Wojciech Rytter
Treść zadania, Opracowanie

Wiktor Kuropatwa
Program

Dostępna pamięć: 256 MB. *OI, etap II, dzień próbny, 12.02.2013*

Spacer

Nazwy miast w Bajtocji mają postać ciągów n bitów. Oczywiście różne miasta mają różne nazwy. Wszystkich miast w Bajtocji jest $2^n - k$. Tak więc tylko k różnych ciągów n bitów nie jest nazwami miast.

Niektóre pary miast w Bajtocji są połączone drogami. Drogi te nie krzyżują się ze sobą poza miastami. Dwa miasta są połączone bezpośrednio drogą wtedy i tylko wtedy, gdy ich nazwy różnią się tylko jednym bitem.

Bajtazar wybiera się na spacer – chce przejść z miasta x do miasta y, korzystając jedynie z istniejących dróg. Twoim zadaniem jest napisanie programu, który sprawdzi, czy taki spacer jest możliwy.

Wejście

W pierwszym wierszu standardowego wejścia zapisane są dwie liczby całkowite n i k oddzielone pojedynczym odstępem ($1 \leqslant n \leqslant 60$, $0 \leqslant k \leqslant 1\,000\,000$, $k < 2^n - 1$, $n \cdot k \leqslant 5\,000\,000$). Oznaczają one odpowiednio liczbę bitów w nazwach miast oraz liczbę różnych ciągów n bitów, które nie są nazwami żadnych miast. W drugim wierszu znajdują się dwa napisy oddzielone pojedynczym odstępem, każdy złożony z n znaków `0` *i/lub* `1`*. Są to nazwy miast x i y. W kolejnych k wierszach są wymienione wszystkie ciągi n bitów, które nie są nazwami miast, po jednym w wierszu. Każdy taki ciąg bitów ma postać napisu złożonego z n znaków* `0` *i/lub* `1`*. Możesz założyć, że nazwy miast x i y nie pojawiają się wśród tych k ciągów bitów.*

W testach wartych łącznie 25% punktów zachodzi dodatkowy warunek $n \leqslant 22$.

Wyjście

Twój program powinien wypisać na standardowe wyjście słowo `TAK`*, jeżeli możliwe jest przejście z miasta x do miasta y, a w przeciwnym przypadku powinien wypisać słowo* `NIE`*.*

Przykład

Dla danych wejściowych:

```
4 6
0000 1011
0110
0111
0011
1101
1010
1001
```

poprawnym wynikiem jest:

```
TAK
```

a dla danych wejściowych:	*poprawnym wynikiem jest:*

`2 2`
`00 11`
`01`
`10`
	`NIE`

Wyjaśnienie do pierwszego przykładu: *Oto przykładowe trasy prowadzące z miasta* `0000` *do miasta* `1011`*:*

- `0000` → `1000` → `1100` → `1110` → `1111` → `1011`,
- `0000` → `0100` → `1100` → `1110` → `1111` → `1011`.

Testy „ocen”:

`1ocen:` *$n = 20$, $k = 215\,766$, zabronione są wszystkie miasta o liczbie zer podzielnej przez 5, chcemy wykonać spacer z $x =$* `100...0` *do $y =$* `011...1`*; odpowiedź* `NIE`*;*

`2ocen:` *$n = 50$, $k = 100\,000$, miasta x i y są połączone bezpośrednio drogą; odpowiedź* `TAK`*;*

`3ocen:` *ładny test z $n = 60$; odpowiedź* `TAK`*.*

Rozwiązanie

Zadanie to było nietypowe i opierało się na ciekawej własności kombinatorycznej pewnego grafu. Własność ta nie jest łatwa do udowodnienia (choć tego od uczestników Olimpiady w czasie zawodów się nie żąda), ale na intuicję można taką (lub podobną) własność zauważyć w oparciu o silny stopień wzajemnych powiązań wewnątrz grafu. Z podanych względów zadanie pojawiło się na sesji próbnej i nie liczyło się do punktacji.

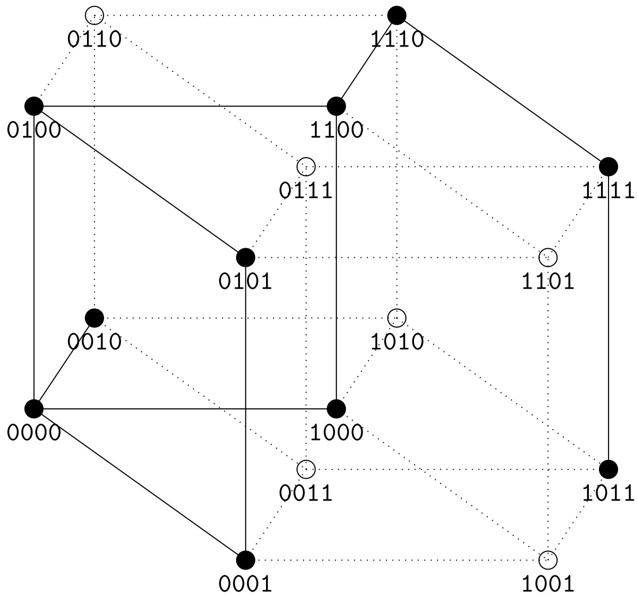
W przypadku $k = 0$ grafem rozważanym w zadaniu jest n-wymiarowa hiperkostka. Ma ona 2^n wierzchołków, a z każdego z nich wychodzi dokładnie n krawędzi (patrz rys. 1). To właśnie ta duża liczba krawędzi wychodzących z każdego wierzchołka powoduje, że zadanie *Spacer* ma rozwiązanie wielomianowe. Własność tę można trochę wzmocnić:

Twierdzenie 1. *Dla każdego podziału wierzchołków hiperkostki na dwa zbiory S i $V \setminus S$ liczba krawędzi prowadzących z jednego zbioru do drugiego jest co najmniej tak duża jak liczność mniejszego ze zbiorów:*

$$|\{(u,v)\colon u \in S, v \in V \setminus S\}| \geqslant \min(|S|, |V \setminus S|).$$

Dowód powyższego twierdzenia znajduje się na końcu opisu. Spójrzmy, co to twierdzenie nam daje.

W naszym zadaniu usunęliśmy ze zbioru wierzchołków hiperkostki pewien podzbiór V_K, o liczności k. To mogło spowodować podział pozostałych wierzchołków na więcej niż jedną spójną składową. Intuicyjnie, jeśli n jest duże, tylko jedna z tych spójnych składowych może być „duża”. Musieliśmy bowiem usunąć wszystkie krawędzie



Rys. 1: Ilustracja do pierwszego przykładu: hiperkostka dla $n = 4$, z której usunięto 6 wierzchołków.

prowadzące pomiędzy składowymi. Gdybyśmy zatem mieli dwie „duże” składowe, to wynikałoby z tego, że musieliśmy usunąć pomiędzy nimi „dużo” krawędzi. A usunęliśmy ich co najwyżej nk.

Uzasadnijmy to formalnie. Załóżmy, że mamy dwie spójne składowe o rozmiarze co najmniej $nk+1$ i niech S będzie jedną z nich. Wtedy z Twierdzenia 1 w hiperkostce pomiędzy S i $V \setminus S$ znajduje się co najmniej $nk+1$ krawędzi (rozmiar $V \setminus S$ jest równy co najmniej $nk+1$, bo znajduje się tam druga z rozważanych składowych). Ponieważ tylko nk z tych krawędzi może wchodzić do zbioru V_K, zatem między S a zbiorem $V \setminus (S \cup V_K)$ znajduje się co najmniej jedna krawędź. Daje to sprzeczność z tym, że S jest spójną składową. Zatem w naszym grafie jest co najwyżej jedna spójna składowa rozmiaru co najmniej $nk+1$.

Dzięki temu jesteśmy już w stanie napisać prosty algorytm działający w czasie $O(nk)$. Jeśli wierzchołki x i y znajdują się w tej samej spójnej składowej, to mamy dwie możliwości:

(1) Składowa ta ma rozmiar mniejszy niż $nk+1$, zatem przeszukując graf z wierzchołka x, znajdziemy ścieżkę do wierzchołka y po co najwyżej $nk+1$ krokach.

(2) Składowa ta jest jedyną składową o rozmiarze co najmniej $nk+1$. Do przetestowania tej możliwości wystarczy przeszukać graf z wierzchołka x, a następnie z wierzchołka y i sprawdzić, czy w obu przypadkach możemy odwiedzić co najmniej $nk+1$ wierzchołków. Po odwiedzeniu takiej liczby wierzchołków, kończymy przeszukiwanie.

Wystarczy zatem napisać funkcję $\mathit{RestrictedSearch}(x, y, l)$, która przeszukuje graf, startując z wierzchołka x, i jeśli znajdzie wierzchołek y lub odwiedzi l wierzchołków, to zwraca **true**. Gdy te przypadki nie wystąpią, a funkcja przejrzy wszystkie osiągalne z x wierzchołki – zwraca **false**. Ścieżka pomiędzy x a y istnieje dokładnie wtedy, gdy zarówno $\mathit{RestrictedSearch}(x, y, nk+1)$, jak i $\mathit{RestrictedSearch}(y, x, nk+1)$ zwrócą **true**.

Rozwiązanie wzorcowe przegląda zatem $O(nk)$ wierzchołków i $O(n^2k)$ krawędzi. Wierzchołki zabronione utrzymywane są w tablicy z haszowaniem, co gwarantuje złożoność czasową rozwiązania $O(n^2k)$. Do przeglądania grafu wykorzystano przeszukiwanie wszerz, ze względu na łatwość implementacji i proste do przewidzenia zużycie pamięci, w przeciwieństwie do rozwiązań rekurencyjnych. Implementacje można znaleźć w plikach `spa.cpp`, `spa1.pas` i `spa2.cpp`.

Dowód własności podziałowej

Zamieńmy każdą krawędź hiperkostki na dwie krawędzie skierowane. *Ścieżką standardową* prowadzącą z wierzchołka u do wierzchołka v nazwiemy taką skierowaną ścieżkę najmniejszej długości, która odpowiada zmianie kolejnych niezgodnych bitów (od lewej do prawej) w ciągach bitów odpowiadających wierzchołkom u i v. Na przykład dla $u = \texttt{0101}$, $v = \texttt{0010}$ ścieżka standardowa odpowiada zmianie kolejno drugiego, trzeciego i czwartego bitu:

$$\texttt{0101} \rightarrow \texttt{0001} \rightarrow \texttt{0011} \rightarrow \texttt{0010}.$$

Ustalmy teraz pewną krawędź skierowaną e i zastanówmy się, ile standardowych ścieżek przechodzi przez e. Rozważmy ścieżkę standardową z u do v, która zawiera e. Jeśli krawędź e dotyczy zmiany i-tego bitu, to ostatnie $n-i$ bitów u oraz pierwsze $i-1$ bitów v są zdeterminowane przez tę krawędź. Poza tym krawędź determinuje i-ty bit u i v. Zatem pozostaje $n-1$ możliwości na wybór niezdeterminowanych bitów w ciągach odpowiadających wierzchołkom u i v. Zatem przez każdą krawędź skierowaną przechodzi dokładnie 2^{n-1} ścieżek standardowych.

Jesteśmy już gotowi do dowodu Twierdzenia 1. Oznaczmy $m = |S|$ i przyjmijmy, że $m \leqslant 2^n - m = |V \setminus S|$. Mamy $m \cdot (2^n - m)$ ścieżek standardowych prowadzących z wierzchołków S do wierzchołków $V \setminus S$. Każda z nich przechodzi przez co najmniej jedną krawędź skierowaną między S i $V \setminus S$. Ponieważ przez jedną taką krawędź przechodzi co najwyżej 2^{n-1} ścieżek standardowych, a $m \leqslant 2^{n-1}$, więc liczba krawędzi pomiędzy S i $V \setminus S$ wynosi co najmniej

$$\frac{m \cdot (2^n - m)}{2^{n-1}} \geqslant \frac{m \cdot 2^{n-1}}{2^{n-1}} = m = |S|.$$