Bartosz Walczak
Treść zadania, Opracowanie

Jakub Pawlewicz
Program

Turniej

Światowa Federacja Gry X organizuje turniej programów grających w tę grę. Startuje w nim n programów ponumerowanych od 1 do n. Zasady rozgrywania turnieju są następujące: dopóki w turnieju pozostaje więcej niż jeden program, losowane są dwa różne spośród nich, które rozgrywają partię gry X. Przegrany (w grze X nie ma remisów) odpada z turnieju, po czym cała procedura jest powtarzana. Program, który pozostanie w turnieju jako jedyny po rozegraniu wszystkich $n-1$ gier, zostaje zwycięzcą.

Federacja dysponuje tabelą wyników poprzednich turniejów. Wiadomo, że programy grają deterministycznie (tzn. w powtarzalny sposób) i tak samo jak w poprzednich turniejach. Zatem jeżeli pewne dwa programy już kiedyś ze sobą grały, to na pewno ich kolejna gra zakończy się tak samo. Jeśli jednak dwa programy jeszcze nigdy nie walczyły ze sobą, rezultatu rozgrywki nie da się przewidzieć — oba mają szansę na wygraną. Federacja chciałaby poznać listę wszystkich tych programów, które mają szansę na wygranie turnieju.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia liczbę uczestniczących programów oraz tabelę wyników ich wcześniejszych gier,*
- *wyznaczy wszystkie programy, które mają szansę wygrać turniej,*
- *wypisze wynik na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera liczbę całkowitą n, $1 \leqslant n \leqslant 100\,000$. Kolejnych n wierszy zawiera tabelę wyników wcześniejszych gier: $i+1$-szy wiersz zawiera liczbę całkowitą k_i, $0 \leqslant k_i < n$, a następnie k_i numerów programów, różnych od i, podanych w kolejności rosnącej — są to numery programów, z którymi program nr i już kiedyś wygrał. Liczby w wierszach są pooddzielane pojedynczymi odstępami. Liczba wszystkich znanych wyników wcześniejszych gier nie przekracza $1\,000\,000$.

Wyjście

Pierwszy i jedyny wiersz standardowego wyjścia powinien zawierać liczbę w wszystkich programów, które mają szansę wygrać turniej, a następnie w liczb będących numerami tych programów, podanymi w kolejności rosnącej. Liczby w wierszu powinny być pooddzielane pojedynczymi odstępami.

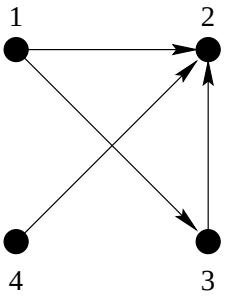
Przykład

Dla danych wejściowych:

```
4
2 2 3
0
1 2
1 2
```

poprawnym wynikiem jest:

```
3 1 3 4
```



Rozwiązanie

Niech $V = \{1, 2, \ldots, n\}$ oznacza zbiór wszystkich programów (każdy program jest reprezentowany przez swój numer). Przez $Win(v)$ oznaczamy zbiór wszystkich programów, z którymi program v na pewno wygra, czyli z którymi już wygrywał we wcześniejszych grach. Piszemy $u \rightsquigarrow v$, gdy $u \neq v$ i $u \notin Win(v)$, czyli gdy u może wygrać partię z v. Zatem jeśli tylko $u \neq v$, to $u \rightsquigarrow v$ lub $v \rightsquigarrow u$.

Każdy program, który może wygrać turniej, nazywamy *programem wygrywającym*. Pozostałe programy to *programy przegrywające*. Zadanie polega na znalezieniu zbioru wszystkich programów wygrywających.

Rozwiązanie zadania opiera się na następującej obserwacji: jeżeli v jest programem wygrywającym oraz $u \rightsquigarrow v$, to u także jest programem wygrywającym. Rozważmy bowiem jakiś przebieg turnieju, w którym v wygrał. Wtedy u musiał przegrać jakąś partię. Rozważmy teraz inny możliwy przebieg turnieju, który różni się od poprzedniego tylko tym, że nie jest rozgrywana partia, w której u przegrał. Wówczas po rozegraniu wszystkich pozostałych partii w turnieju pozostają v i u. Wtedy u może wygrać z v, wygrywając tym samym cały turniej.

Załóżmy, że W jest podzbiorem zbioru wszystkich programów wygrywających. Niech L będzie częścią wspólną zbiorów $Win(v)$ po wszystkich $v \in W$. Jeżeli $u \notin L$, to istnieje program $v \in W$, taki że $u \notin Win(v)$. Wtedy $u \rightsquigarrow v$ lub $u = v$, więc u jest programem wygrywającym. Zatem L jest nadzbiorem zbioru wszystkich programów przegrywających. Ponieważ zawsze $v \notin Win(v)$, zbiory W i L są rozłączne. Jeżeli $W \cup L = V$, to każdy program z W wygrywa z każdym programem spoza W. Wtedy już żaden program spoza W nie może wygrać turnieju, czyli W jest całym zbiorem programów wygrywających. Natomiast jeżeli istnieje $u \notin W \cup L$, to w szczególności $u \notin L$, więc u jest również programem wygrywającym. Wtedy możemy powiększyć zbiór W, dodając do niego u. Ta obserwacja prowadzi nas do następującego schematu algorytmu:

1. Znajdź dowolny program v, który może wygrać turniej. Przypisz $W := \{v\}$, $L := Win(v)$.
2. Dopóki $W \cup L \neq V$, powtarzaj następującą operację: weź dowolny $u \notin W \cup L$, a następnie przypisz $W := W \cup \{u\}$, $L := L \cap Win(u)$.

Po jego zakończeniu W jest szukanym zbiorem programów wygrywających.

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe składa się z dwóch faz. Pierwsza faza wstępnie przetwarza zbiór programów V, dzięki czemu łatwiejsza staje się realizacja drugiej fazy. Druga faza znajduje zbiór programów wygrywających, mniej więcej implementując powyższy schemat.

Pierwsza faza polega na uporządkowaniu zbioru V w ciąg $(v_1, v_2, \ldots, v_n)$, taki że $v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_n$. Takie uporządkowanie zawsze istnieje, a można je wygenerować, stosując odpowiednio zmodyfikowany algorytm sortowania przez wstawianie:

1. Wybierz dowolny $v_1 \in V$.
2. Dla $i = 1, 2, \ldots, n-1$ powtarzaj następujące operacje:
   (a) Wybierz dowolny $v \in V$, który jeszcze nie występuje w ciągu $(v_1, v_2, \ldots, v_i)$.
   (b) Znajdź największą pozycję j w ciągu $(v_1, v_2, \ldots, v_i)$, taką że $v_j \notin \mathit{Win}(v)$, czyli $v_j \rightsquigarrow v$ (jeśli takie j nie istnieje, to można przyjąć $j = 0$).
   (c) Wstaw v do ciągu $(v_1, v_2, \ldots, v_i)$ bezpośrednio za v_j, czyli przypisz $(v_{j+1}, v_{j+2}, \ldots, v_{i+1}) := (v, v_{j+1}, \ldots, v_i)$.

Łatwo się przekonać, że powyższy algorytm rzeczywiście znajduje uporządkowanie $v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_n$. Załóżmy bowiem, że przed wykonaniem kroku pętli zachodzi $v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_i$. Wybieramy największe takie $j \leqslant i$, że $v_j \rightsquigarrow v$. Dlatego jeżeli $j < i$, to $v_{j+1} \not\rightsquigarrow v$, skąd $v \rightsquigarrow v_{j+1}$. Tym samym zachodzi $v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_j \rightsquigarrow v \rightsquigarrow v_{j+1} \rightsquigarrow \ldots \rightsquigarrow v_i$. Zatem po przypisaniu w kroku (c) otrzymujemy $v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_{i+1}$.

Pewnego komentarza wymaga operacja wykonywana w kroku (b) pętli. Chcąc znaleźć odpowiednią pozycję j, możemy jakoś oznaczyć wszystkie te programy, które należą do $\mathit{Win}(v)$, a następnie szukać „od tyłu" największej takiej pozycji $j \leqslant i$, że v_j nie został oznaczony. Programy można oznaczać np. numerem iteracji i. Wtedy w dalszych iteracjach pętli takie oznaczenia nie będą już aktualne. Czas pojedynczego przeszukiwania z zastosowaniem tej metody ogranicza się przez $O(|\mathit{Win}(v)|)$, a więc czas wykonania całej pierwszej fazy wynosi $O(n+m)$, gdzie $m = \sum_{v=1}^{n} |\mathit{Win}(v)|$.

Mając dane uporządkowanie $v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_n$, możemy przejść do drugiej fazy rozwiązania:

1. Przypisz $k := 1$.
2. Dla $i := 1, 2, \ldots$, dopóki $i \leqslant k$, powtarzaj następujące operacje:
   (a) Znajdź największą pozycję j w ciągu $(v_1, v_2, \ldots, v_n)$, taką że $v_j \notin \mathit{Win}(v_i)$, czyli $v_j \rightsquigarrow v_i$ lub $j = i$.
   (b) Przypisz $k := \max\{k, j\}$.

Po zakończeniu powyższego algorytmu szukanym zbiorem programów wygrywających jest $\{v_1, v_2, \ldots, v_k\}$.

Załóżmy, że przed wykonaniem kroku pętli $L = \{v_{k+1}, v_{k+2}, \ldots, v_n\}$ jest nadzbiorem zbioru wszystkich programów przegrywających. Ponieważ $i \leqslant k$, v_i jest programem wygrywającym. Jeżeli $v_j \rightsquigarrow v_i$, to v_j również jest wygrywający. Co więcej, mamy $v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_j$, więc wszystkie programy $v_1, v_2, \ldots, v_j$ są wygrywające. Zatem

$L \cap \{v_{j+1}, v_{j+2}, \ldots, v_n\}$ nadal jest nadzbiorem zbioru wszystkich programów przegrywających, stąd przypisanie w kroku (b). Pętla się kończy wykonywać, gdy $i > k$. Wtedy każdy program ze zbioru $V \setminus L$ wygrywa z każdym ze zbioru L, więc $V \setminus L = \{v_1, v_2, \ldots, v_k\}$ istotnie jest zbiorem wszystkich programów wygrywających.

Widzimy, że faza druga działa analogicznie do przedstawionego wcześniej schematu rozwiązania. Wystarczy przyjąć $W = \{v_1, v_2, \ldots, v_{i-1}\}$. Różnica tkwi w sposobie aktualizacji zbioru L, przez co może on być istotnie mniejszy od części wspólnej zbiorów $Win(v)$, po $v \in W$.

Przeszukiwanie w kroku (a) odbywa się analogicznie do tego w kroku (b) pierwszej fazy i działa w czasie $O(|Win(v_i)|)$. Zatem czas wykonania całej drugiej fazy wynosi $O(n+m)$. Tym samym całe rozwiązanie wzorcowe działa w czasie $O(n+m)$, czyli liniowym względem rozmiaru wejścia.

Powyższe rozwiązanie zostało zaimplementowane w `tur.c` oraz `tur1.pas`.

Rozwiązanie alternatywne

Inne optymalne rozwiązanie można uzyskać, bezpośrednio implementując podany na początku schemat algorytmu.

Zbiór L jest pamiętany na liście, w której programy występują w kolejności rosnących numerów. Programy, które nie występują w żadnym ze zbiorów W i L, przechowujemy w kolejce Q.

Na początku do kolejki Q wstawiamy programy (co najmniej jeden), o których wstępnie stwierdzamy, że są wygrywające. Takie programy można znaleźć np. poprzez symulację turnieju. Wszystkie programy, których nie wstawiliśmy do Q, umieszczamy w liście L.

W pojedynczym kroku pętli pobieramy z kolejki Q jakiś program wygrywający v. Obliczamy $L := L \cap Win(v)$, przeglądając obie listy równolegle w kolejności rosnących numerów i wybierając tylko te programy, które występują w obu listach. Programy, które są w ten sposób usuwane z listy L, są wygrywające, więc dodajemy je do kolejki Q. Algorytm się kończy, gdy kolejka Q jest pusta. Wtedy wszystkie programy nie występujące w L są wygrywające.

Rozwiązanie alternatywne zostało zaimplementowane w `turv1.cpp`.

Testy

Rozwiązania były oceniane na zestawie 16 testów. Testy 1a i 1b, 2a i 2b, 3a i 3b oraz 5a i 5b były pogrupowane w pary. W poniższym zestawieniu n jest liczbą programów, m liczbą znanych wyników partii, a w liczbą programów wygrywających.

nr testu	n	m	w	opis
1a	6	12	4	prosty losowy poprawnościowy
1b	1	0	1	przypadek szczególny $n = 1$
2a	10	40	6	prosty losowy poprawnościowy
2b	2	1	1	przypadek szczególny $n = 2$ i jeden zwycięzca
3a	20	153	4	losowy poprawnościowy
3b	20	0	20	przypadek szczególny $n = 20$ i żadnych krawędzi

nr testu	*n*	*m*	*w*	opis
4	134	5089	71	losowy poprawnościowy
5a	1018	11059	1016	losowy poprawnościowo–wydajnościowy
5b	100000	10	100000	wydajnościowy, bardzo mało krawędzi
6	1230	130141	1170	losowy poprawnościowo–wydajnościowy
7	1790	220000	70	losowy poprawnościowo–wydajnościowy
8	4055	530141	65	losowy poprawnościowo–wydajnościowy
9	100000	1000000	99995	losowy wydajnościowy, dużo zwycięzców
10	100000	1000000	4	losowy wydajnościowy, mało zwycięzców
11	1414	998990	2	strukturalny, wydajnościowy na pierwszą fazę
12	1415	998992	1414	strukturalny, wydajnościowy na drugą fazę

Zawody III stopnia

opracowania zadań