Tomasz Śmigielski
treść zadania

Marek Pawlicki
program wzorcowy, opracowanie

Bramki XOR

Każda **bramka XOR** *ma dwa wejścia i jedno wyjście, a jej działanie opisuje następująca tabelka:*

wejście 1	*wejście 2*	*wyjście*
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

Siecią XOR *nazywamy układ bramek XOR, mający n wejść i jedno wyjście, spełniający następujące warunki:*

(1) Każde wejście sieci XOR jest połączone z przynajmniej jednym wejściem bramki.

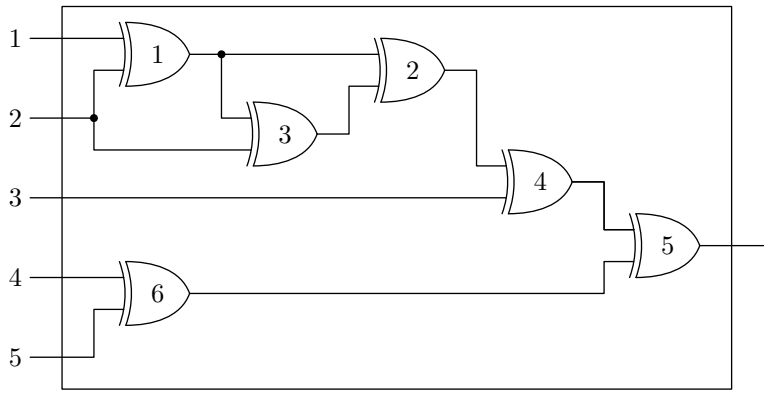
(2) Każde wejście każdej bramki jest połączone z jednym wejściem sieci albo z jednym wyjściem innej bramki.

(3) Wyjście dokładnie jednej bramki jest połączone z wyjściem sieci.

(4) Każde wyjście bramki w sieci jest połączone z przynajmniej jednym wejściem innej bramki albo z jedynym wyjściem sieci.

(5) Istnieje taka numeracja bramek, że do każdego wejścia dowolnej bramki jest podłączone wejście sieci albo wyjście bramki o mniejszym numerze.

Przykład



Przedstawiony na rysunku układ 6 bramek mający 5 wejść i 1 wyjście spełnia warunki 1–5, więc jest siecią XOR.

UWAGA: *Bramki na rysunku zostały ponumerowane dowolnie, ale istnieje numeracja spełniająca warunek określony w punkcie 5.*

Wszystkie wejścia sieci są ponumerowane od 1 do n. Stan wejść sieci XOR opisuje **słowo wejściowe** *utworzone z n cyfr dwójkowych 0 i 1 — przyjmujemy, że i-ta od lewej cyfra danego słowa wejściowego, to stan i-tego wejścia sieci. Dla dowolnego stanu wejść sieć daje na wyjściu 0 albo 1. Każde słowo wejściowe jest dwójkowym zapisem jakiejś liczby naturalnej, więc słowa te można uporządkować zgodnie z ich wartościami liczbowymi. Sieci XOR będziemy testowali podając na wejściu kolejne słowa z ustalonego zakresu i zliczając liczbę otrzymanych w wyniku jedynek.*

Zadanie

Napisz program, który:

- *wczytuje z pliku tekstowego XOR.IN opis sieci XOR: liczbę wejść n, liczbę bramek m, numer bramki połączonej z wyjściem sieci oraz opisy połączeń, a następnie dwa n-bitowe słowa wejściowe — dolne i górne ograniczenie zakresu, w jakim będziemy testowali sieć,*
- *oblicza liczbę jedynek otrzymanych na wyjściu sieci dla słów wejściowych z danego zakresu,*
- *zapisuje wynik w pliku tekstowym XOR.OUT.*

Zakładamy, że $3 \leqslant n \leqslant 100$, $3 \leqslant m \leqslant 3000$ oraz, że bramki danej sieci są ponumerowane w dowolnym porządku liczbami od 1 do m.

Wejście

W pierwszym wierszu pliku tekstowego XOR.IN są zapisane trzy liczby całkowite dodatnie pooddzielane pojedynczym odstępem. Jest to liczba wejść n danej sieci XOR, liczba bramek m oraz numer bramki połączonej z wyjściem sieci.
W kolejnych m wierszach znajdują się opisy połączeń bramek sieci. W i-tym z tych wierszy, dla i od 1 do m, znajduje się opis połączeń dwóch wejść bramki o numerze i, który ma postać dwóch liczb całkowitych nie mniejszych niż $-n$ i nie większych niż m, oddzielonych pojedynczym odstępem. Jeśli odpowiednie wejście do bramki jest połączone z wejściem do sieci o numerze k, to opisem tego połączenia jest liczba ujemna $-k$, a jeśli wejście do bramki jest połączone z wyjściem innej bramki o numerze j, to opisem tego połączenia jest liczba dodatnia j.
W kolejnych 2 wierszach pliku tekstowego XOR.IN są zapisane dwa n-bitowe słowa $\underline{a}$ oraz $\underline{b}$. Jest to dolne i górne ograniczenie zakresu testowania sieci. Zakładamy, że w danym zakresie mieści się nie więcej niż sto tysięcy słów.

Wyjście

W pliku tekstowym XOR.OUT należy zapisać jedną liczbę całkowitą nieujemną — liczbę jedynek, jakie powinniśmy otrzymać na wyjściu poprawnie działającej danej sieci XOR dla słów wejściowych $\underline{s}$ z danego zakresu $\underline{a} \leqslant \underline{s} \leqslant \underline{b}$, gdzie nierówność $\leqslant$ należy rozumieć jako relację porządku zgodnego z wartościami liczbowymi słów dwójkowych.

Przykład

Dla pliku XOR.IN, zawierającego opis przedstawionej powyżej sieci XOR:

```
5 6 5
-1 -2
1 3
1 -2
2 -3
4 6
-4 -5
00111
01110
```

poprawnym rozwiązaniem jest następujący plik XOR.OUT:

```
5
```

Twój program powinien szukać pliku XOR.IN w katalogu bieżącym i tworzyć plik XOR.OUT również w bieżącym katalogu. Plik zawierający napisany przez Ciebie program w postaci źródłowej powinien mieć nazwę XOR.???, gdzie zamiast ??? należy wpisać co najwyżej trzyliterowy skrót nazwy użytego języka programowania. Ten sam program w postaci wykonywalnej powinien być zapisany w pliku XOR.EXE

ROZWIĄZANIE

Kluczem do efektywnego rozwiązania jest znalezienie zależności pomiędzy wartościami na wejściu sieci, a wynikiem na jej wyjściu. W tym celu przyjrzyjmy się bliżej działaniu bramki XOR. Niech $\oplus$ oznacza dodawanie liczb całkowitych modulo 2, tzn. takie, że jego wynikiem jest reszta, jaką daje suma ze zwykłego dodawania przy dzieleniu przez 2. Tak określone działanie jest przemienne i łączne, więc możemy go używać analogicznie do zwykłego dodawania. Stan na wyjściu bramki (lub inaczej: liczbę dwójkową obliczaną przez bramkę) można teraz wyrazić jako $a \oplus b$, gdzie a i b oznaczają wartości na jej wejściach. W ogólności, jeżeli przez $x_1, \ldots, x_n$ oznaczymy stany wejść sieci, to stan na wyjściu każdej z bramek zadanej sieci można wyrazić jako wartość funkcji postaci

$$f(x_1, \ldots, x_n) = a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n \tag{1}$$

gdzie zapis $a_1 x_1$ oznacza zwykłe mnożenie, a $a_1, \ldots, a_n \in \{0, 1\}$ są współczynnikami zależnymi od położenia bramki w sieci. W szczególności stan na wyjściu bramki połączonej z wyjściem sieci jest także opisany powyższą zależnością, a jej funkcję nazywamy **wielomianem sieciowym**. Dla przykładowej sieci z treści zadania wielomiany dla odpowiednich bramek mają postać (f_i oznacza wielomian dla i-tej bramki):

$$\begin{aligned}
f_1(x_1, \ldots, x_5) &= x_1 \oplus x_2 \\
f_2(x_1, \ldots, x_5) &= x_2 \\
f_3(x_1, \ldots, x_5) &= x_1 \\
f_4(x_1, \ldots, x_5) &= x_2 \oplus x_3 \\
f_5(x_1, \ldots, x_5) &= x_2 \oplus x_3 \oplus x_4 \oplus x_5 \\
f_6(x_1, \ldots, x_5) &= x_4 \oplus x_5
\end{aligned}$$

Jak znaleźć wielomian sieciowy? Po przyjrzeniu się postaci wielomianu (1) łatwo zauważyć, że dla $x_i = 1$ oraz $x_j = 0$ dla wszystkich $j \neq i$ wartość na wyjściu sieci jest tożsama ze współczynnikiem a_i jej wielomianu. Aby wyznaczyć cały wielomian, wystarczy obliczyć, jakie wyniki daje sieć na wyjściu dla n słów wejściowych o postaci podanej wyżej dla $i = 1, \ldots, n$. Powyższe obliczenie można prosto zaimplementować w postaci procedury rekurencyjnej wyznaczającej wartość na wyjściu bramki na podstawie wartości na wyjściach bramek podłączonych do jej wejść lub wartości na wejściach sieci. Przyjmijmy następujące globalne deklaracje:

```
const
  MaxLiczbaBramek = 3000;
  MaxLiczbaWejsc = 100;
type
  { Stan binarny }
  TStan = 0..1;
  { Element oznaczający bramkę lub wejście sieci }
  TElement = record
    { Numery bramek podłączonych do wejśc; nieistotne w przypadku wejść sieci }
    Wejscie1, Wejscie2 : Integer;
    { Stan na wyjściu bramki lub stan wejścia sieci }
    Wyjscie : TStan;
    { Oznacza, czy wartość Wyjscie jest wyznaczona }
    WyjscieObliczone : Boolean
  end;
  { Sieć złożona z bramek i wejść }
  TSiec = array [-MaxLiczbaWejsc..MaxLiczbaBramek] of TElement;
  { Współczynniki wielomianu lub stan wejść }
  TWielomian = array [1..MaxLiczbaWejsc] of TStan;
var
  S : TSiec; { Cała sieć }
  N : Integer; { Liczba wejść }
  M : Integer; { Liczba bramek }
  WyjscieSieci : Integer; { Numer bramki podłączonej do wyjścia sieci }
```

Elementy $S[-N]..S[-1]$ opisują odpowiednie wejścia sieci, natomiast elementy $S[1]..S[M]$ przechowują dane o bramkach sieci. Poniższa procedura oblicza wartość na wyjściu bramki o numerze *nr* dla danego stanu wiejść $S[-N]..\mathrm{S}[-1]$:

```
procedure ObliczBramke (nr : Integer);
var
  We1, We2 : Integer;
begin
  We1 := S[nr].Wejscie1; We2 := S[nr].Wejscie2;
  { Oblicz wartości wyjściowe bramek podłączonych do wejść,
    jeżeli nie są jeszcze wyznaczone }
  if not S[We1].WyjscieObliczone then ObliczBramke(We1);
  if not S[We2].WyjscieObliczone then ObliczBramke(We2);
```

```
  { Oblicz wartość na wyjściu }
  S[nr].Wyjscie := (S[We1].Wyjscie + S[We2].Wyjscie) mod 2;
  { Zaznacz, że wartość na wyjściu wyznaczona }
  S[nr].WyjscieObliczone := true
end;
```

Zauważmy, że dzięki pamiętaniu raz obliczonej wartości na wyjściu bramki, liczba kroków obliczeń wykonywanych przy wywołaniu *ObliczBramke*(*WyjscieSieci*) będzie proporcjonalna do liczby bramek m. Zauważmy także, że dzięki warunkowi 5. dla sieci XOR z treści zadania, takie wywołanie zakończy się sukcesem (nie będzie zapętlenia). Możemy teraz, wykorzystując procedurę `ObliczBramke`, łatwo wyznaczyć wielomian sieciowy:

```
var
  WielomianSieci : TWielomian; { Współczynniki wielomianu sieci }
  for i := 1 to N do begin
  { S[-N].Wyjscie..S[-1].Wyjscie := 0, S[-i].Wyjscie := 1 }
  { S[-N].WyjscieObliczone..S[-1].WyjscieObliczone := true }
  { S[1].WyjscieObliczone..S[M].WyjscieObliczone := false }
  { Oblicz wartość na wyjściu sieci... }
  ObliczBramke(WyjscieSieci);
  { ... która jest i-tym współczynnikiem }
  WielomianSieci[i] := S[WyjscieSieci].Wyjscie
end
```

Dobrze zaimplementowany powyższy fragment programu wykonuje liczbę operacji rzędu $m \cdot n$. Obliczenie liczby jedynek dla zadanego przedziału danych wejściowych wydaje się teraz proste:

```
var
  Dane, Koniec : TWielomian; { Słowa z pliku wejściowego }
  LiczbaJedynek : Longint; { Zlicza jedynki }
...
  LiczbaJedynek := 0;
  { Wczytaj wartości graniczne z pliku wejściowego na zmiene Dane i Koniec }
  LiczbaJedynek := LiczbaJedynek + WartoscWielomianu(WielomianSieci, Dane);
  while not RowneDane(Dane, Koniec) do begin
    KolejnyZestaw(Dane);
  LiczbaJedynek := LiczbaJedynek + WartoscWielomianu(Dane)
end;
```

Po uzupełnieniu implementacji procedur: *WartoscWielomianu* — obliczającej wartość wielomianu dla zadanego słowa, *RowneDane* — stwierdzającej, czy dwa słowa są równe, *KolejnyZestaw* — generującej kolejny zestaw wartości wejściowych zgodnie z

porządkiem liczb dwójkowych oraz po zaimplementowaniu wczytania danych o sieci, otrzymujemy gotowy program, który można znaleźć na dołączonej dyskietce.

Złożoność czasowa przedstawionego rozwiązania jest rzędu $mn+ln$, gdzie l oznacza liczbę zestawów wejściowych, natomiast wymagania pamięciowe są rzędu $m+n$.

Możemy sobie wyobrazić, że procedura rekurencyjna podobna do *ObliczBramke* będzie wyznaczać nie wartość na wyjściu bramki lecz cały jej wielomian, na podstawie wielomianów bramek podłączonych do jej wejść lub wielomianów odpowiadającym wejściom sieci. Oczywiście wielomianem odpowiadającym i-temu wejściu sieci jest x_i. Złożoność czasowa rozwiązania opartego na tym pomyśle jest tego rzędu, co złożoność rozwiązania poprzedniego, lecz potrzebna pamięć jest rzędu mn na przechowywanie wielomianów.

Jeżeli wyeliminujemy pamiętanie wyników pośrednich (wyjście bramki lub wielomian), przy zastosowaniu procedury podobnej do *ObliczBramke* otrzymamy nieco prostszy algorytm, za to bardzo nieefektywny (dane dla jednej bramki wyliczane byłyby wielokrotnie).

Dane dla bramki (wyjście bramki lub wielomian) można wyznaczyć w inny sposób niż przez stosowanie rekurencji. W tym celu dla bramek zadanej sieci zdefiniujmy relację $<$ (mniejsza) w ten sposób, że bramka B_1 jest mniejsza od bramki B_2 ($B_1 < B_2$), jeżeli wyjście bramki B_1 podłączone jest do jednego z wejść bramki B_2. Zgodnie z warunkiem 5. dla sieci XOR z treści zadania, jej bramki można ustawić w ciąg tak, aby na prawo od danej bramki w ciągu nie występowały bramki od niej mniejsze. Łatwo zauważyć, że na początku ciągu będzie bramka, której oba wejścia są podłączone do wejść sieci, a na końcu ta, której wyjście jest podłączone do wyjścia sieci. Znalezienie takiego uporządkowania nazywa się **sortowaniem topologicznym**. Z odpowiednim algorytmem i jego omówieniem Czytelnik może się zapoznać np. w książce [21]. Gdy już znajdziemy takie uporządkowanie, wystarczy obliczyć kolejno wartości na wyjściach dla każdej bramki w ciągu. Złożoność czasowa rozwiązania opartego na tym pomyśle jest taka sama, jak rozwiązania pierwszego (!), a złożoność pamięciowa zależy od tego, czy obliczamy wielomian sieci przez symulację, czy przez wyznaczanie wielomianów dla każdej bramki.

Możemy wreszcie zrezygnować z wyznaczania wielomianu sieci i obliczać wartość na jej wyjściu, dla każdych danych wejściowych, stosując rekurencję z pamiętaniem wyników pośrednich lub bez, bądź też sortowanie topologiczne. Jednak złożoność czasowa takich rozwiązań będzie przynajmniej rzędu lm, a ponieważ m może być dużo większe niż n, przy dużych l rozwiązania takie będą znacznie wolniejsze niż rozwiązanie zaproponowane jako pierwsze.

TESTY

Do sprawdzania rozwiązań zawodników użyto 10 testów XOR0.IN–XOR9.IN. Testy można podzielić na dwie grupy: testy sprawdzające szczególne przypadki oraz testy sprawdzające szybkość działania zastosowanego algorytmu. Oto krótki ich opis:

- XOR0.IN — test z treści zadania;

- XOR1.IN — test sprawdzający zachowanie programu, gdy występują w sieci bramki o obu wejściach podłączonych do tego samego punktu;
- XOR2.IN — dla tego testu odpowiedzią jest 0 jedynek;
- XOR3.IN — dla tego testu odpowiedź przekracza zakres typu **Word**.
- XOR4.IN — prosty test sprawdzający poprawność użytej metody;

Testy XOR5.IN–XOR9.IN to testy złożnościowe o wzrastającym stopniu trudności.