Řešení každého příkladu musí obsahovat podrobný popis použitého algoritmu, zdůvodnění jeho správnosti a diskusi o efektivitě zvoleného řešení (tzn. posouzení časových a paměťových nároků programu).

V úlohách P-I-1, P-I-2 a P-I-3 je třeba k řešení připojit odladěný program zapsaný v jazyce Pascal, C nebo C++. Program se odevzdává v písemné formě (jeho výpis je tedy součástí řešení) i elektronicky (na CD, na disketě, případně po dohodě lze i zaslat e-mailem), aby bylo možné otestovat jeho funkčnost. Slovní popis řešení musí být ovšem jasný a srozumitelný, aniž by bylo nutno nahlédnout do zdrojového textu programu. V úloze P-I-4 je nutnou součástí řešení úplný popis všech použitých překládacích strojů.

Řešení úloh domácího kola MO kategorie P vypracujte a odevzdejte buď ve škole, nebo zašlete přímo vaší krajské komisi MO, nejpozději do 15. 11. 2007. Vzorová řešení úloh naleznete po tomto datu na Internetu na adrese *http://mo.mff.cuni.cz/*. Na stejném místě jsou stále k dispozici veškeré aktuální informace o soutěži a také archiv soutěžních úloh a výsledků minulých ročníků.

P-I-1 O zdánlivém kopci

Franta a Pepík se chystali na výlet. Vymysleli si trasu, kudy spolu půjdou, potom každý vytáhnul svoji mapu a začal si v ní trasu prohlížet. Některé body, kterými trasa vedla, měly v mapách vyznačenu nadmořskou výšku.

„To je zajímavé," řekl po chvíli Franta. „Když se dívám jen na nadmořské výšky bodů, kterými budeme procházet, vypadá to, že půjdeme jenom přes jeden kopec – nejprve nahoru a potom dolů."

„Ale kdepak, to není pravda," podivil se Pepík.

Za chvíli zjistili, kde vzniknul problém: měli různě přesné mapy. Některé nadmořské výšky bodů, které byly na Pepíkově mapě vyznačeny, na Frantově mapě chyběly.

Soutěžní úloha: Napište program, který dostane na vstupu seznam nadmořských výšek z Pepíkovy mapy a zjistí, kolik nejvýše z nich může být vyznačeno i na Frantově mapě.

Formát vstupu: První řádek vstupu obsahuje jedno celé číslo N ($1 \leq N \leq 100\,000$) – počet těch bodů na trase výletu, které mají na Pepíkově mapě vyznačenou nadmořskou výšku. Další řádky obsahují celkem N celých čísel z rozsahu 1 až $1\,000\,000\,000$ – nadmořské výšky těchto bodů (v nějakých vám neznámých jednotkách). Výšky jsou uvedeny v pořadí, jak po sobě následují na trase výletu.

Formát výstupu: Výstupem bude jediné celé číslo K – největší počet nadmořských výšek z Pepíkovy mapy, které mohly být vyznačeny i na Frantově mapě.

Jestliže $a_1, \ldots, a_K$ jsou výšky vyznačené na Frantově mapě, pak musí splňovat následující podmínku: pro nějaké i z množiny $\{1, \ldots, K\}$ musí platit $\forall j \in \{1, \ldots, i-1\} : a_j < a_{j+1}$ a zároveň $\forall j \in \{i, \ldots, K-1\} : a_j > a_{j+1}$.

Příklad:

vstup:
12
112 247 211 209 244
350 470 510 312 215
117 217

výstup:
9

Jedna možnost, jak mohly vypadat nadmořské výšky na Frantově mapě:
112, 211, 244, 350, 470, 510, 312, 215, 117

P-I-2 Rezervace místenek

„V horách se našlo zlato!" šeptali si lidé v širokém okolí už několik týdnů. Ti s dobrodružnější povahou si už balili věci a mířili přímo do hor, do legendami opředené oblasti zvané Horní Klondík.

Když ale Horní Klondík zaplavila vlna nových zlatokopů, nastal nečekaný problém. Jediný místní vláček, který v Klondíku měli, byl najednou tak přecpaný, že se do něj polovina zájemců ani nevešla. A nedivte se, že zlatokopa, který se žene za co nejvýhodnější parcelou, něco takového pořádně rozzlobí.

Když už se zdálo, že zanedlouho dojde na každé železniční stanici ke krveprolití, dostali železničáři spásonosný nápad. Budou na vláček prodávat místenky.

Soutěžní úloha: Napište program, který bude zpracovávat rezervace míst na jednu jízdu vlaku. Trať vlaku má $N + 1$ stanic, které si očíslujeme od 0 do N v pořadí, jak na trati leží. Ve vlaku je M míst, takže mezi každou dvojicí po sobě následujících stanic může jet vlakem nejvýše M zlatokopů.

Váš program musí postupně zpracovat několik požadavků na rezervaci míst. Každý požadavek je tvaru „x lidí chce jet ze stanice y do stanice z." Pokud je ještě na každém úseku trati mezi stanicemi y a z ve vlaku aspoň x míst volných, váš program takovýto požadavek přijme, v opačném případě ho odmítne.

Formát vstupu: První řádek vstupu obsahuje tři celá čísla N, M a P ($1 \leq N, M, P \leq 100\,000$) – počet úseků trati, počet míst ve vlaku a počet požadavků na rezervaci.

Následuje P řádků, každý z nich popisuje jeden požadavek na rezervaci v pořadí, v jakém byly tyto požadavky zadány. Přesněji, i-tý z těchto řádků obsahuje tři celá čísla x_i, y_i, z_i oddělená mezerami ($1 \leq x_i \leq M$, $0 \leq y_i < z_i \leq N$). Význam těchto hodnot byl popsán výše.

Formát výstupu: Pro každý požadavek vypište na výstup jeden řádek a na něm buď řetězec `prijat`, jestliže příslušný požadavek bylo ještě možné splnit, nebo řetězec `odmitnut`, pokud už ve vlaku nebylo dost volných míst.

Příklad:

vstup:	*výstup:*
4 6 6	`prijat`
2 1 4	`prijat`
2 1 3	`odmitnut`
3 2 4	`odmitnut`
3 1 2	`prijat`
6 0 1	`prijat`
4 3 4	

Po přijetí prvních dvou požadavků víme, že v úsecích mezi stanicemi 1 a 2 a mezi stanicemi 2 a 3 zůstanou už jen dvě volná místa. Proto musíme odmítnout následující dvě trojčlenné skupiny, které chtějí cestovat i na těchto úsecích tratě. Poslední dva požadavky opět můžeme splnit. U prvního z nich je to zřejmé, u druhého nám ve stanici 3 vystoupí dostatek lidí na to, aby se na poslední úsek trati uvolnila přesně potřebná čtyři místa.

P-I-3 Fibonacciho soustava

Fibonacciho posloupnost vypadá následovně:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

Sestrojíme ji tak, že prvními dvěma členy posloupnosti jsou 0 a 1 a každý následující člen posloupnosti je součtem dvou předcházejících. Matematicky zapsáno:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_{n+2} = F_{n+1} + F_n \qquad (\forall n \geq 0)$$

Podívejme se nyní na to, jak fungují poziční číselné soustavy. Poziční číselná soustava je určena dvěma údaji: množinou používaných cifer a posloupností, která pro každou pozici v zápisu čísla udává hodnotu, jíž se příslušná cifra násobí.

Například naše desítková soustava používá množinu cifer $\{0, 1, 2, \ldots, 9\}$ a jednotlivé pozice v čísle mají hodnoty (zprava doleva) 1, 10, 100, 1000, ...

Fibonacciho číselná soustava je poziční číselná soustava, která používá pouze cifry 0 a 1 a v níž jsou hodnoty jednotlivých pozic postupně rovny členům Fibonacciho posloupnosti (počínaje číslem $F_2 = 1$). Tedy například zápis 10011 ve Fibonacciho soustavě představuje číslo $1 \cdot 8 + 0 \cdot 5 + 0 \cdot 3 + 1 \cdot 2 + 1 \cdot 1 = 11$.

Na rozdíl od běžných číselných soustav ve Fibonacciho soustavě nemají některá čísla jednoznačný zápis. Například také zápis 10100 představuje číslo 11.

Soutěžní úloha:

a) *(3 body)* Zápis čísla ve Fibonacciho soustavě nazveme *pěkný*, jestliže se v něm nikde nevyskytují dvě po sobě jdoucí jedničky. Dokažte, že každé přirozené číslo má ve Fibonacciho soustavě právě jeden pěkný zápis. Napište program, který ze vstupu přečte přirozené číslo N a vypíše jeho pěkný zápis.

b) *(7 bodů)* Napište program, který pro daná k, A a B spočítá, kolik čísel z množiny $\{A, A+1, \ldots, B\}$ má ve svém pěkném zápisu právě k jedniček.

Formát vstupu:

V části a) je na vstupu jediné přirozené číslo N ($1 \leq N \leq 1\,000\,000\,000$).

V části b) jsou na vstupu tři přirozená čísla k, A a B ($1 \leq k \leq 30$, $1 \leq A < B \leq 1\,000\,000\,000$).

Formát výstupu:

V části a) vypište na jeden řádek řetězec nul a jedniček, který představuje pěkný zápis čísla N.

V části b) vypište jedno celé číslo – počet čísel ze zadaného intervalu, která mají ve svém pěkném zápisu právě k jedniček.

Příklady pro část a)

vstup 1:	*výstup 1:*
11	10100
vstup 2:	*výstup 2:*
174591	10100010000000000100010010

Příklady pro část b)

vstup 1:	*výstup 1:*
1 4 13	3

(Pro $k = 1$ je výstupem počet Fibonacciho čísel v daném rozsahu. V zadaném rozsahu leží Fibonacciho čísla 5, 8 a 13.)

vstup 2:	*výstup 2:*
2 4 13	6
vstup 3:	*výstup 3:*
10 102000 103000	86

P-I-4 Překládací stroje

V tomto ročníku olympiády budeme pracovat s překládacími stroji. Ve studijním textu uvedeném za zadáním úlohy jsou tyto stroje popsané.

Soutěžní úloha:

a) *(1 bod)* Všimněte si překládacích strojů B a C z příkladů ve studijním textu. Tyto dva stroje zjevně provádějí překlad „opačnými směry". Mohli bychom proto očekávat, že platí následující tvrzení:
Nechť M je libovolná (třeba i nekonečná) množina, jejímiž prvky jsou nějaké řetězce písmen a, e a i. Potom $C(B(M)) = M$. Slovně popsáno: Když vezmeme množinu M, přeložíme ji pomocí stroje B a výsledek přeložíme pomocí C, dostaneme původní množinu.
Pokud toto tvrzení skutečně platí, dokažte ho. Pokud ne, najděte protipříklad.

b) *(1 bod)* Totéž jako v předcházející části, jen M obsahuje řetězce tvořené znaky ━ a • a zajímá nás, zda musí platit $B(C(M)) = M$.

c) *(3 body)* Řekneme, že řetězec je zajímavý, jestliže obsahuje pouze písmena a a b, přičemž písmen a je dvojnásobný počet než písmen b. Nechť X je množina všech zajímavých řetězců. Tedy například $aaabab \in X$, ale $baba \notin X$.
Nechť Y je množina všech řetězců, které obsahují nejprve několik písmen a a za nimi trojnásobné množství písmen b. Tedy například $abbb \in Y$, ale $aaabab \notin Y$.
Sestrojte překládací stroj, který přeloží X na Y.

d) *(5 bodů)* Na začátku máme množinu M_1, jež obsahuje všechny takové řetězce tvořené z písmen a, b, které obsahují stejný počet písmen a jako b. Tedy například $abbbaa \in M_1$, ale $aabab \notin M_1$.
Nové množiny můžeme sestrojovat následujícími operacemi:

- překladem nějaké již sestrojené množiny nějakým strojem (můžeme použít pokaždé jiný stroj),
- sjednocením dvou již sestrojených množin,
- průnikem dvou již sestrojených množin.

Provedením co nejmenšího počtu operací sestrojte množinu G, která obsahuje právě všechny řetězce, v nichž je nejprve několik písmen a, potom stejný počet písemen b, a nakonec stejný počet písmen c. Tedy například $aabbcc \in G$, ale $abcc \notin G$, a ani $bac \notin G$.

Studijní text:

Překládací stroj přijímá na vstupu řetězec znaků. Tento řetězec postupně čte a podle předem zvolené soustavy pravidel (tedy podle svého programu) občas nějaké znaky zapíše na výstup. Když stroj zpracuje celý vstupní řetězec a úspěšně ukončí svůj výpočet, vezmeme řetězec znaků zapsaný na výstup a nazveme ho *překladem* vstupního řetězce.

Výpočet stroje nemusí být jednoznačně určen. Jinými slovy, soustava pravidel může někdy stroji umožnit, aby se rozhodl o dalším postupu výpočtu. V takovém případě se může stát, že některý řetězec bude mít více různých překladů.

Naopak, může se stát, že v určité situací se podle daných pravidel nebude moci v překladu pokračovat vůbec. V takovém případě se může stát, že některý řetězec nebude mít vůbec žádný překlad.

Formálnější definice překládacího stroje

Každý překládací stroj pracuje nad nějakou předem zvolenou konečnou množinou znaků. Tuto množinu znaků budeme nazývat *abeceda* a značit Σ. V soutěžních úlohách bude vždy Σ známa ze zadání úlohy. Abeceda nebude nikdy obsahovat znak \$, ten budeme používat k označení konce vstupního řetězce.

Stroj si může během překladu řetězce pamatovat informaci konečné velikosti. Formálně tuto skutečnost definujeme tak, že stroj se v každém okamžiku překladu nachází v jednom z *konečně mnoha* stavů. Nutnou součástí programu překládacího stroje je tedy nějaká konečná *množina stavů*, v nichž se stroj může nacházet. Tuto množinu označíme K. Kromě samotné množiny stavů je také třeba uvést, ve kterém stavu se stroj nachází na začátku každého překladu. Tento stav nazveme *počáteční stav*.

Program stroje se skládá z konečného počtu překladových pravidel. Každé pravidlo má tvar čtveřice (p, u, v, q), kde $p, q \in K$ jsou nějaké dva stavy a u, v jsou nějaké dva řetězce znaků z abecedy Σ.

Stavy p a q mohou být i stejné. Řetězec u může být tvořen jediným znakem \$. Řetězce u a v mohou být i stejné. Některý z těchto řetězců může být případně prázdný. Aby se program lépe četl, budeme místo prázdného řetězce psát symbol ε.

Překladové pravidlo má následující význam: „Když je stroj právě ve stavu p a dosud nepřečtená část vstupu začíná řetězcem u, může stroj tento řetězec ze vstupu přečíst, na výstup zapsat řetězec v a změnit svůj stav na q." Všimněte si, že pravidlo tvaru (p, ε, v, q) můžeme použít vždy, když se stroj nachází ve stavu p, bez ohledu na to, jaké znaky ještě zůstávají na vstupu.

Ještě potřebujeme stanovit, kdy překlad úspěšně skončil. V první řadě budeme požadovat, aby překládací stroj přečetl celý vstupný řetězec. Kromě toho umožníme stroji „odpovědět", zda se mu překlad podařil nebo ne. To zařídíme tak, že některé stavy stroje označíme jako *koncové stavy*. Množinu všech koncových stavů budeme značit F.

Formální definice překládacího stroje

Překládací stroj je uspořádaná pětice (K, Σ, P, q_0, F), kde Σ a K jsou *konečné* množiny, $q_0 \in K$, $F \subseteq K$ a P je *konečná* množina překladových pravidel popsaných výše. Přesněji, nechť Σ^* je množina všech řetězců tvořených znaky ze Σ, potom P je *konečná* podmnožina množiny $K \times (\Sigma^* \cup \{\$\}) \times \Sigma^* \times K$.

(Pro každé $q \in K$ budeme množinu pravidel, jejichž první složkou je q, nazývat „překladová pravidla ze stavu q“.)

Chceme-li definovat konkrétní překládací stroj, musíme uvést všech pět výše uvedených objektů.

Když už máme definován konkrétní stroj $A = (K, \Sigma, P, q_0, F)$, můžeme určit, jak tento stroj překládá konkrétní řetězec. Uvedeme nejprve formální definici a potom ji slovně vysvětlíme.

Množina *platných překladů* řetězce u překládacím strojem A je:

$$A(u) = \Big\{ \quad v \quad \Big| \quad \begin{array}{l} \exists n \geq 0\ \exists (p_1, u_1, v_1, r_1), \ldots, (p_n, u_n, v_n, r_n) \in P : \\ (\forall i \in \{1, \ldots, n-1\} : r_i = p_{i+1}) \ \wedge\ p_1 = q_0 \ \wedge\ r_n \in F \ \wedge \\ \quad \wedge\ \exists k \geq 0 : u_1 u_2 \ldots u_n = u\underbrace{\$\ldots\$}_{k} \wedge v_1 v_2 \ldots v_n = v \end{array} \Big\}.$$

Definice stanoví, kdy je řetězec v překladem řetězce u. Vysvětlíme si slovně význam jednotlivých řádků definice:

- První řádek říká, že aby se dalo u přeložit na v, musí existovat nějaká posloupnost překladových pravidel, kterou při tomto překladu použijeme. Další dva řádky popisují, jak tato posloupnost musí vypadat.
- Druhý řádek zabezpečuje, aby stavy v použitých pravidlech byly správné: První pravidlo musí být pravidlem z počátečního stavu, každé další pravidlo musí být pravidlem z toho stavu, do něhož se stroj dostal použitím předcházejícího pravidla.
  Navíc stav, v němž se bude stroj nacházet po skončení výpočtu, musí být koncový.
- Poslední řádek popisuje řetězce, které stroj při použití dotyčných překladových pravidel čte a zapisuje.
  Řetězec, který při použití těchto pravidel stroj přečte ze vstupu, musí být skutečně zadaným řetězcem u, případné může být zprava doplněn vhodným počtem znaků \$.
  Řetězec, který stroj zapíše na výstup, musí být přesně řetězcem v.

K čemu budeme používat překládací stroje?

Překládací stroje nám budou sloužit k získání překladu jedné množiny řetězců na jinou množinu řetězců. Jestliže A je překládací stroj a $M \subseteq \Sigma^*$ nějaká množina řetězců, potom překlad množiny M strojem A je množina

$$A(M) = \bigcup_{u \in M} A(u).$$

Jinými slovy, výslednou množinu $A(M)$ sestrojíme tak, že vezmeme všechny řetězce z M a pro každý z nich přidáme do $A(M)$ všechny jeho platné překlady.

Příklad 1

Mějme abecedu $\Sigma = \{0, \ldots, 9\}$. Nechť M je množina všech řetězců, které představují zápisy kladných celých čísel v desítkové soustavě. Sestrojíme překládací stroj A, pro který bude platit, že překladem této množiny M bude množina zápisů všech kladných celých čísel, která jsou dělitelná třemi.

Řešení

Nejjednodušší bude prostě vybrat z M ta čísla, která jsou dělitelná třemi. Náš překládací stroj bude kopírovat cifry ze vstupu na výstup, přičemž si bude pomocí stavu pamatovat, jaký zbytek po dělení třemi dává dosud přečtené (a zapsané) číslo. Nachází-li se po dočtení vstupu ve stavu odpovídajícím zbytku 0, přejde do koncového stavu.

Formálně A bude pětice $(K, \Sigma, P, 0, F)$, kde $K = \{0, 1, 2, end\}$, $F = \{end\}$ a překladová pravidla vypadají následovně:

$$P = \Big\{(x, y, y, z) \ \Big|\ x \in \{0, 1, 2\} \ \wedge\ y \in \Sigma \ \wedge\ z = (10x + y) \bmod 3\Big\} \ \cup\ \Big\{(0, \$, \varepsilon, end)\Big\}.$$

Příklad 2

Mějme abecedu $\Sigma = \{a, e, i, \bullet, \blacksquare\}$. Sestrojíme překládací stroj B, pro který bude platit, že překladem libovolné množiny M, která obsahuje pouze řetězce z písmen a, e a i, bude množina stejných řetězců zapsaných v morseovce (bez oddělovačů mezi znaky). Zápisy našich písmen v morseovce vypadají následovně: a je $\bullet\,\blacksquare$, e je $\bullet$ a i je $\bullet\bullet$.

Například množinu $M = \{ae, eea, ia\}$ by náš stroj měl přeložit na množinu $\{\bullet\,\blacksquare\,\bullet, \bullet\bullet\bullet\,\blacksquare\}$. (Všimněte si, že řetězce eea a ia mají v morseovce bez oddělovačů stejný zápis.)

Řešení

Překládací stroj B bude číst vstupní řetězec po znacích a vždy zapíše na výstup kód přečteného znaku.

Formálně B bude pětice $(K, \Sigma, P, \heartsuit, F)$, kde $K = \{\heartsuit\}$, $F = \{\heartsuit\}$ a překladová pravidla vypadají takto:

$$P = \{(\heartsuit, a, \bullet\,\blacksquare, \heartsuit),\ (\heartsuit, e, \bullet, \heartsuit),\ (\heartsuit, i, \bullet\bullet, \heartsuit)\}.$$

Všimněte si, že nepotřebujeme nijak zvlášť kontrolovat, zda jsme na konci vstupu. Během celého překladu je totiž stroj v koncovém stavu, takže jakmile přečte poslední znak ze vstupu, bude vytvořený překlad platný.

Příklad 3

Mějme abecedu $\Sigma = \{a, e, i, \bullet, \blacksquare\}$. Sestrojíme překládací stroj C, pro který bude platit, že překladem libovolné množiny M, která obsahuje pouze řetězce tvořené znaky $\bullet$ a $\blacksquare$, bude množina *všech* řetězců z písmen a, e a i, jejichž zápisy v morseovce (bez oddělovačů mezi znaky) jsou obsaženy v množině M. Například množinu $M = \{\bullet\blacksquare\bullet, \bullet\bullet\bullet\blacksquare\}$ by náš stroj měl přeložit na $\{ae, eea, ia\}$.

Řešení

Našemu překládacímu stroji dáme možnost rozhodnout se v každém okamžiku překladu, že bude číst kód nějakého písmena a zapíše na výstup toto písmeno. Potom každé možnosti, jak lze rozdělit vstupní řetězec na kódy písmen, bude odpovídat jeden platný překlad.

Formálně C bude pětice $(K, \Sigma, P, \diamondsuit, F)$, kde $K = \{\diamondsuit\}$, $F = \{\diamondsuit\}$ a překladová pravidla vypadají následovně:

$$P = \{(\diamondsuit, \bullet\blacksquare, a, \diamondsuit),\ (\diamondsuit, \bullet, e, \diamondsuit),\ (\diamondsuit, \bullet\bullet, i, \diamondsuit)\}.$$

Ukážeme si, jak mohl probíhat překlad řetězců z výše uvedené množiny M. Existují tyto tři možnosti:

$(\diamondsuit, \bullet\blacksquare, a, \diamondsuit), (\diamondsuit, \bullet, e, \diamondsuit)$
$(\diamondsuit, \bullet\bullet, i, \diamondsuit), (\diamondsuit, \bullet\blacksquare, a, \diamondsuit)$
$(\diamondsuit, \bullet, e, \diamondsuit), (\diamondsuit, \bullet, e, \diamondsuit), (\diamondsuit, \bullet\blacksquare, a, \diamondsuit)$

Kdybychom zkusili pro libovolný vstupní řetězec z M použít překladová pravidla v jiném pořadí – např. pro vstup $\bullet\bullet\bullet\blacksquare$ použít třikrát pravidlo $(\diamondsuit, \bullet, e, \diamondsuit)$ – nepodaří se nám dočíst vstupní řetězec až do konce.

Příklad 4

Mějme abecedu $\Sigma = \{a, b, c\}$. Nechť množina X obsahuje právě všechny řetězce, v nichž je obsažen stejný počet znaků a a b. Tedy například $abbccac \in X$, ale $cbaa \notin X$.

Nechť množina Y obsahuje právě všechny řetězce, které neobsahují žádné a, neobsahují podřetězec bc, a písmen c je dvakrát více než písmen b. Tedy například $ccccbb \in Y$, ale $cccbcb \notin Y$ a $acacba \notin Y$.

Sestrojíme překládací stroj D, který přeloží X na Y.

Řešení

Budeme překládat jenom některé vhodné řetězce z množiny X. Budou to ty řetězce, které neobsahují žádné c a všechna a v nich předcházejí všem znakům b. Takovýto řetězec přeložíme tak, že nejprve každé a přepíšeme na cc, a potom zkopírujeme na výstup všechna b. Tedy například překladem slova $aabb$ bude slovo $ccccbb$.

Formálně D bude pětice $(K, \Sigma, P, \textbf{čti-a}, F)$, kde $K = \{\textbf{čti-a}, \textbf{čti-b}\}$, $F = \{\textbf{čti-b}\}$ a překladová pravidla vypadají takto:

$$P = \{(\textbf{čti-a}, a, cc, \textbf{čti-a}),\ (\textbf{čti-a}, \varepsilon, \varepsilon, \textbf{čti-b}),\ (\textbf{čti-b}, b, b, \textbf{čti-b})\}.$$

Proč tento překládací stroj funguje? Když vstupní řetězec obsahuje nějaké písmeno c, při jeho překládání se u prvního výskytu c náš stroj zastaví. Proto takové řetězce nemají žádný platný překlad. Podobně nemají platný překlad řetězce, v nichž není dodrženo pořadí písmen a a b. Po přečtení nějaké posloupnosti písmen a přejde stroj pomocí druhého pravidla do stavu **čti-b**, a pokud se poté ještě objeví na vstupu a, stroj se zastaví.

Platné překlady tedy existují skutečně pouze pro slova výše popsaného tvaru. Je zjevné, že překladem každého z nich získáme nějaký řetězec z Y, takže $D(X) \subseteq Y$. Naopak, vybereme-li si libovolné slovo z Y, snadno najdeme slovo z X, které se na něj přeloží. Proto také $Y \subseteq D(X)$, takže $Y = D(X)$.