden podciąg wartości możemy otrzymać przez wybranie dla każdego $0 \leqslant j < 2^{k-1}$ liczby l_j, oznaczającej liczbę wystąpień wartości $2^{k-1}+j$ w tym podciągu. Muszą być przy tym spełnione warunki:

- $0 \leqslant l_j \leqslant L_j$,
- długość podciągu, czyli $\sum_{0 \leqslant j < 2^{k-1}} l_j$, jest liczbą parzystą.

Dla danych l_j liczba podzbiorów indeksów, które dają w efekcie dany podciąg wynosi

$$\prod_{0 \leqslant j < 2^{k-1}} \binom{L_j}{l_j}.$$

Technikę grupowania możemy stosować w dowolnym momencie, nawet dla większych k, gdyż nie musimy rozpatrywać w ogóle tych j, dla których $L_j = 0$.

Metoda ta okazuje się być szybka ze względu na nałożone w zadaniu ograniczenie na liczbę kryształów, co wykazujemy w poniższym lemacie.

Fakt 4 *Liczba różnych podciągów ciągu* $\{m_1,\ldots,m_s\}$ *nie przekracza* $2^{16} = 65536$.

Dowód Oszacujmy od dołu wartość $K(m_1,\ldots,m_n)$. W tym celu policzmy tylko niektóre z możliwych kryształów — rozważmy przypadki, gdy $a_{s+1} = a_{s+2} = \cdots = a_n = 0$ oraz $a_i < 2^{k-1}$ dla $i \leqslant s$. Wtedy możemy dowolnie wybrać wartości a_i dla $1 < i \leqslant s$ oraz wyznaczyć a_1 ze wzoru $a_2 \oplus \cdots \oplus a_s$ (na pewno będzie spełniać ograniczenie $a_1 \leqslant m_1$, bo $m_1 \geqslant 2^{k-1}$). W ten sposób, biorąc pod uwagę tylko niektóre z dopuszczalnych kryształów, już

otrzymaliśmy ich $(2^{k-1})^{s-1}$. Z warunków podanych w zadaniu wiemy, że liczba kryształów jest mniejsza niż 2^{64}, więc muszą zachodzić nierówności

$$2^{(k-1)(s-1)} \leqslant K(m_1,\ldots,m_n) < 2^{64},$$

czyli $(k-1)(s-1) < 64$ i dalej $s-1 < \frac{64}{k-1}$, skąd ostatecznie mamy oszacowanie:

$$s \leqslant \left\lceil \frac{64}{k-1} \right\rceil. \tag{6}$$

Wynika stąd, że dla $k \geqslant 5$ ograniczenie dla s wynosi $s \leqslant \frac{64}{4} = 16$, więc liczba wszystkich podzbiorów $\{1,\ldots,s\}$, a więc i liczba różnych podciągów jest mniejsza niż 2^{16}. To wykazuje prawdziwość faktu dla $k \geqslant 5$.

Przejdźmy teraz do przypadku, gdy $k \leqslant 4$. Liczba wszystkich różnych podciągów ciągu $m_1,\ldots,m_s$ (niekoniecznie parzystej wielkości i rozmiaru mniejszego od s) wynosi

$$P = \prod_{0\leqslant j<2^{k-1}} (L_j+1),$$

gdyż dla każdego j liczbę l_j możemy wybierać spośród liczb $0,\ldots,L_j$. Ponadto wiemy, że $\sum\limits_{0\leqslant j<2^{k-1}} L_j = s$, skąd $\sum\limits_{0\leqslant j<2^{k-1}} (L_j+1) = s+2^{k-1}$. Stosując nierówność Cauchy'ego dla średniej arytmetycznej i geometrycznej mamy:

$$\sqrt[2^{k-1}]{\prod_{0\leqslant j<2^{k-1}} (L_j+1)} \leqslant \frac{\sum\limits_{0\leqslant j<2^{k-1}} (L_j+1)}{2^{k-1}} = \frac{s+2^{k-1}}{2^{k-1}},$$

skąd dostajemy ograniczenie

$$P \leqslant \left(\frac{s+2^{k-1}}{2^{k-1}}\right)^{2^{k-1}}. \tag{7}$$

Stosując ograniczenia (6) i (7) dla $k \leqslant 4$ otrzymamy następujące ograniczenia na liczbę różnych podciągów P. Dla $k=4$ mamy $s \leqslant \lceil \frac{64}{3} \rceil = 22$, więc $P \leqslant \left(\frac{22+8}{8}\right)^8 \approx 39107$. Dla $k=3$ mamy $s \leqslant \lceil \frac{64}{2} \rceil = 32$, więc $P \leqslant \left(\frac{32+4}{4}\right)^4 = 9^4 = 6561$. Dla $k=2$ wiemy, że mamy $s \leqslant 50$, więc $P \leqslant \left(\frac{50+2}{2}\right)^2 = 26^2 = 676$. Podobnie dla $k=1$, $P \leqslant (\frac{50+1}{1})^1 = 51$. ■

Chociaż przedstawione rozwiązanie nie jest optymalne pod względem złożoności (na przykład w porównaniu z przedstawioną dalej metodą 4), jest ono wystarczająco szybkie dla ograniczeń z zadania i zostało użyte w rozwiązaniu wzorcowym.

Metoda 3 — programowanie dynamiczne dla małych k

Problem z dużą liczbą podzbiorów możemy też rozwiązać inaczej. Przy dowodzie faktu 4 uzyskaliśmy ograniczenie (6) na wartość s. Oznacza to, że dla większych k liczba podzbiorów $\{1,\ldots,s\}$ jest na tyle mała, że możemy pozwolić sobie na przejrzenie ich wszystkich. Natomiast dla małych k, dajmy na to dla $k \leqslant 6$, możemy zastosować technikę programowania dynamicznego do wyznaczenia liczności $K(m_1,\ldots,m_n)$.

Przypomnijmy, że liczby $m_1,\ldots,m_n$ są co najwyżej k-bitowe, co między innymi oznacza, że $m_1,\ldots,m_n < 2^k$. Niech $A[t,x]$ dla $0 \leqslant t \leqslant n$ i $0 \leqslant x < 2^k$ oznacza liczbę takich krotek $(a_1,\ldots,a_t)$, że $a_1 \oplus \cdots \oplus a_t = x$ oraz dla $i = 1,\ldots,t$ spełnione są ograniczenia $0 \leqslant a_i \leqslant m_i$. Przy tych oznaczeniach szukaną liczbą jest $A[n,0]$.

Na początku inicjujemy $A[0,0] = 1$ i $A[0,x] = 0$ dla wszystkich $1 \leqslant x < 2^k$. Następnie dla kolejnych $t = 1,\ldots,n$ i każdego $0 \leqslant x < 2^k$ wartość $A[t,x]$ wyliczamy na podstawie wartości $A[t-1,\cdot]$ używając wzoru:

$$A[t,x] = \sum_{a_t=0}^{m_t} A[t-1, x \oplus a_t].$$

Poprawność wzoru wynika stąd, że równość $a_1 \oplus \cdots \oplus a_{t-1} \oplus a_t = x$ jest równoznaczna równości $a_1 \oplus \cdots \oplus a_{t-1} = x \oplus a_t$, a więc liczba wszystkich takich krotek $(a_1,\ldots,a_t)$, że $a_1 \oplus \cdots \oplus a_t = x$ jest równa sumie, po wszystkich możliwych wyborach a_t, liczby takich krotek $(a_1,\ldots,a_{t-1})$, że $a_1 \oplus \cdots \oplus a_{t-1} = x \oplus a_t$.

Czas liczenia $A[t,x]$ dla danych t i x wynosi $O(m_t) \leqslant O(2^k)$, więc całkowita złożoność czasowa tego rozwiązania jest równa $O(n \cdot 2^k \cdot 2^k) = O(n \cdot 2^{2k})$. Zatem dla $k \leqslant 6$ jest ono wystarczająco szybkie.

Metoda 4 — pełne programowanie dynamiczne

Przypomnijmy pierwotne sformułowanie problemu, z którym teraz staramy się uporać. Mamy dane ograniczenia — co najwyżej k-bitowe liczby $m_1,\ldots,m_n$. Przez s oznaczyliśmy największy indeks ograniczenia dokładnie k-bitowego, czyli $m_1 \geqslant \cdots \geqslant m_s > 2^{k-1} - 1 \geqslant m_{s+1} \geqslant \cdots \geqslant m_n$. Poszukujemy krotek $(a_1,a_2,\ldots,a_n)$ spełniających ograniczenia $0 \leqslant a_i \leqslant m_i$ dla $1 \leqslant i \leqslant n$, dla których $a_1 \oplus \cdots \oplus a_n = 0$ oraz nie wszystkie $a_1,\ldots,a_s$ są jednocześnie większe niż $2^k - 1$. W tym celu ustalamy wartości najbardziej znaczących bitów $a_1,\ldots,a_n$. Dla $a_{s+1},a_{s+2},\ldots,a_n$ z racji ograniczeń musimy przyjąć $b_{k-1}^{s+1} = b_{k-1}^{s+2} = \cdots = b_{k-1}^{n} = 0$ i pozostaje nam znaleźć liczbę układów pozostałych bitów. Wiemy przy tym, że suma bitów $b_{k-1}^1,\ldots,b_{k-1}^s$ musi być parzysta i wszystkie te bity nie mogą być równocześnie jedynkami.

Oznaczmy przez $P[t]$ dla $t = 1,\ldots,s+1$ liczby takich ciągów $(a_t,\ldots,a_n)$ spełniających ograniczenia $(m_t,\ldots,m_n)$, dla których zachodzą warunki:

(i) istnieje $t \leqslant i \leqslant s$, taki że $b_{k-1}^i = 0$ oraz

(ii) liczba jedynek wśród bitów $b_{k-1}^t, b_{k-1}^{t+1},\ldots,b_{k-1}^s$ jest parzysta i

(iii) $a_t \oplus \cdots \oplus a_n = 0$.

Analogicznie zdefiniujemy wartości $N[t]$ dla $t = 1,\ldots,s+1$, jako liczbę ciągów $(a_t,\ldots,a_n)$ spełniających ograniczenia $(m_t,\ldots,m_n)$, dla których zachodzą warunki:

(i) istnieje $t \leqslant i \leqslant s$, takie że $b_{k-1}^i = 0$ oraz

(ii') liczba jedynek wśród bitów $b_{k-1}^t, b_{k-1}^{t+1},\ldots,b_{k-1}^s$ jest nieparzysta i

(iii') $a_t \oplus \cdots \oplus a_n = 2^{k-1}$.

Dla tak określonych tablic P i N szukaną przez nas wartością jest $P[1]$. Aby sprawnie policzyć wartości P i N udowodnimy pomocniczy

Lemat 5 *Niech* $1 \leqslant t \leqslant s+1$ *oraz* $0 \leqslant x < 2^{k-1}$. *Liczba takich ciągów* $(a_t, \ldots, a_n)$, *dla których zachodzą ograniczenia* $(m_t, \ldots, m_n)$ *oraz warunki (i), (ii) i* $a_t \oplus \cdots \oplus a_n = x$ *wynosi* $P[t]$. *Z kolei liczba krotek* $(a_t, \ldots, a_n)$, *dla których zachodzą ograniczenia* $(m_t, \ldots, m_n)$ *oraz warunki (i), (ii') i* $a_t \oplus \cdots \oplus a_n = 2^{k-1} + x$ *wynosi* $N[t]$.

Dowód Pokażemy dowód pierwszej części lematu dotyczącej $P[t]$ — dowód dla $N[t]$ jest całkowicie analogiczny. Układy, które mamy zliczyć różnią się od tych z definicji $P[t]$ tylko warunkiem $a_t \oplus \cdots \oplus a_n = x$. Pokażemy, że każdą krotkę spełniającą warunki definicji $P[t]$ można przekształcić w układ spełniający warunki pierwszej części lematu. Ponadto przekształcając różne krotki, dostaniemy różne układy wynikowe.

Niech $(a_t, \ldots, a_n)$ spełnia ograniczenia $(m_t, \ldots, m_n)$ oraz warunki (i), (ii) i (iii). Niech i będzie indeksem, dla którego $b^i_{k-1} = 0$ (jeśli jest takich kilka, to wybieramy najmniejszy z nich). Wiadomo, że znajdziemy $i \leqslant s$, dla którego zachodzi ten warunek. Rozważmy ciąg $(a_t, \ldots, a_{i-1}, a_i \oplus x, a_{i+1}, \ldots, a_n)$. Spełnia on wszystkie warunki z definicji $P[t]$:

(i) bit $b^i_{k-1} = 0$,

(ii) suma bitów $b^t_{k-1}, \ldots, b^s_{k-1}$ nie uległa zmianie, gdyż x jest liczbą $(k-1)$-bitową,

(iii) $a_t \oplus \cdots \oplus a_{i-1} \oplus (a_i \oplus x) \oplus a_{i+1} \oplus \cdots \oplus a_n = (a_t \oplus \cdots \oplus a_n) \oplus x = x$.

Co równie ważne, wartość $a_i \oplus x$ nie przekracza ograniczenia m_i, ponieważ wybraliśmy indeks i, dla którego $b^i_{k-1} = 0$, więc $a_i < 2^{k-1}$ (wtedy także $a_i \oplus x < 2^{k-1}$), a ograniczenie m_i jest równe co najmniej 2^{k-1}.

Pozostaje już tylko zauważyć, że dla dwóch różnych ciągów $A = (a_t, \ldots, a_n)$ i $C = (c_t, \ldots, c_n)$ transformacja daje dwa różne ciągi.

- Jeśli ciągi różnią się którymś z k-tych bitów, to po transformacji nie mogą stać się równe, bo operacja nie wpływa na k-te bity elementów ciągów.
- Jeśli A i C mają identyczne k-te bity, to w trakcie transformacji w obu ciągach wybraliśmy ten sam i-ty element, który poddaliśmy operacji `xor` z x. Niech j będzie indeksem, dla którego $a_j \neq c_j$ na $k-1$ mniej znaczących bitach.
  - Jeśli $i = j$, czyli $a_i \neq c_i$, to także $a_i \oplus x \neq c_i \oplus x$.
  - W przeciwnym razie elementy a_j oraz c_j nie są zmieniane w trakcie transformacji, więc pozostają różne.

Z powyższego rozumowania wynika, że układów spełniających warunki z pierwszej części lematu jest nie mniej, niż układów spełniających warunki z definicji $P[t]$. Zdefiniowanie analogicznej transformacji w drugą stronę pozwala z kolei wykazać, że $P[t]$ jest nie mniejsze niż liczba układów spełniających lemat, co dowodzi tę część twierdzenia. ■

Możemy teraz pokazać, jak wyznaczyć wartości $P[t]$ i $N[t]$ kolejno dla $t = s+1, s, \ldots, 1$. Dla $t = s+1$ mamy $P[t] = N[t] = 0$, gdyż nie możemy spełnić warunku (i). Rozważmy $P[t]$ dla $t \leqslant s$:

1. Załóżmy, że ustawiamy $b^t_{k-1} = 1$. Ponieważ żądamy, aby liczba bitów ustawionych na jeden wśród $b^t_{k-1}, b^{t+1}_{k-1}, \ldots, b^s_{k-1}$ była parzysta, więc tym samym liczba bitów ustawionych na jeden wśród $b^{t+1}_{k-1}, \ldots, b^s_{k-1}$ musi być nieparzysta. Dla każdego a_t spełniającego ograniczenia $2^{k-1} \leqslant a_t \leqslant m_t$ chcemy wyznaczyć liczbę takich krotek $(a_{t+1}, \ldots, a_n)$, że $a_t \oplus a_{t+1} \oplus \cdots \oplus a_n = 0$, czyli że $a_{t+1} \oplus \cdots \oplus a_n = a_t$. Z lematu, przyjmując $x = a_t$, wiemy że jest ich $N[t+1]$. Ponieważ wartość a_t możemy wybrać na $m_t - 2^{k-1} + 1$ sposobów, więc w tym przypadku otrzymujemy $(m_t - 2^{k-1} + 1) \cdot N[t+1]$ krotek spełniających warunki definicji $P[t]$

2. Załóżmy, że ustawiamy $b^t_{k-1} = 0$. Tym razem chcemy, by liczba bitów ustawionych na jeden wśród $b^{t+1}_{k-1}, \ldots, b^s_{k-1}$ była parzysta. Podobnie jak w poprzednim przypadku zauważamy, że dla każdego $0 \leqslant a_t < 2^{k-1}$ liczba takich krotek spełniających równość $a_t \oplus \cdots \oplus a_n = 0$, wynosi $P[t+1]$. Tym razem a_t możemy wybrać na 2^{k-1} sposobów, więc otrzymujemy $2^{k-1} \cdot P[t+1]$ krotek.

   W tym przypadku istnieje jeszcze inna możliwość utworzenia krotek spełniających warunki definicji $P[t]$. Ponieważ mamy już jeden bit ustawiony na zero ($b^t_{k-1} = 0$), możemy dopuścić przypadek, gdy wszystkie pozostałe bity $b^{t+1}_{k-1}, \ldots, b^s_{k-1}$ są równe jeden. Takie układy niestety nie są uwzględnione w definicji $P[t+1]$. Wyznaczmy liczbę takich krotek.

   - Jeśli $s-t$ jest nieparzyste, to nie można spełnić warunku (i) definicji $P[t]$ i liczba szukanych krotek wynosi zero.
   - Niech $s-t$ będzie liczbą parzystą. Ustawiliśmy $b^t_{k-1} = 0$ oraz $b^{t+1}_{k-1} = \ldots b^s_{k-1} = 1$ (bity $b^{s+1}_{k-1}, \ldots, b^n_{k-1}$ są oczywiście zerami z racji ograniczeń $(m_{s+1}, \ldots, m_n)$). Wybierzmy pozostałe bity $a_{t+1}, \ldots, a_n$ i wyznaczmy wartość $a_t = a_{t+1} \oplus \cdots \oplus a_n$ — na pewno spełnia ona ograniczenie $m_t \geqslant 2^k$, gdyż liczba jedynek wśród k-tych bitów $a_{t+1}, \ldots, a_n$ jest parzysta. Pozostaje nam obliczyć, na ile sposobów możemy wybrać wartości $a_{t+1}, \ldots, a_n$ po poczynionych ustaleniach. Dla $i > s$ wartość a_i wybieramy na $m_i + 1$ sposobów. Natomiast dla $t+1 \leqslant i \leqslant s$ — na $m_i - 2^{k-1} + 1$ sposobów. Zatem łączna liczba krotek rozważanego typu wynosi $(m_{t+1} - 2^{k-1} + 1) \cdot \ldots \cdot (m_s - 2^{k-1} + 1) \cdot (m_{s+1} + 1) \cdot \ldots \cdot (m_n + 1)$.

Analogiczne rozważania można przeprowadzić dla $N[t]$. W rezultacie otrzymujemy wzory:

$$P[t] = (m_t - 2^{k-1} + 1) \cdot N[t+1] + 2^{k-1} \cdot P[t+1] + \begin{cases} J[t+1] & s-t \text{ jest parzyste} \\ 0 & s-t \text{ jest nieparzyste} \end{cases}$$

$$N[t] = (m_t - 2^{k-1} + 1) \cdot P[t+1] + 2^{k-1} \cdot N[t+1] + \begin{cases} 0 & s-t \text{ jest parzyste} \\ J[t+1] & s-t \text{ jest nieparzyste} \end{cases}$$

gdzie

$$J[t+1] = (m_{t+1} - 2^{k-1} + 1) \cdot \ldots \cdot (m_s - 2^{k-1} + 1) \cdot (m_{s+1} + 1) \cdot \ldots \cdot (m_n + 1).$$

Tablicę J wyliczamy także stosując metodę programowania dynamicznego. Inicjujemy $J[s+1] = (m_{s+1} + 1) \cdot \ldots \cdot (m_n + 1)$, a następnie dla $1 \leqslant t \leqslant s$ przyjmujemy $J[t] = (m_t - 2^{k-1} + 1) \cdot J[t+1]$.

Przedstawiona metoda jest bardzo szybka — ma złożoność czasową $O(n)$.

Testy

Na zawody zostało przygotowanych 27 testów, za pomocą których zostało utworzonych 20 zestawów testów.

Nazwa	n	Opis
kry1a.in	6	test poprawnościowy
kry1b.in	2	najmniejszy test — dwie jedynki
kry2a.in	7	test poprawnościowy
kry2b.in	4	test poprawnościowy
kry3a.in	7	nieduże liczby (do 20)
kry3b.in	4	test poprawnościowy
kry4.in	10	nieduże liczby (do 20)
kry5.in	13	nieduże liczby (do 20)
kry6.in	20	kilka średnich liczb (ok. 1 000)
kry7.in	25	nieduże liczby (do 25)
kry8.in	10	3 duże liczby (ok. 1 000 000)
kry9.in	3	3 bardzo duże liczby (ok. 2^{32})
kry10.in	25	3 średnie liczby (ok. 20 000) i reszta dwójek
kry11a.in	11	3 bardzo duże liczby (ok. 10^8), reszta dwójki i jedynki
kry11b.in	28	same czwórki, piątki i szóstki
kry12a.in	9	4 duże liczby (od 10^5 do 10^6), 10 i reszta dwójki i jedynki
kry12b.in	34	kilka niedużych liczb (do 20) i reszta 1, 2 i 3
kry13.in	43	małe liczby (do 5)
kry14.in	28	dwie liczby ok. 100 000, dwie ok. 1 000, reszta jedynki
kry15a.in	9	kilka dużych, kilka małych liczb
kry15b.in	30	cztery liczby ok. 4 000, reszta jedynki i kilka dwójek
kry16a.in	5	5 liczb, w tym 3 bardzo duże (ok. 10^8–10^9)
kry16b.in	50	liczby nie większe od 4
kry17.in	28	nieznacznie zmodyfikowany test 14
kry18.in	28	dwie liczby ok. 10^6, dwie ok. 10^3 i reszta jedynki
kry19.in	28	dwie liczby ok. 2^{20}, dwie ok. 2^9 i reszta jedynki
kry20.in	28	dwa razy po 10^6, dwa razy po $2^9 - 1$ i reszta jedynki