Na řešení úloh máte 4,5 hodiny čistého času.

Řešení každého příkladu musí obsahovat:

- **Popis řešení,** to znamená slovní popis použitého algoritmu, argumenty zdůvodňující jeho správnost (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu).
- **Program.** V úlohách **P-III-1** a **P-III-2** je třeba uvést dostatečně podrobný zápis algoritmu, nejlépe ve tvaru zdrojového textu nejdůležitějších částí programu v jazyce Pascal nebo C. Ze zápisu můžete vynechat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů apod. V úloze **P-III-3** zapište úplný program pro paralelizátor.

Hodnotí se nejen správnost programu, ale také kvalita popisu řešení a efektivita zvoleného algoritmu.

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	**výstup**
4 10/1, 10/1, 10/2, 10/9 10/1, 10/1, 10/2, 10/8	0 *(Bez ohledu na rozdělení se všechny příšery zabijí navzájem.)*
vstup	**výstup**
4 7/3, 2/12, 47/47, 5/6 10/1, 4/7, 3/6, 1/1	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 ($2 \leq N \leq 10\,000$), počet úředníků M ($1 \leq M \leq 1\,000\,000$) a číslo K ($1 \leq K \leq 10\,000$) 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 vyřizovat. Pokud není možné požadované potvrzení získat, vypište místo toho zprávu, že to není možné.

Příklad:

vstup	**výstup**	
$N = 4$, $M = 5$, $K = 2$	1	Existuje více způsobů, jak získat čtvrté potvrzení. Můžeme ho dostat
1 4 2	3 4	přímo za potvrzení 1 (u prvního úředníka), ale k tomu potřebujeme
1 2 0		dva doklady. Nebo si můžeme nejdříve zařídit potvrzení 2 (u druhého
2 3 2		úředníka), potom 3 (u třetího) a nakonec 4 (u čtvrtého), ovšem k tomu
3 4 1		potřebujeme předložit třetímu úředníkovi také 2 doklady. Nejlepší je
1 3 1		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 $0 \leq i < R$, $0 \leq j < C$) 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:

vstup — **výstup**

$R = 6$, $C = 5$, $K = 4$ — skončí úspěšně

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

(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.)

vstup — **výstup**

$R = 6$, $C = 5$, $K = 4$ — skončí neúspěšně

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

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

vstup — **výstup**

$R = 5$, $C = 6$, $K = 6$ — skončí neúspěšně

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

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

Studijní text – Piškvorky

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.

Studijní text – Paralelizátor

(Tento studijní text je identický s textem uvedeným u úloh domácího i krajského kola.)

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
   { ošetříme okrajový případ }
   if N = 1 then Reject;

   { zjistíme, kolik má N cifer ve dvojkové soustavě }
   moc2 := 1;
   pocet_cifer := 0;
   while moc2 < N do begin
      moc2 := moc2 * 2;
      inc(pocet_cifer);
   end;

   { vygenerujeme čísla 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 malé dělitele zkoušet nebudeme, prohlásíme za dobré }
   if cislo <= 1 then Accept;
   { ani příliš velké dělitele zkoušet nebudeme }
   if cislo >= N then Accept;
   { jinak zkoušíme, zda vygenerované číslo dělí 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
   { paralelně zkoušíme čísla 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 množiny M nepatří }
   if cislo <= 1 then Reject;
   { zkusíme, zda vygenerované číslo dělí 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 ($1 \leq N \leq 1\,000\,000$).

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`
12 997	1121222212
`nasobek.in`	`nasobek.out`
1379 2	neexistuje
`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 $1 \leq N \leq 5\,000$. Součet délek vyhledávaných slov nepřekročí 100 000.

Pro počet slov na stránce M platí $1 \leq M \leq 200\,000$. 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`
```
3
nasi
vasi
prisli
20
poslali
me
nasi
k
vasim
aby
prisli
vasi
k
nasim
kdyz
neprijdou
vasi
k
nasim
tak
neprijdou
nasi
k
vasim
```

`stranka.out`
```
nasi
k
vasim
aby
prisli
vasi
```