Krzysztof Diks, Wojciech Rytter
Treść zadania, Opracowanie

Jacek Migdał
Program

Dostępna pamięć: 64 MB.

OI, Etap I, 18.10–15.11.2010

Przekładanka

Bajtazar kupił synkowi Bajtkowi zestaw klocków ponumerowanych od 1 do n i ustawił je w rzędzie w pewnej kolejności. Zadaniem Bajtka jest ustawienie klocków w kolejności numerów tych klocków, od najmniejszego do największego. Jedyne ruchy, jakie może wykonywać Bajtek, to:

- *przełożenie ostatniego klocka na początek (ruch typu* `a`*), oraz*
- *przełożenie trzeciego klocka na początek (ruch typu* `b`*).*

Pomóż Bajtkowi i napisz program, który sprawdzi, czy dany układ klocków da się w ogóle ustawić w żądanej kolejności, a jeżeli tak, to poda, jak to zrobić.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n, $1 \leqslant n \leqslant 2\,000$. W drugim wierszu znajduje się n liczb całkowitych z zakresu od 1 do n pooddzielanych pojedynczymi odstępami. Liczby te nie powtarzają się i reprezentują początkowe ustawienie klocków.

Wyjście

Jeśli nie istnieje sekwencja ruchów prowadząca do ustawienia klocków w porządku rosnącym numerów, Twój program powinien wypisać na standardowe wyjście „`NIE DA SIE`*" (bez cudzysłowów).*

W przeciwnym przypadku, w pierwszym wierszu wyjścia powinna znaleźć się jedna liczba całkowita m ($m \leqslant n^2$), oznaczająca liczbę wykonywanych operacji. Przez **operację** *rozumiemy k-krotne wykonanie jednego z ruchów* `a` *lub* `b`*.*

Jeżeli $m > 0$, to w drugim wierszu powinien znaleźć się ciąg m liczb całkowitych z dodanymi pojedynczymi znakami `a` *lub* `b`*. Zapis postaci k*`a` *(dla $0 < k < n$) oznacza k-krotne wykonanie ruchu typu* `a`*. Zapis postaci k*`b` *(dla $0 < k < n$) oznacza k-krotne wykonanie ruchu typu* `b`*.*

Dodatkowo, znaki stowarzyszone z liczbami znajdującymi się w drugim wierszu muszą być ułożone na przemian.

Jeśli istnieje więcej niż jedno rozwiązanie, Twój program może wypisać dowolne z nich.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`4`	`4`
`1 3 2 4`	`3a 2b 2a 2b`

natomiast dla danych:

```
7
1 3 2 4 5 6 7
```

poprawnym wynikiem jest:

```
NIE DA SIE
```

a dla danych:

```
3
1 2 3
```

poprawnym wynikiem jest:

```
0
```

Rozwiązanie

W tym zadaniu tak naprawdę chodzi o posortowanie permutacji liczb $1, 2, \ldots, n$ poprzez wykonanie ciągu operacji dwóch typów: (a) przestawienie ostatniego elementu ciągu na początek, (b) przestawienie trzeciego elementu ciągu na początek.

Czy zawsze jest to możliwe? Żeby odpowiedzieć na to pytanie, potrzebujemy jednej prostej definicji. Parę (a_i, a_j) elementów ciągu $\langle a_1, a_2, \ldots, a_n\rangle$ nazwiemy *inwersją*, jeśli tylko $i < j$ oraz $a_i > a_j$. Dla przykładu, ciąg $\langle 4,\ 5,\ 1,\ 2,\ 3\rangle$ zawiera 6 inwersji. Jeśli ciąg jest uporządkowany rosnąco, liczba jego inwersji wynosi 0.

Zauważmy, że operacja typu (a) zmienia liczbę inwersji w ciągu o nieparzystą liczbę, gdy n jest parzyste, i o parzystą liczbę, gdy n jest nieparzyste. Operacja typu (b) zawsze zmienia liczbę inwersji o parzystą liczbę. Dowody tych prostych obserwacji pozostawiamy Czytelnikowi.

Wynika stąd, że stosując tylko operacje typu (a) i (b), nie da się posortować ciągu długości nieparzystej mającego nieparzystą liczbę inwersji. Przykładowo, nie da się posortować żadnego z ciągów:

$$\langle 1,\ 3,\ 2\rangle,\quad \langle 2,\ 1,\ 3\rangle,\quad \langle 3,\ 2,\ 1\rangle,\quad \langle 2,\ 4,\ 3,\ 5,\ 1\rangle.$$

Pokażemy teraz, że w każdym innym przypadku sortowanie jest możliwe. Ponieważ dla $n \leqslant 3$ rozwiązanie zadania uzyskujemy „na palcach", więc załóżmy, że $n \geqslant 4$.

Najpierw staramy się umieścić $n-2$ najmniejsze elementy na ich końcowych pozycjach. W tym celu będziemy kolejno przesuwać w lewo jedynkę, potem dwójkę, następnie trójkę, itd.

Przesunięcie elementu x o jedną lub dwie pozycje w lewo za pomocą operacji typu (a) i (b) można wykonać następująco:

(OP1) $\langle P,\ u,\ \mathbf{x},\ v,\ S\rangle$ przekształcamy na $\langle P,\ \mathbf{x},\ v,\ u,\ S\rangle$ za pomocą ciągu operacji „$(|S|+3)$`a 2b` $(|P|)$`a`",

(OP2) $\langle P,\ u,\ v,\ \mathbf{x},\ S\rangle$ przekształcamy na $\langle P,\ \mathbf{x},\ u,\ v,\ S\rangle$ za pomocą ciągu operacji „$(|S|+3)$`a b` $(|P|)$`a`",

przy czym P, S to odpowiednio początkowy i końcowy fragment ciągu (prefiks i sufiks), $|P|$ i $|S|$ — długości tych fragmentów, a u, v, x to pojedyncze elementy ciągu.

Jeśli po wykonaniu powyższego otrzymamy ciąg posortowany, to dobrze. W przeciwnym razie pozostał nam do posortowania ciąg $\langle 1, 2, \ldots, n-2, n, n-1\rangle$. Jeśli n jest nieparzyste, to wiemy już, że tego nie da się zrobić — liczba inwersji w wyjściowym ciągu była nieparzysta. A jeśli n jest parzyste, to sortujemy następująco:

(OP3) „`1a 2b` $((n-2)$`a 2b`$)^{(n-4)/2}$ $(n-4)$`a`".

W powyższym zapisie potęgowanie oznacza powtórzenie napisu-podstawy tyle razy, ile wynosi wykładnik. Warto zauważyć, że dla $n = 4$ ostatnia część ma postać `0a`, a zatem, zgodnie z treścią zadania, nie należało jej wypisywać.

Szczególną rolę w naszym algorytmie odgrywają permutacje

$$l_n = \langle 1,\ 2,\ 3,\ \ldots,\ n-2,\ n,\ n-1 \rangle, \quad id_n = \langle 1,\ 2,\ 3,\ \ldots,\ n \rangle.$$

Na przykład

$$l_4 = \langle 1,\ 2,\ 4,\ 3 \rangle, \quad id_4 = \langle 1,\ 2,\ 3,\ 4 \rangle.$$

Używając tych permutacji do uproszczenia zapisu, otrzymujemy następujący pseudokod algorytmu sortowania:

```
1: Algorytm Przekładanka
2: Wczytaj permutację p;
3: for x := 1 to n − 2 do begin
4:    przesuń x w lewo na x-te miejsce, korzystając z operacji OP1 lub OP2
5:       (preferując OP2);
6:    if p = id_n then return;   { koniec }
7:    else if n parzyste then   { wiemy, że p = l_n }
8:       przekształć p na id_n, korzystając z operacji OP3
9:    else wypisz „NIE DA SIE”;
10: end
```

Na koniec pozostawiamy Czytelnikowi sprawdzenie, że powyższy algorytm sortuje dowolną permutację za pomocą co najwyżej n^2 operacji.

Implementacje rozwiązania wzorcowego można znaleźć w plikach `prz.c`, `prz1.pas`, `prz2.cpp` i `prz3.cpp`.

Testy

Rozwiązania zawodników były sprawdzane na 8 grupach testów, z których każda składała się z co najmniej trzech pojedynczych testów.

Nazwa	n	Opis
prz1a.in	1	warunek brzegowy
prz1b.in	4	warunek brzegowy
prz1c.in	15	warunek brzegowy
prz1d.in	2	klocki posortowane
prz1e.in	2	klocki odwrotnie posortowane
prz1f.in	13	nie da się uporządkować
prz2a.in	5	prosty test poprawnościowy
prz2b.in	8	prosty test poprawnościowy
prz2c.in	7	prosty test poprawnościowy, nie da się uporządkować

Nazwa	n	Opis
prz3a.in	100	losowy średni test
prz3b.in	99	losowy średni test, nie da się uporządkować
prz3c.in	98	losowy średni test
prz4a.in	200	losowy średni test
prz4b.in	199	losowy średni test
prz4c.in	198	losowy średni test
prz5a.in	500	losowy średni test
prz5b.in	499	losowy średni test
prz5c.in	498	losowy średni test
prz6a.in	1 600	losowy duży test
prz6b.in	1 599	losowy duży test
prz6c.in	1 598	losowy duży test
prz7a.in	1 800	losowy duży test
prz7b.in	1 799	losowy duży test
prz7c.in	1 798	losowy duży test
prz8a.in	2 000	losowy duży test
prz8b.in	1 999	losowy duży test
prz8c.in	2 000	losowy duży test
prz8d.in	1 989	losowy duży test
prz8e.in	1 999	losowy duży test, nie da się uporządkować