55. ročník matematické olympiády na středních školách

Kategorie P

In: Karel Horák (editor); Martin Mareš (editor); Peter Novotný (editor); Jaromír Šimša (editor); Jaroslav Švrček (editor); Pavel Töpfer (editor); Jaroslav Zhouf (editor): 55. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané ve školním roce 2005/2006. 47. mezinárodní matematická olympiáda. 18. mezinárodní olympiáda v informatice. (Czech). Praha: Jednota českých matematiků a fyziků, 2007. pp. 97–124.

Persistent URL: http://dml.cz/dmlcz/405111

P – I – 1

Pluky

Na monitoru se právě schyluje k velké bitvě mezi armádou hráče a armádou jeho počítače. Síly jsou vyrovnané, obě armády mají stejný počet pluků, ovšem jednotlivé pluky mohou být tvořeny různým počtem vojáků. Na začátku bitvy se pluky obou armád seřadí do dvou řad tak, že proti každému hráčovu pluku stojí právě jeden pluk patřící počítači. Potom začne vlastní boj. Pluky stojící proti sobě na sebe zaútočí. A protože v množství je síla, zvítězí ten z nich, který má více vojáků. Pokud náhodou mají soupeřící pluky stejný počet vojáků, vyhraje pluk patřící počítači.

Hráčova armáda má schopné špiony, kteří před bitvou zjistili, kolik vojáků má nepřítel v kterém pluku a jak jsou jeho pluky rozmístěny. Vaším úkolem je rozmístit na základě těchto informací hráčovy pluky tak, aby co nejvíce z nich svůj souboj vyhrálo.

Soutěžní úloha. Napište program, který vám poradí, jak nejlépe rozmístit pluky, které máte k dispozici. Na vstupu dostanete počet pluků N v každé armádě a počty vojáků v každém z $2N$ pluků na bojišti. Výstupem programu bude jediné celé číslo — maximální počet pluků hráče, které mohou vyhrát svůj souboj při nějakém rozestavení.

Formát vstupu: První řádek vstupního souboru `pluky.in` obsahuje jedno celé číslo N ($1 \leqq N \leqq 10\,000$) — počet pluků v každé z armád. Na druhém řádku je mezerami odděleno N celých čísel $A_1, \ldots, A_N$ ($1 \leqq \leqq A_i \leqq 100\,000\,000$) — počty vojáků v jednotlivých plucích hráče. Na třetím řádku je mezerami odděleno N celých čísel $B_1, \ldots, B_N$ ($1 \leqq B_i \leqq \leqq 100\,000\,000$) — počty vojáků v jednotlivých plucích patřících počítači.

Formát výstupu: Jediný řádek výstupního souboru `pluky.out` bude obsahovat jedno celé číslo — maximální počet pluků hráče, které mohou najednou vyhrát svůj souboj.

Příklady:

pluky.in	pluky.out
5 7 12 1 7 47 7 12 1 7 47	3 *(Pokud hráč rozmístí své pluky správně, zvítězí jeho pluky velikosti 47, 12 a jeden z pluků velikosti 7.)*

pluky.in	pluky.out
4 10 10 10 10 10 10 10 10	0 *(Při jakémkoliv rozestavení všechny pluky hráče prohrají.)*

pluky.in	pluky.out
5 1 3 5 7 9 2 4 6 8 10	4 *(Hráč obětuje svůj nejmenší pluk, pošle ho proti pluku velikosti 10. Ostatní pluky potom lze rozmístit tak, aby vyhrály.)*

P – I – 2

Teleport

Vědcům se konečně podařilo vymyslet efektivní způsob cestování v časoprostoru. Jejich testovací středisko se skládá z několika lokalit. V každé lokalitě je umístěno několik teleportů. Když vstoupíme do teleportu, přemístí nás na předem zadanou lokalitu (což bychom od teleportu očekávali), ale navíc nás přemístí také v čase o zadaný počet minut (buď dopředu, nebo dozadu). Vědci by chtěli zjistit, jak je cestování pomocí teleportů výhodné. Právě se nacházejí u centrálního počítače a chtěli by se jít nasvačit do bufetu. A protože čas jsou peníze, chtěli by být v bufetu co nejdříve. Pohybovat se v čase a prostoru samozřejmě chtějí jen pomocí již postavených teleportů.

Soutěžní úloha. Program dostane na vstupu počet lokalit N, které budeme označovat čísly $1, \ldots, N$. Centrální počítač je umístěn v lokalitě číslo 1, bufet má číslo N. Následuje celkový počet postavených teleportů M a seznam těchto teleportů. Pro každý teleport je určena počáteční lokalita, koncová lokalita a změna času v minutách, jež nastane při průchodu tímto teleportem (kladné číslo znamená posun do budoucnosti, záporné do minulosti a 0 znamená, že se v koncové lokalitě ocitneme ve stejném čase, v jakém jsme nastoupili do teleportu).

Každý teleport se může použít jen tím směrem, který je uveden na vstupu. Mezi dvěma lokalitami může být vybudováno více teleportů. Dokonce může existovat i teleport, který nás přesune pouze v čase (tedy počáteční a koncová lokalita jsou u něj totožné).

Program má určit čas, kdy nejdříve se můžeme dostat do lokality N, jestliže se v lokalitě 1 nacházíme v čase 0. Pokud tam dokážeme být libovolně brzo (tzn. můžeme pomocí teleportů cestovat neomezeně do minulosti), nebo pokud se tam vůbec nemůžeme dostat, program o tom vydá příslušnou zprávu.

Formát vstupu: První řádek vstupního souboru `teleport.in` obsahuje dvě čísla N a M ($2 \leqq N \leqq 1\,000$, $0 \leqq M \leqq 50\,000$) oddělená mezerou. Následuje M řádků, na každém z nich jsou tři čísla A_i, B_i, T_i ($1 \leqq A_i, B_i \leqq N$, $|T_i| \leqq 10\,000$) popisující teleport z lokality A_i do lokality B_i se změnou času T_i minut.

Formát výstupu: Jediný řádek výstupního souboru `teleport.out` bude obsahovat zprávu „`Vedci umrou hlady`“, jestliže se od centrálního počítače nedá dostat do bufetu, resp. zprávu „`Vedci poznaji vznik vesmiru`“, jestliže můžeme cestovat do nekonečna do minulosti. Jinak bude obsahovat jedno celé číslo představující čas v minutách, kdy nejdříve se vědci dokážou dostat do bufetu.

Příklady:

`teleport.in`
```
3 4
1 2 5
2 3 -7
1 3 -1
1 3 16
```
`teleport.out`
```
-2
```
(Prvním teleportem se vědci dostanou do lokality 2 v čase 5, odtud druhým do lokality 3 v čase $5 + (-7) = -2$. Ostatní možnosti jsou horší.)

`teleport.in`
```
2 2
1 1 -1
1 2 0
```
`teleport.out`
```
Vedci poznaji vznik vesmiru
```
(Dříve než se vědci druhým teleportem přesunou do bufetu, mohou prvním odcestovat libovolně daleko do minulosti.)

`teleport.in`
```
4 3
1 2 -1
2 3 0
4 3 10
```
`teleport.out`
```
Vedci umrou hlady
```
(Poslední teleport nemohou vědci použít na přesun z lokality 3 do lokality 4, jedině naopak.)

P – I – 3

Posádky

Žil jednou jeden starý král. Jeho království tvořilo N měst a mezi nimi vedla sem tam nějaká cesta. Jelikož králové bývají od přírody lakomí, v celém království nebylo zrovna mnoho udržovaných cest. Přesněji řečeno, cest bylo právě $N - 1$ a byly vedeny tak, aby se mezi každými dvěma městy v království dalo po cestách dojet (ať už přímo, nebo přes jiná města). V řeči teorie grafů takovéto síti cest říkáme strom.

Na stará kolena krále navštívila teta Paranoia a našeptala mu, že sousedé chtějí napadnout jeho království. Proto se král rozhodl, že lakota musí jít stranou a že postaví ve městech vojenské posádky. Paranoia však šeptala dál: „Zbláznil ses? Když budou dvě posádky v sousedních městech, budou si mezi sebou posílat zprávy. A víš, jak to dopadne... Nech hodně vojáků pohromadě a vzbouří se proti tobě!“

Tři dny a tři noci král nespal, až vymyslel následující kompromis: Vybere několik měst, v nichž postaví vojenské posádky. Aby mu nehrozila vzpoura, rozhodl se, že nikdy nesmějí být pohromadě více než tři posádky. Teď sedí nad mapou a vymýšlí, jak je má jenom rozmístit, aby království bylo co nejlépe zabezpečeno.

Soutěžní úloha. Ještě jednou si formálněji zopakujme, o co králově vlastně jde.

Na vstupu máte zadán počet měst N a popis cest mezi nimi. Cest je právě $N - 1$, nikde se nekřižují, jimi tvořená síť je souvislá a spojuje všechna města. Pro každé město i známe číslo b_i — toto číslo udává, kolik přidá vojenská posádka v i-tém městě k bezpečnosti království. Králově (a vaším) úkolem je vybrat množinu měst, v nichž budou umístěny posádky. Tato množina musí splňovat následující podmínky:

- Každá její souvislá podmnožina má velikost nejvýše 3. (Množinu měst nazýváme souvislou, jestliže se mezi libovolnými dvěma městy z této množiny dá dojet po cestách, aniž bychom při tom navštívili město, které do této množiny nepatří.)
- Ze všech takovýchto množin má maximální možný součet hodnot b_i — bezpečnost království.

Formát vstupu: První řádek vstupního souboru `posadky.in` obsahuje jedno číslo N ($1 \leqq N \leqq 100\,000$) — počet měst v království. Města jsou očíslována od 1 do N. Každý z následujících $N - 1$ řádků obsahuje dvě čísla měst, která jsou spojena cestou. Můžete předpokládat, že síť cest je souvislá.

Poslední řádek vstupního souboru obsahuje N celých čísel $b_1, \ldots, b_N$ ($0 \leqq b_i \leqq 10\,000$), která udávají, kde je jak výhodné umístit vojenskou posádku.

Formát výstupu: První řádek výstupního souboru `posadky.out` bude obsahovat jedno celé číslo — nejlepší dosažitelnou bezpečnost království. Druhý řádek bude obsahovat několik čísel oddělených mezerami — jednu vhodnou množinu měst, pro kterou se uvedené bezpečnosti dosáhne.

Příklady:

```
posadky.in          posadky.out
7                   6
1 2                 1 2 3 5 6 7
2 3                 (Všude je zisk z posádky stejný, chceme jich umís-
3 4                     tit co nejvíce.)
4 5
5 6
6 7
1 1 1 1 1 1 1
```

```
posadky.in          posadky.out
5                   1011
1 5                 2 3 5
2 5                 (Zjevně chceme mít posádku ve městě 5. Potom už
3 5                     ale můžeme vybrat jen dvě z ostatních měst.)
4 5
1 6 5 2 1000
```

```
posadky.in          posadky.out
5                   16
1 5                 1 2 3 4
2 5                 (Ne vždy se vyplatí vybrat město s nejvyšší hod-
3 5                     notou $b_i$.)
4 5
4 4 4 4 5
```

P – I – 4

Paralelizátor

Za sedmero horami a sedmero řekami vymyslel vynálezce Kleofáš podivný stroj, který nazval *paralelizátor*. Na první pohled vypadal paralelizátor jako obyčejný počítač... Byl tu však jeden malý, ale o to důležitější

rozdíl. Za určitých okolností dokázal paralelizátor paralelně (tj. současně) spustit více větví programu, aniž by ho to jakkoliv zpomalilo. Kleofáš rychle pochopil, že jen ze slovního popisu tohoto zázraku by nikdo nebyl moc moudrý, a tak vymyslel i programovací jazyk, v němž je možné psát programy pro jeho paralelizátor.

Programy pro paralelizátor se budou od klasických lišit mimo jiné tím, že nebudou mít žádný výstup. Budeme pouze rozlišovat, zda program skončil *úspěšně* nebo *neúspěšně*. U klasických programů by to znamenalo, že nás zajímá jen tzv. exit code (návratová hodnota) programu.

Kleofášův programovací jazyk je téměř přesnou kopií jazyka Pascal. Oproti klasickému Pascalu v něm nemáme k dispozici generátor náhodných čísel (a tedy například funkci `random`), takže je předem dáno, jak bude výpočet každého programu vypadat. Zato přibyly čtyři nové příkazy: **Accept**, **Reject**, **Both**(x) a **Some**(x) (kde x je proměnná typu integer).

Příkaz **Accept** *úspěšně* ukončí běžící program.

Příkaz **Reject** ukončí běžící program, ale *neúspěšně*. Stejný význam má i provedení standardního Pascalského příkazu **Halt** a ukončení výpočtu programu přechodem přes koncové **End.**, příkaz **Reject** definujeme jen kvůli názornosti.

V následujícím textu budeme *vytvořením kopie programu* rozumět to, že se v operační paměti vytvoří úplně přesná kopie celého programu včetně obsahu jeho proměnných — výsledek bude stejný, jako kdybychom už od začátku daný program spustili ne jednou, ale dvakrát.

Příkaz **Both**(x) zastaví aktuálně běžící program. Vytvoří se dvě jeho identické kopie. V první z nich je hodnota proměnné x nastavena na 0, v druhé na 1. Obě kopie programu jsou paralelně spuštěny, přičemž jejich výpočet pokračuje příkazem následujícím za příslušným příkazem **Both**.

Pokud obě kopie úspěšně skončí, v následujícím taktu procesoru úspěšně skončí i původní program. Jestliže jedna z kopií skončí neúspěšně (druhá přitom skončit ani nemusí), původní program v následujícím taktu skončí také neúspěšně. Ve všech ostatních případech (tj. když jedna kopie nikdy neskončí a druhá buď rovněž nikdy neskončí, nebo skončí úspěšně) původní program nikdy neskončí.

Příkaz **Some**(x) funguje podobně. Rovněž zastaví aktuálně běžící program. Opět se vytvoří dvě jeho identické kopie, v první z nich je hodnota proměnné x nastavena na 0, v druhé na 1. Obě kopie programu jsou paralelně spuštěny, přičemž jejich výpočet pokračuje příkazem následujícím za příslušným příkazem **Some**.

Jakmile některá z kopií úspěšně skončí, v následujícím taktu procesoru úspěšně skončí i původní program. Pokud obě kopie skončí neúspěšně, v následujícím taktu procesoru skončí neúspěšně také původní program. Ve všech ostatních případech (tj. když jedna kopie nikdy neskončí a druhá buď rovněž nikdy neskončí, nebo skončí neúspěšně) původní program nikdy neskončí.

Slovně můžeme tyto operace popsat následovně: Příkaz **Both** provádí „paralelní and" — ověří, zda obě větve úspěšně skončí. Příkaz **Some** provádí „paralelní or" — ověří, zda aspoň jedna z větví úspěšně skončí.

Netrvalo dlouho a Kleofáš si uvědomil, že na takovémto zázračném zařízení dokáže některé problémy řešit až neuvěřitelně rychle. Například testování prvočíselnosti je skutečně snadné.

Příklad 1: V proměnné N je přirozené číslo. Napište program pro paralelizátor, který pro každou hodnotu N skončí, přičemž *úspěšně* skončí právě tehdy, když N je prvočíslo.

ŘEŠENÍ. Pomocí volání příkazu **Both** paralelně vygenerujeme všechna čísla od 2 do $N-1$ a najednou pro každé z nich ověříme, zda dělí N. Každá větev výpočtu úspěšně skončí, jestliže „její" číslo nedělí N. Aby původní program úspěšně skončil, musí úspěšně skončit všechny větve, tedy žádné z vygenerovaných čísel nesmí dělit N. Časová složitost programu je $O(\log N)$.

```
{ VSTUP:  N : integer; }

var moc2, pocet_cifer : integer;
    cislo : integer;
    i,x : integer;

begin
   { oetme okrajov ppad }
   if N = 1 then Reject;

   { zjistme, kolik m N cifer ve dvojkov soustav }
   moc2 := 1;
   pocet_cifer := 0;
   while moc2 < N do begin
      moc2 := moc2 * 2;
      inc(pocet_cifer);
   end;

   { vygenerujeme sla od 0 do 2^pocet_cifer - 1 }
   cislo := 0;
   for i:=1 to pocet_cifer do begin
      Both(x);
      cislo := 2*cislo + x;
   end;
```

```
  { moc mal dlitele zkouet nebudeme, prohlsme za dobr }
  if cislo <= 1 then Accept;
  { ani pli velk dlitele zkouet nebudeme }
  if cislo >= N then Accept;
  { jinak zkoume, zda vygenerovan slo dl N }
  if N mod cislo <> 0 then Accept;
  Reject;
end.
```

Názorně si ukážeme, jak vypadá výpočet paralelizátoru na tomto programu pro $N = 3$ a pro $N = 6$. Kopie programu, které vznikají během výpočtu, budeme číslovat v pořadí, v jakém vznikají.

Pro $N = 3$ bude výpočet probíhat následovně:

- ▷ Spustí se kopie #1 (tedy vlastně originál).
- ▷ Spočítá, že *pocet_cifer* = 2.
- ▷ Spustí se for-cyklus pro $i = 1$.
- ▷ Kopie #1 se zastaví, vzniknou kopie #2 a #3.
- ▷ V kopii #2 je *cislo* = 0, v kopii #3 je *cislo* = 1.
- ▷ V obou běžících kopiích pokračuje for-cyklus pro $i = 2$.
- ▷ Kopie #2 a #3 se zastaví, z #2 vzniknou #4 a #5, z #3 vzniknou #6 a #7.
- ▷ V kopiích #4 až #7 bude mít proměnná *cislo* hodnoty 0 až 3.
- ▷ Kopie #4 a #5 úspěšně skončí, neboť čísla 0 a 1 nechceme testovat jako dělitele.
- ▷ Kopie #2 úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.
- ▷ Kopie #7 úspěšně skončí, neboť ani číslo 3 nechceme testovat.
- ▷ Kopie #6 úspěšně skončí, neboť 2 nedělí 3.
- ▷ Kopie #3 úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.
- ▷ Kopie #1 (tedy původní program) úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.

Pro $N = 6$ bude výpočet probíhat následovně:

- ▷ Podobně jako při $N = 3$ se dostaneme do situace, kdy běží kopie #8 až #15, proměnná *cislo* v nich má hodnoty postupně od 0 do 7.
- ▷ Kopie #8 a #9 (s příliš malým číslem) úspěšně skončí.
- ▷ Kopie #4 (z níž vznikly #8 a #9) úspěšně skončí.
- ▷ Kopie #14 a #15 (s příliš velkým číslem) úspěšně skončí.
- ▷ Kopie #7 (z níž vznikly #14 a #15) úspěšně skončí.
- ▷ Kopie #10 až #13 skončí — a to: #12 a #13 úspěšně (4 ani 5 nedělí 6), #10 a #11 neúspěšně (2 a 3 dělí 6).

- ▷ Kopie #5 skončí neúspěšně (obě její „děti“ skončily neúspěšně), kopie #6 skončí úspěšně.
- ▷ Kopie #2 skončí neúspěšně (neboť kopie #5 skončila neúspěšně), kopie #3 skončí úspěšně.
- ▷ Kopie #1 (tedy původní program) skončí neúspěšně.

Příklad 2: V proměnných N a K jsou přirozená čísla. Napište program pro paralelizátor, který pro každé N skončí, přičemž *úspěšně* skončí právě tehdy, když N má nějakého dělitele z množiny $M = \{2, 3, \ldots, 2^K - 1\}$.

Řešení. Pomocí volání příkazu **Some** paralelně projdeme všechna čísla $m \in M$, stačí nám, když libovolné jedno z nich dělí N.

(Jiný pohled na totéž řešení: Pomocí volání příkazu **Some** „uhodneme“ dělitele $m \in M$ a ověříme, zda jsme ho uhodli správně. Na náš program se můžeme dívat tak, že se nevětví, ale každé volání **Some** „uhodne“ a do x dosadí „správnou“ hodnotu. Jestliže tedy N má v množině M dělitele, najdeme ho, jinak skončíme s nějakým číslem, které N nedělí.)

Časová složitost programu je $O(K)$.

```
{ VSTUP:  N, K : integer; }

var cislo : integer;
    i,x : integer;

begin
   { paraleln zkoume sla od 0 do 2^K - 1 }
   cislo := 0;
   for i:=1 to K do begin
      Some(x);
      cislo := 2*cislo + x;
   end;

   { 0 a 1 do mnoiny M nepat }
   if cislo <= 1 then Reject;
   { zkusme, zda vygenerovan slo dl N }
   if N mod cislo = 0 then Accept;
   Reject;
end.
```

Soutěžní úloha. a) V proměnných *jehla* a *seno* jsou dva znakové řetězce. Napište co nejrychlejší program pro paralelizátor, který pro každý vstup skončí, přičemž *úspěšně* skončí právě tehdy, když se řetězec *jehla* nachází v řetězci *seno* jako souvislý podřetězec. Váš program by tedy měl úspěšně skončit, jestliže například:

$$\textit{jehla} = \texttt{abcd}, \quad \textit{seno} = \texttt{aaabcdddaa},$$
$$\textit{jehla} = \texttt{ddda}, \quad \textit{seno} = \texttt{aaabcdddaa},$$

ale ne v případech:

$$\textit{jehla} = \texttt{abcd}, \quad \textit{seno} = \texttt{aaabcEdddaa},$$
$$\textit{jehla} = \texttt{jajsemjehla}, \quad \textit{seno} = \texttt{vtetokupesenajehlaneni}.$$

b) Nad polem přirozených čísel můžeme postavit „pyramidu". Spodní řádek pyramidy bude tvořit samotné pole. Každý vyšší řádek bude o 1 kratší než předcházející, přičemž i-tý prvek v novém řádku je roven součtu i-tého a $(i+1)$-ního prvku z řádku pod ním, modulo 10 000 (tzn. pokud by součet vyšel větší než 9 999, necháme z něho v pyramidě jen jeho poslední čtyři cifry). Vrchní řádek pyramidy je tvořen jediným číslem.

V proměnné N máme přirozené číslo. V poli A na pozicích 1 až N máme N přirozených čísel menších než 10 000. V proměnné V je nezáporné celé číslo menší než 10 000.

Napište co nejrychlejší program pro paralelizátor, který pro každý vstup skončí, přičemž *úspěšně* skončí právě tehdy, když hodnota V je na vrcholu pyramidy postavené nad polem A.

Příklad:

Vstup:	Výstup:	*(Pyramida vypadá následovně:)*
$N = 4$	skončí neúspěšně	45
$A = (6, 3, 9, 3)$		21 24
$V = 17$		9 12 12
		6 3 9 3
Vstup:	Výstup:	*(Pyramida vypadá následovně:)*
$N = 4$	skončí úspěšně	20
$A = (1, 2, 3, 4)$		8 12
$V = 20$		3 5 7
		1 2 3 4

P – II – 1

Fotbal

V Absurdistánu právě začíná nový ročník fotbalové soutěže. Tento rok se ho zúčastní také slavný tým Dynamo Zbicyklu. Jeho trenér už tři noci pořádně nespal, ale stále ještě nemá připraven plán na tuto sezónu.

Dobře ví, že žádné mužstvo nedokáže vyhrát všechny zápasy, neboť každá výhra stojí hráče mnoho sil. Vymyslel si proto následující zjednodušení:

Aktuální stav jeho mužstva bude popisovat jedno celé číslo S, které udává, kolik mají hráči síly. Na začátku sezóny (v den číslo 0) je síla

mužstva nulová. Každou noc si hráči odpočinou, a proto se jejich síla zvýší o 1. Pokud chtějí nějaký zápas vyhrát, musí se hodně snažit — každá výhra je stojí V síly. Mohou samozřejmě také „hrát na remízu," což je stojí jenom R síly, případně mohou zápas úplně vypustit a prohrát ho, což je nestojí nic. (Jestliže chtějí nějaký zápas vyhrát nebo remizovat, musí na to mít dost sil, síla týmu nemůže nikdy klesnout pod nulu.)

Trenér už zná přesný rozpis ligy, ví tedy, ve kterých dnech má jeho mužstvo volno a kdy hraje nějaký zápas. Napište program, který vypočítá, kolik nejvýše bodů může jeho mužstvo v tomto ročníku ligy získat: za každou výhru jsou tři body a za remízu jeden.

Formát vstupu: Na vstupu jsou zadána celá čísla V, R (vysvětlená výše) a počet zápasů N. Následuje N celých čísel — čísla dní, v nichž hraje naše fotbalové mužstvo zápas.

Můžete předpokládat, že $N \leqq 10\,000$ a $V > R$. Čísla V, R i všechna čísla dní se vejdou do běžné 32bitové celočíselné proměnné. Čísla dní jednotlivých zápasů jsou uvedena v rostoucím pořadí.

Formát výstupu: Program vypíše jediné celé číslo — maximální počet bodů, které může naše mužstvo v soutěži získat.

Příklady:

vstup:
$V = 10$, $R = 3$, $N = 2$
dni zápasů:
3, 13

výstup:
4
(Hráči stihnou nabrat přesně tolik sil, aby dokázali první zápas remizovat a druhý vyhrát.)

vstup:
$V = 20$, $R = 15$, $N = 4$
dni zápasů:
23, 24, 25, 26

výstup:
3
(Mužstvo dokáže vyhrát libovolný jeden z těchto čtyř zápasů.)

vstup:
$V = 30$, $R = 9$, $N = 4$
dni zápasů:
30, 32, 34, 36

výstup:
4
(Ne vždy se vyplatí vyhrát, v tomto případě je výhodnější všechny 4 zápasy remizovat.)

P – II – 2

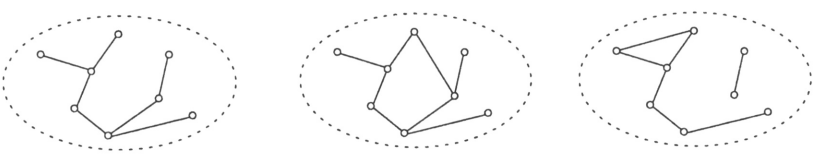
Housenka

Strom je objekt, který má následující vlastnosti:

- ▷ Obsahuje konečný počet význačných míst, kterým říkáme *vrcholy* (jejich počet označíme N). Některé dvojice vrcholů jsou spojeny *hranami*.
- ▷ Je souvislý, tzn. z libovolného vrcholu se můžeme dostat do libovolného jiného vrcholu tak, že postupně projdeme po několika hranách.
- ▷ Obsahuje právě $N - 1$ hran.

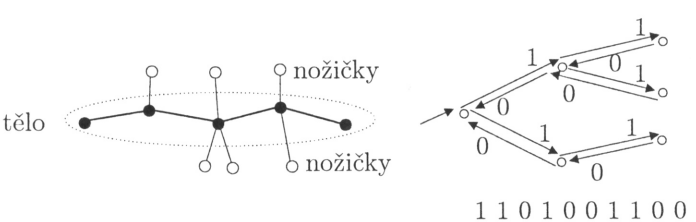
Strom si můžeme představit například jako souvislou silniční síť, kterou tvoří N měst a právě $N - 1$ silnic mezi nimi.

Na následujícím obrázku je levý graf strom, zatímco zbývající dva grafy nikoliv — druhý obsahuje příliš mnoho hran a třetí má sice správný počet hran, ale není souvislý.



Posloupnost na sebe navazujících hran, v níž se žádná hrana neopakuje, se nazývá *cesta*. Všimněte si, že ve stromě vede mezi každými dvěma vrcholy právě jedna cesta.

Housenka je strom, ve kterém existuje taková cesta (tuto cestu pak nazýváme *tělo* housenky), že každý vrchol stromu je buď na této cestě, nebo sousedí s nějakým vrcholem této cesty (pak ho nazýváme *nožička*). Příklad housenky vidíte na následujícím obrázku vlevo.



Existuje několik způsobů, jak je možné zadat strom. My ho popíšeme posloupností nul a jedniček: Zvolíme si jeden libovolný vrchol jako výchozí a začneme se z něj po stromě procházet, přičemž chceme navštívit každý vrchol stromu a chceme projít po každé jeho hraně právě dvakrát (v každém směru jednou). Během této procházky si budeme zapisovat nuly a jedničky následovně: Vždy, když přijdeme do vrcholu, ve kterém jsme ještě nebyli, napíšeme jedničku. Vždy, když se z vrcholu vracíme zpět (po hraně, kterou jsme do něj přišli), napíšeme nulu. Rozmyslete

si, že takto dokážeme (aspoň jedním způsobem) popsat libovolný strom a naopak že z tohoto popisu můžeme strom jednoznačně sestrojit. (Viz předchozí obrázek vpravo.)

Soutěžní úloha. Na základě zadané posloupnosti nul a jedniček sestrojte strom a zjistěte, kolik nejméně vrcholů je z něho třeba odstranit, abychom dostali housenku.

Jinými slovy řečeno, určete v zadaném stromě takovou cestu, pro níž je množina vrcholů, které na ní neleží ani s ní nesousedí, nejmenší možná.

Formát vstupu: Na vstupu je zadána posloupnost nul a jedniček reprezentující strom tak, jak je popsáno výše.

Formát výstupu: Program vypíše jediné celé číslo — minimální počet vrcholů, které je třeba odstranit z původního stromu, abychom dostali housenku.

Příklad:

vstup:

`110100110100110100`

výstup:

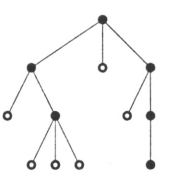
2 *(Je třeba odstranit 2 vrcholy.)*

vstup:

`11011010100010110110000`

výstup:

0 *(Zadaný strom už je housenka.)*



Poznámka. V obou uvedených příkladech vstupu procházku začínáme v „horním“ vrcholu stromu a ostatní vrcholy navštěvujeme v pořadí „zleva doprava.“

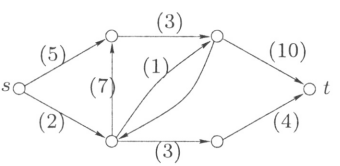
P – II – 3

Myška

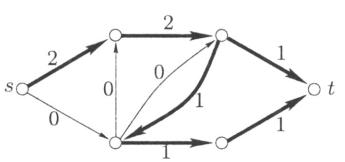
Představte si síť navzájem pospojovaných potrubí. Místa, kde se spojují konce a začátky potrubí, budeme nazývat uzly. Z každého uzlu může vést libovolné množství potrubí, podobně do každého uzlu může libovolně množství potrubí přicházet. Každé potrubí je upraveno tak, že jím voda může téci jen jedním směrem. Jednotlivá potrubí mohou mít různý průřez, takže jimi může protékat za sekundu různé množství vody. (Maximální množství vody, které může protéci potrubím za sekundu, nazýváme *kapacitou* tohoto potrubí.)

Dva uzly v uvažované síti budou mít speciální význam. Jeden z nich nazýváme zdroj (a značíme ho s), druhý nazýváme ústí (a označujeme ho t). *Zdroj* je jediné místo, kde do naší soustavy potrubí může přitékat voda, *ústí* je jediné místo, kde naopak voda může odtékat. Pro jednoduchost budeme předpokládat, že do zdroje ani z ústí žádná potrubí nevedou.

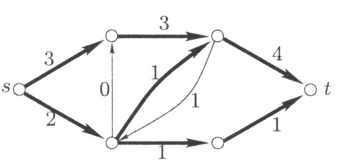
Nyní si představte, že takovouto sítí potrubí necháme protékat vodu a pro každé potrubí si zapíšeme množství vody, které jím za sekundu proteče. Tomuto seznamu čísel říkáme *tok*. *Velikost* tohoto toku je množství vody, které za sekundu vyteče ven ústím (nebo ekvivalentně, které za sekundu přiteče ze zdroje).



a) Příklad sítě potrubí



b) Příklad toku velikosti 2



c) Příklad maximálního toku

Obr. 32

Na obr. 32a) je příklad sítě potrubí, čísla v závorkách představují kapacity jednotlivých potrubí. Na obr. 32b) je vyznačen jeden možný tok pro naši ukázkovou síť potrubí. Silnou čarou jsou znázorněna potrubí, kterými teče nějaká voda, čísla u jednotlivých potrubí udávají množství vody, které za sekundu daným potrubím proteče.

Maximální tok je takový tok, který má pro danou síť potrubí největší možnou velikost. Jinými slovy řečeno, maximální tok popisuje, jakým způsobem lze naší sítí potrubí „protlačit" co největší množství vody za jednu sekundu. Na obr. 32c) je znázorněn maximální tok v naší ukázkové síti.

Soutěžní úloha. Představte si, že máte k dispozici krabičku, které zadáte popis nějaké sítě potrubí (včetně kapacit jednotlivých potrubí) a ona vám určí hodnotu maximálního toku. S její pomocí vyřešte následující problém:

Na vstupu máte zadáno bludiště — neorientovaný graf s N místnostmi (vrcholy grafu) a M chodbami (hrany grafu) mezi nimi. V místnosti 1 se nachází myška. V místnosti N je umístěn sýr. Myška unese najednou nejvýše 1 kousek sýra a chce přemístit co nejvíce sýra z místnosti N do místnosti 1. Nechce ale nikdy vstoupit podruhé do téže místnosti (samozřejmě kromě místností 1 a N), neboť nechce riskovat, že si tam na ni počká kocour, který ji tam po prvním průchodu mohl ucítit. Kolik kousků sýra dokáže myška nejvýše přenést? (Předpokládejte, že místnosti 1 a N nejsou spojeny přímou chodbou, v takovém případě by samozřejmě myška mohla postupně přenosit všechen sýr.)

Řešením této úlohy je tedy program, v němž můžete volat funkci *NajdiMaximalniTok*(...), které zadáte jako parametry popis nějaké sítě potrubí (počet uzlů, počet potrubí, pro každé potrubí jeho začátek, konec a kapacitu, a dále informaci, který uzel je zdrojem a který je ústím) a ona vám vrátí hodnotu maximálního toku v zadané síti. Tuto funkci nemusíte implementovat, přesný formát parametrů si zvolte tak, jak vám bude nejlépe vyhovovat.

Příklad:

vstup:	*výstup*:
$N = 8$, $M = 10$	1
1–2, 1–5, 1–7,	*(Myška může jít po cestě* 1–2–3–8
2–3, 2–4, 3–8,	*pro sýr,* 8–7–1 *zpět. Mohla by*
4–8, 5–6, 6–8,	*ještě jít cestou* 1–5–6–8 *pro sýr,*
7–8	*ale zpět by se už nedostala.)*

Toky v grafech (studijní text). V této části zadání uvádíme formálnější definice výše uvedených pojmů. Pokud je v zadání úlohy všechno jasné, tento text číst nemusíte, použijte ho jen v případě nejasností v neformálním popisu.

Začneme několika definicemi: *Graf* je uspořádaná dvojice (V, E), kde V je konečná množina *vrcholů* grafu a E je konečná množina jeho *hran*. Počet vrcholů označme N a počet hran M, hrany označme e_1 až e_M. Každá hrana e_i spojuje právě dva vrcholy grafu a_i a b_i. Pokud se bude smět procházet po hraně e_i jenom jedním směrem (tj. smíme po ní jít z vrcholu a_i do b_i, ale nikoliv opačně), budeme ji nazývat *orientovaná* hrana, jinak ji budeme nazývat *neorientovaná* hrana. Graf nazveme orientovaný, resp. neorientovaný, jestliže jsou všechny jeho hrany orientované, resp. neorientované.

V grafu budeme mít dva speciální vrcholy. Jeden z nich nazveme

zdroj (značíme s) a druhý *ústí* (značíme t). Dále budeme předpokládat, že každá hrana má stanovenu svoji *kapacitu* $c_i \geq 0$. V orientovaném grafu hrana z a_i do b_i může mít jinou kapacitu než hrana z b_i do a_i, resp. některá z nich ani nemusí existovat.

Funkci f, která přiřazuje každé hraně množství vody, které touto hranou protéká, nazveme *tokem,* jestliže splňuje následující podmínky:

- $f(e_i) \in \mathbb{Z}$ pro $1 \leq i \leq M$. Tedy každou hranou může téci jen celočíselné množství vody.
- $0 \leq f(e_i) \leq c_i$ pro $1 \leq i \leq M$. Tedy žádnou hranou nemůže téci víc vody, než kolik je její kapacita, ani méně než 0.
- Nechť a je vrchol různý od zdroje a ústí. Nechť hrany vedoucí z vrcholu a mají čísla $v_1, \ldots, v_k$. Podobně, nechť hrany vedoucí do vrcholu a mají čísla $p_1, \ldots, p_l$. Potom platí: $\sum_i f(e_{v_i}) = \sum_i f(e_{p_i})$. Tedy ve „vnitřních" vrcholech se voda nemůže hromadit.

Tok je tedy taková funkce f, která nám určuje, kolik vody teče kterým potrubím. Předpokládejme, že do zdroje nevstupují žádné hrany a z ústí nevystupují žádné hrany. Potom můžeme *hodnotu* toku f definovat jako množství vody, které odtéká ze zdroje. *Maximální tok* je tok s největší možnou hodnotou pro daný graf.

P – II – 4

Paralelizátor

V zemi je několik měst a každé z nich je označeno nějakým přirozeným číslem (které se vejde do běžné celočíselné proměnné). Různým městům jsou přiřazena různá čísla, ale jinak není číslování měst nijak systematické.

Mezi některými dvojicemi měst jsou vybudovány cesty, těchto cest je celkem M. Všechny cesty jsou jednosměrné. Všechny křižovatky cest jsou mimoúrovňové, tzn. pokud se vydáme po nějaké cestě, musíme po ní dojít až do toho města, kde tato cesta končí. Můžete předpokládat, že v každém městě aspoň jedna cesta začíná nebo končí.

V poli $C[0..M-1][0..1]$ jsou popsány jednotlivé cesty (i-tá cesta spojuje města s čísly $C[i-1][0]$ a $C[i-1][1]$).

Soutěžní úloha. Napište co nejrychlejší program pro paralelizátor, který pro každý přípustný vstup skončí, přičemž úspěšně skončí právě tehdy, když je síť všech existujících cest silně souvislá — tedy pokud

se z libovolného města můžeme po cestách dostat do libovolného jiného města.

Poznámka. Úlohu lze řešit v lepším čase než lineárním. Pokud se vám však takové řešení nepodaří nalézt, část bodů získáte i za pomalejší řešení.

Příklad 1:

vstup:
$M = 5$
cesty:
$1 \to 2$, $2 \to 3$, $3 \to 1$,
$47 \to 1$, $2 \to 47$

výstup:
skončí úspěšně
(Po prvních třech cestách můžeme volně přecházet mezi městy 1, 2 *a* 3,
díky zbývajícím dvěma se dokážeme dostat i do města 47 *a z něho zase pryč.)*

Příklad 2:

vstup:
$M = 3$
cesty:
$123456 \to 234567$, $23 \to 47$,
$345678 \to 234567$

výstup:
skončí neúspěšně
(Nedokážeme se dostat například z města 47 *do města* 123456.*)*

(Definice paralelizátoru je uvedena v textu úlohy P–I–4.)

Netrvalo dlouho a Kleofáš si uvědomil, že na takovémto zázračném zařízení dokáže některé problémy řešit až neuvěřitelně rychle. Například testování prvočíselnosti je skutečně snadné.

Příklad 1: V proměnné N je přirozené číslo. Napište program pro paralelizátor, který pro každou hodnotu N skončí, přičemž *úspěšně* skončí právě tehdy, když N je prvočíslo.

Řešení. Pomocí volání příkazu **Both** paralelně vygenerujeme všechna čísla od 2 do $N - 1$ a najednou pro každé z nich ověříme, zda dělí N. Každá větev výpočtu úspěšně skončí, jestliže „její" číslo nedělí N. Aby původní program úspěšně skončil, musí úspěšně skončit všechny větve, tedy žádné z vygenerovaných čísel nesmí dělit N. Časová složitost programu je $O(\log N)$.

```
{ VSTUP:  N : integer; }

var moc2, pocet_cifer : integer;
    cislo : integer;
    i,x : integer;

begin
```

```
   { oetme okrajov ppad }
   if N = 1 then Reject;

   { zjistme, kolik m N cifer ve dvojkov soustav }
   moc2 := 1;
   pocet_cifer := 0;
   while moc2 < N do begin
      moc2 := moc2 * 2;
      inc(pocet_cifer);
   end;

   { vygenerujeme sla od 0 do 2^pocet_cifer - 1 }
   cislo := 0;
   for i:=1 to pocet_cifer do begin
      Both(x);
      cislo := 2*cislo + x;
   end;

   { moc mal dlitele zkouet nebudeme, prohlsme za dobr }
   if cislo <= 1 then Accept;
   { ani pli velk dlitele zkouet nebudeme }
   if cislo >= N then Accept;
   { jinak zkoume, zda vygenerovan slo dl N }
   if N mod cislo <> 0 then Accept;
   Reject;
end.
```

Názorně si ukážeme, jak vypadá výpočet paralelizátoru na tomto programu pro $N = 3$ a pro $N = 6$. Kopie programu, které vznikají během výpočtu, budeme číslovat v pořadí, v jakém vznikají.

Pro $N = 3$ bude výpočet probíhat následovně:

- ▷ Spustí se kopie #1 (tedy vlastně originál).
- ▷ Spočítá, že *pocet_cifer* = 2.
- ▷ Spustí se for-cyklus pro i = 1.
- ▷ Kopie #1 se zastaví, vzniknou kopie #2 a #3.
- ▷ V kopii #2 je *cislo* = 0, v kopii #3 je *cislo* = 1.
- ▷ V obou běžících kopiích pokračuje for-cyklus pro i = 2.
- ▷ Kopie #2 a #3 se zastaví, z #2 vzniknou #4 a #5, z #3 vzniknou #6 a #7.
- ▷ V kopiích #4 až #7 bude mít proměnná *cislo* hodnoty 0 až 3.
- ▷ Kopie #4 a #5 úspěšně skončí, neboť čísla 0 a 1 nechceme testovat jako dělitele.
- ▷ Kopie #2 úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.
- ▷ Kopie #7 úspěšně skončí, neboť ani číslo 3 nechceme testovat.
- ▷ Kopie #6 úspěšně skončí, neboť 2 nedělí 3.

▷ Kopie #3 úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.
▷ Kopie #1 (tedy původní program) úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.

Pro $N = 6$ bude výpočet probíhat následovně:

▷ Podobně jako při $N = 3$ se dostaneme do situace, kdy běží kopie #8 až #15, proměnná *cislo* v nich má hodnoty postupně od 0 do 7.
▷ Kopie #8 a #9 (s příliš malým číslem) úspěšně skončí.
▷ Kopie #4 (z níž vznikly #8 a #9) úspěšně skončí.
▷ Kopie #14 a #15 (s příliš velkým číslem) úspěšně skončí.
▷ Kopie #7 (z níž vznikly #14 a #15) úspěšně skončí.
▷ Kopie #10 až #13 skončí — a to: #12 a #13 úspěšně (4 ani 5 nedělí 6), #10 a #11 neúspěšně (2 a 3 dělí 6).
▷ Kopie #5 skončí neúspěšně (obě její „děti" skončily neúspěšně), kopie #6 skončí úspěšně.
▷ Kopie #2 skončí neúspěšně (neboť kopie #5 skončila neúspěšně), kopie #3 skončí úspěšně.
▷ Kopie #1 (tedy původní program) skončí neúspěšně.

Příklad 2: V proměnných N a K jsou přirozená čísla. Napište program pro paralelizátor, který pro každé N skončí, přičemž *úspěšně* skončí právě tehdy, když N má nějakého dělitele z množiny $M = \{2, 3, \ldots, 2^K - 1\}$.

ŘEŠENÍ. Pomocí volání příkazu **Some** paralelně projdeme všechna čísla $m \in M$, stačí nám, když libovolné jedno z nich dělí N.

(Jiný pohled na totéž řešení: Pomocí volání příkazu **Some** „uhodneme" dělitele $m \in M$ a ověříme, zda jsme ho uhodli správně. Na náš program se můžeme dívat tak, že se nevětví, ale každé volání **Some** „uhodne" a do x dosadí „správnou" hodnotu. Jestliže tedy N má v množině M dělitele, najdeme ho, jinak skončíme s nějakým číslem, které N nedělí.)

Časová složitost programu je $O(K)$.

```
{ VSTUP:  N, K : integer; }

var cislo : integer;
    i,x : integer;

begin
   { paraleln zkoume sla od 0 do 2^K - 1 }
   cislo := 0;
   for i:=1 to K do begin
      Some(x);
      cislo := 2*cislo + x;
   end;
```

```
    { 0 a 1 do mnoiny M nepat }
    if cislo <= 1 then Reject;
    { zkusme, zda vygenerovan slo dl N }
    if N mod cislo = 0 then Accept;
    Reject;
end.
```

P – III – 1

Příšery

Na monitoru se zase jednou schyluje k velké bitvě mezi armádou hráče a armádou jeho počítače.

Každou armádu tvoří N příšer. Každou příšeru můžeme popsat dvěma přirozenými čísly: první popisuje její *útok* (útočnou sílu), druhé pak její *obranu* (obranné schopnosti). Příšeru s útokem a a obranou b budeme značit a/b.

Když spolu bojují dvě příšery a útok první je větší než obrana druhé, druhá příšera je zabita. Může se také stát, že se obě příšery zabijí navzájem nebo že obě přežijí. Příšera *vyhraje souboj,* pokud zabije druhou příšeru a sama přežije.

Příšery ovládané počítačem útočí, hráč se musí bránit. Proti každé z příšer počítače musí poslat právě jednu ze svých příšer. Všechny souboje probíhají současně.

Úloha. Napište program, který zjistí, kolik mohou hráčovy příšery maximálně vyhrát soubojů.

Vstup: Na prvním řádku vstupu je celé číslo N ($1 \leq N \leq 10\,000$) — počet příšer, které má každý z hráčů k dispozici. Na druhém řádku jsou uvedeny hráčovy příšery, na třetím pak příšery počítače. Útok i obrana každé příšery je celé číslo od 1 do 1 000 000 000.

Výstup: Vypište jediné celé číslo — největší počet hráčových příšer, které mohou najednou vyhrát své souboje.

Příklady:

vstup	*výstup*
3 9/2, 9/7, 8/8 100/100, 1/1, 7/8	2 *(Nad příšerou 7/8 vyhraje jedině příšera 9/7. Příšeře 1/1 může hráč přiřadit kteroukoliv ze svých zbylých dvou příšer.)*

vstup
4
10/1, 10/1, 10/2, 10/9
10/1, 10/1, 10/2, 10/8

výstup
0
(Bez ohledu na rozdělení se všechny příšery zabijí navzájem.)

vstup
4
7/3, 2/12, 47/47, 5/6
10/1, 4/7, 3/6, 1/1

výstup
4
(Jediné řešení: své příšery hráč přiřadí postupně třetí, první, druhé a čtvrté příšeře počítače.)

P – III – 2

Bürroland

V Bürrolandu vyřešili otázku nezaměstnanosti po svém — zaměstnali hromadu úředníků. Aby měli noví úředníci co na práci, začali vydávat nejrůznější potvrzení, která je potřeba předkládat při různých příležitostech. A jelikož úředníků je mnoho, každý z nich je úzce specializovaný a vydává pouze jeden typ potvrzení. Stejný typ potvrzení ovšem může vydávat více úředníků.

Dostat od úředníka potvrzení, které vydává, není nikterak lehké. Abyste ho dostali, musíte mít potvrzení jiného konkrétního typu a k tomu ještě musíte úředníkovi předložit několik svých osobních dokladů. (Nezáleží na tom, jakých, důležitý je pouze jejich počet.)

Jožin už vlastní jedno potvrzení, ale potřeboval by si vyřídit potvrzení jiného typu. Nyní ho zajímá, jestli je to vůbec možné a pokud ano, jaký *nejmenší počet osobních dokladů* mu k tomu bude stačit. (Pro Jožina je snazší oběhat pár úředníků navíc než najít ve své rodné bažině rodný list.)

Úloha. Je dán počet různých typů potvrzení N ($2 \leqq N \leqq 10\,000$), počet úředníků M ($1 \leqq M \leqq 1\,000\,000$) a číslo K ($1 \leqq K \leqq 10\,000$) udávající počet různých osobních dokladů existujících v Bürrolandu.

Typy potvrzení jsou očíslované od 1 do N. Jožin vlastní potvrzení typu 1 a shání potvrzení typu N.

Pro každého úředníka jsou dána tři čísla: typ potvrzení, které mu je potřeba ukázat, typ potvrzení, které vydává, a počet osobních dokladů, které je nutné mít s sebou (číslo od 0 do K).

Váš program má vypsat nejmenší počet osobních dokladů, které stačí k získání potvrzení typu N, a také pořadí, v jakém máme potvrzení vy-

řizovat. Pokud není možné požadované potvrzení získat, vypište místo toho zprávu, že to není možné.

Příklad:

I vstup	*výstup*
$N = 4$, $M = 5$, $K = 2$	1
1 4 2	3 4
1 2 0	
2 3 2	
3 4 1	
1 3 1	

Existuje více způsobů, jak získat čtvrté potvrzení. Můžeme ho dostat přímo za potvrzení 1 (u prvního úředníka), ale k tomu potřebujeme dva doklady. Nebo si můžeme nejdříve zařídit potvrzení 2 (u druhého úředníka), potom 3 (u třetího) a nakonec 4 (u čtvrtého), ovšem k tomu potřebujeme předložit třetímu úředníkovi také 2 doklady. Nejlepší je získat potvrzení 3 (u pátého úředníka) a potom 4, na což nám stačí jediný doklad.

P – III – 3

Piškvorky

Mach a Šebestová spolu mají rozehranou partii piškvorek. Šebestová hraje s křížky a začínala.

V proměnných R a C je počet řádků a sloupců hrací plochy, v dvourozměrném poli A je na souřadnicích $[i, j]$ (kde $0 \leqq i < R$, $0 \leqq j < C$) znak ‚X', ‚O' nebo ‚.' (zatím prázdné políčko). V proměnné K je délka řady potřebné k výhře partie.

Můžete předpokládat, že vstup korektně popisuje rozehranou partii, ve které je právě na tahu Šebestová. (Čili počet křížků a koleček je stejný a nikde na hrací ploše se ještě nevyskytuje K stejných znaků v řadě vedle sebe.)

Soutěžní úloha. Napište co nejrychlejší program pro paralelizátor, který pro každý přípustný vstup skončí, přičemž *úspěšně* skončí právě tehdy, když v zadané pozici má Šebestová vyhrávající strategii (jinými slovy, pokud existuje postup, který určí, jak má Šebestová hrát a reagovat na Machovy tahy tak, aby vždy vyhrála, ať už Mach hraje jakkoliv).

Příklady:

I vstup
$R = 6$, $C = 5$, $K = 4$

$$A = \begin{pmatrix} \texttt{.....} \\ \texttt{.....} \\ \texttt{.....} \\ \texttt{.OXO.} \\ \texttt{.X...} \\ \texttt{.....} \end{pmatrix}$$

výstup
skončí úspěšně

(Šebestová začne tahem na políčko $[2, 3]$*, čili na políčko v třetím řádku a čtvrtém sloupci, čímž dostane tři znaky* ‚X' *vedle sebe. I když Mach odpoví tahem na políčko* $[5, 0]$ *nebo* $[1, 4]$*, Šebestová může v dalším tahu vždy táhnout na druhé z nich a vyhrát.)*

I vstup
$R = 6$, $C = 5$, $K = 4$

$$A = \begin{pmatrix} \texttt{XOXOX} \\ \texttt{XOXOX} \\ \texttt{OXO.O} \\ \texttt{XO.OX} \\ \texttt{X....} \\ \texttt{OXOXO} \end{pmatrix}$$

výstup
skončí neúspěšně

(Šebestová prvním tahem řadu čtyř nevytvoří, dokonce ani nedokáže zabránit Machovi, aby příštím tahem vyhrál.)

vstup
$R = 5$, $C = 6$, $K = 6$

$$A = \begin{pmatrix} \texttt{......} \\ \texttt{......} \\ \texttt{......} \\ \texttt{......} \\ \texttt{......} \end{pmatrix}$$

výstup
skončí neúspěšně

(Řada šesti se dá vytvořit jedině vodorovně. Mach proto snadno zabrání Šebestové vyhrát. Při optimální hře obou hráčů partie skončí remízou.)

Piškvorky (studijní text). Piškvorky jsou hra, kterou hrají dva hráči na čtverečkovaném papíru obdélníkového tvaru. Na začátku hry se hráči dohodnou na celém kladném čísle K. Každý hráč má svou značku: hráč, který začíná, obvykle používá křížek (‚X'), druhý hráč kolečko (‚O'). Hráči tahají střídavě, v každém tahu hráč zvolí prázdné políčko na hracím plánu a umístí na něj svou značku. Hra končí, pokud některý z hráčů kdekoliv na hracím plánu vytvořil souvislou řadu K svých znaků ve vodorovném, svislém nebo úhlopříčném směru. Pokud se celá plocha zaplní a nikdo nevyhrál, hra končí remízou.

(Definice paralelizátoru je uvedena v textu úlohy P–I–4.)

Netrvalo dlouho a Kleofáš si uvědomil, že na takovémto zázračném zařízení dokáže některé problémy řešit až neuvěřitelně rychle. Například testování prvočíselnosti je skutečně snadné.

Příklad 1: V proměnné N je přirozené číslo. Napište program pro paralelizátor, který pro každou hodnotu N skončí, přičemž *úspěšně* skončí právě tehdy, když N je prvočíslo.

Řešení. Pomocí volání příkazu **Both** paralelně vygenerujeme všechna čísla od 2 do $N-1$ a najednou pro každé z nich ověříme, zda dělí N. Každá větev výpočtu úspěšně skončí, jestliže „její" číslo nedělí N. Aby původní program úspěšně skončil, musí úspěšně skončit všechny větve, tedy žádné z vygenerovaných čísel nesmí dělit N. Časová složitost programu je $O(\log N)$.

```
{ VSTUP:  N : integer; }

var moc2, pocet_cifer : integer;
    cislo : integer;
    i,x : integer;

begin
   { oetme okrajov ppad }
   if N = 1 then Reject;

   { zjistme, kolik m N cifer ve dvojkov soustav }
   moc2 := 1;
   pocet_cifer := 0;
   while moc2 < N do begin
      moc2 := moc2 * 2;
      inc(pocet_cifer);
   end;

   { vygenerujeme sla od 0 do 2^pocet_cifer - 1 }
   cislo := 0;
   for i:=1 to pocet_cifer do begin
      Both(x);
      cislo := 2*cislo + x;
   end;

   { moc mal dlitele zkouet nebudeme, prohlsme za dobr }
   if cislo <= 1 then Accept;
   { ani pli velk dlitele zkouet nebudeme }
   if cislo >= N then Accept;
   { jinak zkoume, zda vygenerovan slo dl N }
   if N mod cislo <> 0 then Accept;
   Reject;
end.
```

Názorně si ukážeme, jak vypadá výpočet paralelizátoru na tomto programu pro $N = 3$ a pro $N = 6$. Kopie programu, které vznikají během výpočtu, budeme číslovat v pořadí, v jakém vznikají.

Pro $N = 3$ bude výpočet probíhat následovně:

- ▷ Spustí se kopie #1 (tedy vlastně originál).
- ▷ Spočítá, že *pocet_cifer* $= 2$.
- ▷ Spustí se for-cyklus pro i $= 1$.
- ▷ Kopie #1 se zastaví, vzniknou kopie #2 a #3.
- ▷ V kopii #2 je *cislo* $= 0$, v kopii #3 je *cislo* $= 1$.
- ▷ V obou běžících kopiích pokračuje for-cyklus pro i $= 2$.
- ▷ Kopie #2 a #3 se zastaví, z #2 vzniknou #4 a #5, z #3 vzniknou #6 a #7.
- ▷ V kopiích #4 až #7 bude mít proměnná *cislo* hodnoty 0 až 3.
- ▷ Kopie #4 a #5 úspěšně skončí, neboť čísla 0 a 1 nechceme testovat jako dělitele.
- ▷ Kopie #2 úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.
- ▷ Kopie #7 úspěšně skončí, neboť ani číslo 3 nechceme testovat.
- ▷ Kopie #6 úspěšně skončí, neboť 2 nedělí 3.
- ▷ Kopie #3 úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.
- ▷ Kopie #1 (tedy původní program) úspěšně skončí, neboť už úspěšně skončily obě kopie, které z ní vznikly.

Pro $N = 6$ bude výpočet probíhat následovně:

- ▷ Podobně jako při $N = 3$ se dostaneme do situace, kdy běží kopie #8 až #15, proměnná *cislo* v nich má hodnoty postupně od 0 do 7.
- ▷ Kopie #8 a #9 (s příliš malým číslem) úspěšně skončí.
- ▷ Kopie #4 (z níž vznikly #8 a #9) úspěšně skončí.
- ▷ Kopie #14 a #15 (s příliš velkým číslem) úspěšně skončí.
- ▷ Kopie #7 (z níž vznikly #14 a #15) úspěšně skončí.
- ▷ Kopie #10 až #13 skončí — a to: #12 a #13 úspěšně (4 ani 5 nedělí 6), #10 a #11 neúspěšně (2 a 3 dělí 6).
- ▷ Kopie #5 skončí neúspěšně (obě její „děti“ skončily neúspěšně), kopie #6 skončí úspěšně.
- ▷ Kopie #2 skončí neúspěšně (neboť kopie #5 skončila neúspěšně), kopie #3 skončí úspěšně.
- ▷ Kopie #1 (tedy původní program) skončí neúspěšně.

Příklad 2: V proměnných N a K jsou přirozená čísla. Napište program pro paralelizátor, který pro každé N skončí, přičemž *úspěšně* skončí právě tehdy, když N má nějakého dělitele z množiny $M = \{2, 3, \ldots, 2^K - 1\}$.

Řešení. Pomocí volání příkazu **Some** paralelně projdeme všechna čísla $m \in M$, stačí nám, když libovolné jedno z nich dělí N.

(Jiný pohled na totéž řešení: Pomocí volání příkazu **Some** „uhodneme“ dělitele $m \in M$ a ověříme, zda jsme ho uhodli správně. Na náš program se můžeme dívat tak, že se nevětví, ale každé volání **Some** „uhodne“ a do x dosadí „správnou“ hodnotu. Jestliže tedy N má v množině M dělitele, najdeme ho, jinak skončíme s nějakým číslem, které N nedělí.)

Časová složitost programu je $O(K)$.

```
{ VSTUP:  N, K : integer; }

var cislo : integer;
    i,x : integer;

begin
   { paraleln zkoume sla od 0 do 2^K - 1 }
   cislo := 0;
   for i:=1 to K do begin
      Some(x);
      cislo := 2*cislo + x;
   end;

   { 0 a 1 do mnoiny M nepat }
   if cislo <= 1 then Reject;
   { zkusme, zda vygenerovan slo dl N }
   if N mod cislo = 0 then Accept;
   Reject;
end.
```

P – III – 4

Násobek

Program: `nasobek.pas` / `nasobek.c` / `nasobek.cpp`
Vstup: `nasobek.in`
Výstup: `nasobek.out`

Nejmenší kladný násobek čísla 13, který je tvořen jen číslicemi 1 a 2, je 221. I číslo 997 má násobky zapsané jen pomocí číslic 1 a 2, nejmenším z nich je $1\,121\,222\,212 = 997 \times 1\,124\,596$. Nejmenším násobkem tří, ve kterém mohou být použity pouze číslice 4 a 7, je číslo 444.

Vaši úlohou je napsat program, který bude taková čísla hledat.

Vstup: Na prvním řádku vstupního souboru je uveden řetězec R tvořený minimálně jednou a maximálně deseti číslicemi (od 0 do 9). Všechny tyto číslice jsou navzájem různé.

Na druhém řádku je uvedeno jedno kladné celé číslo N ($1 \leqq N \leqq \leqq 1\,000\,000$).

Výstup: Vypište jediný řádek a v něm jediné celé číslo — nejmenší kladný násobek čísla N, ve kterém se vyskytují pouze číslice z řetězce R. (Pozor, toto číslo může mít mnoho číslic.)

Pokud číslo N žádný takový násobek nemá, vypište místo toho řetězec „`neexistuje`".

Příklad:

`nasobek.in`	`nasobek.out`
12	1121222212
997	

Příklad:

`nasobek.in`	`nasobek.out`
1379	neexistuje
2	

Příklad:

`nasobek.in`	`nasobek.out`
7654321	47
47	

P – III – 5

Stránka

Program: `stranka.pas` / `stranka.c` / `stranka.cpp`
Vstup: `stranka.in`
Výstup: `stranka.out`

Rozhodli jsme se, že začneme konkurovat světoznámým vyhledávačům, jako jsou například Google a Yahoo. Hlavním klíčem k úspěchu bude samozřejmě prezentace nalezených stránek uživateli. Přesněji, chtěli bychom z každé nalezené stránky ukázat co nejkratší úsek obsahující všechna slova, která uživatel hledal. Vaší úlohou bude napsat program, který takový úsek v dané stránce nalezne.

Soutěžní úloha. Je dáno N slov, která uživatel zadal. Také je dán text stránky obsahující M slov. Napište program, který najde nejkratší úsek stránky, v němž se vyskytují všechna zadaná slova (každé alespoň jednou).

Úsek stránky tvoří několik po sobě jdoucích slov. Délka úseku je rovna součtu jejich délek plus jejich počet minus 1 (za mezery mezi nimi). Tedy například úsek „`Toto je úsek`" má délku 12.

Vstup: Na prvním řádku vstupního souboru je jediné celé číslo N — počet vyhledávaných slov. Následuje N řádků, na každém z nich je jedno vyhledávané slovo. Všechna tato slova jsou navzájem různá.

Na dalším řádku se nachází celé číslo M — počet slov na stránce. Následuje M řádků, na každém z nich je jedno slovo textu stránky, v pořadí, v jakém jsou na stránce uvedena.

Omezení: Každé slovo je řetězec tvořený 1 až 100 malými písmenky anglické abecedy.

Pro počet vyhledávaných slov N platí $1 \leqq N \leqq 5\,000$. Součet délek vyhledávaných slov nepřekročí 100 000.

Pro počet slov na stránce M platí $1 \leqq M \leqq 200\,000$. Součet délek slov na stránce nepřekročí 1 000 000.

Výstup: Vypište nejkratší úsek stránky, v němž se každé vyhledávané slovo vyskytuje alespoň jednou. Pokud je takových úseků více, vypište ten, který je nejblíže k začátku stránky. Úsek vypisujte tak, jak je uveden na vstupu, tedy každé slovo na samostatném řádku.

Pokud se některé vyhledávané slovo v textu stránky nenachází, vypište jediný řádek s textem „`Chybná stránka!`“ (bez uvozovek).

Příklad:

`stranka.in`	`stranka.out`
3	nasi
nasi	k vasim
vasi	aby
prisli	prisli
20	vasi
poslali	
me	
nasi	
k vasim	
aby	
prisli	
vasi	
k nasim	
kdyz	
neprijdou	
vasi	
k nasim	
tak	
neprijdou	
nasi	
k vasim	