Adam Polak
Treść zadania, Opracowanie

Lech Duraj
Program

Dostępna pamięć: 64 MB. *OI, etap II, dzień próbny, 11.02.2014*

Karty

Na stole leży n kart ułożonych w pewnej kolejności. Na każdej karcie zapisane są dwie liczby całkowite: jedna z nich na jednej stronie karty (na awersie), a druga na drugiej stronie karty (na rewersie). Początkowo wszystkie karty leżą na stole awersem do góry. Iluzjonista Bajtazar zamierza przedstawić (i to wielokrotnie!) swój popisowy Wielki Trik z Wyszukiwaniem Binarnym. Aby jednak mógł go zaprezentować, ciąg liczb widocznych na stole musi być niemalejący. W tym celu Bajtazar być może będzie musiał odwrócić niektóre karty tak, aby widoczne były liczby znajdujące się na ich rewersach.

Trik Bajtazara wymaga udziału osoby z publiczności. Jednak niektórzy zgłaszający się na ochotnika widzowie są podstawieni przez konkurentów Bajtazara. Każdy z nich, wchodząc na scenę, błyskawicznym ruchem zamieni ze sobą miejscami dwie spośród leżących na stole kart. Po każdej z takich zamian Bajtazar może znowu odwrócić niektóre karty na drugą stronę, ale nawet mimo tego może nie być w stanie wykonać triku. Będzie wtedy zmuszony wrócić do tradycyjnych metod zabawiania widzów, z udziałem królików i kapeluszy.

Napisz program, który określi, po każdej zamianie kart miejscami, czy Bajtazar może wykonać swoją sztuczkę.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n ($2 \leqslant n \leqslant 200\,000$), oznaczająca liczbę kart. W kolejnych n wierszach opisane są karty, po jednej w wierszu, w takiej kolejności, w jakiej leżą na stole. W i-tym z tych wierszy znajdują się dwie liczby całkowite x_i i y_i ($0 \leqslant x_i, y_i \leqslant 10^7$) oddzielone pojedynczym odstępem. Są to liczby zapisane na i-tej karcie: x_i oznacza liczbę zapisaną na awersie tej karty, a y_i oznacza liczbę zapisaną na jej rewersie. Początkowy ciąg kart nie musi pozwalać na wykonanie triku.

W następnym wierszu wejścia znajduje się jedna liczba całkowita m ($1 \leqslant m \leqslant 1\,000\,000$), oznaczająca liczbę zamian. W kolejnych m wierszach opisane są zamiany: j-ty z tych wierszy zawiera dwie liczby całkowite a_j i b_j ($1 \leqslant a_j, b_j \leqslant n$) oddzielone pojedynczym odstępem, oznaczające, że j-ty z zaproszonych na scenę widzów zamieni miejscami a_j-tą i b_j-tą kartę.

W testach wartych 30% punktów zachodzi dodatkowy warunek: liczby po obu stronach kart są identyczne. W (być może innych) testach wartych 38% punktów zachodzi dodatkowy warunek $n \leqslant 2000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście m wierszy, każdy zawierający pojedyncze słowo `TAK` *lub* `NIE`*. W j-tym wierszu powinno znaleźć się słowo* `TAK`*, jeśli Bajtazar może, po j-tej zamianie kart, ułożyć ciąg niemalejący, odwracając niektóre karty. W przeciwnym wypadku w tym wierszu powinno znaleźć się słowo* `NIE`*.*

Przykład

Dla danych wejściowych:
```
4
2 5
3 4
6 3
2 7
2
3 4
1 3
```

poprawnym wynikiem jest:
```
NIE
TAK
```

Testy „ocen”:

`1ocen`: *$n = 6$, $m = 6$, mały test poprawnościowy;*

`2ocen`: *$n = 7$, $m = 9$, liczby po obu stronach kart są identyczne;*

`3ocen`: *$n = 200\ 000$, $m = 1\ 000\ 000$, każda zamiana o parzystym numerze cofa poprzednią.*

Rozwiązanie

W zadaniu mamy do czynienia z ciągiem n par liczb (x_i, y_i). Na ciągu tym m razy wykonywana jest operacja zamiany miejscami dwóch (niekoniecznie sąsiednich) par. Po każdej operacji musimy odpowiedzieć na pytanie, czy da się z każdej pary wybrać po jednej liczbie tak, by powstały w ten sposób ciąg liczb był niemalejący.

Rozwiązanie wzorcowe

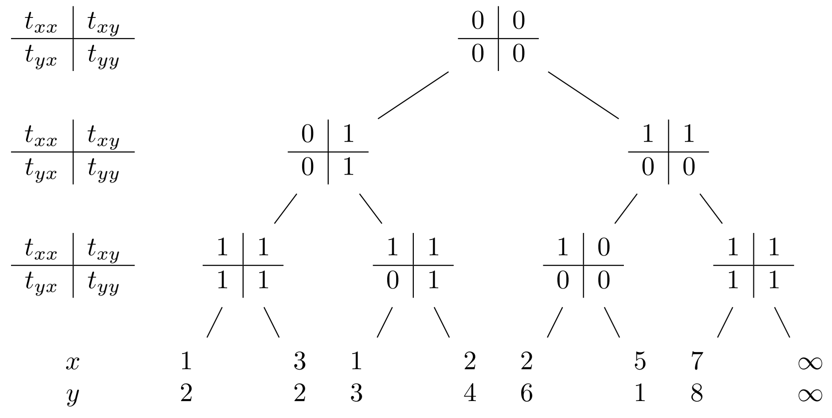
Rozwiązanie opiera się na strukturze danych nazywanej statycznym drzewem przedziałowym[1]. Jest to pełne drzewo binarne, którego liście odpowiadają wyrazom rozważanego przez nas ciągu, zaś węzły wewnętrzne – pewnym jego spójnym podciągom. Węzeł wewnętrzny zawsze reprezentuje podciąg powstały przez sklejenie podciągów reprezentowanych przez synów tego węzła.

Dla uproszczenia załóżmy, że długość ciągu n jest potęgą dwójki. Jeśli tak nie jest, możemy dopisać na końcu ciągu odpowiednią liczbę par, których oba elementy są równe nieskończoność (w praktyce, przy ograniczeniach z treści zadania, za nieskończoność wystarczy przyjąć 10^7). Jak łatwo zauważyć, wydłuży to ciąg co najwyżej dwukrotnie oraz nie zmieni odpowiedzi na postawione w zadaniu pytania.

W rozwiązaniu wzorcowym z każdym węzłem drzewa w skojarzone są cztery zmienne logiczne $t_{xx}[w]$, $t_{xy}[w]$, $t_{yx}[w]$, $t_{yy}[w]$. Załóżmy, że węzeł w reprezentuje podciąg od i-tego do j-tego wyrazu ciągu włącznie. Wówczas zmienna $t_{xx}[w]$ przyjmuje

[1] Statyczne drzewa przedziałowe pojawiały się już wielokrotnie na Olimpiadzie Informatycznej. Znane są również pod nazwą drzew licznikowych oraz drzew turniejowych. Można się o nich co nieco dowiedzieć na stronie `http://was.zaa.mimuw.edu.pl`.

wartość *prawda* wtedy i tylko wtedy, gdy z par w podciągu reprezentowanym przez w da się wybrać po jednej liczbie tak, by powstały ciąg był niemalejący, przy czym z pierwszej pary (x_i, y_i) wybieramy x_i, a z ostatniej pary (x_j, y_j) wybieramy x_j. Pozostałe trzy zmienne są zdefiniowane analogicznie.



Rys. 1: Przykład ilustrujący strukturę danych wykorzystywaną w rozwiązaniu. Wartości logiczne *prawda* są oznaczane jako 1, a *fałsz* jako 0.

Znając wartości zmiennych dla synów, możemy szybko (w czasie stałym) obliczyć je również dla ojca. Na przykład, jeśli synami (odpowiednio lewym i prawym) węzła w są węzły l i p, przedział reprezentowany przez węzeł l kończy się na wyrazie i-tym, a przedział reprezentowany przez węzeł p zaczyna się na wyrazie $(i+1)$-szym, to

$$\begin{aligned} t_{xx}[w] = {} & (t_{xx}[l] \wedge t_{xx}[p] \wedge x_i \leqslant x_{i+1}) \vee \\ & (t_{xx}[l] \wedge t_{yx}[p] \wedge x_i \leqslant y_{i+1}) \vee \\ & (t_{xy}[l] \wedge t_{xx}[p] \wedge y_i \leqslant x_{i+1}) \vee \\ & (t_{xy}[l] \wedge t_{yx}[p] \wedge y_i \leqslant y_{i+1}). \end{aligned}$$

Pozostałe trzy zmienne możemy obliczyć analogicznie.

Na początku rozwiązania konstruujemy drzewo dla wyjściowego ciągu, odwiedzając węzły w kolejności od dołu do góry i obliczając skojarzone z nimi zmienne. Konstrukcja drzewa odbywa się zatem w czasie $O(n)$, gdyż taka jest liczba węzłów drzewa.

Każda operacja zamiany, którą mamy do wykonania, powoduje zmianę wartości dwóch wyrazów ciągu. Po wykonaniu takiej operacji niektóre węzły w drzewie mogą mieć nieaktualne wartości zmiennych – są to jednak wyłącznie węzły leżące na ścieżkach od zmodyfikowanych liści do korzenia drzewa. Możemy „naprawić” te węzły, odwiedzając je od dołu do góry i obliczając na nowo wartości skojarzonych z nimi zmiennych, co zajmuje czas $O(\log n)$, gdyż taka jest wysokość drzewa.

Po wykonaniu operacji możemy odpowiedzieć na postawione w zadaniu pytanie, sprawdzając, czy którakolwiek z czterech zmiennych w korzeniu ma wartość *prawda*.

Przedstawione rozwiązanie działa w czasie $O(n + m \log n)$ i pamięci $O(n)$. Jego implementacje znajdują się w plikach `kar.cpp`, `kar1.pas` oraz `kar2.cpp`.

W trakcie zawodów żaden z zawodników nie nadesłał poprawnego rozwiązania, które byłoby istotnie różne od powyższego.

Rozwiązania powolne

Najprostsze rozwiązanie po wykonaniu każdej operacji zamiany sprawdza wszystkie 2^n sposobów wyboru elementów par w poszukiwaniu takiego, który utworzy ciąg niemalejący. Takie rozwiązanie działa w czasie $O(mn2^n)$ i dostaje 15 punktów. Implementacja znajduje się w pliku `kars6.cpp`.

Nieco bardziej skomplikowane, ale za to dużo szybsze rozwiązanie również oblicza odpowiedzi na pytania niezależnie po każdej operacji zamiany. Czyni to jednak w czasie liniowym. Przegląda ciąg od lewej do prawej, obliczając dla i-tego wyrazu ciągu liczbę p_i – najmniejszą liczbę, jaką może kończyć się ciąg niemalejący wybrany z początkowego fragmentu wyjściowego ciągu kończącego się i-tym wyrazem. Wartości p_i obliczamy zgodnie z wzorami:

$$p_1 = \min(x_1, y_1),$$
$$p_i = \begin{cases} \min(x_i, y_i) & \text{jeśli } p_{i-1} \leqslant \min(x_i, y_i), \\ \max(x_i, y_i) & \text{jeśli } \min(x_i, y_i) < p_{i-1} \leqslant \max(x_i, y_i), \\ \text{odpowiedź } \texttt{NIE} & \text{w przeciwnym przypadku.} \end{cases}$$

Jeżeli dla pewnego i obie liczby x_i i y_i są mniejsze niż poprzednio obliczone p_{i-1}, to wiemy już, że nie da się wybrać ciągu niemalejącego; możemy więc przerwać algorytm i odpowiedzieć `NIE`. Jeżeli natomiast przejrzymy cały ciąg, nie napotykając powyższego problemu, odpowiedź brzmi `TAK`. Takie rozwiązanie działa w czasie $O(mn)$ i dostaje 38 punktów. Implementacja znajduje się w plikach `kars1.cpp` i `kars4.pas`.

Rozwiązanie wariantu z dodatkowym warunkiem $x_i = y_i$

Zgodnie z treścią zadania, w testach wartych 30 punktów zachodzi dodatkowy warunek $x_i = y_i$. W takiej wersji zadanie można rozwiązać dużo prościej, jednak rozwiązanie to ma niewiele wspólnego z rozwiązaniem oryginalnego problemu.

Z tym dodatkowym warunkiem zadanie sprowadza się do następującego: na danym ciągu liczb wykonujemy operacje zamiany dwóch wyrazów i po każdej takiej operacji musimy stwierdzić, czy ciąg jest posortowany niemalejąco. W rozwiązaniu wystarczy pamiętać liczbę takich wyrazów, które są większe od następnego. Ciąg jest posortowany wtedy i tylko wtedy, gdy liczba ta jest równa zero. Zamiana wyrazów o numerach a i b powoduje zmianę statusu – z „większy od następnego” na „nie większy od następnego” lub na odwrót – co najwyżej czterech wyrazów: $a-1$, a, $b-1$ oraz b. Łatwo więc zaktualizować pamiętaną liczbę w czasie stałym.

Przedstawione rozwiązanie działa w czasie $O(n+m)$ i jest zaimplementowane w pliku `karb1.cpp`.

Testy

Przygotowano 13 zestawów testów. Zestawy, w których zachodzi dodatkowy warunek $x_i = y_i$, nazywane są dalej *jednostronnymi*, zaś pozostałe *dwustronnymi*. Testy były tworzone z użyciem następujących technik.

Testy sortujące: Dla każdej karty wybierana jest jedna z dwóch stron, a następnie przez kolejne zamiany ciąg jest sortowany tak, aby był niemalejący względem wartości kart na tych stronach. Sortowanie przebiega jednym z klasycznych algorytmów: przez selekcję, wstawianie, kopcowanie lub sortowanie szybkie. Operacja ta jest powtarzana kilkukrotnie, z różnym wyborem stron kart. W testach sortujących dominują zdecydowanie odpowiedzi `NIE`. W każdym zestawie znajdują się dwa testy sortujące – z małym i dużym zakresem wartości kart – odpowiednio testy a i b.

Testy jednostronne specjalne: Znajdują się w zestawach jednostronnych, jako testy c. Składają się w większości z zamian jednakowych kart, a także krótkich ciągów wracających do punktu wyjścia – wszystko po to, aby mieć stosunkowo dużo odpowiedzi `TAK`.

Testy typu „flipper”: Znajdują się w zestawach dwustronnych, jako testy c. Zamieniają ze sobą tylko trzy karty – początkową, środkową i końcową. Test jest tak skonstruowany, aby zamiany powodowały konieczność obrócenia dużej liczby pozostałych kart.

Testy permutacyjne: Znajdują się w zestawach dwustronnych, jako testy d. Ciągi wartości po jednej i po drugiej stronie kart stanowią permutację $\{1, 2, \ldots, n\}$; zamiany dokonywane są po cyklach permutacji.

W zestawie 1 znajduje się dodatkowo ręcznie utworzony test e.