Łukasz Kowalik
Treść zadania, Opracowanie

Krzysztof Dulęba
Program

OI, Etap III, dzień drugi, 30.03.2006

Zosia

Mała Zosia urządza przyjęcie urodzinowe. Sporządziła wstępną listę n swoich znajomych z przedszkola, których chciałaby zaprosić. Dzieci są jednak bardzo wymagające. Maja powiedziała, że przyjdzie, ale tylko jeśli na przyjęciu nie będzie Kamilki i Emilki, które w zeszłym tygodniu zabrały jej lalkę. Mały Krzyś bawi się tylko z Zosią oraz Kamilką, i nie chce widzieć na przyjęciu innych dzieci. I tak dalej ...

Zosia mówi, że przyjęcie jest **udane***, jeżeli żaden spośród gości nie ma nic przeciwko obecności pozostałych gości. Zosia postanowiła, że nie zaprosi niektórych dzieci, aby przyjęcie było udane. Z drugiej strony, Zosia chciałaby zaprosić jak najwięcej dzieci. Uznała, że jeśli nie będzie mogła zaprosić przynajmniej k dzieci, to w ogóle nie urządzi przyjęcia.*

Zadanie

Pomóż małej Zosi! Napisz program, który:

- *wczyta ze standardowego wejścia liczbę n wszystkich znajomych Zosi, liczbę k oraz opis wymagań dzieci,*
- *sprawdzi, czy można zaprosić co najmniej k dzieci tak, aby przyjęcie było udane,*
- *jeżeli nie jest to możliwe, to wypisze na standardowe wyjście słowo* `NIE`*; jeżeli jest to możliwe, to znajdzie i wypisze na standardowe wyjście najliczniejszą grupę dzieci, które można zaprosić na przyjęcie, tak aby było ono udane.*

Wejście

Pierwszy wiersz standardowego wejścia zawiera dwie liczby całkowite nieujemne, oddzielone pojedynczym odstępem: n — liczbę wszystkich znajomych Zosi ($2 \leqslant n \leqslant 1\,000\,000$), k — minimalną liczbę dzieci, które Zosia chce zaprosić na przyjęcie ($n - 10 \leqslant k < n$). Dzieci są ponumerowane liczbami od 1 do n.

Kolejne wiersze zawierają opis wymagań dzieci. W drugim wierszu znajduje się jedna liczba całkowita m, $1 \leqslant m \leqslant 3\,000\,000$. Każdy z kolejnych m wierszy zawiera parę liczb całkowitych a,b, oddzielonych pojedynczym odstępem ($1 \leqslant a,b \leqslant n$, $a \neq b$). Możesz założyć, że każda para (uporządkowana) pojawia się na wejściu co najwyżej raz. Para a,b oznacza, że dziecko o numerze a nie chce spotkać na przyjęciu dziecka o numerze b.

Wyjście

Jeżeli nie jest możliwe zaproszenie na przyjęcie k dzieci, tak aby było ono udane, to pierwszy i jedyny wiersz standardowego wyjścia powinien zawierać tylko jedno słowo: `NIE`.

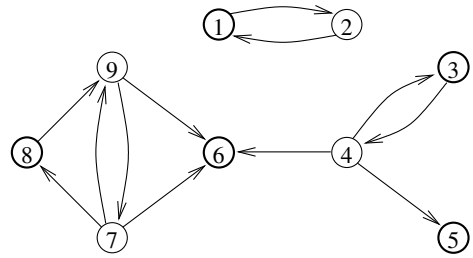
Jeżeli jest to możliwe, to pierwszy wiersz standardowego wyjścia powinien zawierać jedną liczbę całkowitą — maksymalną liczbę dzieci, które można zaprosić na przyjęcie, tak aby było

ono udane. Drugi wiersz standardowego wyjścia powinien wówczas zawierać numery dzieci, które należy zaprosić, podane w rosnącej kolejności i pooddzielane pojedynczymi odstępami. Jeżeli istnieje wiele poprawnych wyników, Twój program powinien wypisać dowolny z nich.

Przykład

Dla danych wejściowych:

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



poprawnym wynikiem jest:

```
5
1 3 5 6 8
```

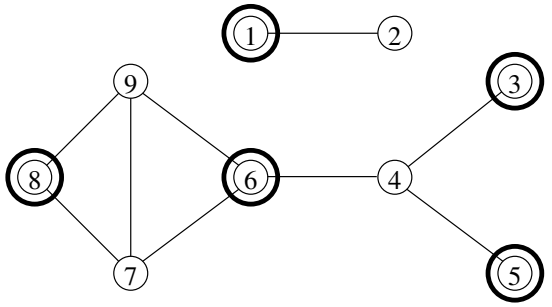
Rozwiązanie

Zosia i grafy

Przyjrzyjmy się rysunkowi z treści zadania. Antypatie wśród przyjaciół Zosi przedstawiono na nim za pomocą grafu skierowanego. *Graf skierowany* to zbiór obiektów (w tym przypadku numerów przyjaciół Zosi) połączonych strzałkami. Strzałka od a do b oznacza, że dziecko o numerze a nie chce spotkać na przyjęciu dziecka o numerze b. Zauważmy od razu, że z punktu widzenia postawionego zadania kierunek strzałki nie ma tu żadnego znaczenia – spośród dwóch osób połączonych strzałką możemy zaprosić tylko jedną, niezależnie od tego, kto kogo nie lubi. Przypomnijmy jeszcze, że w teorii grafów połączenia między obiektami nazywamy *krawędziami*, a same obiekty *wierzchołkami*. Gdy krawędzie nie mają kierunku (możemy je przedstawiać graficznie za pomocą zwykłych linii), mamy do czynienia z grafem nieskierowanym.

Zdefiniujmy *graf antypatii* jako graf nieskierowany, którego wierzchołkami są numery dzieci oraz wierzchołek a jest połączony krawędzią z wierzchołkiem b, jeżeli dziecko o numerze a nie chce widzieć na przyjęciu dziecka o numerze b lub odwrotnie, dziecko o numerze b nie chce widzieć na przyjęciu dziecka o numerze a. Zauważmy, że graf antypatii może różnić się od grafu z treści zadania, w którym po prostu zamieniono strzałki na linie — po takiej prostej modyfikacji niektóre wierzchołki mogą być połączone podwójnie, a takiej

sytuacji nie dopuszczamy w grafie antypatii. Na rysunku 1 przedstawiono graf antypatii dla przykładu z treści zadania.



Rys. 1: Przykładowy graf antypatii i zbiór niezależny

Zosia buduje graf w pamięci komputera

Zajmiemy się teraz zagadnieniem budowy grafu antypatii — zastanowimy się, w jaki sposób można go zbudować w pamięci komputera na podstawie danych wejściowych dla zadania. Czytelnicy znający dobrze algorytmy grafowe mogą pominąć ten rozdział i przeskoczyć do następnego punktu.

Najprostszym sposobem przechowywania informacji o grafie w pamięci jest wykorzystanie kwadratowej tablicy `T` zawierającej zera i jedynki. Jedynka w komórce `T[a,b]` oznacza, że wierzchołki o numerach `a` i `b` są połączone krawędzią, zero — że takiej krawędzi nie ma. Tablicę `T` można bardzo łatwo wypełnić na podstawie danych wejściowych, ma ona jednak wielką wadę: zajmuje zbyt dużo miejsca! Zwróćmy uwagę, że Zosia z zadania jest bardzo towarzyską dziewczynką i może mieć nawet 10^6 przyjaciół z przedszkola, a więc cała tablica miałaby 10^{12} (bilion!) komórek.

Alternatywną metodą przechowywania grafów w pamięci są listy sąsiedztwa. W tej reprezentacji dla każdego wierzchołka tworzymy listę numerów jego sąsiadów. List jest tyle co wierzchołków, a każdej krawędzi grafu odpowiadają dokładnie 2 elementy list sąsiedztwa, a więc wszystkie listy zajmują w pamięci komputera $O(m+n)$ komórek.

Teraz zastanówmy się, jak utworzyć listy sąsiedztwa na podstawie danych wejściowych. Sprawa wydaje się prosta: po przeczytaniu wiersza z parą liczb a i b, dodajemy a do listy wierzchołka b oraz b do listy wierzchołka a. Jest tylko jeden problem: niektóre krawędzie grafu antypatii zapiszemy podwójnie (np. krawędź łączącą 1 i 2 w przykładzie z zadania). Aby tego uniknąć, moglibyśmy, wstawiając a do listy wierzchołka b, sprawdzać, czy a już na tej liście jest. Niestety zajmuje to czas proporcjonalny do długości listy. Gdyby jakiś wierzchołek miał wielu sąsiadów — na przykład był połączony ze wszystkimi pozostałymi $n-1$ wierzchołkami — to takie sprawdzenia zajęłyby czas proporcjonalny do $1+2+\ldots+(n-3)+(n-2)$, czyli $\Theta(n^2)$.

Posłużymy się innym fortelem. Utworzymy listy sąsiedztwa w zwykły sposób, a następnie usuniemy z nich podwójne krawędzie. Aby łatwo takie podwójne krawędzie znaleźć, wystarczy posortować każdą z list sąsiedztwa — wtedy podwójne krawędzie będą występować obok siebie. W językach C/C++ wystarczy użyć jednej z funkcji sortujących ze standardowych bibliotek. Funkcje te są skonstruowane najczęściej

w oparciu o algorytm szybkiego sortowania (*Quicksort*), który działa w oczekiwanym czasie $O(k \log k)$ dla ciągu k-elementowego. Sortowanie wszystkich ciągów, których całkowita długość wynosi $2m$, a maksymalna długość jest ograniczona przez $n-1$, zajmie czas $O(\sum_{i=1}^{n} \deg(i) \log \deg(i))$, gdzie $\deg(i)$ oznacza stopień wierzchołka i, czyli liczbę jego sąsiadów i równocześnie długość jego listy sąsiedztwa. Stąd otrzymujemy: $\sum_{i=1}^{n} \deg(i) \log \deg(i) \leqslant \sum_{i=1}^{n} \deg(i) \log(n-1) = \log(n-1) \sum_{i=1}^{n} \deg(i) = 2m \log(n-1)$ i dowiadujemy się, że całkowity (oczekiwany) czas sortowania wynosi $O(m \log n)$.

Powyższa metoda jest dostatecznie dobra dla ograniczeń z rozwiązywanego zadania, można jednak w prosty sposób posortować listy sąsiedztwa w czasie liniowym. Z tego sposobu mogą także skorzystać osoby programujące w Pascalu, które nie mają dostępu do gotowych funkcji sortujących. Użyjemy sortowania kubełkowego — w tym celu potrzebna będzie dodatkowa tablica list `kubełki[1..n]`. Sam algorytm działa następująco:

1. Utwórz tablicę `kubełki[1..n]` zawierającą puste listy.
2. Dla każdego wierzchołka i, od 1 do n, wykonuj, dopóki lista sąsiedztwa i jest niepusta:
   (a) Usuń wierzchołek z listy sąsiedztwa i. Niech j będzie numerem tego wierzchołka.
   (b) Wstaw i do listy `kubełki[`j`]`.
3. Dla każdego j, od 1 do n, wykonuj, dopóki lista `kubełki[`j`]` jest niepusta:
   (a) Usuń element z listy `kubełki[`j`]`. Niech i będzie tym elementem.
   (b) Wstaw j do listy sąsiedztwa wierzchołka i.

Fakt, że listy sąsiedztwa po wykonaniu algorytmu są posortowane, wynika z kolejności wstawiania elementów do tych list w punkcie 3. Zwróćmy uwagę, że w powyższym algorytmie sortujemy wszystkie listy równocześnie! Sortowanie kubełkowe każdej listy z osobna zajęłoby znacznie więcej czasu.

Zosia i trudne problemy

Dowolny podzbiór wierzchołków grafu, z których żadne dwa nie są połączone krawędzią, nazywamy *zbiorem niezależnym*. Przykładowy zbiór niezależny przedstawiono na rysunku 1 (wierzchołki zbioru niezależnego wyróżniono dodatkową obwódką). Zauważmy, że w naszym zadaniu należy znaleźć zbiór niezależny rozmiaru co najmniej k.

Wprowadźmy jeszcze jedno pojęcie. *Pokryciem wierzchołkowym* grafu nazywamy dowolny podzbiór C wierzchołków grafu taki, że każda krawędź grafu ma co najmniej jeden z końców w zbiorze C (mówimy, że krawędź jest pokryta przez ten wierzchołek). Spójrzmy ponownie na rysunek 1. Wierzchołki niezaznaczone tworzą pokrycie wierzchołkowe. Czy to tylko przypadek? Rozważmy dowolny graf G o zbiorze wierzchołków V i dowolny zbiór niezależny N w grafie G. Wybierzmy dowolną krawędź w grafie G. Z definicji zbioru niezależnego wiemy, że co najmniej jeden z jej końców jest poza N — innymi słowy co najmniej jeden z jej końców należy do zbioru $V \setminus N$. Stąd wynika, że $V \setminus N$ jest pokryciem wierzchołkowym. To samo rozumowanie można przeprowadzić w drugą stronę. Otrzymamy wtedy następujący fakt:

Fakt 1 *W dowolnym grafie G o zbiorze wierzchołków V podzbiór wierzchołków N jest zbiorem niezależnym wtedy i tylko wtedy, gdy zbiór $V \setminus N$ jest pokryciem wierzchołkowym.*

Z powyższego faktu płynie wniosek, że nasze zadanie można sformułować także następująco: „znajdź pokrycie wierzchołkowe rozmiaru co najwyżej $n-k$". Widzimy jasno, że problem znajdowania zbioru niezależnego (pierwsze sformułowanie zadania) jest równoważny problemowi znajdowania pokrycia wierzchołkowego (drugie sformułowanie), tzn. jeśli mamy efektywny sposób rozwiązywania pierwszego problemu, to umiemy też rozwiązywać problem drugi, i odwrotnie.

Problemy zbioru niezależnego i pokrycia wierzchołkowego wydają się tak naturalne, że byłoby bardzo dziwne, gdyby zostały postawione po raz pierwszy na tegorocznej Olimpiadzie Informatycznej. Istotnie, znajdziemy je łatwo w podręcznikach algorytmiki (np. problem pokrycia wierzchołkowego jest opisany w książce Cormena i innych [18]; zamiast problemu zbioru niezależnego jest tam opisany mocno z nim związany problem kliki). Dowiemy się stamtąd, że oba te problemy należą do grupy tzw. problemów *NP-zupełnych*. Nie będziemy tutaj dokładnie opisywać, co to pojęcie oznacza. Wspomnijmy jedynie, że dla żadnego z tych problemów nie znaleziono dotąd algorytmu działającego w czasie wielomianowym (tzn. w czasie $O(n)$, $O(n^3)$, $O(n^{15})$ ani w czasie $O(n^c)$, gdzie c jest dowolną stałą). Co więcej, gdyby ktoś rozwiązał dowolny problem NP-zupełny w czasie wielomianowym, to takiego algorytmu można użyć do rozwiązania w czasie wielomianowym dowolnego innego problemu NP-zupełnego. Ponieważ dotąd znaleziono mnóstwo problemów NP-zupełnych w wielu różnych działach informatyki, a pomimo intensywnych badań ciągle dla nich nie wymyślono algorytmów wielomianowych, więc wydaje się mało prawdopodobne, żeby w końcu taki algorytm został odkryty.

Zosia upraszcza zadanie

Na szczęście w treści zadania znajduje się informacja, która znacznie upraszcza nasz problem. Mianowicie, Zosia godzi się na nie zaproszenie jedynie kilku spośród swoich przyjaciół, bowiem $k \geqslant n-10$ — oznaczmy liczbę tych pechowców przez $l = n-k$. Innymi słowy, zadanie polega na sprawdzeniu, czy graf antypatii zawiera pokrycie wierzchołkowe rozmiaru l, gdzie $l \leqslant 10$. Naiwny algorytm wymagający przejrzenia i przetestowania wszystkich podzbiorów zbioru wierzchołków, które mają nie więcej niż 10 elementów, działa w czasie $O(\sum_{i \leqslant 10} \binom{n}{i}) = O(n^{10})$, a więc wielomianowym! Taka złożoność jest dla nas jednakże zupełnie nieakceptowalna. Aby uzyskać lepsze rozwiązanie, w inny sposób wykorzystamy fakt, że poszukiwane pokrycie wierzchołkowe jest bardzo małe.

Algorytm I: Zosia szybko się zniechęca

Przypomnijmy, że poszukujemy pokrycia wierzchołkowego rozmiaru co najwyżej l. Rozważmy dowolną krawędź uv grafu antypatii. Jeden z jej końców musi znaleźć się w pokryciu. Możemy więc usunąć z grafu wierzchołek u wraz z wszystkimi jego krawędziami, a następnie rekurencyjnie rozwiązać zadanie dla tak otrzymanego mniejszego grafu, poszukując pokrycia wierzchołkowego rozmiaru co najwyżej $l-1$. Jeśli okaże się, że takie pokrycie istnieje, wystarczy dodać do niego wierzchołek u, aby otrzymać pokrycie

całego grafu. Jeśli natomiast nie istnieje, ponownie wstawiamy do grafu wierzchołek u i wszystkie uprzednio usunięte krawędzie. Następnie usuwamy wierzchołek v wraz z przyległymi krawędziami i postępujemy analogicznie, jak dla wierzchołka u. Jeśli i w tym przypadku nie udało się znaleźć pokrycia, oznacza to, że w oryginalnym grafie nie ma pokrycia rozmiaru co najwyżej l — odpowiedź brzmi więc „NIE".

Opisany powyżej algorytm wydaje się mało odkrywczy: jest to tylko pewien sposób sprawdzania wszystkich możliwości. Zauważmy jednak, że przy każdym wywołaniu rekurencyjnym ograniczenie na rozmiar pokrycia zmniejsza się o 1. W chwili, gdy to ograniczenie osiągnie wartość 0, wystarczy sprawdzić, czy graf zawiera *puste* pokrycie wierzchołkowe, a to ma miejsce jedynie wówczas, gdy nie ma w nim już żadnych krawędzi. Taki test można oczywiście łatwo wykonać w czasie liniowym. Całe drzewo wywołań rekurencyjnych jest więc stosunkowo płytkie — ma co najwyżej $l+1$ poziomów, a więc zawiera nie więcej niż 2^l wywołań procedury rekurencyjnej. Ponieważ usuwanie wierzchołka, podobnie jak sprawdzenie, czy zbiór pusty jest pokryciem, zajmuje czas $O(n)$, a więc całkowity czas działania algorytmu to $O(n2^l)$ — pomijając wstępną fazę tworzenia grafu antypatii. Trzeba przyznać, że to znaczna poprawa w stosunku do wcześniejszych rozwiązań. Rozmiar grafu (liczbę wierzchołków n) udało się bowiem usunąć w złożoności z funkcji wykładniczej — nie występuje ona już ani w podstawie, ani w wykładniku.

Algorytm II: Zosia nie zaprasza największych urwisów

Zosia dobrze wie, że na przyjęcie nie należy zapraszać największych urwisów, bo pobiją inne dzieci lub w najlepszym razie zabiorą im zabawki. Podobnie, wierzchołek stopnia większego niż l (czyli taki, który jest połączony z więcej niż l wierzchołkami) musi należeć do pokrycia wierzchołkowego, jeśli rozmiar tego pokrycia nie może przekraczać l. Gdyby bowiem nie należał, to aby pokryć wszystkie wychodzące z niego krawędzie, należałoby wybrać wszystkich jego sąsiadów, a tych jest więcej niż l. Dlatego możemy rozpocząć algorytm od usunięcia z grafu wszystkich wierzchołków stopnia większego niż l (powiedzmy, że jest ich m), zapisując na boku ich numery, by potem dodać je do pokrycia oryginalnego grafu. Jeśli wierzchołków do usunięcia jest więcej niż l, to oczywiście musimy odpowiedzieć „NIE" i możemy zakończyć pracę. W przeciwnym przypadku w okrojonym grafie szukamy pokrycia rozmiaru l', gdzie $l' = l - m$. Poczynimy teraz kluczową, choć bardzo prostą, obserwację:

Fakt 2 *Jeśli graf G ma wierzchołki stopni co najwyżej l oraz zawiera pokrycie wierzchołkowe rozmiaru co najwyżej l', to liczba krawędzi w G nie przekracza $l \cdot l'$.*

Ponieważ każda krawędź, podobnie jak każdy kij, ma dwa końce, otrzymujemy następujący wniosek.

Wniosek 3 *Liczba wierzchołków o niezerowym stopniu w grafie G nie przekracza $2ll'$.*

Możemy wykorzystać powyższy wniosek w zaproponowanym poprzednio algorytmie. Jeśli po usunięciu wierzchołków stopnia większego niż l zobaczymy, że liczba wierzchołków o niezerowym stopniu przekracza $2ll'$, to możemy śmiało odpowiedzieć „NIE" i zakończyć działanie algorytmu. Jeśli nie przekracza, zadanie nie jest jeszcze rozwiązane, ale rozmiar badanego grafu drastycznie się zmniejszył – jest ograniczony przez $2l^2$. Zastanówmy się

jeszcze, jakim kosztem została osiągnięta taka redukcja, czyli jaka jest złożoność czasowa procesu wykrywania „urwisów". Aby wyznaczyć stopnie wszystkich wierzchołków, dla każdego wierzchołka zliczamy przylegające do niego krawędzie. Czas tej operacji można podzielić na czas zerowania liczników, czyli $O(n)$, i czas powiększania liczników, czyli $O(m)$ – razem $O(n+m)$. Choć usuwanie jednego wierzchołka stopnia większego niż l może zająć nawet czas $\Theta(n)$, to zauważmy, że usuwanie wszystkich takich wierzchołków zajmuje czas $O(n+m)$, gdyż każdy wierzchołek i każda krawędź jest usuwana co najwyżej raz. Wreszcie, zliczanie wierzchołków o niezerowym stopniu trwa jedynie $O(n)$. A więc cała pierwsza faza algorytmu zajmuje jedynie czas liniowy — $O(n+m)$. Co więcej, czas drugiej fazy będzie zależał już tylko od l.

Jeśli w drugiej fazie algorytmu użyjemy naiwnego podejścia, czyli sprawdzania wszystkich podzbiorów zbioru wierzchołków rozmiaru co najwyżej l, czas drugiej fazy będzie ograniczony przez $O(\binom{2l^2}{l})$. Najlepszą złożoność czasową można oczywiście uzyskać, łącząc pomysły z algorytmów I i II. Dostajemy w ten sposób rozwiązanie o złożoności czasowej $O(n+m+l^2 2^l)$.

Szukajcie jądra problemu!

Podejście zaprezentowane w tym opracowaniu jest ogólną metodą „radzenia" sobie z problemami NP-zupełnymi. Polega ono na zmodyfikowaniu problemu poprzez wprowadzenie dodatkowego parametru — w naszym przypadku jest to rozmiar poszukiwanego pokrycia wierzchołkowego. Następnie próbujemy skonstruować taki algorytm, którego złożoność zależy w sposób wykładniczy jedynie od wprowadzonego parametru, a nie od rozmiaru danych wejściowych. W przypadku algorytmów grafowych, z reguły cel ten osiągamy szybko redukując rozwiązywanie problemu w oryginalnym grafie do badania mniejszego grafu, którego rozmiar zależy już tylko od wartości parametru. Metoda ta nazywana jest „redukcją do jądra problemu".

Testy

Rozwiązania były testowane na zestawie 27 testów połączonych w 10 grup. Zawodnik otrzymywał punkty za testy w danej grupie jedynie wtedy, gdy jego program zwrócił dostatecznie szybko poprawne rozwiązania dla wszystkich testów z grupy. W każdej grupie znajdował się przynajmniej jeden test, w którym poszukiwane pokrycie wierzchołkowe istnieje, oraz przynajmniej jeden trudny przypadek, pozwalający wykryć niepoprawne algorytmy, polegające na wyborze do pokrycia wierzchołkowego l kolejnych wierzchołków o największych stopniach. Duże testy były tworzone w sposób losowy, poprzez wybranie kilku specjalnych wierzchołków „centralnych" połączonych z prawie wszystkimi pozostałymi wierzchołkami.
Oto charakterystyka plików testowych:

Nazwa	n	k	m	Opis
zos1.in	7	1	9	test wygenerowany ręcznie

Nazwa	n	k	m	Opis
zos2.in	14	5	36	test wygenerowany ręcznie
zos3.in	7	2	21	test wygenerowany ręcznie
zos4.in	51	42	9	test wygenerowany ręcznie
zos5a.in	102	93	323	test losowy
zos5b.in	3	1	6	test losowy
zos5c.in	11	3	47	test losowy
zos6a.in	70000	69990	349884	test wygenerowany ręcznie z dodanym centrum
zos6b.in	769341	769331	2461243	test losowy
zos6c.in	775231	775223	2481305	test losowy
zos7a.in	70000	69990	279942	test wygenerowany ręcznie z dodanym centrum
zos7b.in	1000000	999995	3000000	maksymalny test z odpowiedzią NIE
zos7c.in	679109	679101	2173981	test losowy
zos7d.in	1000000	999990	986713	graf z pokryciem wielkości 9 oraz jeden wierzchołek połączony z prawie wszystkimi innymi
zos8a.in	70000	69990	279925	test wygenerowany ręcznie z dodanym centrum
zos8b.in	277114	277105	1773719	test losowy
zos8c.in	824849	824842	2638722	test losowy
zos8d.in	900000	899990	70	cykl długości 70; odpowiedź NIE
zos9a.in	70000	69990	139957	test wygenerowany ręcznie z dodanym centrum
zos9b.in	618922	618913	2971106	test losowy
zos9c.in	950357	950347	3000000	test losowy
zos9d.in	120	110	35	każda spójna składowa jest pojedynczą krawędzią (odpowiedź NIE)
zos10a.in	70000	69990	209941	test wygenerowany ręcznie z dodanym centrum
zos10b.in	678692	678688	2172173	test losowy
zos10c.in	423166	423156	2709748	test losowy
zos10d.in	900000	899990	70	cykl długości 70; odpowiedź NIE