74. ročník Matematické olympiády – 2024/2025

Úlohy ústředního kola kategorie P – 2. soutěžní den

Na řešení úloh máte 4.5 hodiny čistého času. Při soutěži je zakázáno používat jakékoliv pomůcky kromě psacích potřeb a přiděleného počítače (tzn. knihy, kalkulačky, mobily, apod.).

Řešením každého příkladu je zdrojový kód programu zapsaný v jednom z následujících programovacích jazyků: C, C++ 20, Python 3.11, C# 11, Java 11 nebo Pascal. Vzhledem k různé výkonnosti stejného algoritmu implementovaného v různých programovacích jazycích nezaručujeme, že je ve všech podporovaných jazycích možné získat plný počet bodů – může se stát, že program nestihne doběhnout do časového limitu, i když implementuje algoritmus s optimální časovou složitostí. Časové limity jsou (s rezervou) nastavené dle autorských řešení implementovaných v C++.

Řešení odevzdáváte pomocí soutěžního systému CMS, který ho automaticky otestuje na připravených sadách testovacích dat. Detaily hodnocení naleznete v letáku s popisem prostředí. Podrobnější informace o testovacích datech najdete na konci zadání každé úlohy.

Zadání naleznete na další straně.

P-III-4 Barevný graf

Jirka nabyl dojmu, že je aktuální logo Matematické olympiády příliš nudné. Jeho tvar je totiž předvídatelný a neobsahuje žádné zajímavé barvy. Rozhodl se proto přijít se svým vlastním návrhem. Protože má rád informatiku, chce, aby výsledné logo tvořilo graf, a má v úmyslu použít k jeho vytvoření přesně definovaný postup.

Soutěžní úloha

Na začátku Jirka namaluje na plátno N různobarevných vrcholů, číslovaných od 0 do $N-1$. Dále provede nahodile Q operací dvojího typu:

- Buď vybere dvě různé na plátně přítomné barvy i a j a všechny vrcholy barvy i přebarví na barvu j. Barva i tak zanikla a Jirka ji již nikdy nepoužije.
- Nebo vybere dvě různé na plátně přítomné barvy i a j a přidá do grafu neorientovanou hranu z každého vrcholu barvy i do každého vrcholu barvy j. Poznamenejme, že i předtím mohou být některé vrcholy barvy i spojené hranou s některými vrcholy barvy j; v tom případě pro ně nic neděláme, spojené hranou tedy zůstanou.

Jirka takto již vytvořil první návrh a nyní by jej zajímalo, jaká je nejmenší vzdálenost mezi vybranými vrcholy u a v vzniklého grafu. Vzdáleností se rozumí nejmenší počet hran, přes které je nutné přejít, abychom se z vrcholu u dostali do vrcholu v. Můžete předpokládat, že taková posloupnost hran (cesta) pro zadané vrcholy u a v existuje.

Formát vstupu

Na prvním řádku dostanete přirozené číslo N ($1 \leq N \leq 10^6$), značící počet vrcholů grafu, a celé číslo Q ($0 \leq Q \leq 3 \cdot 10^6$), udávající celkový počet Jirkou provedených operací. Vrcholy i barvy jsou reprezentovány celými čísly v rozsahu od 0 do $N-1$ a platí, že vrchol i má na začátku barvu i. Druhý řádek obsahuje čísla u a v značící startovní a cílový vrchol hledané nejkratší cesty ($0 \leq u, v \leq N-1$).

Následuje Q řádků s jednotlivými operacemi v pořadí, v jakém je Jirka prováděl. Každý takový řádek obsahuje písmeno značící typ operace a dvě čísla i a j zpracovávaných barev ($0 \leq i, j \leq N-1$, $i \neq j$). Je zaručeno, že na plátně existuje aspoň jeden vrchol barvy i a aspoň jeden vrchol barvy j. Pokud Jirka přebarví všechny vrcholy barvy i barvou j, řádek bude ve formě `p` i j. Pokud spojí každý vrchol barvy i s každým vrcholem barvy j, řádek bude ve formě `c` i j.

Formát výstupu

Výstup by měl obsahovat jediné číslo, a to délku nejkratší cesty v grafu z vrcholu u do vrcholu v. Slibujeme, že taková cesta vždy existuje.

Bodování

Je pět sad vstupů s různými omezeními. Popisy sad a dodatečná omezení uvádíme v tabulce:

sada	body	max N	max Q	další omezení
1	2	10^6	$3 \cdot 10^6$	Jirka pouze přidává hrany.
2	1	10^6	$3 \cdot 10^6$	Jirka nejprve jen přebarvuje vrcholy a poté už pouze přidává hrany.
3	1	300	900	
4	2	10 000	30 000	
5	4	10^6	$3 \cdot 10^6$	

Příklady

Vstup:

```
6 5
0 4
c 0 1
p 4 5
c 3 5
p 1 2
c 2 5
```

Výstup:

```
2
```

Existuje cesta $0 \to 1 \to 4$ *a dá se ukázat, že je nejkratší. Hrana mezi vrcholy* 1 *a* 4 *existuje, protože* 1 *byla přebarvena barvou* 2 *a vrchol* 4 *barvou* 5. *Poté Jirka propojil všechny vrcholy barvy* 2 *s každým vrcholem barvy* 5 *a propojil tedy i tyto dva vrcholy.*

Vstup:

```
6 5
0 5
c 1 0
c 1 2
c 2 3
c 3 4
c 4 5
```

Výstup:

```
5
```

V tomto případě existuje pouze jedna cesta, a to $0 \to 1 \to 2 \to 3 \to 4 \to 5$.

P-III-5 Jednosměrná jednokolejka

Mezi Slonovem a Taškovem vede kvůli velmi nehostinnému terénu a nedostatečnému financování pouze jedna jednosměrná jednokolejka. Ve Slonově mají i tak všichni velmi rádi vlaky, a tak je tato jednokolejka značně využívaná a dochází na ní k velkým zpoždění. Často se totiž stává, že se rychlík zasekne za pomalejším vlakem a musí s ním jet pomalu.

Lukáš, který tuto slavnou jednokolejku podědil po svém prastrýci, už má dost stížností na podobně slavná zpoždění. Po dlouhém zamyšlení došel k tomu, že čím menší bude celkové zpoždění, tím méně bude stížností! Rozhodl se tedy s vaší pomocí úplně přerovnat odjezdy, a minimalizovat tak celkové zpoždění.

Soutěžní úloha

V plánu je poslat jednokolejkou n vlaků. Vlak i je připraven k odjezdu v čase p_i a je jednoho ze tří druhů: osobní vlak, rychlík nebo expres. závislosti na druhu vlaku trvá projetí trasy Slonov-Taškov $t_{\mathrm{Os}} \geq t_{\mathrm{R}} \geq t_{\mathrm{Ex}}$ minut.

Cesta vlaku vypadá následovně: Poté, co je vlak připraven k odjezdu, může ještě čekat libovolně dlouho na nádraží ve Slonově. Na nádraží může čekat libovolné množství vlaků a čekající vlaky mohou být pouštěny na jednokolejku v libovolném pořadí. V libovolný okamžik můžeme pustit libovolně mnoho z vlaků, které jsou připraveny.

Jakmile vlaky pustíme na jednokolejku v nějakém pořadí, musí už v tomto pořadí zůstat až do konce jednokolejky. Pokud tedy před vlakem A, který by normálně dorazil v čase a, jede pomalejší vlak B, který dorazí až v čase $b > a$, pak vlak A nemůže dorazit dříve než v čase b.

Vlak typu X připravený k odjezdu v čase p má očekávaný čas příjezdu do Taškova $p + t_X$. Jestliže do cíle dorazí až v čase a, má tedy zpoždění $a - (p + t_X)$. Určete nejmenší možný součet zpoždění všech vlaků, kterého lze dosáhnout.

Formát vstupu

První řádek obsahuje jedno kladné celé číslo n ($1 \leq n \leq 400$) udávající počet vlaků. Na dalším řádku jsou tři kladná celá čísla t_{Os}, t_{R}, t_{Ex} ($1 \leq t_{\mathrm{Ex}} \leq t_{\mathrm{R}} \leq t_{\mathrm{Os}} \leq 10^5$) oddělená mezerami, popisující, jak dlouho projíždí vlak daného typu jednokolejkou. Následuje n řádků popisující vlaky; i-tý z nich obsahuje nejprve nezáporné celé číslo p_i ($0 \leq p_i \leq 10^9$) udávající čas připravení i-tého vlaku, a následně řetězec popisující druh vlaku: `Os` pro osobní vlak, `R` pro rychlík, a `Ex` pro expresní vlak. Vlaky jsou seřazeny vzestupně podle času připravení.

Formát výstupu

Výstupem je jediné celé číslo: nejmenší možný celkový počet minut zpoždění.

Bodování

Je osm sad vstupů s různými omezeními. Popisy sad a dodatečná omezení uvádíme v tabulce:

sada	body	max n	další omezení
1	1	8	—
2	1	14	—
3	1	40	—
4	1	100	$t_{\mathrm{Os}} \leq 4$, $t_{\mathrm{R}} = 1$, žádné expresy, jedinečné časy připravenosti
5	1	100	$p_i \leq 1000$, $t_{\mathrm{Os}} \leq 100$, $t_{\mathrm{R}} \leq 100$, $t_{\mathrm{Ex}} \leq 100$
6	2	100	žádné expresy
7	1	100	—
8	2	400	—

Omezení *jedinečné časy připravenosti* znamená, že v daný čas může být připraven k odjezdu maximálně jeden vlak, tedy $p_i \neq p_j$ pro všechny vlaky $i \neq j$.

Příklady

Vstup:
```
5
12 8 4
3 Os
5 R
7 Os
9 Ex
11 Os
```

Výstup:
```
6
```

První dva vlaky pustíme ihned potom, co budou připraveny k odjezdu. První vlak tedy dorazí v čase $3 + 12 = 15$. *Druhý by dorazil v* $5 + 8 = 13$, *což znamená, že ten první ho zdrží o 2 minuty. Třetí vlak zdržíme o 2 minuty, aby pak nezdržoval následující expres. Tento expres pak může dorazit nejdříve v čase* $9 + 4 = 13$, *takže jej stejně zdrží první vlak o dvě minuty, neboť ten dokončí cestu až v čase* 15. *Pátý vlak vypustíme ihned v čase* 11. *K žádnému dalšímu zpoždění nedojde. Menšího celkového zpoždění nejde dosáhnout.*

Vstup:
```
5
4 1 1
3 R
5 Os
6 R
7 R
8 R
```

Výstup:
```
3
```

P-III-6 Počítáme s tříděním

K této úloze se vztahuje studijní text uvedený na následujících stranách, který je totožný se studijním textem ze zadání domácího a krajského kola.

Ondrova řada procesorů se opět vrací, ovšem tentokrát v praktické úloze! Čeká vás jen jedna úloha, zato s více sadami vstupů. Neodevzdáváte přímo program pro Ondrův procesor, ale *generátor* takového programu. To je program v jednom z podporovaných klasických jazyků, který pro zadané N vypíše program pro Ondrův procesor s N jádry, jenž úlohu řeší.

Úloha

Každé jádro i dostane jako vstup dvě čísla. První z nich je identifikační číslo a_i jeho *idolu*, jednoho z jader. Druhé z nich je jeho oblíbené číslo b_i. Výstupem jádra i by mělo být oblíbené číslo jeho idolu, neboli b_{a_i}.

Čísla a_i budou vždy od 0 do $N-1$, zato b_i může být jakékoliv 64-bitové číslo. Může stát, že je jádro natolik egoistické, že je svým vlastním idolem. Počet jader N bude vždy *mocnina dvojky* a alespoň 2.

Příklad

Vstup:

```
 1  5  5  5  6  0  5  7
78 38 60 57 -3 48 13 35
```

Výstup:

```
38 48 48 48 13 78 48 35
```

Vstup a výstup je uvedený ve formátu, který používá simulátor.

Vstup a výstup generátoru

Vámi napsaný generátor dostane na vstupu celé číslo N a znaky d a p, oddělené mezerami. Číslo N ($N \in \{2^1, 2^2, \ldots, 2^{16}\}$) udává počet jader. Znaky d a p popisují další omezení, které budou ve vstupech programu pro Ondrův procesor zaručeně platit. Jejich význam najdete v těchto dvou tabulkách:

d	význam
`S`	Všechna jádra mají stejný idol.
`R`	Žádné dvě jádra nemají stejný idol.
`-`	*žádná další omezení*

p	význam
`P`	Každé jádro se sudým ID má idol s lichým ID, obdobně každé jádro s lichým ID má idol se sudým ID.
`-`	*žádná další omezení*

Výstupem generátoru má být program pro Ondrův procesor, ve formátu, který je popsaný ve studijním textu. Váš program smí obsahovat nejvýše **6 144** instrukcí, jinak nebude uznán.

Bodování

Je celkem šest sad vstupů:

sada	body	max N	d	p
1	1	2^7	`-`	`-`
2	1	2^{16}	`S`	`-`
3	2	2^{16}	`R`	`P`
4	1	2^{16}	`-`	`P`
5	3	2^{16}	`R`	`-`
6	1	2^{11}	`-`	`-`
7	1	2^{16}	`-`	`-`

Příklad pro generátor

Vstup:

```
2 - -
```

Výstup:

```
jádro:    input 1
číslo:    input 2
id:       id
egoista:  = jádro id
vlevo:    left číslo
          if egoista číslo vlevo
```

Každé jádro nejprve zjistí, jestli svým vlastním idolem. Pokud ano, pak vrátí své vlastní oblíbené číslo, jinak vrátí oblíbené číslo jádra nalevo od něj. Tento program funguje pouze pro dvě jádra.

Odevzdávání

Svůj generátor odevzdejte v soutěžním systému, jako řešení jakékoliv jiné úlohy.

Pokud váš generátor nedoběhne do časového limitu, bude ukončen signálem nebo z jiného důvodu selže, pak bude ve detailech vyhodnocení uvedená příslušná hláška. Pokud váš generátor vygeneruje neplatný nebo příliš dlouhý program, pak to tam bude také uvedené. Hláška „Nesprávný výstup“ může znamenat jedině to, že generátor vygeneroval platný program pro Ondrův procesor, ale tento program na nějakém vstupu nevyřešil úlohu správně.

V detailech vyhodnocení najdete také sekci *Podúloha s příklady 1*. V ní najdete výsledek vyhodnocení vašeho generátoru a jeho výstupu na vstupu uvedeném výše v části *Příklad pro generátor*.

Simulátor

Abyste mohli programy pro Ondrův procesor vyzkoušet, budete mít opět k dispozici simulátor. Simulátor umí spustit daný program na daném vstupu a vypsat vám jeho výstup. Také umí vygenerovat tabulku s výsledky všech instrukcí.

Simulátor existuje ve dvou verzích. První z nich má podobu webového rozhraní, jaké znáte z minulých kol. Odkaz na něj najdete na ploše. Jím vygenerovaná tabulka je interaktivní a umožní vám zobrazit si pořadí jader po každé vykonané instrukci

`sort`. Druhá z nich má podobu aplikace pro příkazový řádek. Je nainstalovaná na vašem počítači, můžete ji spustit příkazem `sort-machine`. Kromě grafické tabulky umí vypsat výsledky každé instrukce i ve formátu CSV. Informace o použití program vypíše při spuštění s přepínačem `--help`.

Od domácího kola jsme simulátor naučili v ladicí tabulce zobrazovat i komentáře. Webová verze je zobrazuje vždy, verze pro příkazový řádek, jen pokud je spuštěna s přepínačem `-i`.

V každém případě musíte zadat program a vstup. Formát programu je popsaný ve studijním textu. Vstupní soubor musí obsahovat tolik řádků, jako má každé z jader vstupů. Na prvním řádku by měl být první vstup všech jader od prvního po poslední, na druhém druhý vstup všech jader a tak dále. Hodnoty v rámci řádku mohou být odděleny jednou nebo více mezerami či tabulátory. Pokud chcete simulovat program, který nemá žádný vstup, musíte specifikovat počet jader (a vstup můžete nechat prázdný). V opačném případě počet jader uvádět *nemusíte*.

Simulátor nekontroluje limity ze zadání. Webová verze simulátoru omezuje délku programu a vstupu. Verze pro příkazový řádek žádná omezení nevymáhá, ale upozorňujeme, že ke spuštění programu s M instrukcemi na N jádrech budete potřebovat přibližně $8 \cdot N \cdot M$ bajtů paměti.

Studijní text

Ondra si před deseti lety během zkoušky z algoritmizace vylosoval třídicí algoritmy, které se nenaučil, a ze zkoušky dostal čtyřku. Tehdy přísahal, že se třídicím algoritmům pomstí. Pomstít se algoritmu ale není vůbec jednoduché, protože algoritmus není možné zničit ani uvěznit. Algoritmus může porazit jedině tím, že ho překoná.

Proto Ondra posledních deset let strávil vývojem úplně nového modelu procesoru, který umí třídit v konstantním čase. Je si jistý, že za pár let bude počítač s Ondrovým procesorem v každé domácnosti. Na třídicí algoritmy pak všichni do pár dalších let zajisté zapomenou!

Stroje na masovou výrobu integrovaných obvodů jsou ale velmi drahé, takže Ondra potřebuje pár ukázkových programů, aby získal přízeň investorů. Protože Ondra nikdy algoritmizaci nedostudoval, chce, abyste mu s tím pomohli.

Ondrův procesor

Ondrův procesor má N jader, každé má jiné identifikační číslo od 0 do $N - 1$. Jádra jsou uspořádána v řadě. Na začátku jsou seřazená podle identifikačních čísel s jádrem 0 vlevo, jejich pořadí se ale v průběhu výpočtu může měnit.

Programy pro něj jsou tvořeny posloupností instrukcí, které všechna jádra vykonávají souběžně, tedy nejprve všechna jádra současně vykonají první instrukci, poté všechna jádra vykonají druhou instrukci, a tak dále. Existuje mnoho různých instrukcí, jejich seznam najdete níže. Instrukce číslujeme od jedné.

Každá instrukce spočítá hodnotu, které budeme říkat výsledek. Každé jádro si pamatuje výsledky všech instrukcí, které za běh programu vykonalo. Většina in-

strukcí pracuje s výsledky předchozích instrukcí, například instrukce `+ 1 2` sečte výsledek první a druhé instrukce.

Jádra mohou číst paměť svých sousedů pomocí speciálních instrukcí `left` a `right`. Instrukce `left` i přečte výsledek i-té instrukce vykonané jádrem nalevo od jádra, co ji vykoná, obdobně `right` i přečte výsledek i-té instrukce jádra napravo. Čtení probíhá cyklicky, tedy první jádro je napravo od posledního.

Nakonec tu máme nejdůležitější instrukci, a to `sort`, která změní pořadí jader. Každé jádro si nejprve zvolí číslo zvané klíč (argument instrukce `sort`). Poté procesor pomocí komplikovaných obvodů jádra seřadí. Jádro s nejmenším klíčem skončí nalevo a jádro s největším klíčem skončí napravo. Třídění je stabilní, tj. jádra se stejným klíčem si zachovají vzájemné pořadí. Jádra si při přesuvání zachovávají svoje identifikační čísla i historii výsledků instrukcí, jádro s identifikačním číslem 0 tedy po přesunutí už nemusí být nalevo.

Aritmetika

Ondrův procesor používá 64-bitová znaménková čísla ve dvojkovém doplňku, muže tedy reprezentovat čísla v rozsahu -2^{63} až $2^{63}-1$. Pokud se výsledek nějaké operace nevejde do 64 bitů, pak budou přebytečné nejvýznamnější bity zahozeny.

Vstup a výstup

Každé jádro má několik vstupů (počet záleží na problému) očíslovaných od 1. Vstup číslo k může jádro přečíst pomocí instrukce `input` k. Za výstup jádra považujeme výsledek poslední instrukce, kterou vykoná.

Instrukční sada

K dispozici jsou následující instrukce. V popisu instrukcí označíme výsledek i-té instrukce pro specifikované jádro r_i.

`const` x	Vrátí číslo x.
`id`	Vrátí identifikační číslo jádra.
`input` k	Vrátí k-tou vstupní hodnotu tohoto jádra.
`copy` i	Vrátí r_i.
`sort` i	Seřadí jádra podle klíče r_i, vrátí r_i.
`left` i	Vrátí r_i jádra nalevo.
`right` i	Vrátí r_i jádra napravo.
`+` a b	Vrátí $r_a + r_b$.
`-` a b	Vrátí $r_a - r_b$.
`*` a b	Vrátí $r_a \cdot r_b$.
`/` a b	Vrátí dolní celou část $r_a \div r_b$, pokud $r_b \neq 0$, jinak vrátí 0.
`%` a b	Vrátí zbytek po dělení $r_a \div r_b$, pokud $r_b \neq 0$, jinak vrátí 0.
`|` a b	Vrátí bitovou disjunkci (OR) r_a a r_b.
`&` a b	Vrátí bitovou konjunkci (AND) r_a a r_b.
`^` a b	Vrátí bitovou exkluzivní disjunkci (XOR) r_a a r_b.
`=` a b	Vrátí 1, pokud $r_a = r_b$, jinak vrátí 0.
`!=` a b	Vrátí 1, pokud $r_a \neq r_b$, jinak vrátí 0.
`<` a b	Vrátí 1, pokud $r_a < r_b$, jinak vrátí 0.

`>` a b	Vrátí 1, pokud $r_a > r_b$, jinak vrátí 0.
`<=` a b	Vrátí 1, pokud $r_a \leq r_b$, jinak vrátí 0.
`>=` a b	Vrátí 1, pokud $r_a \geq r_b$, jinak vrátí 0.
`if` c a b	Pokud $r_c \neq 0$, pak vrátí r_a, jinak vrátí r_b.

Formát programu

Programy pro Ondrův procesor se zapisují do textového souboru. Každá instrukce je zapsaná na svém vlastním řádku. Jméno instrukce a její argumenty jsou oddělené alespoň jednou mezerou či tabulátorem. V kódu mohou být prázdné řádky, které nic nedělají a nepočítají se mezi instrukce. Program může obsahovat i komentáře, které začínají znakem „`#`“ a končí na konci řádku. Například program pro výpočet druhé mocniny čísla procesoru můžeme zapsat takto:

```
id    # Získáme identifikační číslo procesoru
* 1 1 # Vynásobíme ho sebou samým
```

Instrukce je možné pojmenovat; stačí na začátek řádku připsat jméno následované znakem „`:`“. Jméno může obsahovat písmena, číslice a podtržítka, nesmí se skládat jen z číslic. Jméno instrukce je pak možné použít místo jejího čísla:

```
procesor: id
mocnina:  * procesor procesor
```

Stejné jméno můžete použít vícekrát, v tom případě jméno vždy označuje poslední přecházející takto pojmenovanou instrukci. Instrukce také může mít více jmen. Například program pro sečtení tří vstupů můžete zapsat takto:

```
součet: const 0
vstup:  input 1
součet: + součet vstup
vstup:  input 2
součet: + součet vstup
vstup:  input 3
součet: + součet vstup
```

Příklad #1: hledání minima

Každé jádro dostane jako vstup jedno číslo, každé jádro jiné. Jádro s nejmenším číslem by mělo mít výstup 1, ostatní jádra výstup 0.

Řešení:

```
vstup:        input 1
              sort vstup
vstup_nalevo: left vstup
              > vstup_nalevo vstup
```

Jádra seřadíme podle čísla na vstupu, čímž dostaneme jádro s nejmenším vstupem na začátek řady. Poté každé jádro zjistí, jestli má jádro nalevo od něj větší

vstup než ono samo. Pokud tomu tak je, pak se musí nacházet na začátku řady, a tedy mít nejmenší vstup.

Příklad #2: hledání duplikátů

Každé jádro dostane jako vstup dvě čísla. Jádro musí odpovědět 1, pokud existuje jiné jádro se stejným vstupem, jinak musí odpovědět 0.

Řešení:

```
vstup_1:  input 1
vstup_2:  input 2
          sort vstup_2
          sort vstup_1

soused_1: left vstup_1
soused_2: left vstup_2
stejné_1: = soused_1 vstup_1
stejné_2: = soused_2 vstup_2
vlevo:    & stejné_1 stejné_2

soused_1: right vstup_1
soused_2: right vstup_2
stejné_1: = soused_1 vstup_1
stejné_2: = soused_2 vstup_2
vpravo:   & stejné_1 stejné_2

          | vlevo vpravo
```

Jádra seřadíme nejprve podle druhého vstupu, poté podle prvního vstupu. Díky tomu, že třídění zachovává pořadí jader se stejným klíčem, budou nakonec jádra seřazená primárně podle prvního vstupu a sekundárně podle druhého vstupu (a terciárně podle identifikačního čísla). Jádra se stejnými vstupy tedy skončí u sebe. Poté stačí, aby se každé jádro podívalo na levého a pravého souseda, a zjistilo, jestli některý má stejný vstup.