Jakub Radoszewski
Treść zadania, Opracowanie

Marek Cygan
Program

OI, Etap III, dzień próbny, 5–04–2005

Dziuple

W Bajtocji rosną dwa bardzo wysokie pionowe drzewa, a w każdym z nich są wydrążone jedna pod drugą dziuple dla ptaków. Pewnego dnia w dziuplach postanowiło zamieszkać n bardzo szybkich ptaszków. Niektóre ptaszki znają się wzajemnie, więc po wprowadzeniu się chciałyby mieć możliwość odwiedzania się nawzajem w swoich dziuplach. Ptaszki latają bardzo szybko i zawsze po liniach prostych. Chcąc uniknąć niebezpieczeństwa zderzenia postanowiły rozlokować się w dziuplach w taki sposób, żeby:

- *każde dwa ptaszki, które chciałyby się odwiedzać, mieszkały na różnych drzewach oraz*
- *dla dowolnych dwóch par znajomych ptaszków, odcinki łączące dziuple znajomych ptaszków nie przecinały się (co najwyżej mogą mieć wspólny koniec).*

Dodatkowo, ptaszki chcą mieszkać jak najniżej. Na każdym z drzew zajmują więc pewną liczbę dolnych dziupli. W każdym z drzew jest więcej dziupli niż wszystkich ptaszków.

Jak wiadomo, ptaszki mają niewielkie rozumki. Dlatego też poprosiły Cię — znanego ornitologa — o pomoc w sprawdzeniu, na ile różnych sposobów mogą rozlokować się w dziuplach.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis relacji znajomości wśród ptaszków,*
- *obliczy, na ile różnych sposobów można rozmieścić ptaszki w dziuplach spełniając podane powyżej wymagania,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu wejścia zapisano trzy liczby całkowite n, m oraz k, oznaczające odpowiednio: liczbę ptaszków, liczbę różnych par ptaszków znających się nawzajem oraz liczbę której należy użyć przy podawaniu wyniku (por. p. Wyjście), $2 \leqslant n \leqslant 1\,000\,000$, $1 \leqslant m \leqslant 10\,000\,000$, $2 \leqslant k \leqslant 2\,000\,000$. Ptaszki są ponumerowane od 1 do n. W kolejnych m wierszach podane są pary znających się nawzajem ptaszków, po jednej parze w wierszu. W $(i+1)$-szym wierszu są zapisane dwie liczby całkowite a_i i b_i oddzielone pojedynczym odstępem, $1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$. Są to numery znajomych ptaszków. Każda (nieuporządkowana) para znajomych ptaszków jest podana dokładnie raz.

Wyjście

Niech r będzie liczbą różnych rozmieszczeń ptaszków w dziuplach, spełniających podane warunki. Twój program powinien wypisać w pierwszym wierszu wyjścia jedną liczbę całkowitą: resztę z dzielenia r przez k. Jeżeli nie istnieje szukane rozmieszczenie ptaszków, to poprawnym wynikiem jest 0.

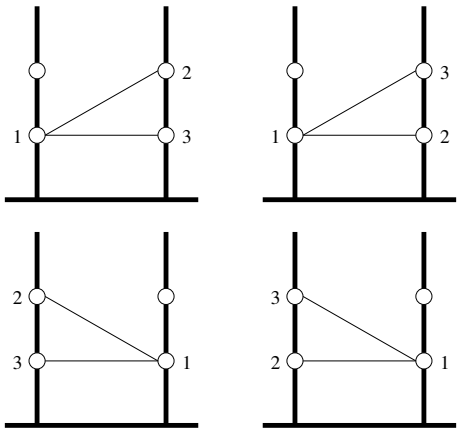
Przykład

Dla danych wejściowych:

```
3 2 10
1 2
1 3
```

poprawnym wynikiem jest:

```
4
```



Rozwiązanie

Grafy pladzielne

Rozpoczniemy od sformułowania problemu z zadania w języku teorii grafów. W tym celu zbudujemy *graf znajomości*, w którym wierzchołki to ptaszki. Wierzchołki połączymy w grafie krawędzią (nieskierowaną), jeśli odpowiadające im ptaszki znają się wzajemnie. Naszym zadaniem jest rozmieszczenie grafu na dwóch drzewach — ponieważ słowo drzewo ma swoje znaczenie w języku grafów, więc by nie mylić drzew prawdziwych z drzewami-grafami przyjmiemy od tej pory, że drzewa z dziuplami, w których chcą osiedlić się ptaszki to *lipy*.

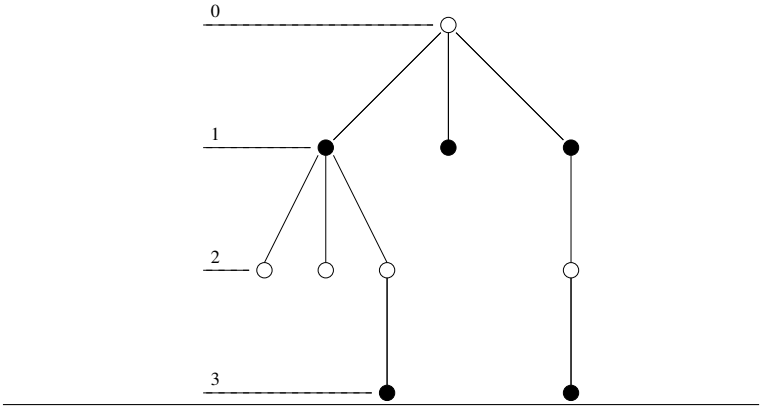
Definicja 1 Łatwo zauważyć, że nie każdy graf da się rozmieścić na dwóch lipach. Aby było to możliwe, graf musi być:

- dwudzielny — wierzchołki (ptaszki) trzeba podzielić na dwa zbiory (lipy) tak, by nie było krawędzi łączących wierzchołki należące do tego samego zbioru (tzn. ptaszki mieszkające na tej samej lipie nie mogą się znać i odwiedzać);
- w pewien sposób planarny — graf trzeba narysować w ten sposób, że wierzchołki z każdego zbioru (lipy) są rozmieszczone na prostych równoległych, krawędzie są narysowane jako odcinki i w takiej reprezentacji żadne dwie krawędzie nie przecinają się poza wierzchołkami.

Na potrzeby opisu rozwiązania graf spełniający powyższe warunki nazwiemy *grafem pladzielnym*.

Spróbujmy scharakteryzować grafy pladzielne dokładniej. Nietrudno zauważyć, że graf pladzielny nie może zawierać cykli — w przeciwnym wypadku nie dałoby się go narysować bez przecinających się krawędzi. Dlatego musi być *lasem* (w sensie grafowym — czyli grafem acyklicznym), a więc każda jego spójna składowa musi być *drzewem* (czyli acyklicznym grafem spójnym).

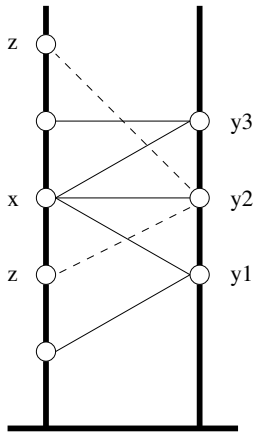
Sprawdźmy, czy każde drzewo może być składową spójności grafu pladzielnego. Każde drzewo jest grafem dwudzielnym. Wystarczy wybrać dowolny wierzchołek w grafie i nazwać go *korzeniem* (w ten sposób *ukorzeniamy* drzewo), a następnie pokolorować na biało wszystkie wierzchołki, których odległość od korzenia jest parzysta, i na czarno pozostałe (takie kolorowanie jest przedstawione na rysunku 1). W ten sposób dzielimy wierzchołki drzewa na dwie grupy takie, że wewnątrz żadnej grupy nie ma krawędzi, co dowodzi dwudzielności tego grafu[1].



Rys. 1: Dwukolorowanie drzewa pozwalające wykazać, że drzewo jest dwudzielne.

Niestety, nie każde drzewo jest grafem pladzielnym. Zauważmy, że problem stwarza sytuacja, gdy jeden wierzchołek x ma trzech sąsiadów y_1, y_2 i y_3, z których każdy ma stopień (czyli liczbę sąsiadów) co najmniej dwa (patrz rysunek 2). Rozlokowując pladzielnie takie wierzchołki na dwóch lipach musimy umieścić x na jednej lipie, a y_1, y_2, y_3 na drugiej — przypuśćmy, że rozmieścimy je w takim właśnie porządku: y_1 najniżej, y_2 pośrodku, a y_3 najwyżej. Niech z będzie sąsiadem y_2 różnym od x. Wierzchołek z musimy umieścić na tej samej lipie co x — i tu mamy problem. Jeśli umieścimy go poniżej x, to krawędź (y_2, z) będzie przecinać się z (y_1, x). Jeśli umieścimy go powyżej x, to wtedy krawędź (y_2, z) będzie przecinać się z (y_3, x). Można sprawdzić, że zmiana porządku y_1, y_2, y_3 nic tu nie pomoże.

[1] Więcej informacji na temat własności drzew i lasów można znaleźć na przykład w książce [26].



Rys. 2: Próby rozmieszczenia drzewa niepladzielnego na dwóch lipach.

W ten sposób doszliśmy do wniosku, że aby drzewo było pladzielne, jego wierzchołki stopnia większego niż jeden (nazywane *wierzchołkami wewnętrznymi*) muszą tworzyć ścieżkę — nazwiemy ją *osią* tego drzewa.

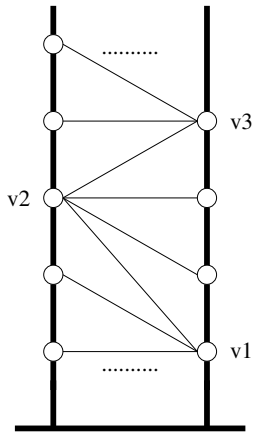
Lemat 1 *Drzewa, w których wierzchołki wewnętrzne tworzą jedną ścieżkę są jedynymi spójnymi grafami pladzielnymi.*

Dowód Pokazaliśmy już, że inne spójne grafy nie są pladzielne. Pozostaje pokazać, że opisane drzewo da się rozmieścić na dwóch lipach. Niech $(v_1, v_2, \ldots, v_t)$ będzie osią drzewa. Pozostałe wierzchołki drzewa — o stopniu jeden (nazywane *liśćmi*) — można podzielić na zbiory sąsiadów wierzchołków z osi: $S(v_1), S(v_2), \ldots, S(v_t)$. Widzimy, że drzewo można rozmieścić planarnie na dwóch lipach. Wystarczy:

- wierzchołki osi o nieparzystych indeksach $(v_1, v_3, \ldots)$ umieścić kolejno od dołu na jednej lipie;
- wierzchołki osi o parzystych indeksach $(v_2, v_4, \ldots)$ umieścić kolejno od dołu na drugiej lipie;
- sąsiadów v_i, czyli $S(v_i)$, umieścić na przeciwnej lipie niż v_i pomiędzy v_{i-1} oraz v_{i+1}.

■

W ten sposób stworzyliśmy pełną charakteryzację składowych grafu pladzielnego.



Rys. 3: Rozmieszczenie drzewa pladzielnego na dwóch lipach.

Zliczanie rozmieszczeń ptaszków w dziuplach

Wiemy już jak rozpoznać graf pladzielny i jak rozmieścić go na lipach przynajmniej na jeden sposób. Przystąpmy teraz do policzenia wszystkich możliwości.

Podzielmy nasz graf na spójne składowe — wiemy już, że są to drzewa pladzielne. Teraz osobno rozpatrzymy *duże* spójne składowe mające więcej niż jeden wierzchołek i co najmniej jedną krawędź (powiedzmy, że jest to p pierwszych składowych: $T_1, T_2, \ldots, T_p$) i *małe* składowe — wierzchołki izolowane (powiedzmy, że jest to s kolejnych składowych $X_1, X_2, \ldots, X_s$).

Rozmieszczenie dużej składowej

Rozpoczniemy od wyznaczenia liczby możliwych rozmieszczeń składowej T_i na dwóch lipach. Niech $(v_1, v_2, \ldots, v_t)$ będzie osią drzewa, a $S_1, S_2, \ldots, S_t$, zbiorami sąsiadów stopnia jeden kolejnych wierzchołków z osi. Zauważmy, że:

- jeśli $t = 1$ to oś drzewa można rozmieścić na lipach na dwa sposoby: $(\{v_1\}, \emptyset)$ oraz $(\emptyset, \{v_1\})$;
- także dla $t = 2$ istnieją dwa sposoby rozmieszczenia osi na lipach: $(\{v_1\}, \{v_2\})$ oraz $(\{v_2\}, \{v_1\})$;
- dla $t > 2$ są już cztery sposoby rozmieszczenia osi na lipach: $(\{v_1, v_3, \ldots\}, \{v_2, v_4 \ldots\})$ i $(\{v_2, v_4, \ldots\}, \{v_1, v_3, \ldots\})$ oraz $(\{v_t, v_{t-2}, \ldots\}, \{v_{t-1}, v_{t-3} \ldots\})$ i $(\{v_{t-1}, v_{t-3}, \ldots\}, \{v_t, v_{t-2}, \ldots\})$, gdzie w zbiorach wypisujemy wierzchołki począwszy od najniżej umieszczonych na lipie.

Po rozmieszczeniu wierzchołków osi, wierzchołki sąsiadujące z v_j możemy dowolnie rozmieścić pomiędzy v_{j-1} oraz v_{j+1}. Czyli możemy je rozlokować na $|S_j|!$ sposobów.

Stąd jedną dużą składową możemy rozmieścić na

$$L(T_i) \quad = \quad c_t \cdot |S_1|! \cdot |S_2|! \cdot \cdots \cdot |S_t|! \tag{1}$$

sposobów, gdzie $c_t = 2$ dla $t \leqslant 2$ i $c_t = 4$ dla $t > 2$.

Wzajemne rozmieszczenie dużych składowych

Dla każdej pary składowych T_i, T_j ($1 \leqslant i < j \leqslant p$) są możliwe tylko dwa ich wzajemne położenia na lipach. Albo składowa T_i leży (cała!) poniżej składowej T_j, albo składowa T_j leży (także cała) poniżej składowej T_i. Jakiekolwiek zaburzenie tego porządku powoduje, że krawędzie z tych składowych przecinają się i rozmieszczenie przestaje być pladzielne. Stąd istnieje

$$p! \tag{2}$$

możliwości wzajemnego rozmieszczenia dużych składowych.

Rozmieszczenie małych składowych

Małe składowe możemy rozmieszczać dowolnie (ptaszki nie mające znajomych, nie będą nikomu przeszkadzać). Jeśli na pierwszej lipie mamy już x wierzchołków, a na drugiej y, to składową X_i możemy umieścić na $x+1+y+1$ sposobów. Tak więc jeśli przez w oznaczymy liczbę wierzchołków we wszystkich dużych składowych, to pierwszą małą składową X_1 możemy umieścić na $w+2$ sposobów, składową X_2 na $w+3$ sposobów,... Ogólnie, składową X_i możemy umieścić na $(w+2+(i-1))$ sposobów. Stąd liczba rozmieszczeń małych składowych w każdym rozmieszczeniu dużych składowych wynosi

$$W \quad = \quad (w+2) \cdot (w+3) \cdot \cdots \cdot (w+2+s-1) \tag{3}$$

Ze wzorów (1), (2) oraz (3) widzimy, że liczba wszystkich rozmieszczeń ptaszków na lipach wynosi:

$$r \quad = \quad L(T_1) \cdot L(T_2) \cdot \cdots \cdot L(T_p) \cdot p! \cdot W$$

i może być bardzo duża. Dlatego w zadaniu wystarczy podać tylko resztę z dzielenia r przez k (tak dużej liczby ptaszki i tak z pewnością nie mogłyby ogarnąć swoimi niewielkimi rozumkami), stąd wszystkie obliczenia można wykonywać na liczbach całkowitych w standardowej reprezentacji.

Rozwiązanie wzorcowe

Schemat algorytmu rozwiązania wzorcowego jest następujący.

1. Wczytujemy wartości m i n, a następnie sprawdzamy, czy $m < n$. Jeśli nie, to graf z pewnością nie jest acykliczny, czyli nie może być także pladzielny. Nie musimy więc wczytywać krawędzi grafu — możemy od razu wypisać odpowiedź 0 i skończyć.
2. Jeśli $m < n$, to wczytujemy opis krawędzi grafu. Następnie sprawdzamy, czy jest to graf pladzielny:
   (a) czy jest acykliczny (tzn. nie zawiera cyklu) oraz
   (b) czy każdy wierzchołek ma co najwyżej dwóch sąsiadów o stopniu większym niż jeden.

3. Jeśli wykryjemy, że graf nie jest pladzielny, to wypisujemy odpowiedź 0 i kończymy.
4. Dzielimy graf na spójne składowe (właściwie można to zrobić przy okazji sprawdzania acykliczności w punkcie 2(a)).
5. Następnie obliczamy stopniowo wynik:
   (a) dla dużych składowych T obliczamy wartości $L(T)$ (patrz wzór (1)) i wymnażamy je;
   (b) domnażamy przez liczbę permutacji dużych składowych (patrz wzór (2));
   (c) domnażamy przez liczbę położeń małych składowych wśród dużych (patrz wzór (3)).

Sprawdzanie acykliczności grafu i znajdowanie spójnych składowych (kroki 2(a) i 4) wykonujemy za pomocą algorytmu przeszukiwania w głąb (DFS, patrz na przykład [17]), który działa w czasie $O(n+m)$, co w naszym przypadku jest równe $O(n)$. Oddzielnie obliczamy stopnie wierzchołków grafu i sprawdzamy drugi warunek pladzielności dla każdej składowej. Całość nadal działa w czasie $O(n)$ i wymaga pamięci $O(n+m) = O(n)$.

Wszystkie obliczenia w kroku 5 są wykonywane modulo k na liczbach całkowitych 64-bitowych (typ Int64 w Pascalu i long long w C/C++), aby uniknąć przekroczeń zakresu typów całkowitych.

Opisane rozwiązanie znajduje się na dysku dołączonym do książeczki, zaimplementowane w języku Pascal (`dzi.pas`) i C++ (`dzi1.cpp`).

Alternatywna metoda szukania spójnych składowych

Sprawdzanie acykliczności i wyznaczanie spójnych składowych, które w rozwiązaniu wzorcowym były wykonywane za pomocą algorytmu DFS, można zrobić przy pomocy struktury danych dla zbiorów rozłącznych, tzw. „Find & Union” (patrz na przykład [17]).

1. Początkowo z każdego wierzchołka v grafu tworzymy zbiór jednoelementowy S_v (w algorytmie w jednym zbiorze będziemy umieszczać wierzchołki, o których już wiemy, że są połączone w grafie i należą do jednej składowej spójności).
2. Następnie przeglądamy kolejno wszystkie krawędzie grafu i dla każdej krawędzi (u,v):
   (a) szukamy zbioru U, do którego należy wierzchołek u;
   (b) szukamy zbioru V, do którego należy wierzchołek v;
   (c) jeśli $U = V$, to widzimy, że w grafie jest cykl (bo już wcześniej znaleźliśmy połączenie pomiędzy u i v) i kończymy odpowiadając, że graf nie jest pladzielny;
   (d) w przeciwnym razie łączymy zbiory U i V w jeden (wiemy już, że ich wierzchołki należą do tej samej składowej).
3. Na zakończenie mamy zbiory $V_1, V_2, \ldots$ zawierające wierzchołki poszczególnych składowych grafu.

4. W trakcie przeglądania krawędzi możemy przy okazji obliczać stopnie wierzchołków grafu; wówczas po wyznaczeniu składowych możemy sprawdzić pladzielność drzew-składowych.

Pozostaje jeszcze opisać strukturę danych dla zbiorów wierzchołków, która umożliwi sprawne wykonywanie wszystkich powyższych operacji. Ale tutaj już odeślemy Czytelnika do literatury (patrz na przykład [12] lub [17]). Opisana tam implementacja pozwala wykonać przedstawiony algorytm w czasie $O(n \cdot \log^* n)$. Funkcja $\log^* n$ to tak zwany logarytm iterowany, który definiujemy jako

$$\log^* n = \min\{i \geqslant 0 : \log^{(i)} n \leqslant 1\}$$

gdzie $\log^{(i)}$ to *i*-krotne złożenie funkcji logarytm dwójkowy ze sobą. Logarytm iterowany jest funkcją *bardzo, bardzo* wolno rosnącą:

$$\begin{aligned} \log^* 2 &= 1 \\ \log^* 4 &= 2 \\ \log^* 16 &= 3 \\ \log^* 65536 &= 4 \\ \log^*(2^{65536}) &= 5 \end{aligned}$$

więc dla danych n z rozważanego zakresu $log^*(n)$ nie przekracza 5 i rozwiązanie działa praktycznie w czasie liniowo zależnym od n.

Rozwiązanie to znajduje się na dysku dołączonym do książeczki, zaimplementowane w języku Pascal (`dzi2.pas`) i C++ (`dzi3.cpp`).

Rozwiązania niepoprawne

Błędy najczęściej występujące w rozwiązaniach zawodników, to:

- używanie procedur rekurencyjnych o dużej liczbie parametrów i zmiennych lokalnych (zwłaszcza algorytmu DFS) — dla niektórych danych (grafów o długich ścieżkach) zagłębienie rekursji było tak duże, że programy przekraczały założone limity pamięci; aby uniknąć takich błędów należało zaprogramować rekurencję na własnym stosie;
- wczytywanie za każdym razem całego grafu do pamięci, co w przypadku $m \geqslant 1\,000\,000$ było zbyt kosztowne pamięciowo (i czasowo);
- pomijanie pewnych możliwości rozmieszczenia ptaszków w dziuplach.

Testy

Zadanie było testowane na 10 grupach testów. Grupowanie testów przeciwdziałało uzyskiwaniu punktów przez programy zawsze wypisujące 0.

Nazwa	n	m	Opis
dzi1a.in	4	2	test poprawnościowy
dzi1b.in	3	2	test poprawnościowy
dzi1c.in	4	3	test poprawnościowy
dzi2a.in	7	1	test z wierzchołkami izolowanymi
dzi2b.in	16	8	test z wierzchołkami izolowanymi
dzi3a.in	50 342	50 325	graf zawiera cykle
dzi3b.in	8	5	test poprawnościowy
dzi4a.in	35	31	graf acykliczny, niepladzielny
dzi4b.in	12	11	graf acykliczny, niepladzielny
dzi4c.in	14	11	test poprawnościowy
dzi5.in	100 001	74 695	specyficzny test
dzi6a.in	904 000	5 400 000	wczytywanie wszystkich krawędzi zbędne
dzi6b.in	13	12	test poprawnościowy
dzi7.in	10 040	10 030	test losowy
dzi8a.in	500 012	495 949	duży test losowy
dzi8b.in	700 000	699 995	dwa drzewa o głębokości 350 000
dzi9a.in	1 000 000	990 677	duży test losowy
dzi9b.in	1 000 000	999 995	dwa drzewa o głębokości 500 000
dzi10a.in	1 000 000	999 998	drzewo o głębokości 1 000 000
dzi10b.in	1 000 000	997 659	duży test losowy