Zawody I stopnia

Zawody I stopnia — opracowania zadań

Tomasz Śmigielski
Treść zadania

Marcin Mucha
Opracowanie, Program

Mapa gęstości

Dane są:

- *liczby naturalne* $n > r \geq 0$,
- F *— tabelka* $n \times n$ *wypełniona liczbami ze zbioru* $\{0, 1\}$*; kolumny i wiersze tabelki są ponumerowane od 1 do* n*; liczbę znajdującą się w* i*-tej kolumnie i* j*-tym wierszu tabelki oznaczamy przez* $F[i, j]$.

Jeśli $[i, j]$ *i* $[i', j']$ *są dwoma miejscami w tabelce* F*, to odległością między nimi nazywamy liczbę* $max\left(|i - i'|, |j - j'|\right)$.

Należy obliczyć tabelkę W, $n \times n$ *(do elementów tej tabelki odwołujemy się tak samo, jak do elementów tabelki* F*) taką, że* $W[i, j]$ *jest sumą wszystkich liczb z tabelki* F *leżących w odległości co najwyżej* r *od* $[i, j]$.

Zadanie

Napisz program, który:

- *wczyta z pliku tekstowego* `map.in` *liczby* n, r *oraz tabelkę* F,
- *obliczy tabelkę* W,
- *zapisze ją do pliku tekstowego* `map.out`.

Wejście

W pierwszym wierszu pliku tekstowego `map.in` *znajdują się dwie dodatnie liczby całkowite oddzielone pojedynczą spacją:* n *i* r*, gdzie* $0 \leq r < n \leq 250$.
W kolejnych n *wierszach znajduje się opis tabelki* F*. Każdy z tych wierszy zawiera* n *liczb ze zbioru* $\{0, 1\}$*, pooddzielanych pojedynczymi odstępami,* i*-ta liczba zapisana w* $j+1$*-szym wierszu to* $F[i, j]$.

Wyjście

Plik tekstowy `map.out` *powinien zawierać dokładnie* n *wierszy, w* i*-tym wierszu powinny być zapisane kolejno wartości* $W[1, i] \ldots W[n, i]$ *pooddzielane pojedynczymi odstępami.*

Przykład

Dla pliku wejściowego `map.in`*:*

```
5 1
1 0 0 0 1
1 1 1 0 0
1 0 0 0 0
0 0 0 1 1
0 1 0 0 0
```

poprawną odpowiedzią jest plik wyjściowy `map.out`*:*

```
3 4 2 2 1
4 5 2 2 1
3 4 3 3 2
2 2 2 2 2
1 1 2 2 2
```

Rozwiązanie

W całym opracowaniu (które jest jednak nieco bardziej teoretyczne od programu) będziemy zakładali, że funkcja F jest określona na całym zbiorze $Z \times Z$, przy czym poza rozważanym kwadratem się zeruje. To pozwoli na nieco swobodniejsze posługiwanie się sumami, a rozważenie przypadków brzegowych w programie jest dość proste (jeśli przypadki brzegowe będą w którymś momencie istotne, to zostanie to odnotowane).

Rozwiązanie optymalne polega na dynamicznym policzeniu sum:

$$S[i,j] = \sum_{1 \leq x \leq i} \sum_{1 \leq y \leq j} F[x,y]$$

Poszukiwane wartości wyrażają się przez $S[i,j]$ następująco:

$$W[i,j] = S[i,j] - S[i-r-1,j] - S[i,j-r-1] + S[i-r-1,j-r-1].$$

Obliczanie sum $S[i,j]$ można wykonać w naturalny sposób w miejscu (tzn. w tablicy F) w czasie $O(n^2)$. Wyliczanie każdego $W[i,j]$ odbywa się już potem w czasie stałym. Tak więc cały algorytm ma złożoność $O(n^2)$. Program implementujący ten algorytm znajduje się w pliku `map.pas`.

Inne rozwiązania

Zawsze można $W[i,j]$ obliczyć przy pomocy czterech zagnieżdżonych pętli, czyli po prostu z definicji. Taki algorytm ma złożoność $O(n^2r^2)$ i nie należy do najszybszych. Jego implementacja znajduje się w pliku `map1.pas`.

Chwila zastanowienia wystarcza na znalezienie następującej sprytnej formuły:

$$W[i,j] = W[i,j-1] + \sum_{i-r \leq k \leq i+r} F[k,j+r] - \sum_{i-r \leq k \leq i+r} F[k,j-r-1]$$

Innymi słowy: jeśli przesuwamy kwadracik o jedno pole w prawo, to musimy jedną kolumnę odjąć, a drugą dodać. Wystarczy więc "faktycznie" policzyć $W[i,j]$ dla $j=1$, a dalej możemy już w czasie $2r$ korzystać z powyższej formuły. Jaki jest łączny czas działania takiego algorytmu? Na "faktyczne obliczenia" — $O(nr^2)$. Na poprawki $O(n^2r)$. Łącznie $O(n^2r)$. Program implementujący ten algorytm znajduje się w pliku `map2.pas`.

Oczywiście analogiczną formułę można zastosować, aby policzyć $W[i,1]$ przy pomocy $W[i-1,1]$. Wtedy musielibyśmy "faktycznie" policzyć tylko $W[1,1]$. Wydaje się to jednak bezcelową komplikacją, skoro i tak program będzie miał złożoność $O(n^2r)$. Nie jest to jednak takie proste. Poprawka bowiem nie zawsze wymaga czasu $2r$. Czasami dodawana lub odejmowana kolumna jest poza zakresem i wtedy nic nie trzeba robić. Im większe r, tym częściej się to zdarza. W szczególności, jeśli r jest bardzo duże (np. $r = n-1$), to program "poprawiający" w pionie i w poziomie będzie działał w czasie $O(n^2)$, a "poprawiający" tylko w poziomie w czasie $O(n^3)$, z tego prostego powodu, że będzie musiał policzyć wszystkie $W[i,1]$. W związku z powyższym program "poprawiający" i w pionie i w poziomie można uznać za oddzielne rozwiązanie, a jego implementacja znajduje się w pliku `map3.pas` (choć pesymistycznie nie jest on szybszy od programu "poprawiającego" tylko w poziomie).

Testy

Wszystkie testy są losowe (nie bardzo widać, co mogłoby być lepsze). Różnią się rozmiarami, wartością r (z treści opracowania wynika, że generalnie najtrudniejsze testy to takie, dla których $r \approx n/2$) i czasem gęstością jedynek. Testy $1-4$ to testy poprawnościowe. W szczególności test 1 sprawdza odporność na $r=0$, a test 3 na r bardzo duże. Dalej idą coraz trudniejsze testy wydajnościowe. Testy $11-13$ to najtrudniejsze testy jakie udało się wymyśleć. Po drodze jest test 8, który może przejść nawet słaby algorytm, o ile przypadek małej liczby jedynek rozważa oddzielnie. Test 10 jest z kolei celowo dobrany pod algorytm poprawiający w obu kierunkach (tak naprawdę na testach $11-13$ ten algorytm też radzi sobie nieco lepiej).

Wydaje się, że świetnie napisany i zoptymalizowany algorytm, działający w czasie $O(n^2r)$, przejdzie większość testów (być może nawet wszystkie). Oto charakterystyki poszczególnych testów:

- `map0.in` — test z treści zadania;
- `map1.in` — $n=5$,$r=0$;
- `map2.in` — $n=10$, $r=5$;
- `map3.in` $n=20$, $r=19$;

- `map4.in` $n = 50$, $r = 3$;
- `map5.in` $n = 100$, $r = 50$;
- `map6.in` $n = 150$, $r = 50$;
- `map7.in` $n = 200$, $r = 100$;
- `map8.in` $n = 250$, $r = 10$, rzadka macierz;
- `map9.in` $n = 250$, $r = 50$;
- `map10.in` $n = 250$, $r = 249$;
- `map11.in` $n = 250$, $r = 125$;
- `map12.in` $n = 250$, $r = 125$;
- `map13.in` $n = 250$, $r = 125$.

Wojciech Guzicki
Treść zadania, Opracowanie

Marek Pawlicki
Program

Przedziały

Dany jest ciąg n przedziałów domkniętych $[a_i; b_i]$, gdzie $i = 1, 2, \ldots, n$. Suma tych przedziałów może być przedstawiona w postaci sumy parami rozłącznych przedziałów domkniętych. Zadanie polega na znalezieniu przedstawienia tej sumy w postaci sumy minimalnej liczby parami rozłącznych przedziałów domkniętych. Przedziały tworzące to przedstawienie należy zapisać w pliku wyjściowym w rosnącej kolejności. Mówimy, że dwa przedziały rozłączne $[a; b]$ i $[c; d]$ są ustawione w rosnącej kolejności wtedy i tylko wtedy, gdy $a \leq b < c \leq d$.

Zadanie

Napisz program, który:

- *wczyta z pliku tekstowego* `prz.in` *opis ciągu przedziałów,*
- *wyznaczy parami rozłączne przedziały spełniające warunki zadania,*
- *zapisze wyznaczone przedziały w rosnącej kolejności do pliku tekstowego* `prz.out` *.*

Wejście

W pierwszym wierszu pliku tekstowego `prz.in` *znajduje się jedna liczba całkowita n, spełniająca nierówności $3 \leq n \leq 50\,000$. Jest to liczba przedziałów. W $(i + 1)$-szym wierszu pliku, $1 \leq i \leq n$, znajduje się opis przedziału $[a_i; b_i]$ w postaci dwóch liczb całkowitych a_i i b_i oddzielonych pojedynczym odstępem, będących odpowiednio jego początkiem i końcem, $1 \leq a_i \leq b_i \leq 1\,000\,000$.*

Wyjście

W kolejnych wierszach pliku tekstowego `prz.out` *należy zapisać opisy znalezionych parami rozłącznych przedziałów. W każdym wierszu ma być zapisany opis jednego przedziału w postaci dwóch liczb całkowitych oddzielonych pojedynczym odstępem, będących odpowiednio początkiem i końcem tego przedziału. Przedziały w pliku wyjściowym powinny być zapisane w rosnącej kolejności.*

Przykład

Dla pliku wejściowego `prz.in`*:*

```
5
5 6
1 4
10 10
6 9
8 10
```

poprawną odpowiedzią jest plik wyjściowy `prz.out`*:*

```
1 4
5 10
```

Rozwiązanie

Przedziały $[a_i; b_i]$ sortujemy w kolejności wzrastania początków. Jako bieżący bierzemy pierwszy przedział. Niech będzie to przedział $[a; b]$. Niech kolejnym przedziałem będzie $[c; d]$. Jeśli $c \leq b$ i $d > b$, to modyfikujemy przedział bieżący: będzie nim odtąd $[a; d]$. Jeśli natomiast $c > b$, to zapisujemy przedział bieżący, a nowym przedziałem bieżącym będzie odtąd $[c; d]$. Po wyczerpaniu wszystkich przedziałów zapisujemy przedział bieżący.

Załóżmy, że dane o przedziałach zostały zapisane w tablicy `P` następującej postaci:

```
type
   Przedzial=record
      Poczatek, Koniec : longint;
   end
   Przedzialy=array[1.. n] of Przedzial;
var
   P: Przedzialy;
```

Załóżmy następnie, że tablica `P` jest już posortowana w kolejności wzrastania początków. Opisany wyżej algorytm można zapisać w postaci następującego fragmentu programu:

```
Biezacy:= P[1];
for i:= 2 to n do
  if ( P[ i]. Poczatek ≤ Biezacy.Koniec)
      and ( P[ i]. Koniec > Biezacy.Koniec) then
         Biezacy.Koniec := P[ i]. Koniec
  else if P[ i]. Poczatek > Biezacy.Koniec then begin
         Zapisz( Biezacy);
         Biezacy:= P[ i]
     end;
Zapisz( Biezacy);
```

Procedura `Zapisz` zapisuje dane o przedziale `Biezacy` do pliku wyjściowego.

Jak widać, zadanie to jest dość łatwe. Jednak przy implementacji tego prostego algorytmu mogą wystąpić pewne trudności. Spróbujemy je teraz omówić.

1. Czas działania programu zależy od wyboru procedury sortującej. Nie będziemy tu omawiać różnych sposobów sortowania; algorytmy sortujące były wielokrotnie opisywane w sprawozdaniach z wcześniejszych Olimpiad Informatycznych, można też je znaleźć w wielu popularnych podręcznikach. Wspomnimy tu tylko o tym, że wybór wolno działającej procedury sortującej (np. sortowanie przez wybieranie, sortowanie przez wstawianie, sortowanie bąbelkowe – działających w czasie $O(n^2)$) spowoduje przekroczenie dopuszczalnego limitu czasu dla danych zapisanych w plikach `PRZ7.IN` i `PRZ8.IN`. Zadanie można też rozwiązać z pominięciem sortowania, dołączając kolejny przedział w odpowiednim miejscu do aktualnej listy przedziałów parami rozłącznych. Taki program też będzie działał na ogół w czasie $O(n^2)$. Testy sprawdzające zostały dobrane w taki sposób, by odróżnić rozwiązania wykorzystujące sortowanie w czasie kwadratowym od rozwiązań wykorzystujących sortowanie w czasie $O(n \log n)$ (sortowanie szybkie lub sortowanie stogowe) lub sortowanie w czasie $O(n)$ (sortowanie pozycyjne). Program wzorcowy `PRZ.PAS` używa właśnie sortowania pozycyjnego.

2. Druga trudność polega na braku pamięci. Jeśli dysponujemy kompilatorem dopuszczającym używanie dowolnie długich tablic, to z tą trudnością się nie zetkniemy. Jeśli jednak użyjemy np. kompilatora *Turbo Pascala*, to nie będziemy mogli zadeklarować jednej tablicy zawierającej dane o wszystkich przedziałach. Mianowicie początek i koniec każdego przedziału jest liczbą typu *longint*; zajmuje więc 4 bajty. Dane o każdym przedziale zajmują więc 8 bajtów. Maksymalna liczba przedziałów (50 tysięcy) będzie wymagać 400 000 bajtów. Te dane musimy zapisać w kilku tablicach, używając do tego zmiennych dynamicznych. Można teraz wybrać jedno z kilku rozwiązań. Pierwsze polega na posortowaniu każdej tablicy oddzielnie i wybieraniu jako kolejnego przedziału najmniejszego z przedziałów najmniejszych w tych tablicach (podobnie jak czyni się to w procedurze sortowania przez łączenie). Drugie rozwiązanie polega na traktowaniu tych kilku tablic jak jednej długiej tablicy. Zamiast odwołania `P[i]` do i-tego miejsca w tablicy `P` musimy teraz za każdym razem **obliczyć**, do którego miejsca w której tablicy się odwołujemy. Zaprogramowanie jednoczesnego sortowania takich tablic wymaga trochę staranności, ale nie jest bardzo trudne.

Do tego zadania ułożono 8 testów:

- `prz1.in`: trzy przedziały $[1; 1]$;
- `prz2.in`: 10 przedziałów zachodzących na siebie;

- `prz3.in`: 10 przedziałów rozłącznych, zapisanych w odwrotnej kolejności;
- `prz4.in`: 20 przedziałów takich, że każdy następny zawiera poprzedni;
- `prz5.in`: 6 przedziałów stykających się brzegami;
- `prz6.in`: 1000 losowo wybranych przedziałów;
- `prz7.in`: 20000 losowo wybranych przedziałów;
- `prz8.in`: 50000 losowo wybranych przedziałów.

Wszystkie rozwiązania poprawne (również działające w czasie kwadratowym) przechodziły przez testy od 1 do 6; przez ostatnie dwa testy przechodziły tylko programy wykorzystujące szybsze algorytmy sortowania.

Wojciech Rytter
Treść zadania, Opracowanie

Marcin Stefaniak
Program

Liczby antypierwsze

Oznaczmy przez LiczbaDzielników*(i) liczbę dzielników liczby i (włącznie z 1 i i). Dodatnią liczbę całkowitą nazywamy* **antypierwszą***, gdy ma ona więcej dzielników niż wszystkie dodatnie liczby całkowite mniejsze od niej. Przykładowymi liczbami antypierwszymi są liczby: 1, 2, 4, 6, 12 i 24.*

Zadanie

Napisz program, który:

- *wczyta z pliku tekstowego* `ant.in` *dodatnią liczbę całkowitą n,*
- *wyznaczy największą liczbę antypierwszą nie przekraczającą n,*
- *zapisze wyznaczoną liczbę w pliku tekstowym* `ant.out`*.*

Wejście

W jedynym wierszu pliku tekstowego `ant.in` *znajduje się jedna liczba całkowita* n, $1 \leq n \leq 2\,000\,000\,000$.

Wyjście

W jedynym wierszu pliku `ant.out` *Twój program powinien zapisać dokładnie jedną liczbę całkowitą — największą liczbę antypierwszą nie przekraczającą n.*

Przykład

Dla pliku wejściowego `ant.in`*:*
`1000`
poprawną odpowiedzią jest plik wyjściowy `ant.out`*:*
`840`

Rozwiązanie

Efektywne rozwiązanie zadania opiera się na prostych własnościach liczb antypierwszych:

- liczb antypierwszych mniejszych od zadanego n jest bardzo mało;
- liczby antypierwsze mają bardzo proste rozkłady względem liczb pierwszych;
- liczby pierwsze występujące w tych rozkładach są małe.

Aby obliczyć liczbę dzielników liczby $n > 1$, można rozłożyć ją na czynniki pierwsze. Jeśli

$$n = p_1^{a_1} * p_2^{a_2} * \ldots * p_k^{a_k}$$

to

$$\mathit{LiczbaDzielników}(n) = (a_1 + 1) * (a_2 + 1) * \ldots * (a_k + 1).$$

Widać, że liczba dzielników danej liczby nie zależy od wielkości liczb pierwszych występujących w jej rozkładzie na czynniki pierwsze, a jedynie od krotności, z jaką do tego rozkładu wchodzą.

Przykład
Przykładowymi liczbami antypierwszymi są liczby 1, 2, 4, 6, 12, 24 oraz *duże* liczby:

$$27935107200 \;=\; 2^7 3^3 5^2 7^1 11^1 13^1 17^1 19^1$$

$$2248776129600 \;=\; 2^6 3^3 5^2 7^2 11^1 13^1 17^1 19^1 23^1$$

$$\mathit{LiczbaDzielników}(27935107200) = 8 \cdot 4 \cdot 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2$$
$$\mathit{LiczbaDzielników}(2248776129600) = 7 \cdot 4 \cdot 3 \cdot 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$$

Przyjmijmy, że przez p_i rozumiemy i-tą kolejną liczbę pierwszą, a przez a_i rozumiemy krotność p_i w rozkładzie n na czynniki pierwsze (możliwe, że $a_i = 0$).

Jeżeli ciąg liczb a_i nie jest nierosnący, to n nie może być liczbą antypierwszą, gdyż wtedy moglibyśmy tak zamienić miejscami pewne krotności, że otrzymalibyśmy mniejszą liczbę o tej samej liczbie dzielników. Dlatego możemy ograniczyć poszukiwania liczb antypierwszych do liczb o „nierosnących" rozkładach na czynniki pierwsze. Łatwo możemy wygenerować wszystkie takie rozkłady dla liczb nie przekraczających danego ograniczenia górnego (kolejne rozkłady wyznaczamy w porządku leksykograficznym).

Jak się okazuje, liczb antypierwszych $\leq 2\,000\,000\,000$ jest niedużo, dokładnie 68. Pierwsze 41 liczb antypierwszych przedstawiono w poniższej tabeli, razem z potęgami w rozkładach na kolejne liczby pierwsze.

2	1					
4	2					
6	1	1				
12	2	1				
24	3	1				
36	2	2				
48	4	1				
60	2	1	1			
120	3	1	1			
180	2	2	1			
240	4	1	1			
360	3	2	1			
720	4	2	1			
840	3	1	1	1		
1260	2	2	1	1		
1680	4	1	1	1		
2520	3	2	1	1		
5040	4	2	1	1		
7560	3	3	1	1		
10080	5	2	1	1		
15120	4	3	1	1		
20160	6	2	1	1		
25200	4	2	2	1		
27720	3	2	1	1	1	
45360	4	4	1	1		
50400	5	2	2	1		
55440	4	2	1	1	1	
83160	3	3	1	1	1	
110880	5	2	1	1	1	
166320	4	3	1	1	1	
221760	6	2	1	1	1	
277200	4	2	2	1	1	
332640	5	3	1	1	1	
498960	4	4	1	1	1	
554400	5	2	2	1	1	
665280	6	3	1	1	1	
720720	4	2	1	1	1	1
1081080	3	3	1	1	1	1
1441440	5	2	1	1	1	1
2162160	4	3	1	1	1	1
2882880	6	2	1	1	1	1

Niech $p_1, p_2, \ldots, p_t$ będzie najdłuższym ciągiem kolejnych liczb pierwszych takich, że ich iloczyn nie przekracza n. Liczymy najpierw t. Istotnym jest to, że t jest stosunkowo małe, zatem liczba liczb pierwszych i ciągów do rozważenia nie jest bardzo duża. Aby znaleźć największą liczbę antypierwszą mniejszą lub równą n, rozważmy (najlepiej w kolejności leksykograficznej) wszystkie ciągi $i_1, i_2, \dots i_t$ spełniające dwa poniższe warunki.

$$p_1^{i_1} p_2^{i_2} \dots p_t^{i_t} \leq n$$

$$i_1 \geq i_2 \geq i_3 \dots \geq i_t \geq 0$$

Jak się okazuje, takich ciągów, potencjalnych kandydatów na liczby antypierwsze $\leq 2\,000\,000\,000$, jest niedużo – około 1500. Bez trudu możemy więc je przesortować i jeden raz przeglądając znaleźć największą liczbę antypierwszą nie przekraczającą danego ograniczenia górnego.

Obserwacja
Przyjrzyjmy się strukturze pierwszych 40 liczb antypierwszych. W tabeli pokazane są wszystkie liczby antypierwsze mniejsze od 3 milionów. Zauważmy, że ostatnie potęgi, poza dwoma przypadkami, są równe 1. Przedostatnie potęgi są też nieduże. Nie jest to przypadek, ponieważ zachodzi poniższy fakt (dosyć trudny do uzasadnienia, ale prawdziwy). Korzystając z tego można ograniczyć przestrzeń rozpatrywanych danych.

Fakt
Dla każdej liczby antypierwszej, oprócz liczb 2 i 36, ostatnia potęga w rozkładzie na liczby pierwsze jest równa 1, natomiast potęgi druga i trzecia od końca nie przekraczają nigdy 4.
Zauważmy olbrzymią różnicę w liczbie liczb antypierwszych mniejszych od zadanej liczby n (dostatecznie dużej) w porównaniu z liczbą liczb pierwszych. Właśnie dzięki temu testowanie liczb antypierwszych jest znacznie łatwiejsze od testowania liczb pierwszych.

Testy

Zadanie testowane było na zestawie 21 danych testowych.

- `ant0.in` — $n = 1000$, wynik: 840;
- `ant1.in` — $n = 1$, wynik: 1;
- `ant2.in` — $n = 3$, wynik: 2;
- `ant3.in` — $n = 5$, wynik: 4;
- `ant4.in` — $n = 100000$, wynik: 83160;
- `ant5.in` — $n = 8$, wynik: 6;
- `ant6.in` — $n = 5041$, wynik: 5040;
- `ant7.in` — $n = 20159$, wynik: 15120;
- `ant8.in` — $n = 15120$, wynik: 15120;
- `ant9.in` — $n = 75000$, wynik: 55440;
- `ant10.in` — $n = 150000$, wynik: 110880;
- `ant11.in` — $n = 1000000$, wynik: 720720;
- `ant12.in` — $n = 3000$, wynik: 2520;
- `ant13.in` — $n = 8000000$, wynik: 7207200;
- `ant14.in` — $n = 15000000$, wynik: 14414400;
- `ant15.in` — $n = 30000$, wynik: 27720;
- `ant16.in` — $n = 60000000$, wynik: 43243200;
- `ant17.in` — $n = 110000000$, wynik: 73513440;
- `ant18.in` — $n = 354218765$, wynik: 294053760;
- `ant19.in` — $n = 600000000$, wynik: 551350800;
- `ant20.in` — $n = 1000000000$, wynik: 735134400;
- `ant21.in` — $n = 2000000000$, wynik: 1396755360.

Marcin Jurdziński
Treść zadania, Opracowanie

Tomasz Waleń
Program

Gra w zielone

Gra **w zielone** *jest grą dla dwóch graczy — nazwijmy ich Ania i Bolek — polegającą na przesuwania pionka po planszy.*

Część pól planszy jest pokolorowana na zielono, a pozostałe są białe. Wszystkie pola są ponumerowane liczbami naturalnymi z zakresu $1 \ldots (a+b)$, pola o numerach z zakresu $1 \ldots a$ należą do Ani, natomiast pola o numerach $(a+1) \ldots (a+b)$ należą do Bolka.

Dla każdego pola dany jest zbiór **następników**, *zawierający te pola planszy, do których można z niego przejść w jednym ruchu. Zbiory te zostały tak dobrane, że z pola należącego do Ani można przejść w jednym ruchu tylko na pole należące do Bolka i odwrotnie.*

Na początku gry ustawiamy pionek na dowolnym polu, a następnie gracze na przemian przestawiają pionek ze swojego pola na dowolny następnik tego pola — należący do przeciwnika. Grę rozpoczyna właściciel pola, z którego zaczynamy rozgrywkę. Zakładamy, że wszystkie pola mają niepuste zbiory następników, a więc zawsze można wykonać ruch. Gra kończy się w momencie, gdy pionek stanie po raz drugi na tym samym polu. Jeśli w sekwencji ruchów, od pierwszego do powtórnego zajęcia tego pola, pionek stanął przynajmniej raz na polu zielonym, to wygrywa Ania, w przeciwnym przypadku wygrywa Bolek.

Powiemy, że Ania ma **strategię wygrywającą dla danego pola początkowego**, *jeśli istnieje metoda gwarantująca jej wygraną w rozgrywce zaczynającej się od tego pola, niezależnie od tego, jakie ruchy będzie wykonywał Bolek.*

Zadanie

Napisz program, który:

- *wczyta z pliku tekstowego* `gra.in` *opis planszy do gry w zielone,*
- *znajdzie zbiór pól planszy, dla których Ania ma strategię wygrywającą,*
- *zapisze wynik w pliku tekstowym* `gra.out`.

Wejście

W pierwszym wierszu pliku tekstowego `gra.in` *zapisane są dwie nieujemne liczby całkowite a, b oddzielone pojedynczym odstępem, oznaczające odpowiednio: liczbę pól należących do Ani i liczbę pól należących do Bolka. Liczby a, b spełniają warunek: $1 \leq a + b \leq 3\,000$. W kolejnych $a+b$ wierszach opisano pola planszy — najpierw pola należące do Ani, a następnie pola należące do Bolka. Wiersz o numerze $i+1$, dla $1 \leq i \leq a+b$, zawiera na początku liczby całkowite z, k oddzielone pojedynczym odstępem, oznaczające odpowiednio kolor pola o numerze i (0 oznacza kolor biały, 1 — kolor zielony), oraz liczbę następników tego pola. Następnie w tym wierszu zapisane jest k liczb całkowitych ($1 \leq k < a+b$), pooddzielanych pojedynczymi odstępami, będącymi numerami następników danego pola. Liczba pól zielonych na planszy nie przekracza 100. Suma liczb następników wszystkich pól na planszy nie przekracza $30\,000$.*

Wyjście

Pierwszy wiersz pliku tekstowego `gra.out` *powinien zawierać dokładnie jedną liczbę całkowitą p, oznaczającą liczbę pól, dla których Ania ma strategię wygrywającą. Następne p wierszy powinno zawierać numery tych pól zapisane w kolejności rosnącej — każda liczba powinna zostać zapisana w osobnym wierszu.*

Przykład

Dla pliku wejściowego `gra.in`*:*

```
5 3
0 2 6 7
0 3 6 7 8
0 1 8
1 1 7
1 1 8
```

```
1 2 1 2
0 2 1 2
0 2 3 4
```

poprawną odpowiedzią jest plik wyjściowy `gra.out`*:*

```
5
1
2
4
6
7
```

Rozwiązanie

Ścisłe sformułowanie faktu, że w grze w zielone Ania ma strategię wygrywającą z danego pola początkowego nie jest banalne i jest zaledwie naszkicowane w treści zadania. Jego sprecyzowanie oraz algorytmiczną charakteryzację należy zatem do pewnego stopnia traktować jako część zadania postawionego przed uczestnikami olimpiady.

Przez P będziemy oznaczać zbiór wszystkich pól na planszy, a przez Z zbiór wszystkich pól zielonych.

Aby ułatwić opis strategii wygrywających w grze w zielone oraz ich poszukiwanie, wygodnie jest rozważyć następującą uproszczoną wersję gry. Niech $X \subseteq Z$ będzie pewnym zbiorem zielonych pól na planszy. *Uproszczona gra w zielone* toczy się w sposób podobny do zwyczajnej gry w zielone. Różnica polega na tym, że w uproszczonej grze w zielone rozgrywka może się zakończyć wcześniej: gdy pionek odwiedzi zielone pole ze zbioru X po raz pierwszy, rozgrywka zostaje przerwana i Ania zwycięża.

Przez *Wymuś*(X) oznaczamy zbiór pól, z których Ania ma strategię wygrywającą w uproszczonej grze w zielone. Zauważ, że *Wymuś*(X) jest zbiorem pól z których Ania może wymusić przesunięcie pionka na pewne zielone pole ze zbioru X w taki sposób, że pionek nie staje na żadnym polu dwa razy.

W poniższym fakcie wykazujemy, że rozwiązanie uproszczonej gry w zielone dla $X = Z$, tj. obliczenie zbioru *Wymuś*(Z), jest pomocne w ustaleniu pewnego zbioru pól, z których Bolek ma strategię wygrywającą w normalnej grze w zielone.

Fakt 1 (Strategia wygrywająca dla Bolka) *Bolek ma strategię wygrywającą z każdego pola, które nie jest w zbiorze Wymuś*(Z).

Dowód. Fakt, że pole p nie należy do zbioru *Wymuś*(Z) oznacza, że w uproszczonej grze w zielone rozpoczynającej się w polu p, Bolek potrafi zapobiec dojściu do jakiegokolwiek zielonego pola. Cała rozgrywka toczy się więc i kończy w białej części planszy — to gwarantuje zwycięstwo Bolka w zwyczajnej grze w zielone z pola p. □

Przykład 2 *W tym i w poniższych przykładach ilustrujemy pojęcia i fakty opisane w tym opracowaniu na przykładzie gry w zielone z przykładu z treści zadania.*
Zauważ, że $Z = \{4,5,6\}$ *oraz Wymuś*$(Z) = \{1,2,4,5,6,7\}$. *W polach spoza zbioru Wymuś*(Z)*, tj. w polach ze zbioru* $\{3,8\}$*, Bolek ma strategię wygrywającą polegającą na wybraniu pola 3 jako następnika w polu 8.*

Następnie spróbujemy ustalić jakie warunki gwarantują, że strategia wygrywająca dla Ani w uproszczonej grze w zielone daje Ani zwycięstwo także w normalnej grze w zielone. W tym celu przydatne okazuje się pojęcie pułapki dla Bolka.

Definicja 3 (Pułapka dla Bolka) *Powiemy, że zbiór pól Wymuś*(X) *jest* pułapką *dla Bolka, jeśli dla każdego pola p ze zbioru* X *mamy:*

1. *pole p należy do Ani i* istnieje *następnik pola p znajdujący się w zbiorze Wymuś*(X)*; lub*
2. *pole p należy do Bolka i* każdy *następnik pola p znajduje się w zbiorze Wymuś*(X).

Przykład 4 *Niech* $X = \{4,6\}$. *Zbiór Wymuś*$(X) = \{1,2,4,6,7\}$ *jest pułapką dla Bolka, ponieważ:*

- *istnieje następnik pola 4 (tj. pole 7) w zbiorze Wymuś*(X)*, oraz*
- *każdy następnik pola 6 (tj. zarówno pole 1 jak i pole 2) należy do zbioru Wymuś*(X).

Fakt 5 (Strategia wygrywająca dla Ani) *Jeśli zbiór Wymuś*(X) *jest pułapką dla Bolka, to Ania ma strategię wygrywającą z każdego pola w zbiorze Wymuś*(X).

Dowód. Rozważmy następującą strategię dla Ani:

- w każdym białym polu Ani ze zbioru *Wymuś*(X), Ania gra zgodnie z jej strategią wygrywającą w uproszczonej grze w zielone;
- w każdym zielonym polu Ani ze zbioru *Wymuś*(X), tj. w każdym polu Ani ze zbioru X, Ania wybiera następnika który należy do zbioru *Wymuś*(X): taki następnik istnieje, bo zbiór *Wymuś*(X) jest pułapką dla Bolka.

Zauważ, że w każdej rozgrywce zgodnej z tą strategią, pionek nigdy nie opuszcza zbioru *Wymuś*(X). Co więcej, żaden cykl (tj. fragment rozgrywki rozpoczynający się i kończący w tym samym polu) w takiej rozgrywce nie składa się tylko z białych pól, ponieważ strategia Ani na białych polach jest wygrywająca w uproszczonej grze w zielone. Dlatego każda rozgrywka zgodna z powyższą strategią jest wygrywająca dla Ani w zwyczajnej grze w zielone. □

Pozostaje nam do rozwiązania sytuacja, w której zbiór *Wymuś*(Z) nie jest pułapką dla Bolka. Niech $Z_1 \subseteq Z$ będzie zbiorem zielonych pól, które spełniają warunki 1 lub 2 definicji pułapki dla Bolka, dla $X = Z$.

Zauważ, że w zielonych polach nie należących do zbioru Z_1, Bolek ma następującą strategię wygrywającą: najpierw przesuń pionek w jednym kroku do pola nie należącego do zbioru *Wymuś*(Z), a następnie użyj strategii wygrywającej zagwarantowanej przez Fakt 1. Możemy zatem odrzucić zielone pola spoza zbioru Z_1, jako "bezużyteczne" dla Ani.

Aby ponowić próbę użycia Faktu 5 do wyznaczenia strategii wygrywającej dla Ani, rozważmy zbiór *Wymuś*(Z_1). Jeśli ten zbiór jest pułapką dla Bolka, to rozwiązanie gry w zielone jest gotowe (można wykazać — patrz dowód ogólniejszego Faktu 7 poniżej — że także we wszystkich białych polach spoza zbioru *Wymuś*(Z_1) Bolek ma strategię wygrywającą.) W przeciwnym wypadku należy ponownie odrzucić zielone pola, które okazały się bezużyteczne dla Ani, itd.

Powyższe rozumowanie motywuje następującą metodę obliczania kolejnych, coraz mniejszych zbiorów zielonych pól, które kandydują do roli pól dających Ani strategię wygrywającą (poprzez Fakt 5).

```
1:  procedure NajwiększaPułapkaDlaBolka
2:  begin
3:      Z_0 := Z
4:      i := 0
5:      while Wymuś(Z_i) nie jest pułapką dla Bolka do
6:      begin
7:          Z_{i+1} := zbiór pól z Z_i spełniających warunki 1 lub 2 definicji 3
8:          i := i + 1
9:      end
10: end
```

Przykład 6
Z przykładu 2 mamy $Z_0 = Z = \{4, 5, 6\}$ *oraz Wymuś*$(Z_0) = \{1, 2, 4, 5, 6, 7\}$. *Zauważ, że* Z_0 *nie jest pułapką dla Bolka, bo pole 5 należy do Ani, ale jedyny następnik pola 5, tj. pole 8,* nie należy *do zbioru Wymuś*(Z_0). *Pole 5 jest jedynym polem z* Z_0 *nie spełniającym warunków 1 lub 2 definicji 3, więc* $Z_1 = \{4, 6\}$. *Z przykładu 4 wiemy, że zbiór Wymuś*(Z_1) *jest pułapką dla Bolka.*

Zauważ, że jeśli warunek kontynuacji wykonywania pętli 5:–9: jest spełniony, to istnieje przynajmniej jedno zielone pole w zbiorze Z_i *nie* spełniające warunków 1 lub 2 definicji 3. Dlatego ciąg zbiorów Z_i jest malejący, a stąd wynika, że liczba iteracji pętli 5:–9: wynosi $O(k)$.

Niech ℓ będzie najmniejszą liczbą dla której zbiór *Wymuś*(Z_ℓ) jest pułapką dla Bolka. Oczywiście Ania ma strategię wygrywającą ze zbioru *Wymuś*(Z_ℓ) — wynika to z Faktu 5. Aby wykazać, że *Wymuś*(Z_ℓ) to zbiór *wszystkich* pól z których Ania ma strategię wygrywającą, wystarczy udowodnić, że z każdego pola spoza zbioru *Wymuś*(Z_ℓ), strategię wygrywającą ma Bolek.

Fakt 7 (Strategia wygrywająca dla Bolka: ogólny przypadek)
Dla każdego $i \leq \ell$, *Bolek ma strategię wygrywającą z każdego pola spoza zbioru Wymuś*(Z_i).

Dowód. Dowód przebiega przez indukcję ze względu na i. Zauważ, że dla $i = 0$ teza wynika wprost z Faktu 1.

Załóżmy, że dla pewnego $i < \ell$, Bolek ma strategię wygrywającą z każdego pola spoza zbioru *Wymuś*(Z_i). Wykażemy, że wtedy Bolek ma także strategię wygrywającą z każdego pola spoza zbioru *Wymuś*(Z_{i+1}).

Wystarczy udowodnić, że Bolek ma strategię wygrywającą z każdego pola ze zbioru *Wymuś*$(Z_i) \setminus$ *Wymuś*(Z_{i+1}) — dla pól nie należących do zbioru *Wymuś*(Z_i) teza wynika z hipotezy indukcyjnej.

Rozważmy dwa rodzaje pól ze zbioru *Wymuś*$(Z_i) \setminus$ *Wymuś*(Z_{i+1}):

1. Jeśli pole jest zielone, to nie spełnia warunków 1 lub 2 definicji 3, więc Bolek może z tego pola w jednym kroku wymusić przesunięcie pionka poza zbiór $\mathit{Wymu\acute{s}}(Z_i)$, skąd ma strategię wygrywającą (z hipotezy indukcyjnej).

2. Jeśli pole jest białe, to Bolek korzysta ze strategii w uproszczonej grze w zielone, która zapobiega dojściu pionka do zbioru $\mathit{Wymu\acute{s}}(Z_{i+1})$. Jeśli w wyniku rozgrywki pionek stanie na jednym z zielonych pól w zbiorze $\mathit{Wymu\acute{s}}(Z_i)\setminus \mathit{Wymu\acute{s}}(Z_{i+1})$, wtedy Bolek postępuje jak w przypadku 1 i wygrywa. W przeciwnym razie rozgrywka toczy się i kończy w białej części zbioru $\mathit{Wymu\acute{s}}(Z_i)\setminus \mathit{Wymu\acute{s}}(Z_{i+1})$, więc jest wygrywająca dla Bolka. □

Aby precyzyjnie oszacować koszt czasowy działania algorytmu *NajwiększaPułapkaDlaBolka* musimy ustalić, jaki jest koszt rozwiązywania uproszczonej gry w zielone, tj. obliczania zbioru $\mathit{Wymu\acute{s}}(X)$. Powiemy, że warunek $\mathit{AniaWymusza}(T,p)$ zachodzi dla pewnego zbioru pól T, oraz pola p, jeśli mamy:

- pole p należy do Ani i *istnieje* następnik pola p należący do zbioru T, lub
- pole p należy do Bolka i *każdy* następnik pola p należy do zbioru T.

Ćwiczenie 8 (Poprawność algorytmu *ObliczZbiórWymuś*)
Wykaż, że poniższa procedura ObliczWymuś(X) wyznacza poprawnie zbiór pól z których Ania ma strategię wygrywającą w uproszczonej grze w zielone, tj. zbiór Wymuś(X).

```
procedure  ObliczWymuś(X)
begin
   W := X
   while istnieje pole p ∉ W takie, że AniaWymusza(W,p) do
      dodaj pole p do zbioru W
   return(W)
end
```

Procedurę *ObliczWymuś* można zaimplementować tak, aby działała w czasie $O(m)$, gdzie m jest sumą liczb następników dla wszystkich pól. Przykładowa implementacja znajduje się na załączonej dyskietce.

Powyższą algorytmiczną analizę struktury strategii wygrywających dla Ani i Bolka w grze w zielone można podsumować w następujący sposób.

Twierdzenie 9 (Poprawność algorytmu *NajwiększaPułapkaDlaBolka*)
Niech ℓ będzie najmniejszą liczbą, dla której w procedurze NajwiększaPułapkaDlaBolka zbiór Wymuś(Z_ℓ) jest pułapką dla Bolka. Wtedy Ania ma strategię wygrywającą z pól ze zbioru Wymuś(Z_ℓ), a Bolek ma strategię wygrywającą ze wszystkich pozostałych pól. Algorytm można zaimplementować tak, by działał w czasie $O(k\cdot m)$, gdzie k jest liczbą zielonych pól, a m jest sumą liczb następników dla wszystkich pól.

Inne rozwiązania

W tym podrozdziale omówimy alternatywny warunek wystarczający i konieczny dla istnienia strategii wygrywających dla Bolka lub Ani w Grze w Zielone. Interesującą cechą tego warunku jest jego lokalność: jeśli każde pole jest w odpowiedniej, dość łatwej do sprawdzenia relacji ze zbiorem jego następników, to mamy gwarancję, że istnieje "globalna" strategia wygrywająca dla odpowiedniego gracza.

Definicja 10 (Dobre etykietowanie dla Bolka)
Rozważmy funkcję $\beta : P \to \{0,1,2,\ldots,k,\infty\}$: z każdym polem $p \in P$, związana jest etykieta $\beta(p)$, która jest liczbą naturalną nie większą niż k, lub symbolem ∞. Przyjmujemy umownie, że ∞ jest większa od każdej liczby naturalnej, czyli $i < \infty$, dla każdej liczby naturalnej i. Powiemy, że etykietowanie β jest dobre dla Bolka w polu *p, jeśli $\beta(p) = \infty$ lub spełnione są następujące warunki:*

- *jeśli pole p należy do Ani, to dla* każdego *następnika r pola p zachodzi warunek PostępBolka$(\beta;p,r)$;*
- *jeśli pole p należy do Bolka, to* istnieje *następnik r pola p taki, że zachodzi warunek PostępBolka$(\beta;p,r)$;*

gdzie warunek PostępBolka$(\beta;p,r)$ jest spełniony wtedy i tylko wtedy gdy:

1. *$\beta(p) > \beta(r)$, jeśli pole p jest zielone, oraz*
2. *$\beta(p) \geq \beta(r)$, jeśli pole p jest białe.*

Mówimy, że etykietowanie β jest dobre *dla Bolka, jeśli β jest dobre dla Bolka we wszystkich polach planszy.*

Przykład 11 *Zauważ, że zbiór P pól Gry w Zielone z treści zadania jest równy $P = \{1, 2, 3, \ldots, 8\}$. Niech etykietowanie $\beta : P \to \{0, 1, 2, 3, \infty\}$ będzie zdefiniowane przez następującą tabelę.*

p	1	2	3	4	5	6	7	8
$\beta(p)$	∞	∞	1	∞	3	∞	∞	1

Etykietowanie β jest dobre dla Bolka ponieważ:

- *pole 3 należy do Ani, pole 8 jest jedynym następnikiem pola 3, oraz zachodzi PostępBolka(β; 3, 8), tj. $\beta(3) \geq \beta(8)$; zauważ, że wystarczy tu nieostra nierówność, bo pole 3 jest białe;*
- *pole 5 należy do Ani, pole 8 jest jedynym następnikiem pola 5, oraz zachodzi PostępBolka(β; 5, 8), tj. $\beta(5) > \beta(8)$; zauważ, że wymagana tu jest ostra nierówność, bo pole 5 jest zielone;*
- *pole 8 należy do Bolka i dla pola 3, które jest następnikiem pola 8, zachodzi PostępBolka(β; 8, 3), tj. $\beta(8) \geq \beta(3)$.*

Za tą—na pierwszy rzut oka—zawiłą definicją kryje się następująca prosta intuicja. O etykiecie $\beta(p)$ pola p — o ile ta etykieta jest liczbą naturalną, a nie symbolem ∞ — można myśleć jako o górnym ograniczeniu na liczbę zielonych pól, które Bolek może być zmuszony (przez Anię) odwiedzić w rozgrywce, jeśli Bolek wybiera zawsze następnika o jak najmniejszej etykiecie. Wykażemy, że wybieranie następnika o najmniejszej etykiecie gwarantuje Bolkowi zwycięstwo w grze w zielone, jeśli etykietowanie jest dobre dla Bolka.

Fakt 12 (Strategia wygrywająca dla Bolka)
Jeśli etykietowanie $\beta : P \to \{0, 1, 2, \ldots, k, \infty\}$ jest dobre dla Bolka, to Bolek ma strategię wygrywającą z każdego pola p takiego, że $\beta(p) \neq \infty$.

Dowód: Wykażemy, że następująca strategia jest wygrywająca dla Bolka: w każdym polu p należącym dla Bolka, Bolek wybiera następnika r pola p, o jak najmniejszej etykiecie $\beta(r)$. Zauważmy, że z definicji dobrego etykietowania dla Bolka wynika, że wtedy dla każdej rozgrywki $p_1, p_2, p_3, \ldots, p_i$ mamy

$$\beta(p_1) \geq \beta(p_2) \geq \beta(p_3) \geq \cdots \geq \beta(p_i).$$

Niech $p_j = p_i$, dla pewnego $j < i$, tj. $p_j = p_i$ jest pierwszym polem na którym pionek stanął dwa razy. Z powyższego warunku wynika, że $\beta(p_j) \geq \beta(p_{j+1}) \geq \beta(p_{j+2}) \geq \cdots \geq \beta(p_i) = \beta(p_j)$, więc musi być $\beta(p_j) = \beta(p_{j+1}) = \beta(p_{j+2}) = \cdots = \beta(p_i)$. Z definicji dobrego etykietowania dla Bolka — a dokładniej z punktu 1 definicji warunku *PostępBolka*(β, p, r) — wynika, że pola $p_j, p_{j+1}, p_{j+2}, \ldots, p_i$ są białe. Inaczej mówiąc, żadne pole odwiedzone pomiędzy pierwszą i drugą wizytą w polu $p_j = p_i$ nie jest zielone, czyli rozgrywka jest wygrywająca dla Bolka. □

Powyższy fakt oznacza, że istnienie dobrego etykietowania β dla Bolka, dla którego zachodzi $\beta(p) \neq \infty$, jest warunkiem wystarczającym dla istnienia strategii wygrywającej dla Bolka z pola p. Poniżej naszkicujemy dowód faktu, że jest to również warunek konieczny, oraz przedstawimy wydajny sposób znajdowania dobrych etykietowań. Rozważmy następującą procedurę poszukiwania dobrego etykietowania dla Bolka:

```
procedure PodnośEtykietowanie
begin
    dla każdego pola p do β(p) := 0
    while β nie jest dobre dla Bolka do
    begin
        wybierz pole p w którym β nie jest dobre dla Bolka
        β(p) := Popraw(β, p)
    end
end
```

gdzie *Popraw*(β, p) to najmniejsza etykieta nie mniejsza od $\beta(p)$, która przypisana polu p gwarantuje, że etykietowanie β staje się dobre w polu p. Wartość *Popraw*(β, p) można obliczyć na przykład tak:

$$\mathit{Popraw}(\beta, p) = \begin{cases} \beta(p) & \text{jeśli } \beta \text{ jest dobre dla Bolka w polu } p, \\ \mathit{MaxMin}(\beta, p) & \text{jeśli } p \text{ jest białe,} \\ \infty & \text{jeśli } p \text{ jest zielone i } \mathit{MaxMin}(\beta, p) \geq k, \\ 1 + \mathit{MaxMin}(\beta, p) & \text{jeśli } p \text{ jest zielone i } \mathit{MaxMin}(\beta, p) < k; \end{cases}$$

gdzie $MaxMin(\beta, p)$ jest zdefiniowane w następujący sposób:

$$MaxMin(\beta, p) = \begin{cases} \max\{\beta(r)\ :\ r \text{ jest następnikiem } p\} & \text{jeśli } p \text{ należy do Ani,} \\ \min\{\beta(r)\ :\ r \text{ jest następnikiem } p\} & \text{jeśli } p \text{ należy do Bolka.} \end{cases}$$

Warto tu zwrócić uwagę na to, że $Popraw(\beta, p) \geq \beta(p)$ oraz, że $Popraw(\beta, p) > \beta(p)$, jeśli etykietowanie β nie jest dobre w polu p.

Przykład 13 *Dla skrócenia i uproszczenia ilustracji działania algorytmu PodnośEtykietowanie wprowadzamy następującą notację na oznaczenie ciągu kilku operacji Popraw:*

$$\begin{aligned} Popraw(\beta, [p_1, p_2]) &= Popraw\big(Popraw(\beta, p_1), p_2\big) \\ Popraw(\beta, [p_1, p_2, p_3]) &= Popraw\Big(Popraw\big(Popraw(\beta, p_1), p_2\big), p_3\Big) \\ &\ \vdots \end{aligned}$$

Rozważmy następujące wykonanie procedury PodnośEtykietowanie:

p	1	2	3	4	5	6	7	8
$\beta_0(p)$	0	0	0	0	0	0	0	0
$\beta_1(p) = Popraw\big(\beta_0, [4,5,6]\big)(p)$	0	0	0	1	1	1	0	0
$\beta_2(p) = Popraw\big(\beta_1, [1,2]\big)(p)$	1	1	0	1	1	1	0	0
$\beta_3(p) = Popraw(\beta_2, 7)(p)$	1	1	0	1	1	1	1	0
$\beta_4(p) = Popraw\big(\beta_3, [4,6,1,2,7]\big)(p)$	2	2	0	2	1	2	2	0
$\beta_5(p) = Popraw\big(\beta_4, [4,6,1,2,7]\big)(p)$	3	3	0	3	1	3	3	0
$\beta_6(p) = Popraw\big(\beta_5, [4,6,1,2,7]\big)(p)$	∞	∞	0	∞	1	∞	∞	0

Etykietowanie β_6 jest dobre dla Bolka. Zauważ, że etykietowanie β_6 jest "mniejsze" niż etykietowanie β z przykładu 11, w tym sensie, że dla każdego pola $p \in P$, mamy $\beta_6(p) \leq \beta(p)$.

Można myśleć o procedurze *PodnośEtykietowanie* jako o metodzie przybliżania etykietowania β „z dołu", do „najmniejszego" dobrego etykietowania. Istotnie, początkowe etykietowanie $\beta = \beta_0$, gdzie $\beta_0(p) = 0$, dla każdego pola p, jest „mniejsze" niż jakiekolwiek inne etykietowanie. (Dla ścisłości, mówimy, że etykietowanie β jest „mniejsze" lub równe niż etykietowanie β', jeśli $\beta(p) \leq \beta'(p)$, dla każdego pola p.) Kolejne „poprawki" polegające na podnoszeniu wartości etykietowania β w polu, w którym β nie jest dobre dla Bolka, zachowują własność, że etykietowanie β jest mniejsze lub równe niż każde dobre etykietowanie. Jest tak dlatego, ponieważ w procedurze *PodnośEtykietowanie* wartość etykiety w pewnym polu jest zawsze podnoszona tylko o tyle, o ile to jest konieczne, aby etykietowanie stało się dobre dla Bolka w tym polu.

Przykład 14 *Etykietowanie $\beta_6 : P \to \{1, 2, 3, \infty\}$:*

p	1	2	3	4	5	6	7	8
$\beta_6(p)$	∞	∞	0	∞	1	∞	∞	0

z przykładu 13 jest najmniejszym *dobrym etykietowaniem dla Bolka w grze z treści zadania. Inaczej mówiąc, etykietowanie β_6 jest mniejsze nie tylko od dobrego etykietowania beta dla Bolka z przykładu 11, ale jest nie większe niż* każde *dobre etykietowanie dla Bolka w tej grze.*

Zauważ, że każde pole może być poprawione przez procedurę *PodnośEtykietowanie* co najwyżej $k + 1$ razy, zatem liczba powtórzeń pętli 4:–8: wynosi $O(k \cdot n)$.

Co więcej, etykietowanie β po zakończeniu procedury *PodnośEtykietowanie* jest dobrym etykietowaniem dla Bolka; w „najgorszym" przypadku mamy $\beta(p) = \infty$, dla każdego pola p — „największe" dobre etykietowanie. Z faktu 12 wynika, że Bolek ma strategię wygrywającą z każdego pola p takiego, że $\beta(p) \neq \infty$. *Najmniejsze* dobre etykietowanie dla Bolka, tj. etykietowanie β obliczane przez procedurę *PodnośEtykietowanie*, jest szczególnie interesujące ponieważ można z niego łatwo odczytać rozwiązanie gry w zielone, tj. zbiór pól z których istnieje strategia wygrywająca dla Ani.

Twierdzenie 15 (Strategia wygrywająca dla Ani)
Etykietowanie $\beta : P \to \{0, 1, 2, \ldots, k, \infty\}$ obliczone przez procedurę PodnośEtykietowanie ma taką własność, że Ania ma strategię wygrywającą z pola p wtedy i tylko wtedy gdy $\beta(p) = \infty$.

Dowód tego twierdzenia nie jest natychmiastowy i wymaga dość subtelnej argumentacji. Poniżej szkicujemy przydatne w tym celu pojęcia oraz ogólny tok rozumowania. Wypełnienie szczegółów dowodu pozostawiamy jako ćwiczenie dla dociekliwego czytelnika.

Definicja 16 (Dobre etykietowanie dla Ani)
Rozważmy etykietowanie $\alpha : P \to \{0, 1, 2, \ldots, n-k, \infty\}$. Powiemy, że etykietowanie α jest dobre dla Ani w polu p, *jeśli $\alpha(p) = \infty$ lub spełnione są następujące warunki:*

- *jeśli pole p należy do Ani, to* istnieje *następnik r pola p taki, że zachodzi warunek $PostępAni(\alpha; p, r)$,*
- *jeśli pole p należy do Bolka, to dla* każdego *następnika r pola p, zachodzi warunek $PostępAni(\alpha; p, r)$;*

gdzie warunek $PostępAni(\alpha; p, r)$ oznacza, że $\alpha(p) > \alpha(r)$, jeśli pole p jest białe. Mówimy, że etykietowanie α jest dobre *dla Ani, jeśli α jest dobre dla Ani we wszystkich polach planszy.*

Przykład 17 *Etykietowanie $\alpha : P \to \{0, 1, 2, 3, 4, 5, \infty\}$ zdefiniowane przez następującą tabelę jest dobre dla Ani.*

p	1	2	3	4	5	6	7	8
$\alpha(p)$	1	1	∞	0	∞	0	2	∞

Podobnie jak w przypadku dobrych etykietowań dla Bolka, można skonstruować strategię wygrywającą dla Ani, jeśli dane jest dobre etykietowanie dla Ani. Dowód następującego łatwego faktu pozostawiamy czytelnikowi.

Ćwiczenie 18 (Strategia wygrywająca dla Ani) *Wykaż, że jeśli etykietowanie $\alpha : P \to \{0, 1, 2, \ldots, n-k, \infty\}$ jest dobre dla Ani, to Ania ma strategię wygrywającą z każdego pola p takiego, że $\alpha(p) \neq \infty$.*

W celu wykazania Twierdzenia 15 do procedury *PodnośEtykietowanie* dodamy instrukcje obliczające etykietowanie $\alpha : P \to \{0, 1, 2, \ldots, n-k, \infty\}$. Zauważ, że te dodatkowe instrukcje nie mają wpływu na obliczenie etykietowania β i służą nam tylko jako pomoc w dowodzie Twierdzenia 15. Rozważmy następującą modyfikację procedury *PodnośEtykietowanie*:

```
procedure PodnośEtykietowanie'
begin
  dla każdego pola p do begin β(p) := 0; α(p) := ∞ end
  while β nie jest dobre dla Bolka do
  begin
    wybierz pole p w którym β nie jest dobre dla Bolka;
    β(p) := Popraw(β, p);
    if β(p) = ∞ then α(p) := Ustal(α, p)
  end
end
```

gdzie

$$Ustal(\beta, p) = \begin{cases} 0 & \text{jeśli pole } p \text{ jest zielone,} \\ 1 + MinMax(\beta, p) & \text{jeśli pole } p \text{ jest białe;} \end{cases}$$

oraz

$$MinMax(\alpha, p) = \begin{cases} \min\{\alpha(r) \;:\; r \text{ jest następnikiem } p\} & \text{jeśli } p \text{ należy do Ani,} \\ \max\{\alpha(r) \;:\; r \text{ jest następnikiem } p\} & \text{jeśli } p \text{ należy do Bolka.} \end{cases}$$

Przykład 19
Przekonaj się, że etykietowanie α obliczane przez procedurę PodnośEtykietowanie' jest równe etykietowaniu α z Przykładu 17.

Ćwiczenie 20 (Dobre etykietowanie dla Ani)
Wykaż, że etykietowanie: $\alpha : P \to \{0, 1, 2, \ldots, n-k, \infty\}$ obliczane przez procedurę PodnośEtykietowanie' jest dobre dla Ani.

Wskazówka. *Wykaż, że gdyby istniało pole p takie, że $\alpha(p) \neq \infty$ i α nie jest dobrym etykietowaniem dla Ani w polu p, to $\beta(p) \neq \infty$, gdzie β jest* najmniejszym *dobrym etykietowaniem dla Bolka.*

Powyższą analizę dobrych etykietowań dla Ani i Bolka można podsumować w następujący sposób.

Twierdzenie 21 (Poprawność algorytmu *PodnośEtykietowanie*)
Niech β będzie etykietowaniem obliczonym przez procedurę PodnośEtykietowanie. Wtedy Ania ma strategię wygrywającą z każdego pola p dla którego zachodzi $\beta(p) = \infty$, a Bolek ma strategię wygrywającą ze wszystkich pozostałych pól.

Ćwiczenie 22 (Wydajna implementacja algorytmu)
Zaimplementuj algorytm PodnośEtykietowanie w taki sposób, aby czas działania twojego programu był $O(k \cdot m)$.

Wskazówka. *Zaprojektuj strukturę danych umożliwiającą ustalanie w czasie $O(1)$, czy istnieje pole p, w którym aktualne etykietowanie nie jest dobre dla Bolka, oraz poprawianie wartości etykietowania w polu p i aktualizację struktury danych w czasie $O(d)$, gdzie d jest sumą liczb, następników pola p oraz jego „poprzedników", tj. pól dla których p jest następnikiem.*

Ciekawy problem 23 *Przyjmijmy, że liczba zielonych pól wynosi $\Omega(n)$. Czy istnieje algorytm rozwiązujący gry w zielone działający w czasie $o(n \cdot m)$? W szczególności, czy istnieje algorytm rozwiązujący gry w zielone działający w czasie $O(m)$?*

Testy

Ocena rozwiązań zadania "Gra w zielone", przedstawionych przez uczestników olimpiady, została oparta na zachowaniu się programów na kolekcji jedenastu testów, tj. jedenastu plików wejściowych zawierających opisy plansz różnych gier w zielone:

- `gra0.in`: Plik z treści zadania.
- `gra1.in`, `gra2.in`: Niewielkie plansze, zaprojektowane głównie z myślą o sprawdzaniu, czy program poprawnie rozwiązuje zadanie, nie wymagające — ze względu na mały rozmiar — aby algorytm był bardzo wydajny.
- `gra3.in`: Niewielka plansza o kilkudziesięciu polach z losowo dobranymi następnikami.
- `gra4.in`*: Plansza o umiarkowanej wielkości, z losowo dobranymi następnikami; mała średnia liczba następników.
- `gra5.in`*: Plansza o umiarkowanej wielkości, z losowo dobranymi następnikami; duża średnia liczba następników.
- `gra6.in`*: Plansza o dużej wielkości z losowo dobranymi następnikami.
- `gra7.in`*: Plansza o umiarkowanej wielkości, w której wszystkie pola przeciwnika o większych numerach niż numer danego pola są następnikami tego pola.
- `gra8.in`*: Plansza o bardzo dużej wielkości, z losowo dobranymi następnikami; mała średnia liczba następników.
- `gra9.in`*: Plansza o bardzo dużej wielkości, z losowo dobranymi następnikami; duża średnia liczba następników.
- `gra10.in`*: Plansza o umiarkowanej wielkości, na której przedstawione rozwiązania działają w czasie równym pesymistycznemu oszacowaniu, tj. $O(k \cdot m)$.

Testy oznaczone gwiazdką zawierają dodatkowe małe fragmenty planszy zaprojektowane z myślą o weryfikacji poprawności rozwiązania.