Marian M. Kędzierski — Treść zadania
Marian M. Kędzierski, Błażej Osiński — Opracowanie
Błażej Osiński — Program

Dostępna pamięć: 32 MB. *OI, Etap III, dzień pierwszy, 01.04.2009*

Tablice

Rozważmy tablicę o wymiarach $n \times m$ wypełnioną **różnymi** *liczbami całkowitymi. Na tej tablicy możemy wykonywać następujące operacje:*

1. *zamiany dwóch wierszy*
2. *zamiany dwóch kolumn.*

Powiemy, że dwie tablice są **podobne**, *jeżeli przy pomocy pewnej sekwencji powyższych operacji wykonanych na pierwszej tablicy możemy z niej otrzymać drugą.*

Napisz program, który dla danego zestawu par tablic stwierdzi, które pary zawierają tablice podobne.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita t ($1 \leqslant t \leqslant 10$) oznaczająca liczbę par tablic. W następnych liniach znajdują się opisy kolejnych par tablic.

Opis pary tablic zaczyna się od wiersza zawierającego dwie liczby całkowite n oraz m ($1 \leqslant n, m \leqslant 1000$) oddzielone pojedynczym odstępem, oznaczające odpowiednio liczbę wierszy oraz liczbę kolumn obu tablic.

W następnych n wierszach znajduje się opis pierwszej tablicy. W i-tym spośród tych wierszy znajduje się m liczb całkowitych a_{ij} ($-1\,000\,000 \leqslant a_{ij} \leqslant 1\,000\,000$) pooddzielanych pojedynczymi odstępami, oznaczających kolejne liczby w i-tym wierszu pierwszej tablicy.

W następnych n wierszach znajduje się opis drugiej tablicy. W i-tym spośród tych wierszy znajduje się m liczb całkowitych b_{ij} ($-1\,000\,000 \leqslant b_{ij} \leqslant 1\,000\,000$) pooddzielanych pojedynczymi odstępami, oznaczających kolejne liczby w i-tym wierszu drugiej tablicy.

Wszystkie liczby występujące w jednej tablicy są parami różne.

Wyjście

Twój program powinien wypisać na standardowe wyjście t wierszy. W k-tym z nich powinno znaleźć się jedno słowo „`TAK`”, jeżeli tablice w k-tej wczytanej parze są podobne, zaś słowo „`NIE`” w przeciwnym przypadku.

Przykład

Dla danych wejściowych:

```
2
4 3
1 2 3
4 5 6
7 8 9
10 11 12
11 10 12
8 7 9
5 4 6
2 1 3
2 2
1 2
3 4
5 6
7 8
```

poprawnym wynikiem jest:

```
TAK
NIE
```

Wyjaśnienie do przykładu: *Pierwsza para zawiera tablice podobne. Aby przetworzyć pierwszą tablicę na drugą, wystarczy zamienić ze sobą pierwsze dwie kolumny, a następnie pierwszy wiersz z ostatnim i drugi wiersz z trzecim.*

Druga para zawiera tablice, które nie są podobne. Aby to stwierdzić, wystarczy zauważyć, że zbiory wartości w ich komórkach są różne.

Rozwiązanie

Rozwiązanie siłowe

Dla uproszczenia opisu nazwijmy wymienione w treści zadania operacje na tablicy *operacjami elementarnymi*, zaś tablice, na których operujemy — *macierzami*[1].

Najprostsze rozwiązanie naszego zadania polega na generowaniu wszystkich możliwych macierzy, jakie można utworzyć z pierwszej macierzy przy użyciu operacji elementarnych, i sprawdzaniu, dla każdej z nich, czy otrzymaliśmy drugą macierz. Niestety takie rozwiązanie siłowe jest bardzo powolne, ponieważ liczba możliwych wyników ciągów operacji to iloczyn liczb permutacji wierszy i kolumn, a zatem złożoność czasowa takiego rozwiązania wyniosłaby $\Omega(n! \cdot m!)$. Wobec ograniczeń z zadania, oczywiście nie możemy sobie na to pozwolić.

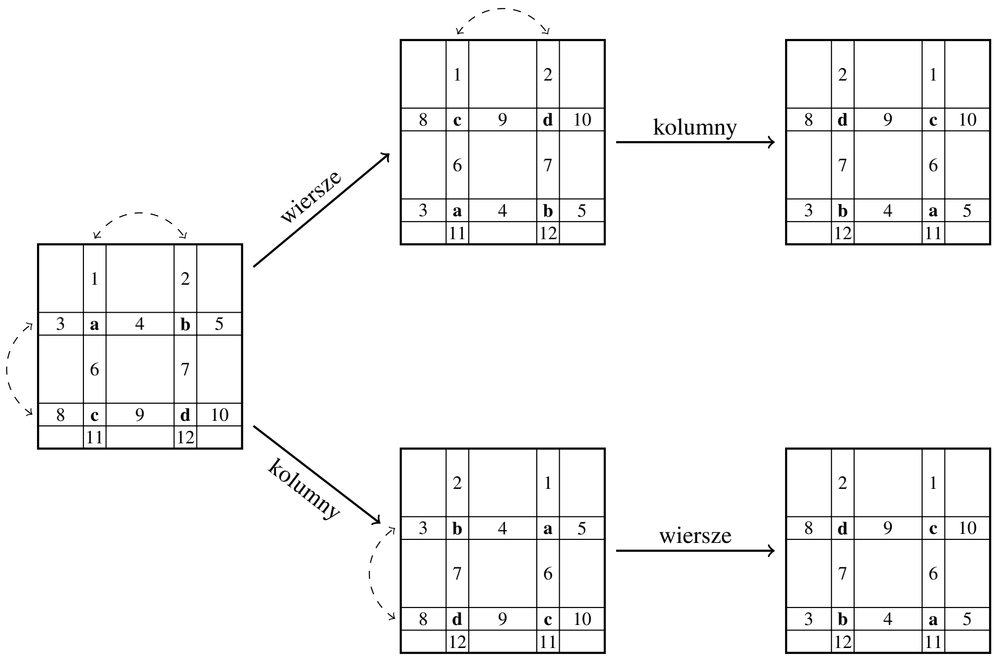
Kluczowe spostrzeżenia

Kluczowym spostrzeżeniem prowadzącym do rozwiązania zadania jest to, że operacje elementarne nie mieszają zawartości wierszy między sobą. Innymi słowy, jeżeli dwie licz-

[1] W rzeczywistości istnieje w algebrze liniowej pojęcie *operacji elementarnej* na *macierzy* i obejmuje ono nieco szerszą klasę operacji niż tutaj rozważane. Od nich także wzięła się inspiracja do niniejszego zadania.

by a i b znajdują się w tym samym wierszu, to po wykonaniu dowolnej liczby operacji elementarnych będą one nadal w jednym wierszu, potencjalnie o innym numerze. Można to łatwo udowodnić, zauważając, że operacja zamiany wierszy nie zmienia zawartości poszczególnych wierszy, a jedynie ich kolejność, natomiast operacja zamiany kolumn jedynie przestawia elementy w ramach każdego z wierszy. Podobnie rzecz ma się z kolumnami.

Drugim kluczowym spostrzeżeniem jest to, że jeżeli chcemy wykonać kolejno dwie operacje elementarne, jedną na wierszach, a drugą na kolumnach, to kolejność ich wykonania nie ma znaczenia — w obu przypadkach otrzymamy to samo (patrz rys. 1).



Rys. 1: Rysunkowy dowód faktu, że operacje zamiany miejscami dwóch wierszy i dwóch kolumn są przemienne.

Z powyższego spostrzeżenia wynika w szczególności, że jeżeli istnieje ciąg operacji elementarnych przekształcający jedną macierz w drugą, to istnieje też ciąg złożony z tych samych operacji i dający taki sam wynik, na którego początku są tylko operacje na wierszach, a na końcu na kolumnach.

Rozwiązanie wzorcowe

Jak wykorzystać poczynione spostrzeżenia? Otóż nadal będziemy próbowali przetworzyć pierwszą macierz tak, aby stała się identyczna z drugą, jednak tym razem nie musimy już badać wszystkich permutacji, gdyż wiemy, że wiele z nich będzie prowadziło do tego samego wyniku.

Spróbujmy uszeregować wiersze pierwszej macierzy tak, aby zgadzały się z wierszami drugiej macierzy. Oczywiście problemem jest to, że odpowiadające sobie wiersze w obu macierzach nie są identyczne (mogą różnić się kolejnością liczb). I tutaj przychodzi nam z pomocą istotna własność, wyraźnie wspomniana w treści zadania: liczby w komórkach każdej macierzy są parami różne. Dzięki temu możemy identyfikować poszczególne wiersze na przykład z najmniejszymi liczbami, które się w nich znajdują. Jeżeli bowiem macierze mają być podobne, to najmniejsze wyrazy odpowiadających sobie wierszy muszą być równe.

Aby więc dobrać odpowiednią kolejność wierszy pierwszej macierzy, wystarczy w każdej z macierzy posortować tablice złożone z par: numer wiersza i jego najmniejszy wyraz (sortujemy według najmniejszego wyrazu). W ten sposób albo od razu dowiadujemy się, że macierze nie są podobne (jeżeli zbiory najmniejszych wyrazów nie zgadzają się), albo jesteśmy w stanie przyporządkować każdemu wierszowi pierwszej macierzy odpowiedni wiersz drugiej macierzy. Jeżeli się to udało, to zmieniamy odpowiednio kolejność wierszy pierwszej macierzy, zaś drugą pozostawiamy bez zmian. W ten sposób otrzymujemy takie dwie macierze, że dla każdego $i \in \{1, 2, \ldots, n\}$ najmniejsza liczba w i-tym wierszu każdej z nich jest taka sama oraz wyjściowe macierze są podobne wtedy i tylko wtedy, gdy pierwszą z nowo powstałych macierzy można przekształcić w drugą jedynie za pomocą zamian kolumn miejscami. Następnie w identyczny sposób możemy uszeregować kolumny (niezależnie od wyniku przyporządkowania wierszy).

Po wykonaniu tych dwóch kroków (uszeregowania wierszy i kolumn) albo przekonamy się, że dwie rozważane macierze nie mogą być podobne, albo otrzymamy jedyne permutacje kolumn i wierszy pierwszej macierzy reprezentujące zestaw operacji elementarnych, który może przeprowadzać ją na drugą macierz. Aby jednak upewnić się co do podobieństwa, musimy jeszcze faktycznie wykonać te operacje na pierwszej macierzy, a następnie sprawdzić, czy rzeczywiście otrzymaliśmy w ten sposób drugą macierz.

Złożoność czasowa tego algorytmu to $O(n \cdot m + n \cdot \log n + m \cdot \log m)$, jeżeli użyjemy sortowania działającego w czasie liniowo-logarytmicznym (np. sortowania przez scalanie). Jednak dzięki stosunkowo małemu zakresowi liczb w komórkach macierzy (od $-d$ do d dla $d \leqslant 1\,000\,000$) każde z sortowań możemy wykonać w czasie liniowym względem d (sortowanie przez zliczanie), co prowadzi do złożoności czasowej $O(n \cdot m + d)$.

Algorytm wzorcowy działa podobnie, tyle że w drugiej fazie (czyli przy znajdowaniu przyporządkowania kolumn) korzysta z wyniku uszeregowania wierszy i porównuje ze sobą jedynie dowolne dwa odpowiadające sobie wiersze. Na końcu, podobnie jak wyżej, stosuje uzyskane permutacje do pierwszej macierzy i sprawdza, czy otrzymano drugą. Jego implementacje znajdują się w następujących plikach: `tab.cpp`, `tab1.c`, `tab2.pas`, `tab3.java`.

Inne rozwiązania

Powyższe rozwiązanie nie jest jedynym, za które przewidziana była maksymalna liczba punktów. Można także podać wiele innych, równie dobrych metod. Jedną z nich jest podejście oparte na sprowadzeniu macierzy do pewnej *postaci kanonicznej*.

Jest to dosyć częsta metoda sprawdzania, czy dwa elementy jakiegoś zbioru są w relacji równoważności[2]. Nie jest tu istotne, co rozumiemy przez „postać kanoniczną” — ważne jest tylko, żeby był to jakiś ściśle określony reprezentant każdej klasy równoważności[3], którego jesteśmy w stanie łatwo wygenerować z każdego elementu zbioru. W tym przypadku chodzi nam o znalezienie jakiejś funkcji, która każdą macierz przekształci na podobną do niej, przy czym dwie macierze podobne do siebie zawsze przekształci na taką samą.

Inne, klasyczne przykłady takiego podejścia to:

- Badanie, czy dwa słowa są anagramami, które polega na zliczeniu wystąpień każdej litery w obu słowach i porównaniu otrzymanych statystyk.
- Badanie, czy zbiory wyrazów w dwóch ciągach są równe, które polega na posortowaniu każdego z ciągów i porównaniu ich posortowanych postaci.

Znalezienie „postaci kanonicznej” często polega na jakimś rodzaju sortowania. W tym wypadku wystarczy, jeżeli posortujemy według pewnego kryterium najpierw wiersze, a potem kolumny obu macierzy. Możemy np. sortować wiersze względem minimalnych elementów, a następnie kolumny względem elementów w pierwszym wierszu. Rozwiązanie takie można zaimplementować w takiej samej złożoności jak wzorcowe.

Jeżeli dobrze przyjrzymy się temu algorytmowi, to odkryjemy, że jest on bardzo podobny do poprzedniego, z tą różnicą, że w algorytmie wzorcowym sprowadzaliśmy pierwszą macierz do drugiej (permutując kolumny i wiersze pierwszej macierzy tak, aby zgadzały się z drugą), natomiast w niniejszym rozwiązaniu permutujemy wiersze i kolumny obu macierzy naraz. Implementacja tego rozwiązania znajduje się w pliku `tab4.cpp`.

Można wymienić wiele podobnych algorytmów, w których konstruowane postaci kanoniczne są nieco inne, nie zawsze jednak otrzymamy w ten sposób efektywne rozwiązanie. Wszystko zależy od tego, jaki rodzaj postaci kanonicznej wybierzemy i jak szybko jesteśmy w stanie ją otrzymywać z dowolnej macierzy.

Haszowanie

Można także posłużyć się zupełnie inną, ciekawą metodą, opartą na haszowaniu. Daje ona prostą heurystykę, która z dużym prawdopodobieństwem działa poprawnie, mimo iż nie można zagwarantować jej poprawności w ogólnym przypadku.

Podejście to polega na skonstruowaniu dla każdej z macierzy swoistego „zbioru kolumn”, w którym każda kolumna jest reprezentowana przez swoje posortowane elementy zakodowane w jednej liczbie całkowitej, oraz porównaniu tych zbiorów dla obu macierzy. Następnie operację tę należy powtórzyć dla wierszy.

W jaki sposób zakodować ciąg wyrazów z kolumny w jednej liczbie całkowitej? Niech $a_1 < a_2 < \ldots < a_n$ będą elementami pewnej kolumny posortowanymi rosnąco i niech Q będzie pewną liczbą całkowitą (najlepiej pierwszą i większą niż wszystkie a_i). Wtedy liczba

$$Z = a_1 \cdot Q^{n-1} + a_2 \cdot Q^{n-2} + \ldots + a_{n-1} \cdot Q + a_n$$

[2] *Relacja równoważności* to taka relacja pomiędzy elementami pewnego zbioru, która jest *zwrotna* (czyli każdy element jest sam ze sobą w relacji), *symetryczna* (jeżeli a jest w relacji z b, to b jest w relacji z a) oraz *przechodnia* (jeżeli a jest w relacji z b oraz b jest w relacji z c, to również a jest w relacji z c).

[3] *Klasa równoważności* (inaczej klasa abstrakcji) to maksymalny ze względu na zawieranie podzbiór elementów, którego każde dwa elementy są ze sobą w danej relacji równoważności.

reprezentuje cały ciąg (a_i). Nazwijmy ją *charakterystyką* rozważanej kolumny. W przypadku większych testów (lub dużej liczby Q) obliczenie Z często będzie powodowało przepełnienie zakresu liczby. W takiej sytuacji operacje są wykonywane modulo 2^{32} dla typu całkowitego 32-bitowego i modulo 2^{64} dla typu 64-bitowego. Z tego powodu może się zdarzyć, że dwie różne kolumny (odpowiednio dwa wiersze) zostaną odwzorowane na tę samą liczbę Z i w efekcie macierze zostaną uznane za podobne, mimo iż podobne nie są (taką sytuację nazywamy *konfliktem* w haszowaniu).

Zauważmy, że operacje elementarne nie zmieniają zbioru charakterystyk kolumn i wierszy macierzy. Jeżeli więc okaże się, że zbiór charakterystyk kolumn (wierszy) jest inny w przypadku pierwszej macierzy niż w przypadku drugiej, to nie jest możliwe, żeby macierze te były do siebie podobne. Jeżeli jednak zbiory te będą identyczne i nie wystąpią żadne konflikty haszowania, to można pokazać, że macierze rzeczywiście są podobne. W tym celu należy dowieść, że ze zbioru charakterystyk wierszy i kolumn macierzy można odtworzyć całą macierz z dokładnością do podobieństwa, czyli zrekonstruować pewną postać kanoniczną macierzy. Uzasadnienie tego faktu pozostawiamy Czytelnikowi.

Powyższa metoda nie daje wprawdzie gwarancji poprawności, jest jednak łatwa do zapisania i trudno jest przygotować testy, dla których działa ona niepoprawnie (dlatego że nie wiadomo, jaka stała Q została użyta w programie). Rozwiązanie takie odpowiednio zaimplementowane z bardzo dużym prawdopodobieństwem uzyskiwało komplet punktów. Gdybyśmy chcieli to prawdopodobieństwo zwiększyć, moglibyśmy powtórzyć obliczenie kilkukrotnie z użyciem różnych wartości Q. Rozwiązanie to, podobnie jak większość rozwiązań opartych na haszowaniu, jest godne polecenia, ponieważ prawdopodobieństwo błędnej odpowiedzi spowodowanej użyciem haszowania jest wielokrotnie mniejsze niż prawdopodobieństwo błędu implementacyjnego w przypadku wybrania bardziej skomplikowanego algorytmu. Jeżeli jednak chcemy mieć pewność, że zastosowany algorytm jest poprawny, to niestety nie jest to podejście satysfakcjonujące.

Przykład implementacji powyższej heurystyki znajduje się w pliku `tab5.cpp`. Dodajmy na koniec, że analogiczne rozwiązania, które jako charakterystyki wierszy i kolumn obierają jakieś prostsze wartości, np. sumy, minima czy maksima elementów, już poprawne nie są — znalezienie odpowiednich kontrprzykładów na takie rozwiązania pozostawiamy Czytelnikowi jako ciekawe ćwiczenie.

Testy

Rozwiązania zawodników były sprawdzane na 10 zestawach danych testowych:

Nazwa	max **n**	max **m**	**Opis**
tab1.in	10	10	bardzo małe i proste testy
tab2.in	10	10	małe testy, kwadratowe macierze
tab3.in	20	20	małe testy, prostokątne macierze
tab4.in	130	130	średniej wielkości testy, kwadratowe macierze
tab5-6.in	131	140	średniej wielkości testy, prostokątne macierze
tab7-10.in	1000	1000	duże testy