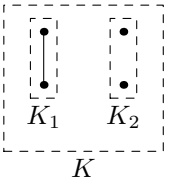
**Tomasz Waleń**
Treść zadania, Opracowanie

**Rafał Rusin**
Program

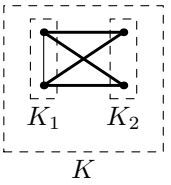
# Kaglony

**Kaglony** *to narodowa ulubiona potrawa mieszkańców Bajtocji. Kaglony mają bardzo charakterystyczną budowę. Glon składający się z jednej komórki jest kaglonem. Mając dwa kaglony* $K_1$ *i* $K_2$*, można je połączyć w następujący sposób:*

- *biorąc wszystkie komórki z* $K_1$ *i* $K_2$*, oraz wszystkie połączenia z* $K_1$ *i* $K_2$*,*




- *biorąc wszystkie komórki z* $K_1$ *i* $K_2$*, wszystkie połączenia z* $K_1$ *i* $K_2$*, oraz dodając nowe połączenia: każdą komórkę z* $K_1$ *łączymy z każdą komórką z* $K_2$*.*




*Otrzymujemy w wyniku nowy kaglon* $K$*.*

*Niestety niedawno wrogie państwo Bitocji rozpoczęło sprzedaż glonów imitujących kaglony. Glony te są na tyle podobne, że na pierwszy rzut oka trudno odróżnić je od oryginału, dlatego też rząd Bajtocji poprosił Cię o napisanie programu, który umożliwiłby sprawdzanie czy dany glon jest kaglonem.*

## Zadanie

*Napisz program który:*

- *wczyta ze standardowego wejścia opisy glonów,*
- *sprawdzi, które z nich są poprawnymi kaglonami,*
- *zapisze na standardowym wyjściu odpowiedź.*

## Wejście

*W pierwszym wierszu standardowego wejścia zapisana jest jedna liczba całkowita* $k$*,* $1 \leqslant k \leqslant 10$*, liczba badanych glonów. W kolejnych wierszach zapisane jest* $k$ *opisów glonów. Pojedynczy opis ma następującą postać: w pierwszym wierszu zapisane są dwie liczby*

*całkowite $n$ i $m$, oddzielone pojedynczym odstępem, $1 \leqslant n \leqslant 10\,000$, $0 \leqslant m \leqslant 100\,000$, odpowiednio liczba komórek i liczba połączeń. Komórki są ponumerowane od 1 do $n$. W kolejnych $m$ wierszach opisane są połączenia, w każdym z tych wierszy zapisano dwie liczby całkowite $a$, $b$ oddzielone pojedynczym odstępem, $a \neq b$, $1 \leqslant a,b \leqslant n$, oznaczające, że komórki $a$ i $b$ są połączone. Każde połączenie wymienione jest jeden raz.*

### Wyjście

*Na standardowym wyjściu należy zapisać $k$ wierszy. W $i$–tym wierszu należy zapisać jedno słowo:*

- `TAK` — *jeśli $i$–ty glon jest poprawnym kaglonem,*
- `NIE` — *w przeciwnym przypadku.*

### Przykład

*Dla danych wejściowych:*

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

*poprawnym wynikiem jest:*

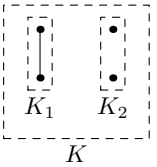
```
TAK
NIE
TAK
```

## Rozwiązanie

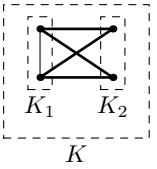
### Najprostsze rozwiązanie

Dobrym dowodem na to, że dany graf jest kaglonem, może być drzewo opisujące sposób otrzymania grafu. Liście tego drzewa odpowiadają wierzchołkom grafu, a węzły wewnętrzne odpowiadają operacjom, które łączą grafy z poddrzew. Zgodnie z treścią zadania istnieją dwie metody łączenia poddrzew:

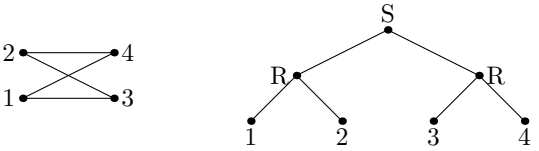
- równoległa — łączymy grafy z poddrzew, nie dodając żadnych dodatkowych krawędzi;




- szeregowa — łączymy grafy z poddrzew, ale tym razem dodajemy krawędzie pomiędzy każdą parą wierzchołków z różnych poddrzew.




Na rysunku 1 przedstawiono *ka–drzewo* dla przykładowego grafu.




Rysunek 1: Przykładowe ka–drzewo

Dopełnieniem grafu $G = (V,E)$ będziemy nazywać graf $G' = (V,E')$ o tym samym zbiorze wierzchołków $V$ i krawędziach $E' = \{(u,v) : (u,v) \notin E\}$.

Łatwo zauważyć, że gdy dla kaglonu $G$, ostatnią operacją było połączenie:

- równoległe — graf $G$ składa się z co najmniej dwóch spójnych składowych,
- szeregowe — graf $G'$ (dopełnienie $G$) składa się z co najmniej dwóch spójnych składowych.

Jeśli graf $G$ ($|G| > 1$) nie spełnia żadnego z tych warunków, czyli jednocześnie $G$ i $G'$ są spójne — to graf $G$ nie jest kaglonem.

Powyższe rozumowanie prowadzi do następującego algorytmu na sprawdzanie, czy graf $G$ jest kaglonem:

```
1:  function CzyKaglon(G)
2:  begin
3:      if |G|=1 then   return TRUE;
4:      wyznacz spójne składowe G — G_1,...,G_k;
5:      if k>1 then
6:          return CzyKaglon(G_1) and ... and CzyKaglon(G_k);
7:      else begin
8:          wyznacz G' — dopełnienie grafu G;
9:          wyznacz spójne składowe G' — G'_1,...,G'_j
10:         oraz odpowiadające im podgrafy G — G_1,...,G_j
```

```
11:         if j = 1 then
12:             return FALSE;
13:         else
14:             return CzyKaglon(G_1) and ... and CzyKaglon(G_j);
15:     end
16: end
```

Niestety takie rozwiązanie jest dosyć wolne, wymaga $O(n^3)$ czasu oraz $O(n^2)$ pamięci (w pesymistycznym przypadku graf $G'$ może mieć rozmiar $\Omega(n^2)$).

## Rozwiązanie wzorcowe

Problem rozpoznawania kaglonów można rozwiązać nawet w czasie $O(n+m)$, jednak takie rozwiązanie jest dosyć skomplikowane. Dalej opiszemy rozwiązanie o trochę gorszej złożoności czasowej, jednak znacznie prostsze. Rozwiązanie składa się z dwóch kroków:

- wyznaczenia pewnego obiektu kombinatorycznego, który miałby świadczyć o tym, że graf jest ka–glonem;
- a następnie weryfikacji poprawności takiego świadka (jeśli graf nie jest ka–glonem, to w tym kroku się o tym przekonamy).

Bardzo dobrym świadkiem na to, że graf jest ka–glonem, mogłoby być ka–drzewo. Niestety efektywne wyznaczenie ka–drzewa jest zadaniem dosyć skomplikowanym. W naszym rozwiązaniu świadkiem będzie taka permutacja wierzchołków grafu, która odpowiada kolejności liści w pewnym ka–drzewie badanego grafu. Taką permutację będziemy nazywać *permutacją faktoryzującą*. W rozdziale *Weryfikacja* opisany jest algorytm, który pozwala sprawdzić w czasie liniowym czy dla danej permutacji wierzchołków istnieje odpowiadające mu ka–drzewo.

### Obliczenie permutacji faktoryzującej

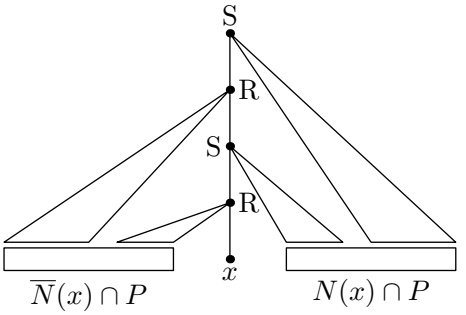
Początkowo nie dysponujemy żadną wiedzą o kolejności wierzchołków w permutacji faktoryzującej, czyli rozpoczynamy obliczenia z $P = (V)$. W trakcie obliczeń uzyskujemy dodatkowe informacje na temat kolejności wierzchołków, jednak nadal nie znamy dokładnego ich uporządkowania, w takiej sytuacji częściowo obliczoną permutację możemy reprezentować jako sekwencję rozłącznych podzbiorów wierzchołków:

$$P = (C_1, C_2, \ldots, C_k), \text{ gdzie } C_i \subseteq V$$

Oczywiście każdy wierzchołek $v \in V$ należy do dokładnie jednego ze zbiorów $C_i$. Naszym celem jest takie przekształcenie $P$, by każdy ze zbiorów $C_i$ miał rozmiar 1, czyli $P$ stał się permutacją wierzchołków $V$.

W pierwszym kroku algorytmu (gdy sekwencja $P$ jest po prostu zbiorem wierzchołków), wybieramy dowolny wierzchołek $x$ i dzielimy $P$ na $(P \cap \overline{N}(x), \{x\}, P \cap N(x))$, gdzie $N(x)$ oznacza zbiór sąsiadów wierzchołka $x$, a $\overline{N}(x)$ zbiór tych wierzchołków, które nie sąsiadują

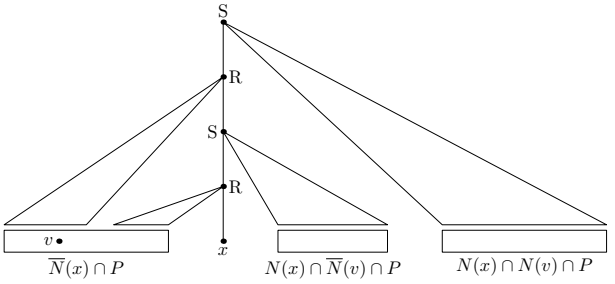
z $x$. Dlaczego możemy tak arbitralnie ustalić kolejność wierzchołków? Zauważmy, że dowolne ka–drzewo możemy tak uporządkować, by zawierało liście właśnie w tej kolejności — $P \cap \overline{N}(x), \{x\}, P \cap N(x)$.




Rysunek 2: Podział zbioru $P$ na $P \cap \overline{N}(x), \{x\}, P \cap N(x)$

Jednak taki podział wierzchołków nie jest jeszcze wystarczający — nie doprowadziliśmy jeszcze do sytuacji w której następne sprawdzenia będą wykonywane rekurencyjnie.

Tak więc należy uporządkować zbiory $P \cap \overline{N}(x), P \cap N(x)$, tak by każdy z nich zawierał wierzchołki z poddrzew wyznaczonych przez przodków wierzchołka $x$ w ka–drzewie. Wybierając dowolny wierzchołek $v$ z $\overline{N}(x)$ możemy podzielić wierzchołki z $N(x)$ na te, które nie są połączone z $v$ i na te, które sąsiadują z $v$ — czyli $N(x) \cap \overline{N}(v)$ i $N(x) \cap N(v)$. Powyższe rozumowanie przedstawione jest na rysunku 3. Podobny podział zbioru $\overline{N}(x)$ można uzyskać używając jako wierzchołków rozdzielających $v \in N(x)$.




Rysunek 3: Wynik operacji *Polepsz*$(v, (\overline{N}(x), \{x\}, N(x)))$

Bardziej formalnie procedura podziału zbiorów na wierzchołki poddrzew ma następującą postać:

```
1: function Polepsz(v, P)
2: begin
3:     for C_i ∈ P do
4:         if C_i ∩ N(v) ≠ C_i and C_i ∩ N(v) ≠ ∅ then
5:             zastąp C_i, przez zbiory C_i ∩ N̄(v), C_i ∩ N(v) (w takiej kolejności)
6: end
```

Gdy wykonamy już podziały dla wszystkich wierzchołków z $v \in N(x), \overline{N}(x)$, zbiory $C_i \in P$ będą odpowiadać wierzchołkom poddrzew o korzeniach będących przodkami wierzchołka $x$. Stąd dla każdego z poddrzew (zbiorów $C_i$) można rekurencyjnie obliczyć odpowiadającą mu permutację. Dzięki wielokrotnemu zastosowaniu powyższej metody uzyskujemy coraz dokładniejszą informację na temat permutacji $P$. Pełny kod procedury obliczania permutacji faktoryzującej ma następującą postać:

```
1:  function Kaglon(x,P)
2:  begin
3:      if |P|=1 then  return (x);
4:      P:=(N̄(x),{x},N(x));
5:      niech A mniejszy ze zbiorów N̄(x), N(x), a B większy;
6:      for x ∈ A do
7:          Polepsz(x, P)
8:      end;
9:      for C_i ∈ P ∩ B do
10:         niech y będzie dowolnym wierzchołkiem należącym do C_i;
11:         Polepsz(y, P);
12:         C'_i:=Kaglon(y, C_i);
13:         zastąp C_i przez C'_i w P
14:     end;
15:     for C_i ∈ P ∩ A do
16:         niech y będzie dowolnym wierzchołkiem należącym do C_i;
17:         C'_i:=Kaglon(y, C_i);
18:         zastąp C_i przez C'_i w P
19:     end;
20:     return P
21: end
```

Procedura *Polepsz*$(x,P)$ wymaga czasu proporcjonalnego do $N(x)$. Teraz wystarczy zauważyć, że każdy wierzchołek może być argumentem dla procedury *Polepsz* co najwyżej $O(\log n)$ razy — wierzchołek ma szansę na wielokrotne wykorzystanie tylko, gdy należy do mniejszego ze zbiorów $\overline{N}(x)$, $N(x)$ — czyli zbioru $A$. Stąd całkowity czas potrzebny na obliczenie permutacji faktoryzującej jest rzędu $O(n+m\log n)$.

### Weryfikacja

O dwóch wierzchołkach $x$, $y$ mówimy, że są bliźniakami (*twins*$(x,y)$), jeśli $N(x) = N(y)$ lub $N(x) \cup \{x\} = N(y) \cup \{y\}$. Gdy przeanalizujemy postać ka–drzewa, można zauważyć, że dwa wierzchołki $x$, $y$ są bliźniakami wtedy i tylko wtedy, gdy są braćmi w ka–drzewie. Stąd pojęcie "bliźniaków" może być pomocne w weryfikacji poprawności permutacji faktoryzującej — wystarczy przeglądać permutację z lewa na prawo i lokalizować sąsiadujące wierzchołki.

```
1: i:=0;
2: z:=x_1; { pierwszy wierzchołek w permutacji P }
3: while i < n − 1 and z ≠ x_n do begin
```

```
4:      if twins(z, prev(z)) then
5:          usuń prev(z) z P;
6:          i:=i+1
7:      else if twins(z, next(z)) then
8:          usuń z z P;
9:          z:=next(z);
10:         i:=i+1
11:     else
12:         z:=next(z)
13: end;
14: if P=1 then
15:     return TRUE;
16: else
17:     return FALSE;
```

Jeśli permutacja zostanie zredukowana do pojedynczego wierzchołka oznacza to, że permutacja odpowiada pewnemu ka–drzewu, a więc graf $G$ jest kaglonem. Weryfikacja permutacji wymaga czasu $O(n+m)$.

## Testy

Zadanie testowane było na zestawie 10 danych testowych, których opisy zawiera poniższa tabela.

| Nr | $k$ | max $n$ | max $m$ |
|---|---|---|---|
| 1 | 10 | 8 | 14 |
| 2 | 10 | 169 | 216 |
| 3 | 10 | 10000 | 40 |
| 4 | 10 | 9910 | 99000 |
| 5 | 10 | 9910 | 99125 |
| 6 | 10 | 9910 | 99002 |
| 7 | 10 | 10000 | 99000 |
| 8 | 10 | 9521 | 95210 |
| 9 | 10 | 10000 | 99894 |
| 10 | 10 | 450 | 52364 |