XIII Międzynarodowa Olimpiada Informatyczna, Tampere 2001

XIII Międzynarodowa Olimpiada Informatyczna — treści zadań

Depot

Pewna fińska firma dysponuje wielkim prostokątnym magazynem. Boki magazynu są nazwane kolejno (idąc po obwodzie): lewy, górny, prawy i dolny. Wyodrębnienie w magazynie wierszy i kolumn podzieliło go na kwadraty o identycznych wymiarach. Wiersze są ponumerowane od góry liczbami 1, 2, ... Podobnie, kolumny są ponumerowane od lewej liczbami $1, 2, \ldots$.

W magazynie przechowywane są kontenery zawierające bezwartościowe rupiecie. Kontenery mają różne numery identyfikacyjne. Każdy kontener zajmuje jeden kwadrat. Magazyn jest tak duży, że liczba kontenerów, które kiedykolwiek mogą znaleźć się w magazynie, jest mniejsza zarówno od liczby wierszy, jak i liczby kolumn. Kontenery nie są nigdy usuwane z magazynu, ale czasami do magazynu są przysyłane nowe. Wejście do magazynu znajduje się w jego górnym lewym rogu.

Magazynier rozmieścił kontenery w okolicach górnego lewego rogu w taki sposób, że potrafi je odnajdywać na podstawie ich numerów identyfikacyjnych. W tym celu stosuje następującą metodę.

Przypuśćmy, że numerem identyfikacyjnym nowo wstawianego kontenera jest k *(w skrócie kontenera* k*). Magazynier przegląda pierwszy wiersz, rozpoczynając od lewej strony, i szuka pierwszego kontenera o numerze identyfikacyjnym większym od* k*. Jeśli takiego kontenera nie ma, kontener* k *jest umieszczany bezpośrednio za ostatnim kontenerem w tym wierszu. Jeśli taki kontener* l *został znaleziony, to w miejsce kontenera* l *wstawiany jest kontener* k*, a kontener* l *jest umieszczany w następnym wierszu za pomocą tej samej metody. Jeśli magazynier dojdzie do wiersza bez kontenerów, to nowy kontener jest umieszczany w pierwszym (z lewej strony) kwadracie tego wiersza.*

Załóżmy, że do magazynu trafiły kontenery 3,4,9,2,5,1 w tej właśnie kolejności. Wówczas rozmieszczenie tych kontenerów w magazynie jest następujące:

```
1 4 5
2 9
3
```

Do magazynu przyszedł Pan Kierownik i przeprowadził z magazynierem następującą rozmowę:

Kierownik: *Czy kontener 5 przyszedł przed kontenerem 4?*

Magazynier: *Ależ Panie Kierowniku, to jest niemożliwe!*

Kierownik: *Ach, jesteś w stanie określić kolejność napływania kontenerów, znając tylko ich rozmieszczenie w magazynie!*

Magazynier: *Eeee, no tak ogólnie, to nie. Na przykład, kontenery, które aktualnie znajdują się w magazynie mogły przyjść w kolejności 3,2,1,4,9,5, lub w kolejności 3,2,1,9,4,5, lub w jednej z 14 innych kolejności.*

Pan Kierownik nie chciał wyjść na głupka, więc poszedł sobie. Pomóż Panu Kierownikowi i napisz program, który mając dane rozmieszczenie kontenerów w magazynie obliczy wszystkie możliwe kolejności, w których mogły one przybyć do magazynu.

WEJŚCIE

Plik wejściowy nazywa się `depot.in`*. Pierwszy wiersz zawiera jedną liczbę całkowitą* R *— liczbę wierszy zawierających co najmniej po jednym kontenerze. Kolejne* R *wierszy w pliku wejściowym zawiera informacje o wierszach* $1, \ldots, R$*, poczynając od góry. Na początku każdego z tych wierszy jest zapisana liczba całkowita* M *— liczba kontenerów w tym wierszu. Następnie zapisanych jest* M *liczb całkowitych — numery identyfikacyjne kontenerów umieszczonych w tym wierszu, poczynając od lewej. Wszystkie numery identyfikacyjne* I *spełniają nierówności:* $1 \leq I \leq 50$*. Dla liczby kontenerów* N *w magazynie zachodzi* $1 \leq N \leq 13$.

WYJŚCIE

Plik wyjściowy nazywa się `depot.out`*. Liczba wierszy w pliku wyjściowym jest równa liczbie możliwych kolejności przybycia kontenerów do magazynu. Każdy z tych wierszy zawiera* N *liczb całkowitych — numery identyfikacyjne kontenerów w potencjalnej kolejności ich przybycia do magazynu. Różne wiersze powinny opisywać różne kolejności.*

PRZYKŁAD

Przykład 1:

depot.in	depot.out
3	3 2 1 4 9 5
3 1 4 5	3 2 1 9 4 5
2 2 9	
1 3	

Przykład 2:

depot.in	depot.out
2	3 1 2
2 1 2	1 3 2
1 3	

PUNKTACJA

Jeśli plik wyjściowy zawiera niepoprawną kolejność, lub nie zawiera żadnych kolejności, to Twój program dostanie 0 punktów za taki test. W przeciwnym razie punkty za test są przyznawane w następujący sposób. Jeśli plik wyjściowy zawiera wszystkie możliwe kolejności bez powtórzeń, to Twój program dostanie 4 punkty. Jeśli plik wyjściowy zawiera co najmniej połowę możliwych kolejności i każdą z nich dokładnie raz, to Twój program dostanie 2 punkty. Jeśli plik wyjściowy zawiera mniej niż połowę możliwych kolejności, lub niektóre z nich się powtarzają, Twój program dostanie 1 punkt.

Double Crypt

Rozwinięty Standard Szyfrowania (z angielskiego AES) korzysta z nowego algorytmu szyfrowania. Używa się w nim trzech bloków, każdy złożony ze 128 bitów. Dla danego bloku komunikatu p (tekst jawny) i bloku klucza k, funkcja szyfrująca E w standardzie AES oblicza zaszyfrowany blok c (tekst zaszyfrowany):

$$c = E(p, k).$$

Funkcją odwrotną dla funkcji szyfrującej E jest funkcja deszyfrująca D taka, że:

$$D(E(p, k), k) = p,\ E(D(c, k), k) = c.$$

W Podwójnym Szyfrowaniu AES używa się kolejno dwóch niezależnych bloków kluczy k1 i k2:

$$c_2 = E(E(p, k_1), k_2)$$

.

W tym zadaniu dana jest także liczba całkowita s. We wszystkich kluczach tylko $4 \cdot s$ skrajnie lewych bitów ma znaczenie, podczas gdy wszystkie pozostałe bity (skrajnie prawe $128 - 4 \cdot s$) są równe 0.

Twoim zadaniem jest odtworzenie pary kluczy szyfrujących dla pewnych komunikatów zaszyfrowanych za pomocą Podwójnego AES. Znasz zarówno tekst jawny p, odpowiadający mu podwójnie zaszyfrowany tekst c2, jak i strukturę kluczy szyfrujących zadaną przez liczbę całkowitą s.

Algorytmy szyfrowania i deszyfrowania w systemie AES są dostępne w bibliotece. Do oceny przedkładasz odtworzone klucze, a nie program do ich odtwarzania.

WEJŚCIE

Zestawy danych dla tego zadania są zapisane w plikach tekstowych o nazwach double1.in — double10.in. Każdy plik wejściowy składa się z trzech wierszy. Pierwszy wiersz zawiera liczbę całkowita s, w drugim wierszu znajduje się blok jawnego tekstu p, natomiast trzeci wiersz zawiera blok tekstu zaszyfrowanego c2 otrzymany za pomocą Podwójnego Szyfrowania AES. Oba bloki są zapisane w postaci 32-znakowych słów szesnastkowych (o cyfrach `0,..., 9, A, ..., F`*). Biblioteka zawiera procedurę konwersji słów na bloki. Dla każdego zestawu danych wejściowych istnieje rozwiązanie.*

WYJŚCIE

Do oceny przedstawiasz 10 plików wyjściowych odpowiadających danym plikom wejściowym. Każdy plik wyjściowy składa się z trzech wierszy. Pierwszy wiersz zawiera tekst:

`#FILE double` *I*

gdzie I jest numerem odpowiedniego pliku wejściowego. Drugi wiersz zawiera blok klucza k1, a trzeci wiersz blok klucza k2, takie że: $c2 = E(E(p, k1), k2)$. Oba bloki muszą być zapisane jako 32-znakowe słowa szesnastkowe (o cyfrach `0,..., 9, A, ..., F`*). Biblioteka zawiera procedurę do konwersji bloków na słowa. Jeśli istnieje wiele rozwiązań wystarczy, że przedstawisz tylko jedno z nich.*

PRZYKŁAD

Plik przedstawiony w tym przykładzie ma numer 0.

`double0.in`	*Możliwy plik wyjściowy*
`1`	`#FILE double 0`
`00112233445566778899AABBCCDDEEFF`	`A0000000000000000000000000000000`
`6323B4A5BC16C479ED6D94F5B58FF0C2`	`70000000000000000000000000000000`

BIBLIOTEKA

Biblioteka dla FreePascala (Linux: `aeslibp.p`*,* `aeslibp.ppu`*,* `aeslibp.o`*; Windows:* `aeslibp.p`*,* `aeslibp.ppw`*,* `aeslibp.ow*`*):*

```
type

  HexStr = String [ 32 ]; { only '0'..'9', 'A'..'F' }

  Block  = array [ 0..15 ] of Byte; { 128 bits }


procedure HexStrToBlock ( const hs: HexStr; var b: Block );

procedure BlockToHexStr ( const b: Block; var hs: HexStr );

procedure Encrypt ( const p, k: Block; var c: Block );

  { c = E(p,k) }

procedure Decrypt ( const c, k: Block; var p: Block );

  { p = D(c,k) }
```

Do dyspozycji masz program `aestoolp.pas`*, który pokazuje jak korzystać z takiej biblioteki.*
Biblioteka dla GNU C/C++ (Linux i Windows: `aeslibc.h`*,* `aeslibc.o*`*):*

```
typedef char HexStr[33]; /* '0'..'9', 'A'..'F', '\0'-terminated */

typedef unsigned char Block[16];  /* 128 bits */


void hexstr2block ( const HexStr hs, /* out-param */ Block b );

void block2hexstr ( const Block b, /* out-param */ HexStr hs );

void encrypt ( const Block p, const Block k, /* out-param */ Block c );

  /* c = E(p,k) */

void decrypt ( const Block c, const Block k, /* out-param */ Block p );

  /* p = D(c,k) */
```

Program `aestoolc.c` *pokazuje jak korzystać z tej biblioteki.*

OGRANICZENIA

Liczba określająca liczbę istotnych cyfr szesnastkowych w kluczu spełnia nierówność $1 \leq s \leq 5$.
Wskazówka: *Dobry program potrafi dla każdego dozwolonego pliku wejściowego odtworzyć klucze w czasie mniejszym niż 10s.*

Krzysztof Diks, Marcin Kubica
Tłumaczenie

Ioiwari Game

Rodzina gier Mankala jest znana ludzkości od dawna. W tym zadaniu rozważamy grę specjalnie wymyślona na potrzeby IOI. W grze bierze udział dwóch graczy. Do dyspozycji mają okrągłą planszę z siedmioma dołkami na obwodzie, a dodatkowo każdy z graczy ma swój dołek–bank. Gra rozpoczyna się od losowego rozłożenia 20 kamieni w dołkach na planszy w taki sposób, że w każdym z nich znajdują się co najmniej 2 i co najwyżej 4 kamienie. Gracze wykonują ruchy na przemian. Ruch gracza polega na wyborze niepustego dołka na planszy i wzięciu z niego wszystkich kamieni do ręki. Mając kamienie w ręku, gracz bierze pod uwagę kolejne dołki (w kierunku ruchu wskazówek zegara), zaczynając od następnego po opróżnionym, i wykonuje następujące czynności:

Więcej niż jeden kamień w ręce: Jeżeli w rozważanym dołku jest 5 kamieni, to bierze jeden kamień z dołka i przekłada go do swojego banku, w przeciwnym przypadku, odkłada jeden kamień z ręki do rozważanego dołka.

Jeden kamień w ręce: Jeżeli w rozważanym dołku jest co najmniej jeden, a co najwyżej 4 kamienie, to przekłada wszystkie kamienie z tego dołka oraz jeden z ręki do swojego banku, w przeciwnym przypadku (gdy w dołku jest 0 lub 5 kamieni) odkłada jedyny kamień z ręki do banku przeciwnika.

Gra się kończy, gdy wszystkie dołki na planszy są puste, a zwycięża ten gracz, który ma więcej kamieni w swoim banku.

Gracz, który rozpoczyna ma zawsze strategię wygrywającą. Twoim zadaniem jest napisanie programu, który gra w Ioiwari jako gracz rozpoczynający i wygrywa. Przeciwnik, program testujący, gra optymalnie, tzn. jeśli tylko dać mu szansę, to wygra, a Twój program przegra.

WEJŚCIE/WYJŚCIE

Twój program powinien czytać ze standardowego wejścia i pisać na standardowe wyjście. Twój program to gracz nr 1, a jego przeciwnik to gracz nr 2. Po rozpoczęciu, Twój program musi najpierw wczytać siedem liczb całkowitych $p1, \ldots p7$*, zapisanych w jednym wierszu: początkowe liczby kamieni odpowiednio w dołkach* $1, \ldots, 7$ *na planszy. Dołki na planszy są ponumerowane od 1 do 7, zgodnie z ruchem wskazówek zegara. Na samym początku gry banki graczy są puste. Twój program powinien grać jak następuje:*

Jeśli jest to ruch Twojego programu, to powinien on wypisać na standardowe wyjście numer dołka opisujący ruch.

Jeśli jest to ruch przeciwnika, to Twój program powinien wczytać ze standardowego wejścia numer (opróżnianego) dołka określający ten ruch.

NARZĘDZIA

Masz do dyspozycji program (ioiwari2 pod Linuxem, ioiwari2.exe pod Windowsami), który dla jednego początkowego rozmieszczenia kamieni gra optymalnie jako gracz nr 2. Program ten wypisuje na standardowe wyjście pierwszy wiersz, jaki powinien wczytać Twój program, opisujący początkowe rozmieszczenie kamieni w dołkach na planszy: `4 3 2 4 2 3 2`*. Następnie program ten gra, próbując czytać ze standardowego wejścia ruchy gracza nr 1 i wypisując swoje ruchy na standardowe wyjście. Możesz uruchomić swój program i ioiwari2 w oddzielnych okienkach i ręcznie wymieniać dane pomiędzy programami. Przebieg dialogu jest zapisywany w pliku ioiwari.out.*

WYTYCZNE PROGRAMISTYCZNE

W poniższych przykładach, ostatnia liczba całkowita z wejścia jest wczytywana na zmienną `last`*, a zmienna* `mymove` *zawiera Twój ruch.*

Jeśli programujesz w C++ i używasz iostreams, powinieneś stosować następującą konwencję wczytywania ze standardowego wejścia i wypisywania na standardowe wyjście:

```
cout<<mymove<<endl<<flush;
cin>>last;
```

Jeśli programujesz w C lub C++ i używasz scanf i printf, powinieneś stosować następującą konwencję wczytywania ze standardowego wejścia i wypisywania na standardowe wyjście:

```
printf("%d\n",mymove); fflush (stdout);
scanf ("%d", &last);
```

Jeśli programujesz w Pascalu, powinieneś stosować następującą konwencję wczytywania ze standardowego wejścia i wypisywania na standardowe wyjście:

```
Writeln(mymove);
Readln(last);
```

PRZYKŁAD

Oto poprawny ciąg 6 ruchów:

	Zawartości dołków i banków po wykonaniu ruchu								
Operacja/nr dołka	*1.*	*2.*	*3.*	*4.*	*5.*	*6.*	*7.*	*Bank1*	*Bank2*
Sytuacja początkowa	*4*	*3*	*2*	*4*	*2*	*3*	*2*	*0*	*0*
Ruch gracza nr 1: 2	*4*	*0*	*3*	*5*	*0*	*3*	*2*	*3*	*0*
Ruch gracza nr 2: 3	*4*	*0*	*0*	*4*	*1*	*4*	*0*	*3*	*4*
Ruch gracza nr 1: 5	*4*	*0*	*0*	*4*	*0*	*0*	*0*	*8*	*4*
Ruch gracza nr 2: 4	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*8*	*9*
Ruch gracza nr 1: 5	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*10*	*9*
Ruch gracza nr 2: 7	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*11*	*9*

PUNKTACJA

Za zwycięstwo w jednym teście Twój program otrzyma 4 punkty, za remis 2 punkty, a w pozostałych sytuacjach 0 punktów.

Mobile phones

Czwarta generacja stacji przekaźnikowych telefonii komórkowej w regionie Tampere działa w następujący sposób. Cały region jest podzielony na kwadraty. Kwadraty tworzą macierz rozmiaru $S \times S$, w której wiersze i kolumny są ponumerowane od 0 do $S - 1$. W każdym kwadracie znajduje się jedna stacja. Liczba aktywnych telefonów wewnątrz każdego kwadratu może się zmieniać, ponieważ telefony mogą przemieszczać się pomiędzy kwadratami oraz być wyłączane lub włączane. Od czasu do czasu, każda stacja przesyła raport o zmianie liczby aktywnych telefonów w jej obszarze wraz z informacją o jej położeniu (wiersz–kolumna w macierzy).

Napisz program, który dostaje te raporty i odpowiada na pytania o aktualną liczbę wszystkich aktywnych telefonów w zadanym prostokątnym obszarze.

WEJŚCIE/WYJŚCIE

Dane wejściowe mają postać liczb całkowitych i są czytane ze standardowego wejścia. Odpowiedzi są wypisywane na standardowe wyjście, również jako liczby całkowite. Format danych wejściowych jest następujący. Każde pojedyncze dane zajmują jeden wiersz i składają się z numeru instrukcji, po którym następują jej parametry (liczby całkowite), zgodnie z następującą tabelką:

Instrukcja	*Parametry*	*Znaczenie*
0	S	*Ustalenie rozmiarów macierzy na $S \times S$ i wypełnienie jej zerami. Instrukcja ta pojawia się tylko raz i zawsze jako pierwsza.*
1	$X\ Y\ A$	*Dodanie A do liczby aktywnych telefonów w kwadracie (X, Y) macierzy. Wartość A może być dodatnia lub ujemna.*
2	$L\ B\ R\ T$	*Zapytanie o aktualną, łączną liczbę aktywnych telefonów w kwadratach (X, Y), dla $L \leq X \leq R, B \leq Y \leq T$.*
3		*Zakończenie programu. Instrukcja ta pojawia się tylko raz, jako ostatnia.*

Wszystkie dane wejściowe mieszczą się w podanych zakresach i nie trzeba tego sprawdzać. W szczególności, jeśli A jest ujemne, to możesz założyć, ze liczba telefonów w kwadracie nie spadnie poniżej 0. Numeracja wierszy i kolumn w macierzy zaczyna się od 0, np. dla macierzy 4×4, mamy $0 \leq X \leq 3$ i $0 \leq Y \leq 3$.

Twój program powinien odpowiadać wyłącznie na instrukcje nr 2. W tym przypadku, powinien on wypisać na standardowe wyjście jeden wiersz zawierający jedną liczbę całkowitą — odpowiedź na zapytanie.

WYTYCZNE PROGRAMISTYCZNE

W poniższych przykładach zmienna całkowita `last` *reprezentuje ostatnią liczbę całkowitą wczytaną z wiersza, a* `answer` *jest zmienną całkowitą zawierającą odpowiedź.*

Jeśli piszesz w C++ i używasz iostreams, powinieneś stosować następującą konwencję wczytywania ze standardowego wejścia i wypisywania na standardowe wyjście:

```
cin>>last;
cout<<answer<<endl<<flush;
```

Jeśli piszesz w C lub C++ i stosujesz `scanf` *i* `printf`*, powinieneś stosować następującą konwencję wczytywania ze standardowego wejścia i wypisywania na standardowe wyjście:*

```
scanf ("%d", &last);
printf("%d\n",answer); fflush (stdout);
```

Jeśli piszesz w Pascalu, powinieneś stosować następującą konwencję wczytywania ze standardowego wejścia i wypisywania na standardowe wyjście:

```
Read(last); ... Readln;
Writeln(answer);
```

PRZYKŁAD

stdin	*stdout*	*opis*
0 4		*Ustalenie rozmiaru macierzy na* 4×4.
1 1 2 3		*Do kwadratu (1,2) dodaj +3.*
2 0 0 2 2		*Zapytanie dotyczące prostokąta* $0 \leq X \leq 2$, $0 \leq Y \leq 2$.
	3	*Odpowiedź na zapytanie.*
1 1 1 2		*Do kwadratu (1, 1) dodaj +2.*
1 1 2 -1		*Do kwadratu (1, 2) dodaj -1.*
2 1 1 2 3		*Zapytanie dotyczące prostokąta* $1 \leq X \leq 2$, $1 \leq Y \leq 3$.
	4	*Odpowiedź na zapytanie.*
3		*Koniec.*

OGRANICZENIA

Wymiary macierzy	$S \times S$	$1 \times 1 \leq S \times S \leq 1024 \times 1024$
Wartość w kwadracie V w dowolnym momencie	V	$0 \leq V \leq 2^{15} - 1 (= 32767)$
Wielkość zmiany	A	$-2^{15} \leq A \leq 2^{15} - 1 (= 32767)$
Liczba instrukcji na wejściu	U	$3 \leq U \leq 60002$
Maksymalna łączna liczba telefonów w macierzy	M	$M = 2^3 0$

Spośród 20 zestawów danych, w 16 rozmiar macierzy nie przekracza 512×512.

UWAGA: *Program umożliwiający testowanie rozwiązań przez sieć podaje testowanemu programowi na standardowe wejście zawartość Twojego pliku testowego.*

Score

Score (wynik) jest grą dwuosobową, w której gracze przesuwają ten sam jeden pionek po planszy. Plansza do gry składa się z N pozycji ponumerowanych od 1 do N i strzałek. Każda strzałka prowadzi od jednej pozycji do innej. Każda pozycja należy do dokładnie jednego z graczy, którego nazywamy właścicielem tej pozycji. Dodatkowo każdej pozycji jest przypisana dodatnia wartość. Wszystkie wartości są różne. Pozycja 1 jest pozycją startową. Początkowo wynik (ang. score) każdego z graczy wynosi 0.

Reguły gry są następujące. Oznaczmy przez C pozycję pionka na planszy przed wykonaniem ruchu. Na początku gry C jest pozycją startową (nr 1). Ruch w grze polega na wykonaniu następujących czynności:

(1) Jeśli wartość C jest większa niż aktualny wynik właściciela tej pozycji, to jego wynik wzrasta do wartości C. W przeciwnym przypadku wynik właściciela nie zmienia się. Wynik drugiego z graczy nie ulega zmianie.

(2) Następnie właściciel C wybiera jedną ze strzałek wychodzących z aktualnej pozycji pionka i przesuwa go na pozycje, do której prowadzi strzałka. Nowa pozycją staje się aktualną pozycją pionka. Zwróć uwagę, że ten sam gracz może wykonać kilka kolejnych ruchów z rzędu.

Gra kończy się, gdy pionek powraca na pozycję startową. Zwycięzcą gry jest gracz z większym końcowym wynikiem. Strzałki są zawsze poprowadzone w następujący sposób:

- *Z każdej aktualnej pozycji pionka zawsze wychodzi co najmniej jedna strzałka.*
- *Każda pozycja P jest osiągalna z pozycji startowej, tzn. istnieje ciąg strzałek prowadzący z pozycji startowej do P.*
- *Masz gwarancję, że gra zawsze się kończy w skończonej liczbie ruchów.*

Napisz program, który gra w Score i wygrywa. W każdej grze, w której będzie uczestniczył Twój program podczas testowania, ma on możliwość zwyciężyć, niezależnie od tego czy akurat rozpoczął grę, czy nie. Przeciwnik zawsze gra optymalnie, tzn. jeśli dasz mu tylko szansę, to wygra, a Twój program przegra.

WEJŚCIE/WYJŚCIE

Twój program wczytuje dane ze standardowego wejścia i wypisuje wyniki na standardowe wyjście. Twój program to gracz nr 1, natomiast przeciwnik to gracz nr 2. Po uruchomieniu Twój program powinien najpierw przeczytać następujące dane ze standardowego wejścia.

Pierwszy wiersz zawiera jedną liczbę całkowita — liczbę pozycji N, $1 \leq N \leq 1\,000$. Pozycje są ponumerowane od 1 do N. Każdy z kolejnych N wierszy zawiera N liczb całkowitych opisujących strzałki. Jeśli istnieje strzałka prowadząca od pozycji i do pozycji j, to j–tą liczbą w i–tym z tych wierszy jest 1, w przeciwnym przypadku 0.

Kolejny wiersz zawiera N liczb całkowitych opisujących właścicieli poszczególnych pozycji. Jeśli właścicielem pozycji i jest gracz nr 1 (Twój program), to i–ta z tych liczb jest równa 1, w przeciwnym przypadku i–ta liczba jest równa 2.

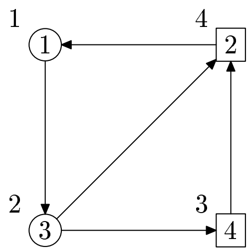
Kolejny wiersz zawiera N liczb całkowitych — wartości poszczególnych pozycji. Jeśli i–ta z tych liczb jest równa j, to wartością pozycji i jest j. Wszystkie wartości j są różne i spełniają nierówności $1 \leq j \leq N$.

Następnie rozpoczyna się gra.

Pozycją początkowa pionka jest pozycja nr 1. Twój program powinien grać następująco i zakończyć działanie po powrocie pionka na pozycję nr 1:

- *Jeśli jest to ruch Twojego programu, to Twój program powinien wypisać na standardowe wyjście numer następnej pozycji P, $1 \leq P \leq N$.*
- *Jeśli jest to ruch przeciwnika, to Twój program powinien wczytać ze standardowego wejścia numer następnej pozyji P, $1 \leq P \leq N$.*

Rozważmy następujący przykład. Plansza do gry jest przedstawiona na rysunku 1. Pozycje zaznacznone kółkami należą do gracza nr 1, a pozycje zaznacznone kwadratami należą do gracza nr 2. Wewnątrz każdej pozycji znajduje się jej wartość, a numer pozycji jest zapisany na zewnątrz. Poniżej przedstawiono rozgrywkę.



Rys. 1

stdin	*stdout*	*objaśnienie*
4		*N*
0 1 0 0		*Opis strzałek wychodzących z pozycji 1*
0 0 1 1		*Opis strzałek wychodzących z pozycji 2*
0 0 0 1		*Opis strzałek wychodzących z pozycji 3*
1 0 0 0		*Opis strzałek wychodzących z pozycji 4*
1 1 2 2		*Właściciele pozycji*
1 3 4 2		*Wartości pozycji*
	2	*Ruch gracza 1*
	4	*Ruch gracza 1*
1		*Gracz 2 wykonuje ruch na pozycję startową — gra się kończy.*

Po zakończeniu gry wynikiem Gracza 1 jest 3, natomiast Gracza 2 — 2. Gracz 1 wygrywa.

WYTYCZNE PROGRAMISTYCZNE

W przykładach poniżej `target` *jest całkowitoliczbową zmienną oznaczającą pozycję.*

Jeśli programujesz w C++ i korzystasz z iostreams, powinieneś stosować następującą konwencję czytania ze standardowego wejścia i pisania na standardowe wyjście:

```
cin>>target;
cout<<target<<endl<<flush;
```

Jeśli programujesz w C lub C++ i korzystasz ze `scanf` *i* `printf`*, powinieneś stosować następującą konwencję czytania ze standardowego wejścia i pisania na standardowe wyjście:*

```
scanf ("%d", &target);
printf("%d\n",target); fflush (stdout);
```

Jeśli programujesz w Pascalu, powinieneś stosować następującą konwencję czytania ze standardowego wejścia i pisania na standardowe wyjście:

```
Readln(target);
Writeln(target);
```

NARZĘDZIA

Do dyspozycji masz program (`score2` *w Linuxie,* `score2.exe` *w Windowsach), który wczytuje opis planszy do gry z pliku score.in zapisany w formacie przedstawionym na poprzedniej stronie. Program ten wypisze powyższą informację w tym samym formacie na standardowe wyjście. To wyjście może być wykorzystane jako wejście dla Twojego programu, do celów testowych. Następnie program* `score2` *gra używając strategii losowej — wczytuje ruchy Twojego programu ze standardowego wejścia i zapisuje swoje ruchy na standardowe wyjście.*

SPOSÓB OCENY

Dla jednego zestawu danych Twój program zdobędzie komplet punktów, jeśli wygra grę, a nie zdobędzie żadnych punktów, jeśli ją przegra. W trakcie oceny przez sprawdzaczkę Twój program jest uruchamiany dwukrotnie. Ograniczenie czasowe dla pierwszego uruchomienia jest o 1s większe od ograniczenia czasu podanego dla tego zadania. Ruchy (wejścia i wyjścia Twojego programu) wykonywane w trakcie gry są zapamiętywane. Następnie Twój program jest wykonywany dla danych wczytywanych z pliku i mierzony jest oficjalnie jego czas działania. Twój program MUSI wykonywać dokładnie TAKIE SAME RUCHY, jak podczas pierwszego wykonania (tzn. musi być DETERMINISTYCZNY).

Twofive

Tajne komunikaty pomiędzy Świętym Mikołajem i jego małymi pomocnikami szyfruje się zwykle w języku J-25. Alfabet tego języka jest taki sam jak alfabet angielski, z jednym wyjątkiem: nie ma w nim litery "Z", tzn. alfabet ten zawiera 25 liter od "A" do "Y", w takiej samej kolejności, co alfabet angielski. Każde słowo w języku J–25 składa się z dokładnie 25 różnych liter. Takie słowo można zapisać w tablicy 5×5*, wpisując je w kolejne wiersze; np. słowo* `ADJPTBEKQUCGLRVFINSWHMOXY` *zostanie zapisane następująco:*

```
A    D    J    P    T
B    E    K    Q    U
C    G    L    R    V
F    I    N    S    W
H    M    O    X    Y.
```

Po wpisaniu poprawnego słowa z języka J-25 do tablicy, litery we wszystkich wierszach i kolumnach będą uporządkowane rosnąco. Tak więc słowo `ADJPTBEKQUCGLRVFINSWHMOXY` *jest poprawne, a* `ADJPTBEGQUCKLRVFINSWHMOXY` *nie (porządek rosnący nie jest zachowany w drugiej i trzeciej kolumnie).*

Święty Mikołaj ma słownik, który jest listą wszystkich poprawnych słów języka J-25, ułożonych w porządku rosnącym (leksykograficznie) wraz z ich numerami porządkowymi zaczynającymi się od 1. Np. w tym słowniku `ABCDEFGHIJKLMNOPQRSTUVWXY` *jest słowem nr 1, zaś* `ABCDEFGHIJKLMNOPQRSUTVWXY` *jest słowem nr 2. W słowie nr 2, w porównaniu ze słowem nr 1, litery "U" i "T" są przestawione.*

Niestety słownik ten jest bardzo duży. Napisz program, który wyznacza numer porządkowy dowolnego zadanego słowa, a także słowo odpowiadające zadanemu numerowi. W słowniku nie ma więcej niż 231 słów.

WEJŚCIE

Plik wejściowy ma nazwę `twofive.in` *i składa się z dwóch wierszy. Pierwszy wiersz zawiera jednoliterowy napis: "W" lub "N". Jeżeli jest to "W", to drugi wiersz zawiera poprawne słowo języka J-25, tzn. 25-znakowy napis. Jeżeli pierwszy wiersz zawiera "N", to drugi wiersz zawiera numer porządkowy poprawnego słowa z języka J-25.*

WYJŚCIE

Plik wyjściowy ma nazwę `twofive.out` *i składa się z jednego wiersza. Jeśli drugi wiersz pliku wejściowego zawiera słowo 25-literowe, to plik wyjściowy zawiera numer porządkowy tego słowa. Jeśli drugi wiersz pliku wejściowego zawiera liczbę, to plik wyjściowy zawiera 25-literowe słowo o tym numerze porządkowym.*

PRZYKŁAD

twofive.in	twofive.out
W ABCDEFGHIJKLMNOPQRSUTVWXY	2

twofive.in	twofive.out
N 2	ABCDEFGHIJKLMNOPQRSUTVWXY