Teoretická informatika

Zdeněk Sawa

Katedra informatiky, FEI,
Vysoká škola báňská – Technická univerzita Ostrava
17. listopadu 2172/15, Ostrava-Poruba 708 00
Česká republika

24. září 2023

Přednášející

Jméno: doc. Ing. Zdeněk Sawa, Ph.D.

E-mail: zdenek.sawa@vsb.cz

Místnost: EA413

Web: http://www.cs.vsb.cz/sawa/ti

Na těchto stránkách najdete:

- Informace o předmětu
- Slidy z přednášek
- Zadání příkladů na cvičení
- Učební text (starší, nepokrývá některá témata)
- Aktuální informace
- Odkaz na stránku s animacemi

V rámci MS Teams (tým „*Teoretická informatika 2023/24*“):

- některé další učební texty
- podklady pro referáty

Požadavky

- **Zápočet** (35 bodů):
  - **Referát** (15 bodů) — je nutné získat minimálně 5 bodů

    možnost opravy za maximálně 10 bodů,
    při opravě je nutné získat minimálně 1 bod
  - **Zápočtová písemka** (20 bodů) — je nutné získat minimálně 10 bodů

    bude možnost opravy — maximálně za 17 bodů,
    nutné minimum je stále 10 bodů

  Celkově je třeba z referátu a zápočtové písemky získat v součtu minimálně 15 bodů.

- **Zkouška** (65 bodů):
  - bude probíhat písemnou formou
  - je nutné získat minimálně 25 bodů

Teoretická informatika

Teoretická informatika — vědní obor na pomezí mezi informatikou a matematikou

- zkoumání obecných otázek týkajících se algoritmů a výpočtů
- zkoumání různých formalismů pro popis algoritmů
- zkoumání různých prostředků pro popis syntaxe a sémantiky formálních jazyků (zejména s důrazem na programovací jazyky)
- matematický přístup k analýze a řešení problémů (dokazování obecně platných matematických tvrzení týkajících se algoritmů)

Teoretická informatika

Příklady některých typických otázek studovaných v teoretické informatice:

- Je možné daný problém řešit pomocí nějakého algoritmu?
- Pokud je možné daný problém řešit pomocí algoritmu, jaká je výpočetní složitost tohoto algoritmu?
- Existuje pro daný problém nějaký efektivní algoritmus, který ho řeší?
- Jak se přesvědčit o tom, že daný algoritmus je skutečně korektním řešením daného problému?
- Jaké instrukce musí umět vykonat stroj, který by mohl provádět daný algoritmus?

Algoritmy a problémy

Algoritmus — mechanický postup, jak něco spočítat (může být vykonáván počítačem)

Algoritmy slouží k řešení různých **problémů**.

Příklad algoritmického problému:

Vstup: Přirozená čísla a a b.

Výstup: Přirozené číslo c takové, že $c = a + b$.

Algoritmy a problémy

Algoritmus — mechanický postup, jak něco spočítat (může být vykonáván počítačem)

Algoritmy slouží k řešení různých **problémů**.

Příklad algoritmického problému:

Vstup: Přirozená čísla a a b.

Výstup: Přirozené číslo c takové, že $c = a + b$.

Konkrétní vstup nějakého problému se nazývá **instance** problému.

Příklad: Instancí výše uvedeného problému je například dvojice čísel 728 a 34.

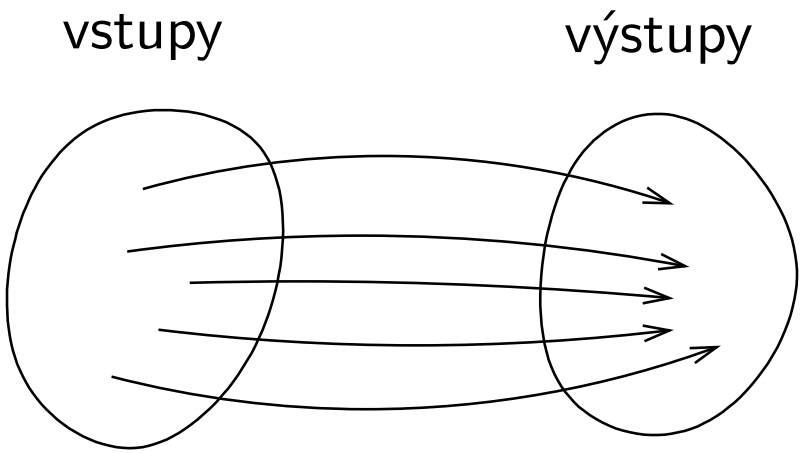
Výstupem pro tuto instanci je číslo 762.

Problémy

Problém

V zadání **problému** musí být určeno:

- co je množinou možných vstupů
- co je množinou možných výstupů
- jaký je vztah mezi vstupy a výstupy



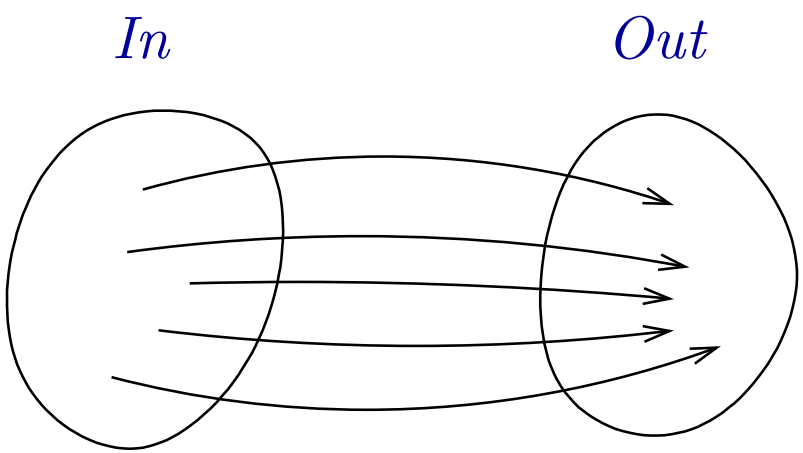
Problémy

Problém

Formálně tedy můžeme **problém** definovat jako trojici (In, Out, R), kde:

- In je množina možných vstupů
- Out je množina možných výstupů
- $R \subseteq In \times Out$ je relace přiřazující každému vstupu možné odpovídající výstupy. Tato relace musí splňovat

$$\forall x \in In : \exists y \in Out : R(x, y).$$



Příklady algoritmických problémů

Problém „Třídění“

Vstup: Sekvence prvků $a_1, a_2, \ldots, a_n$.

Výstup: Prvky sekvence $a_1, a_2, \ldots, a_n$ seřazené od nejmenšího po největší.

Příklad instance problému a jí odpovídajícího výstupu:

- Vstup: $8, 13, 3, 10, 1, 4$
- Výstup: $1, 3, 4, 8, 10, 13$

Poznámka: Pro daný problém často existuje celá řada různých algoritmů, které jej korektně řeší, např. pro problém třídění:

- insertion sort, selection sort, bubble sort, merge sort, quicksort, heapsort, ...

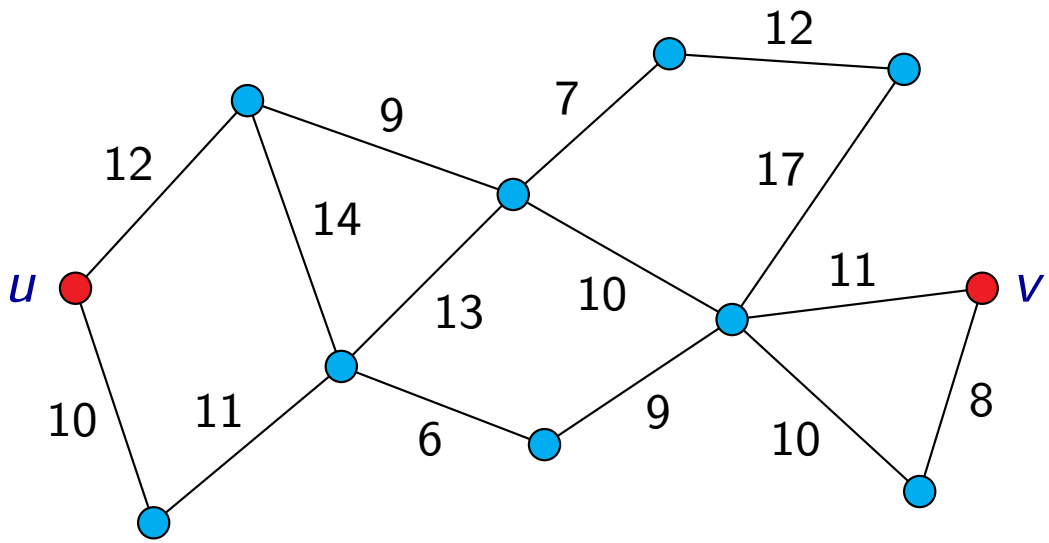
Příklady algoritmických problémů

Problém „Hledání nejkratší cesty v (neorientovaném) grafu“

Vstup: Neorientovaný graf $G = (V, E)$ s ohodnocením hran a dvojice vrcholů $u, v \in V$.

Výstup: Nejkratší cesta z vrcholu u do vrcholu v. (Nebo informace, že žádná taková cesta neexistuje.)

Příklad:



Algoritmy a problémy

Algoritmus **řeší** daný problém pokud:

- Se pro každý vstup po konečném počtu kroků zastaví.
- Pro každý vstup vydá správný výstup.

Korektnost algoritmu — ověření toho, že daný algoritmus skutečně řeší daný problém

Výpočetní složitost algoritmu:

- **časová složitost** — jak závisí doba výpočtu na velikosti vstupu
- **prostorová** (nebo též **paměťová**) **složitost** — jak závisí množství použité paměti na velikosti vstupu

Formální jazyky

Oblast teoretické informatiky zabývající se otázkami týkajícími se **syntaxe**.

- **Abeceda** — množina **symbolů** (nebo též **znaků**)
- **Slovo** — sekvence symbolů z určité abecedy
- **Jazyk** — množina slov

Slova a jazyky se v informatice objevují na mnoha místech:

- Reprezentace vstupních a výstupních dat
- Reprezentace kódu programů
- Manipulace s řetězci znaků nebo se soubory
- ...

Formální jazyky

Formálně:

- **Abeceda** — libovolná neprázdná konečná množina **symbolů** (**znaků**)
  Příklad: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$
- **Slovo** — libovolná konečná posloupnost symbolů z dané abecedy
  Příklad: $\mathtt{cabcbba}$

Množina všech slov nad abecedou Σ se označuje zápisem Σ^*.

Poznámka: Speciálním případem slova je prázné slovo (tj. slovo délky 0). Pro označení prázného slova budeme používat symbol ε.

Formální jazyky

Definice

(Formální) jazyk L v abecedě Σ je nějaká libovolná podmnožina množiny Σ^*, tj. $L \subseteq \Sigma^*$.

Příklad: Předpokládejme, že $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

- Jazyk $L_1 = \{\mathtt{aab}, \mathtt{bcca}, \mathtt{aaaaa}\}$
- Jazyk $L_2 = \{w \in \Sigma^* \mid \text{počet výskytů symbolů } \mathtt{b} \text{ ve slově } w \text{ je sudý}\}$

Kódování vstupu a výstupu

U algoritmických problémů často předpokládáme, že vstupy i výstupy jsou kódovány jako slova v nějaké abecedě Σ, tj. $In = Out = \Sigma^*$.

Různé jiné objekty (čísla, posloupnosti čísel, grafy, ...) pak zapisujeme (kódujeme) jako slova v této abecedě.

Příklad: Například u problému „Třídění“ bychom mohli zvolit jako abecedu $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ,\}$.

Vstupem by pak mohlo být například slovo

$$826,13,3901,128,562$$

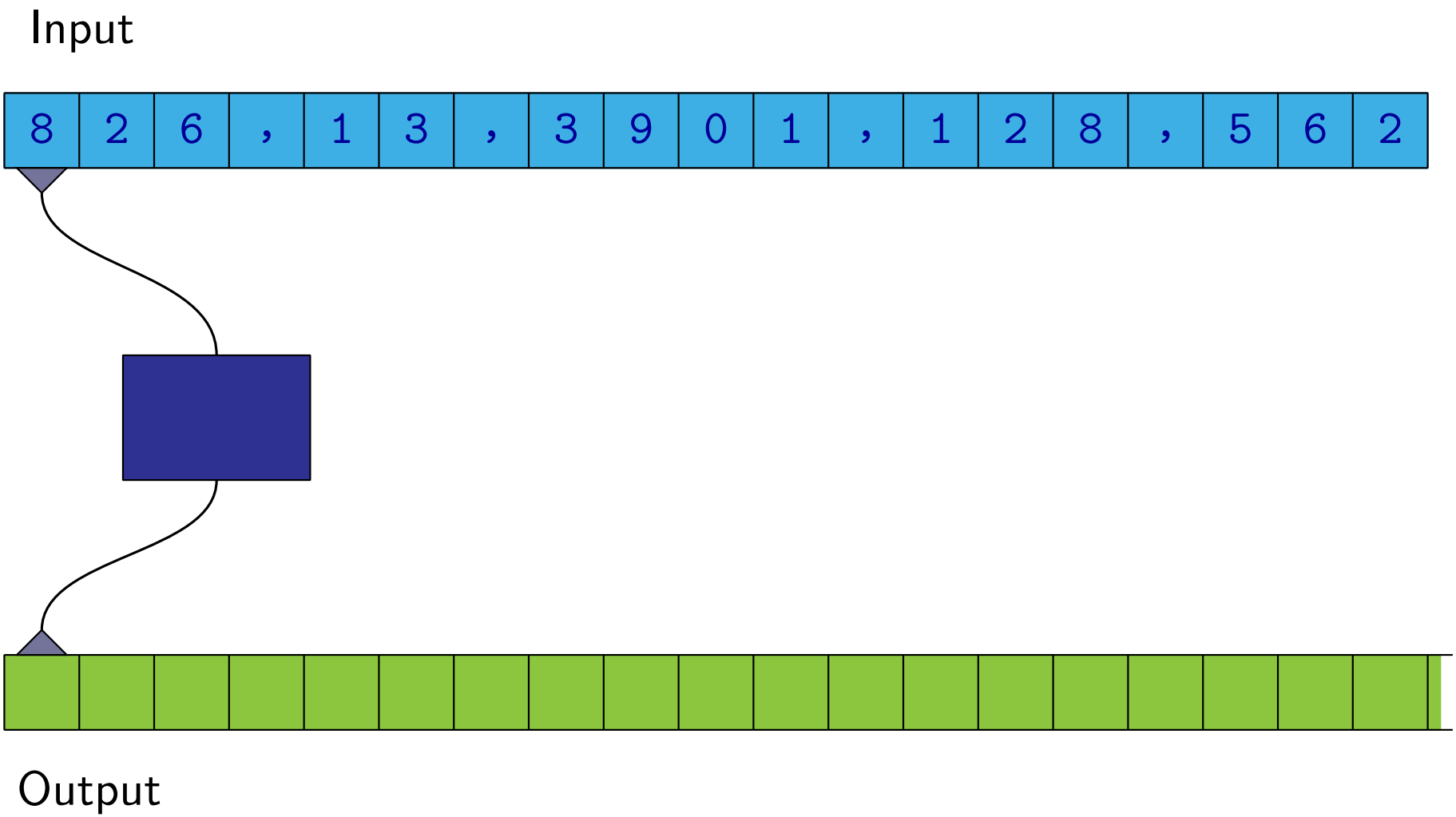
a výstupem slovo

$$13,128,562,826,3901$$

Poznámka: Ne každé slovo ze Σ^* musí reprezentovat nějaký vstup. Kódování bychom ale měli zvolit tak, abychom byli schopni snadno poznat ta slova, která nějaký vstup reprezentují.

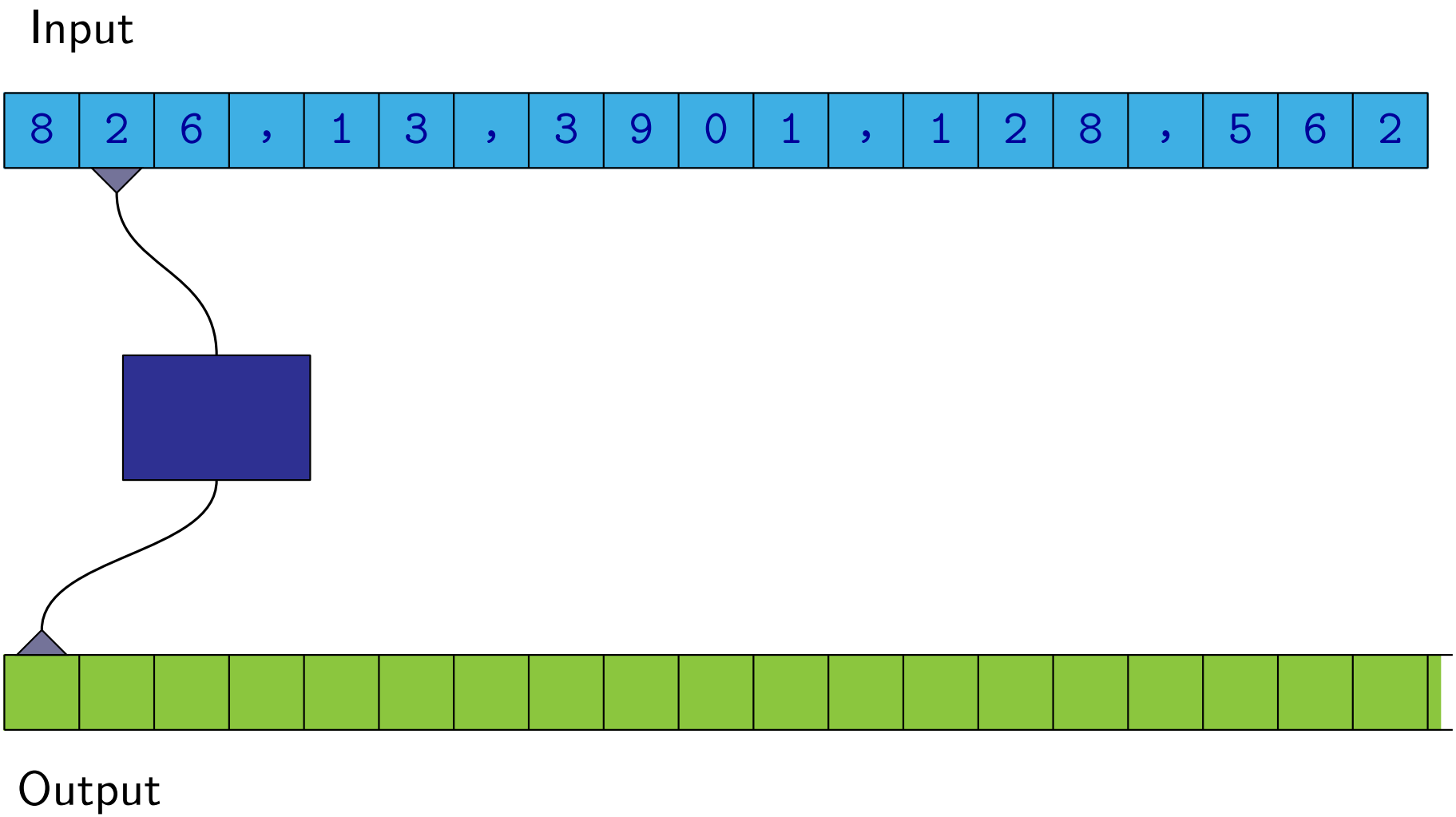
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



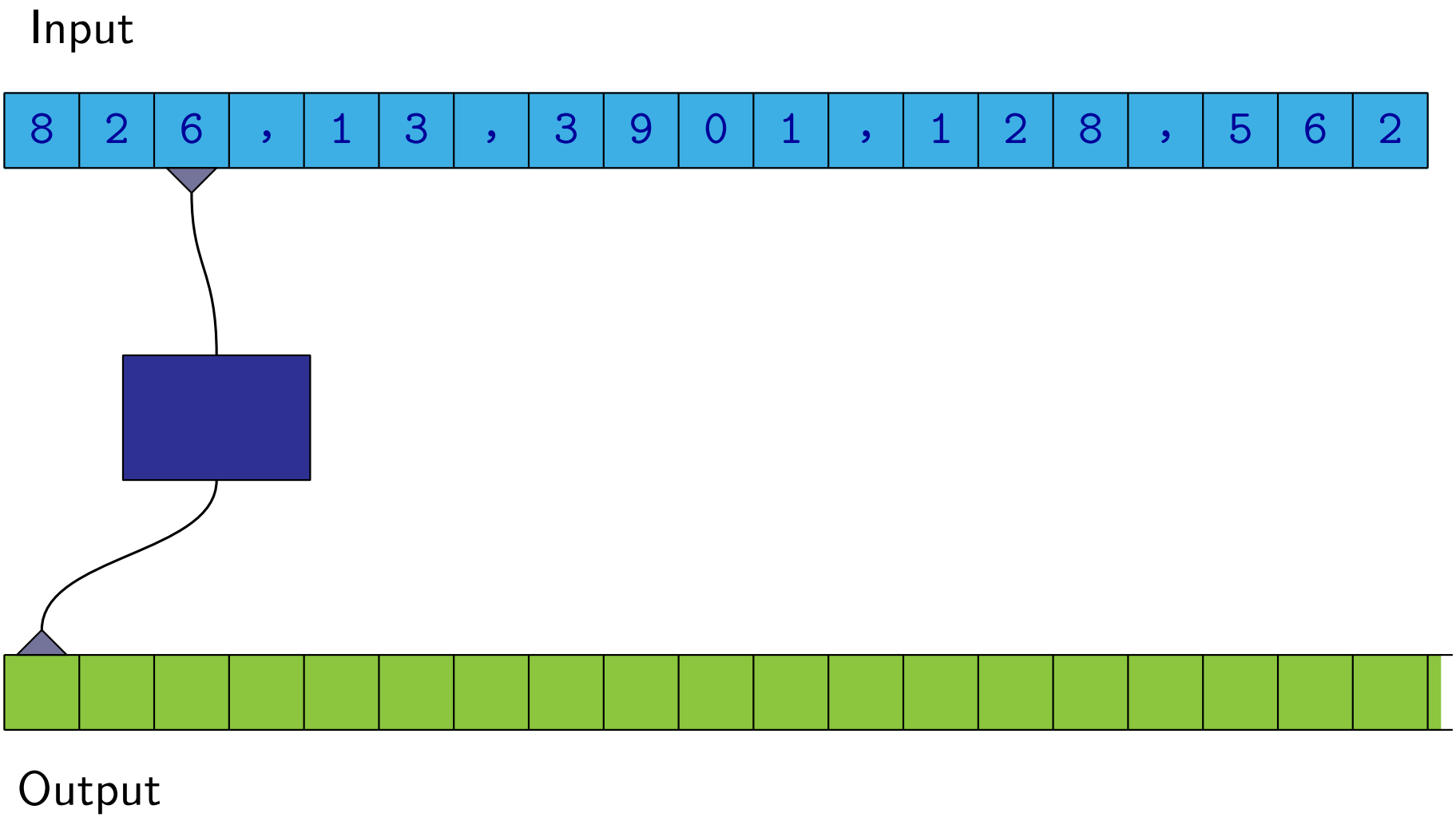
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



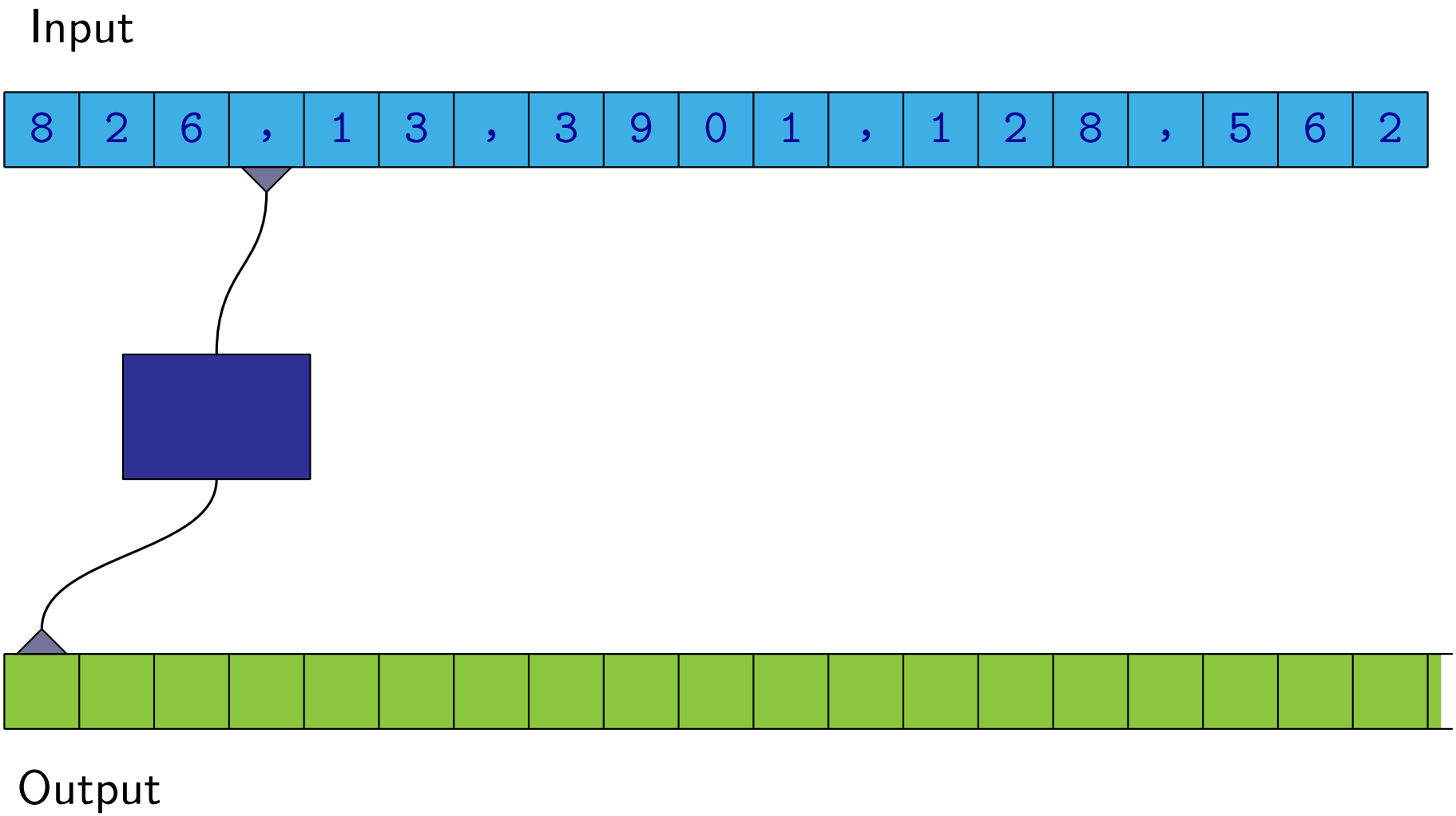
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



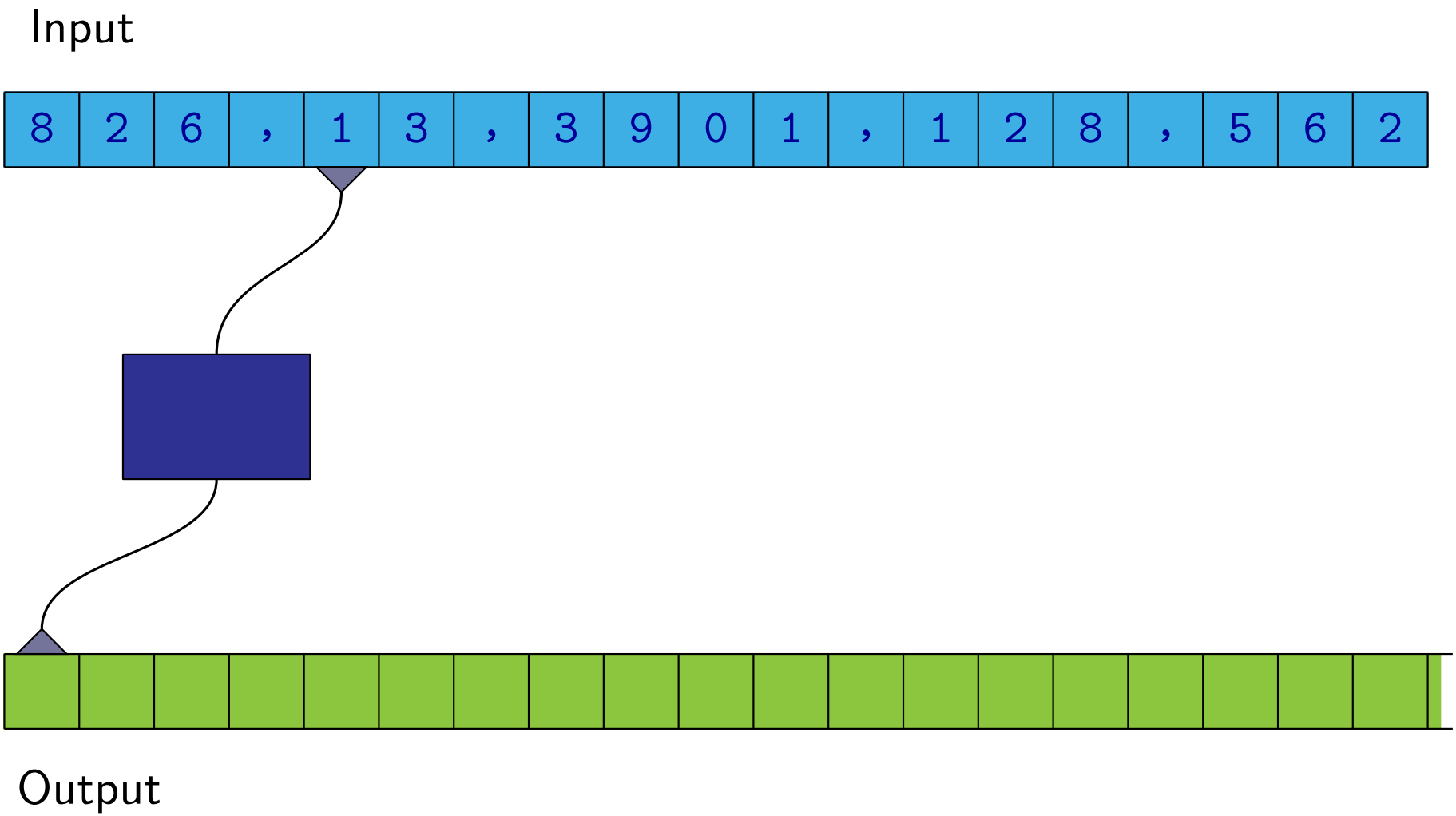
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



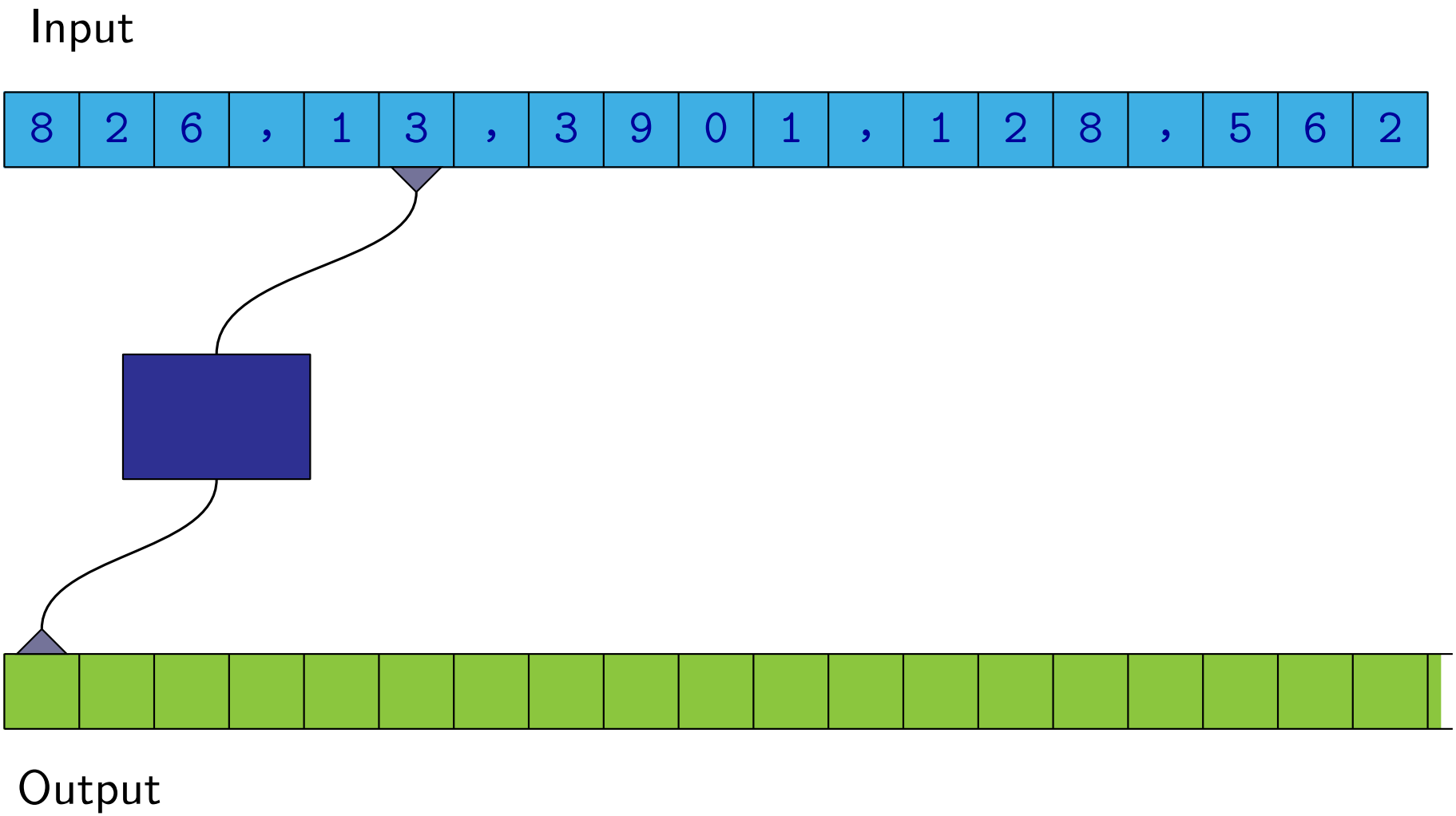
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



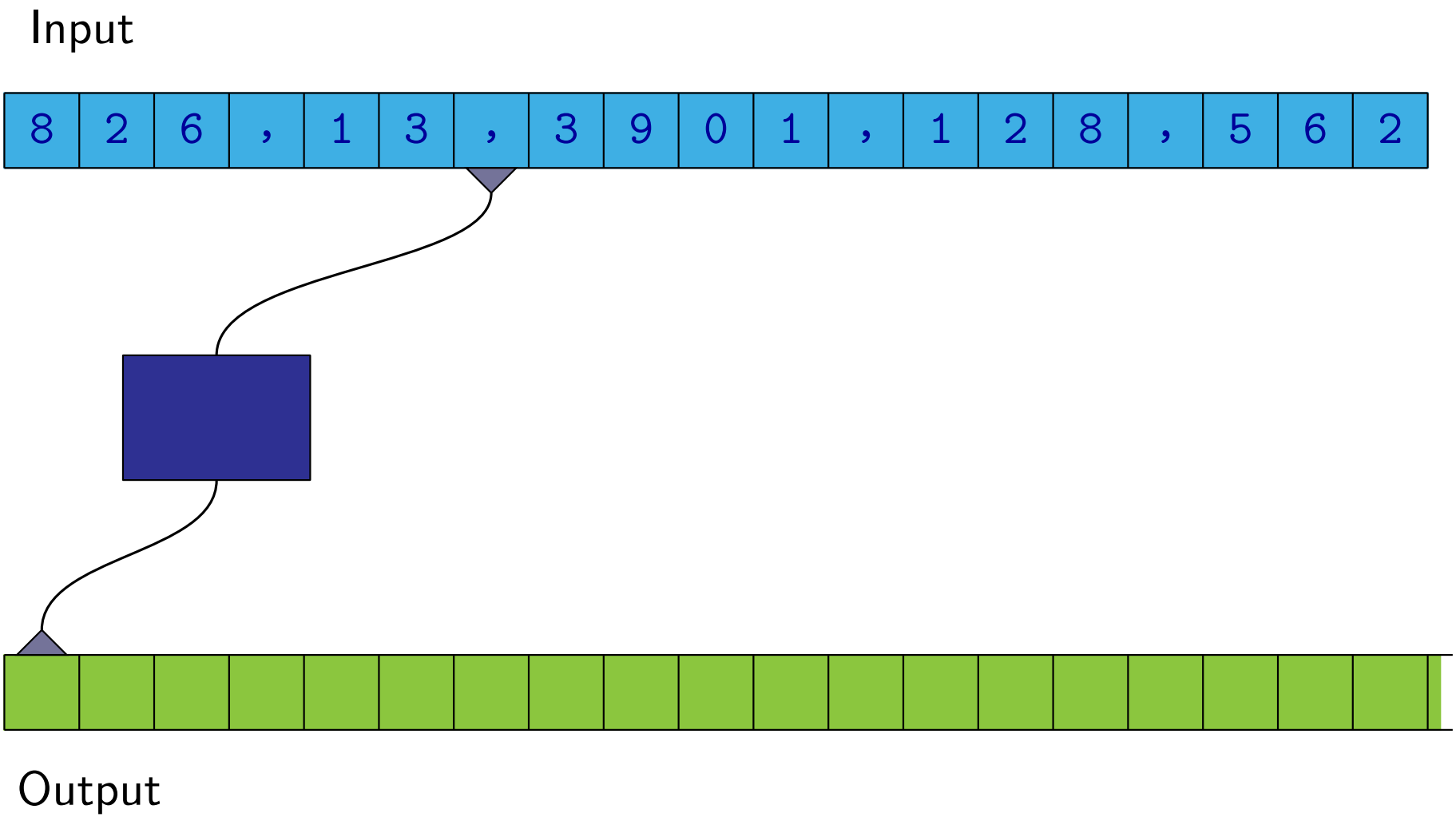
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



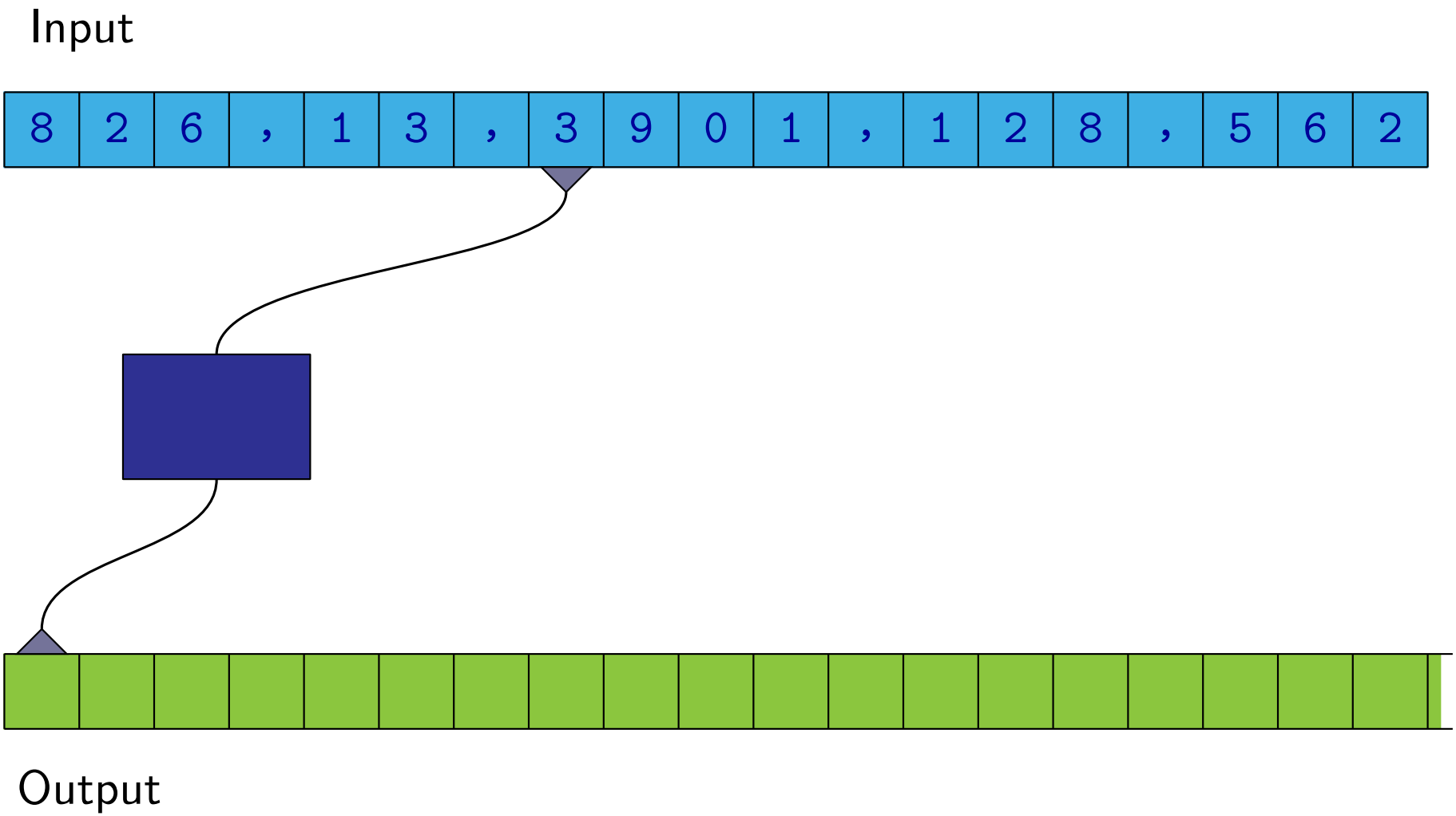
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



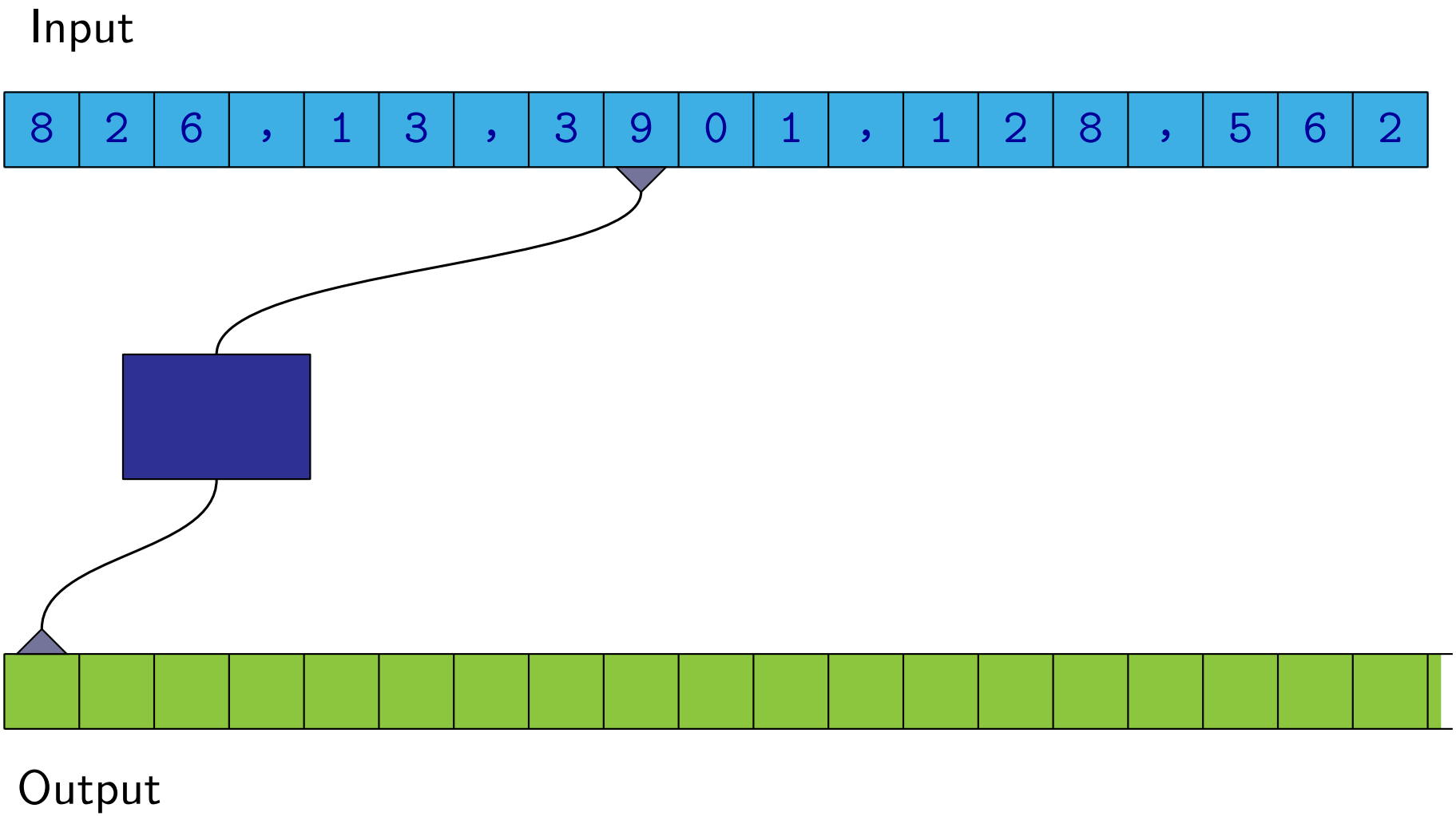
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



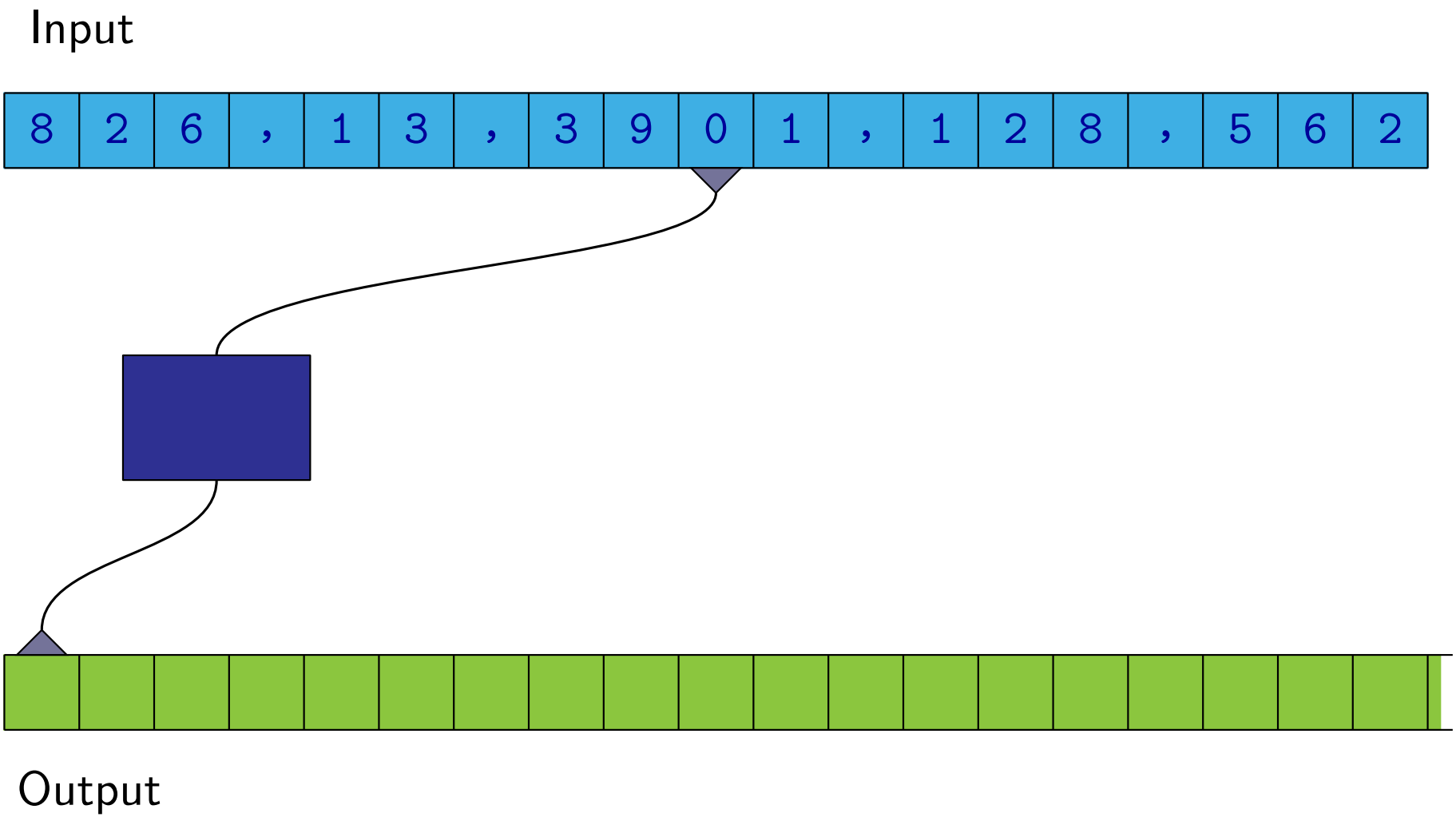
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



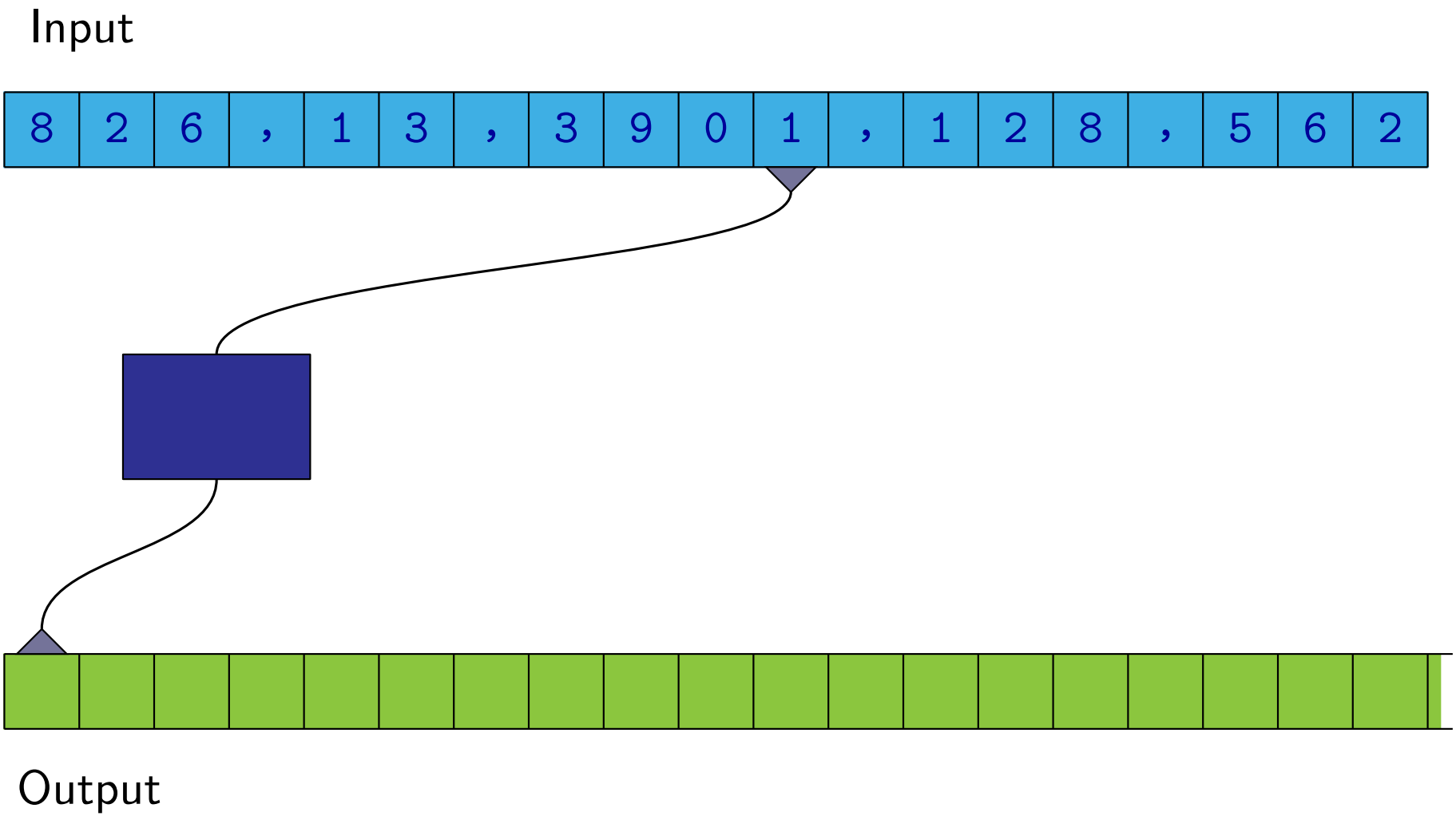
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



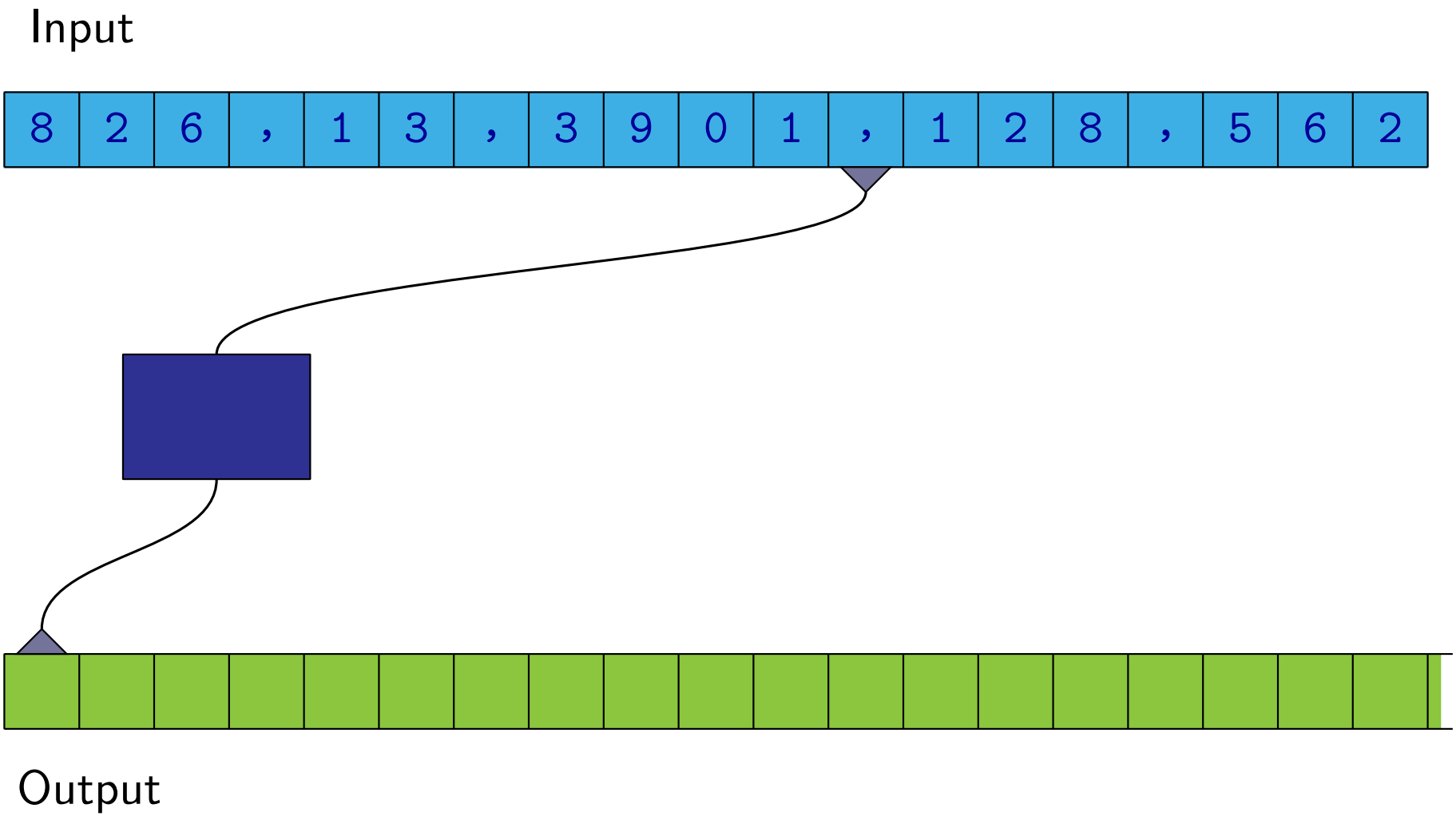
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



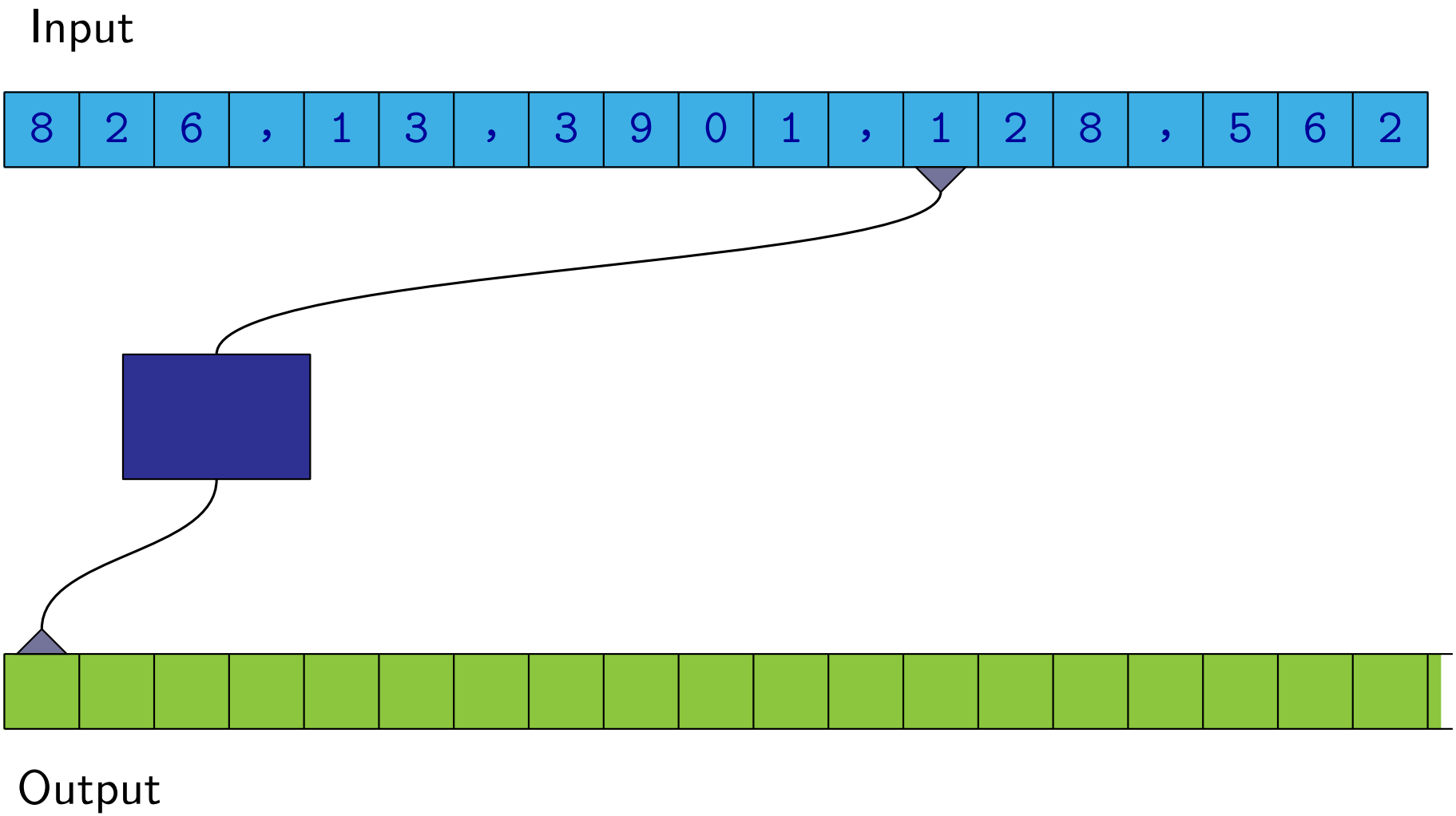
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



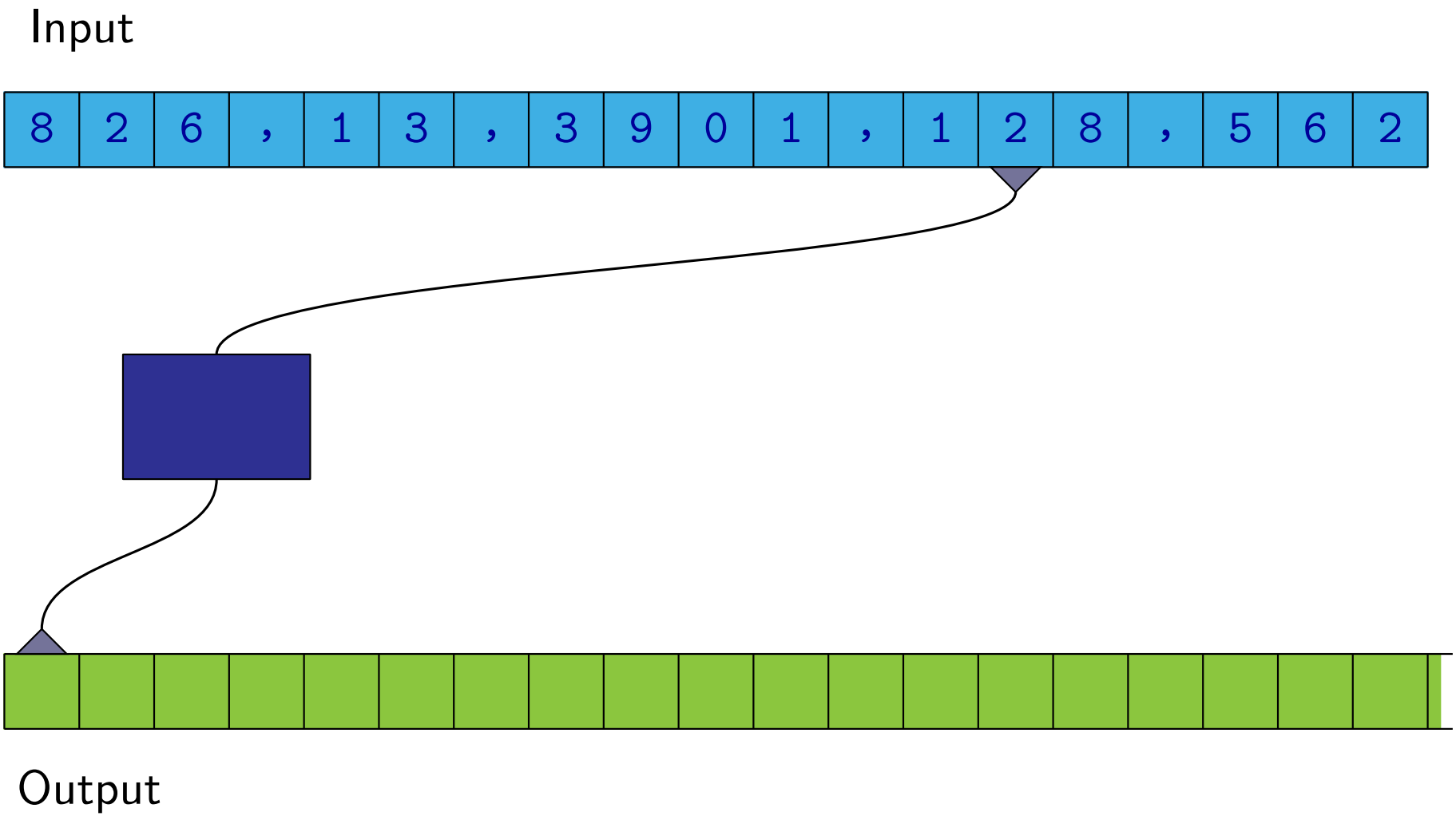
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



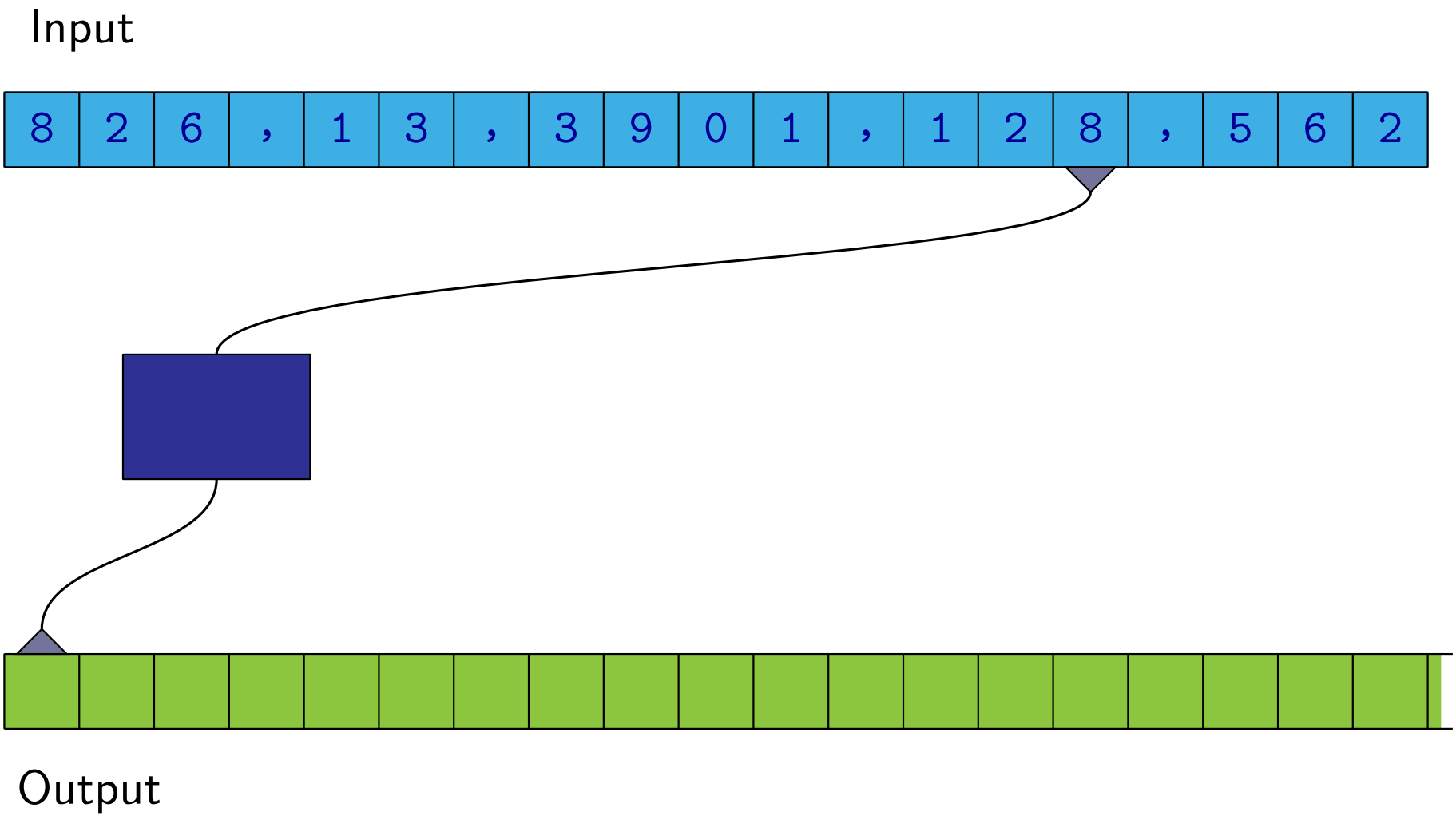
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



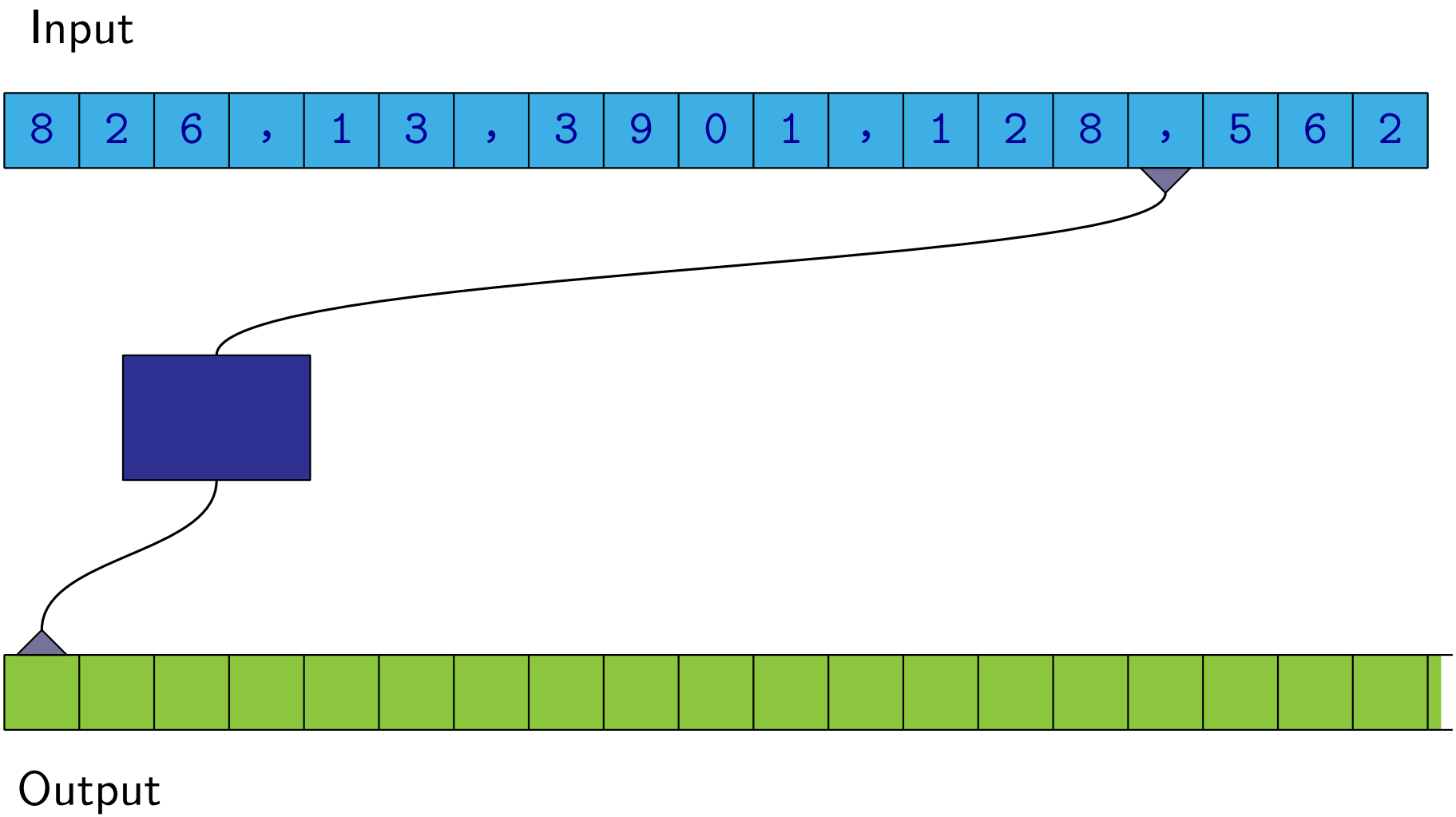
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



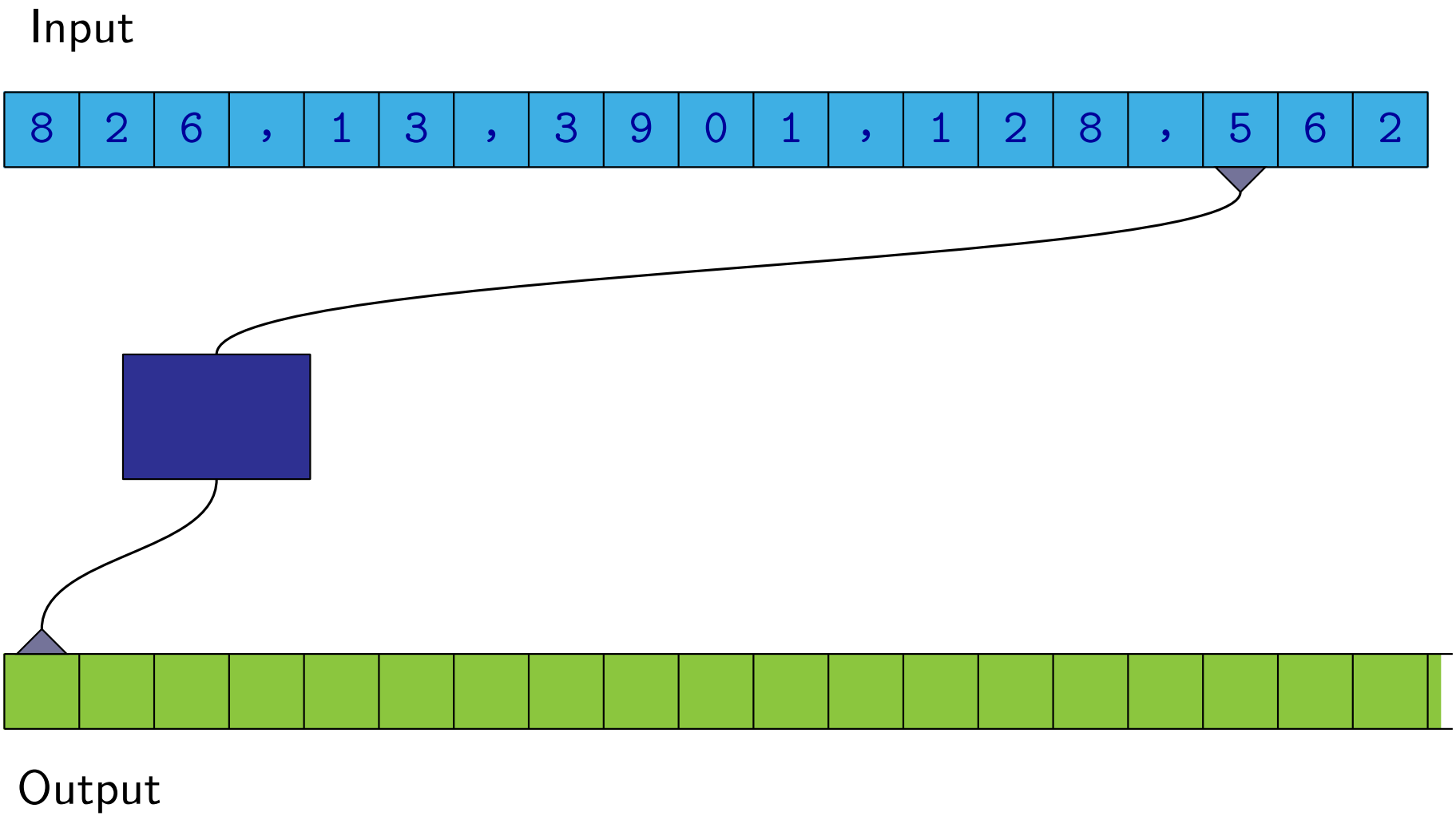
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



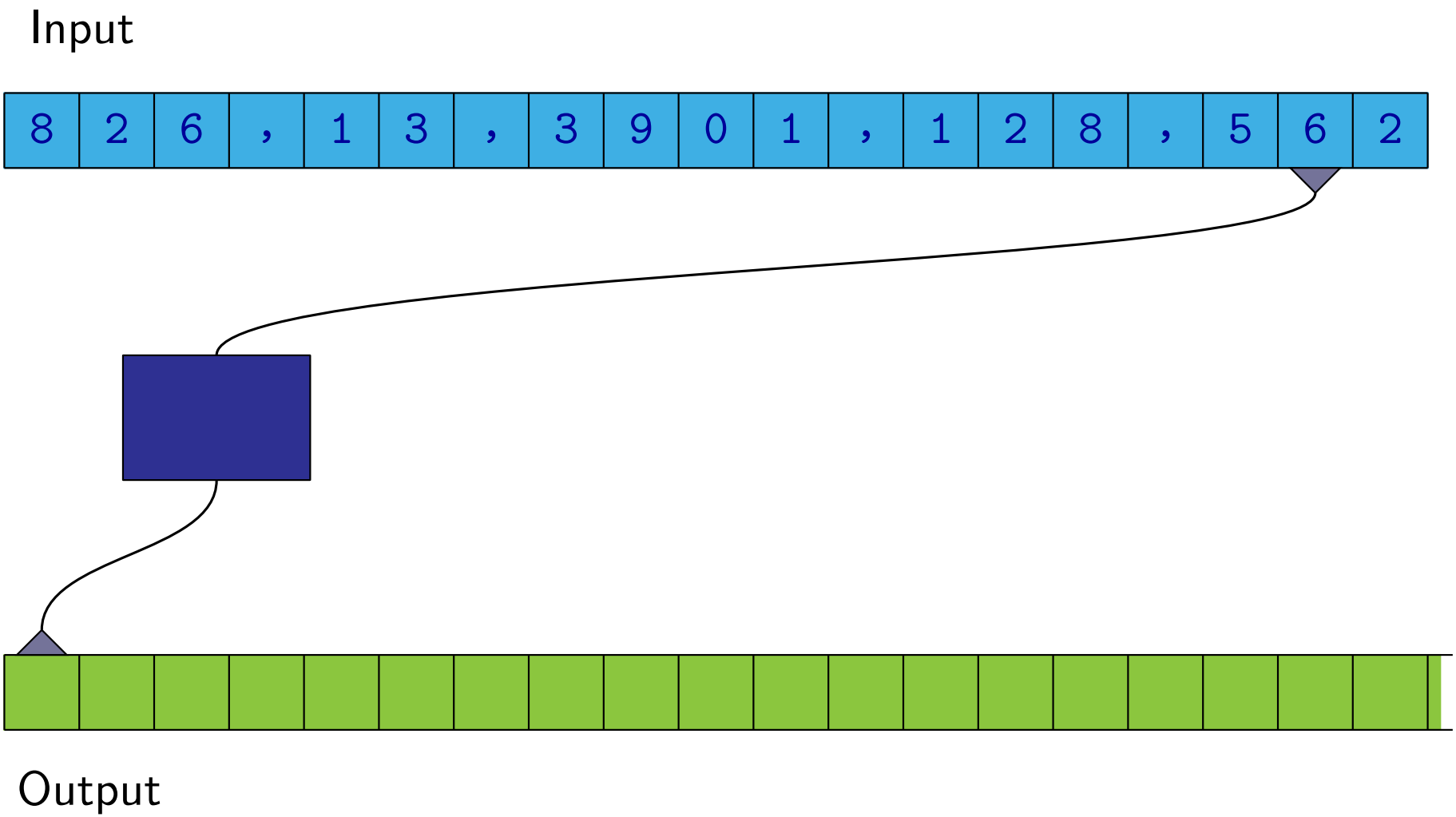
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



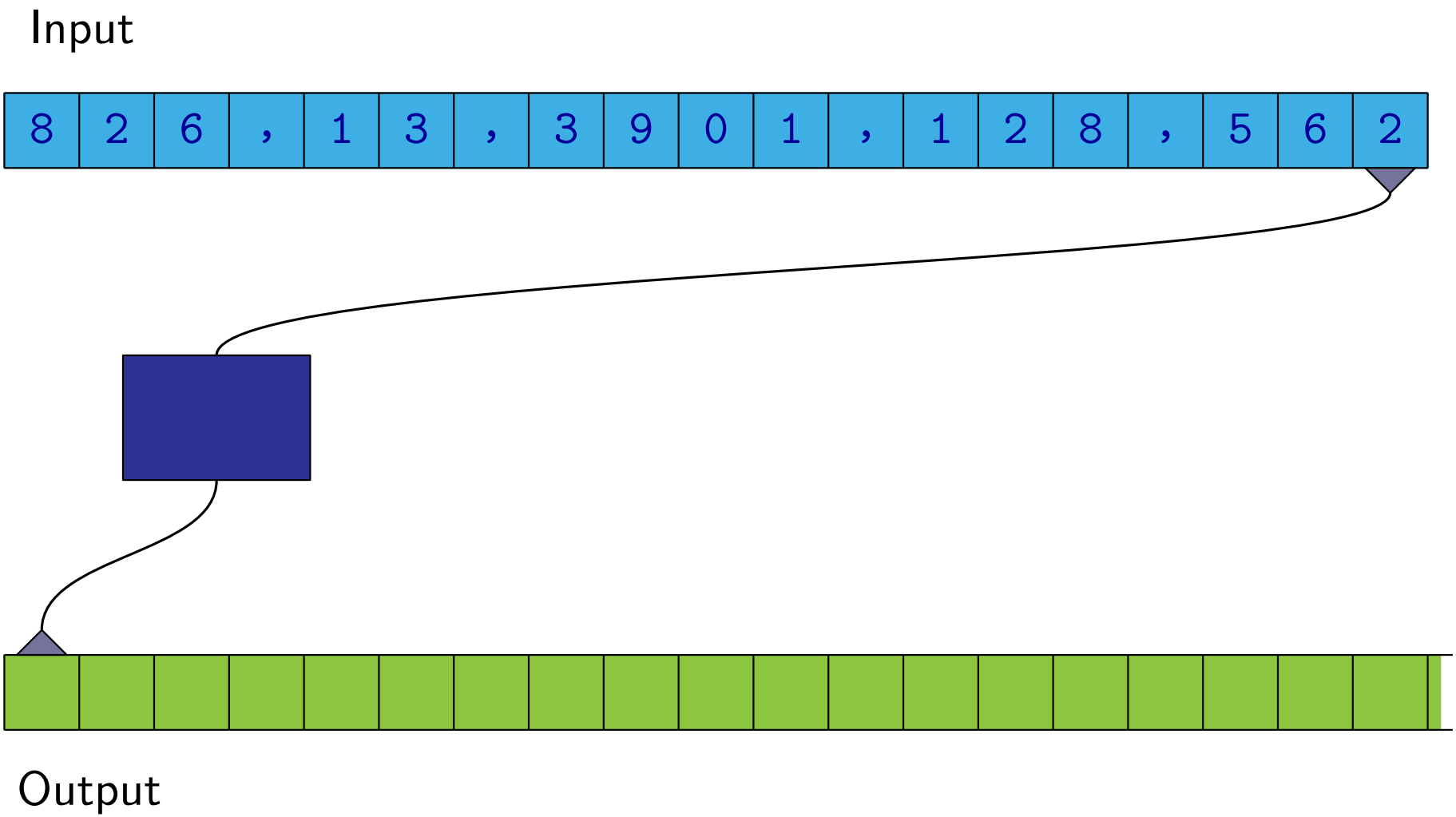
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



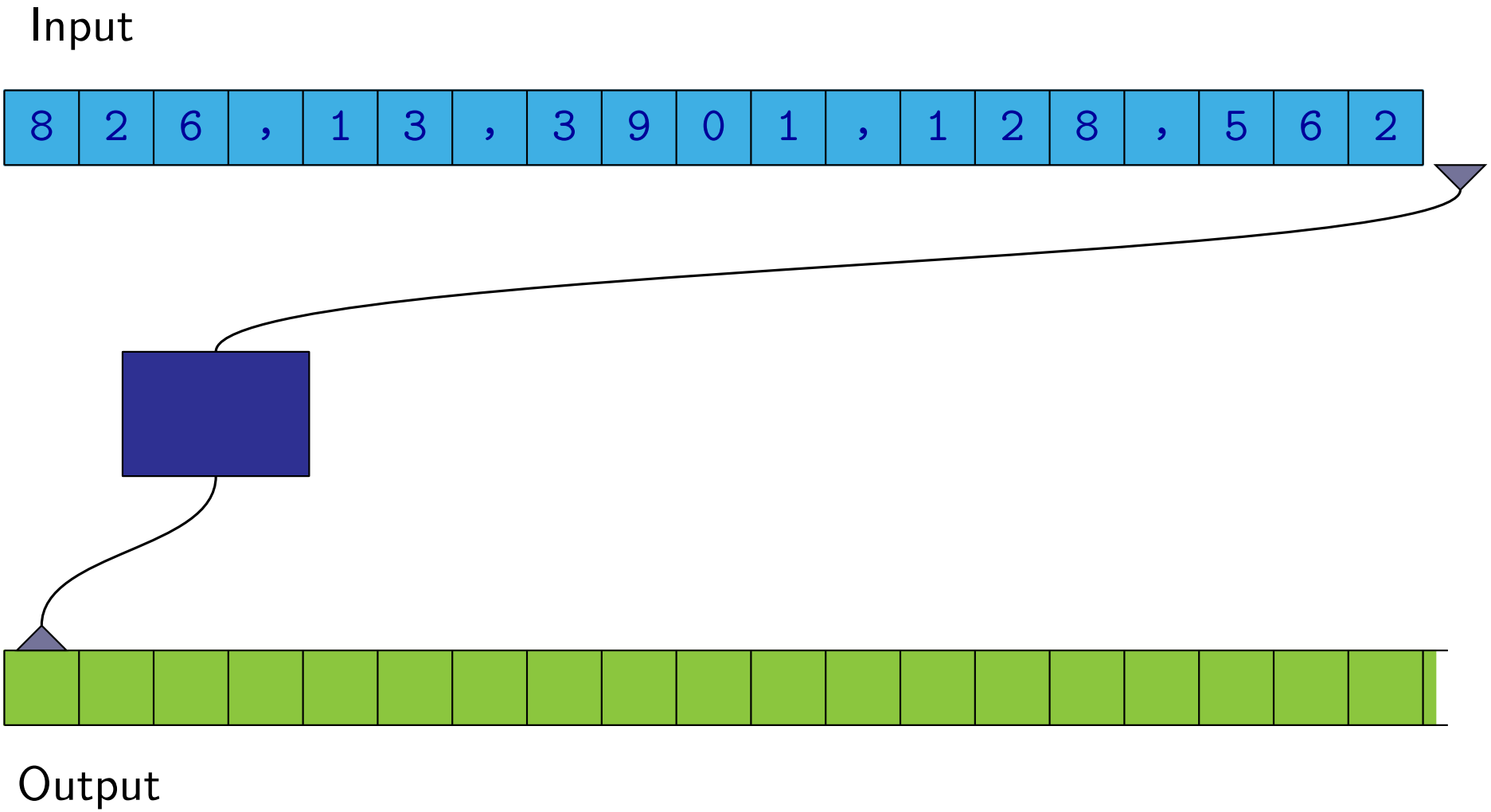
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



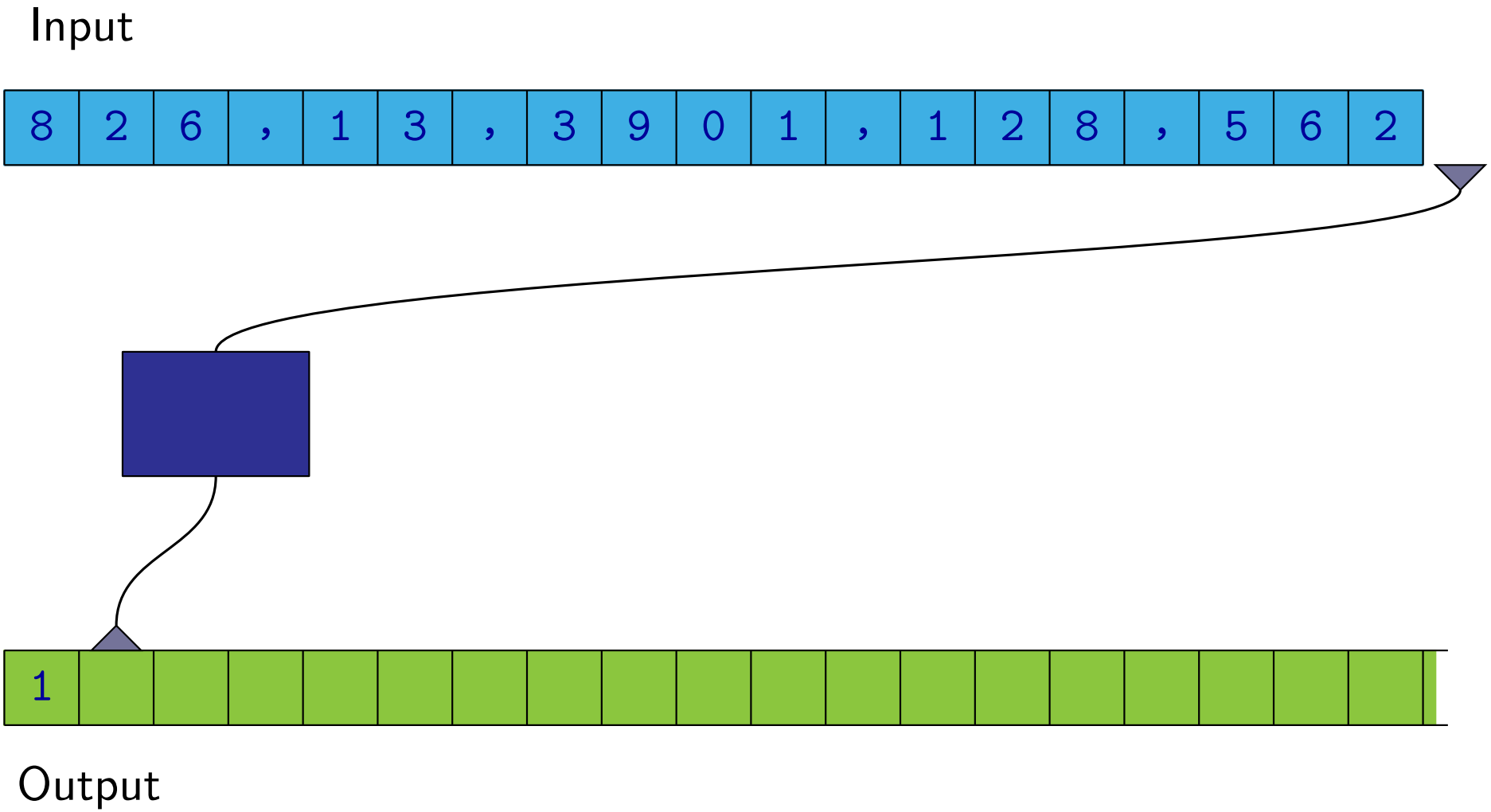
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



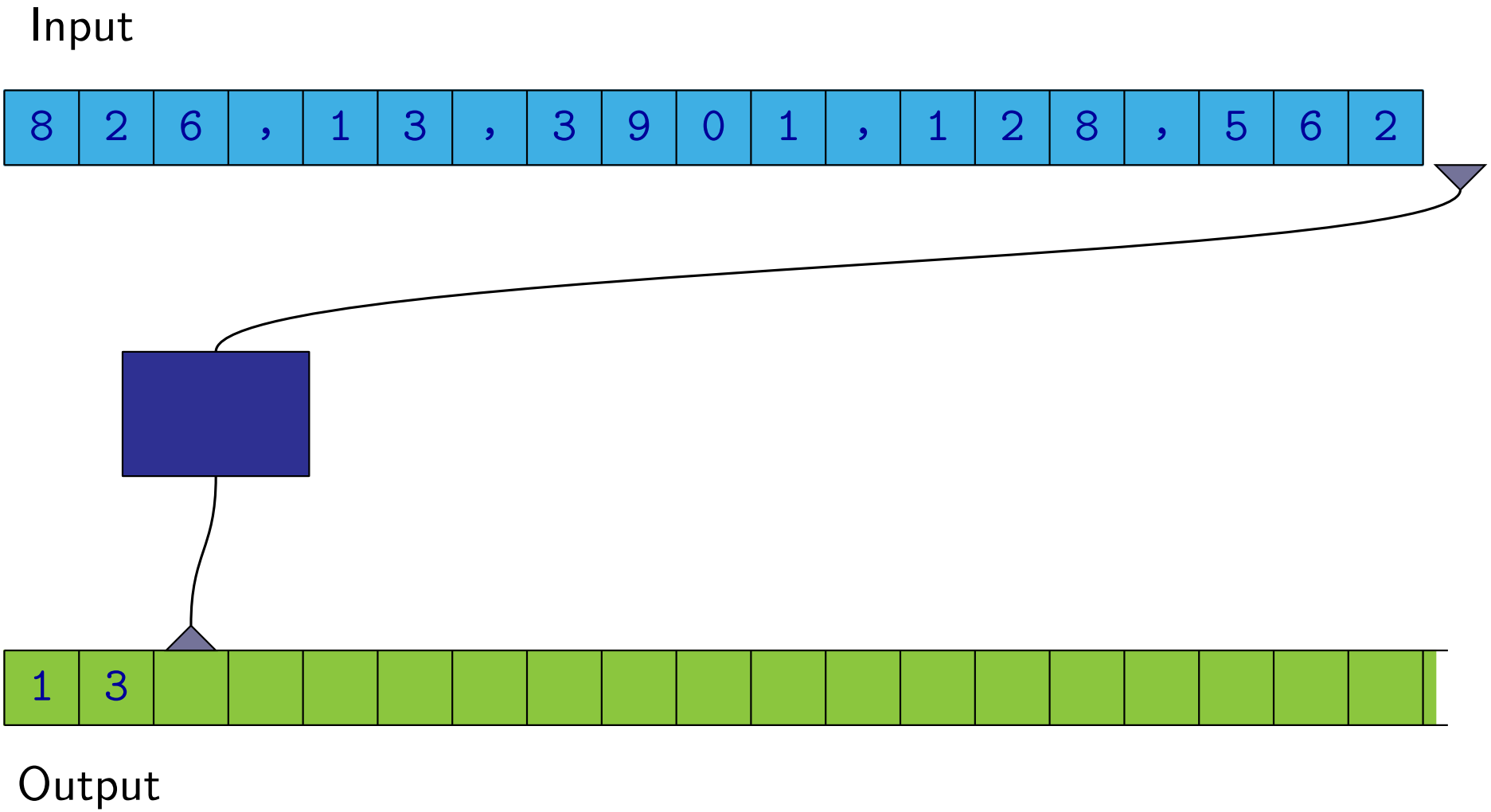
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



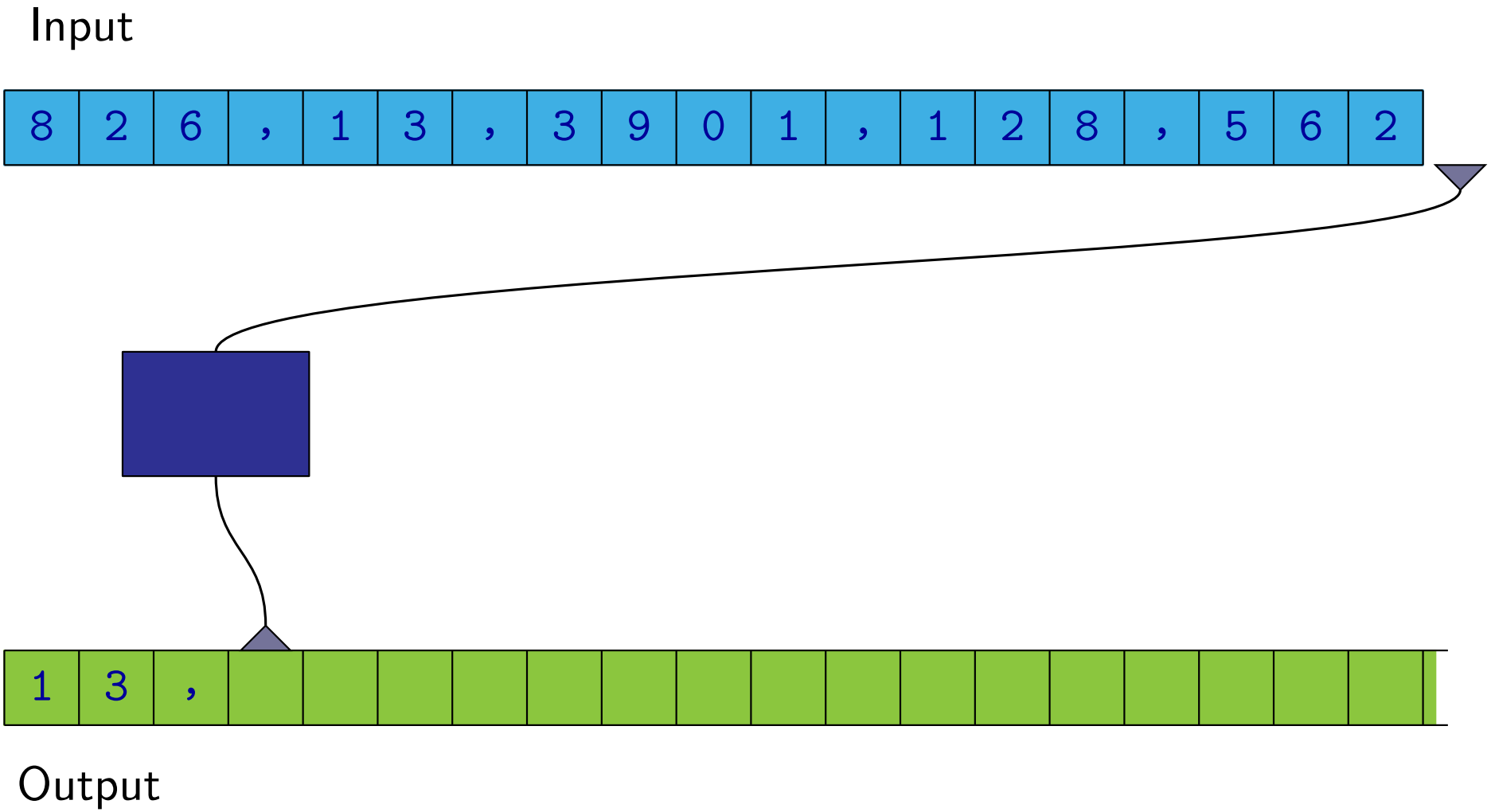
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



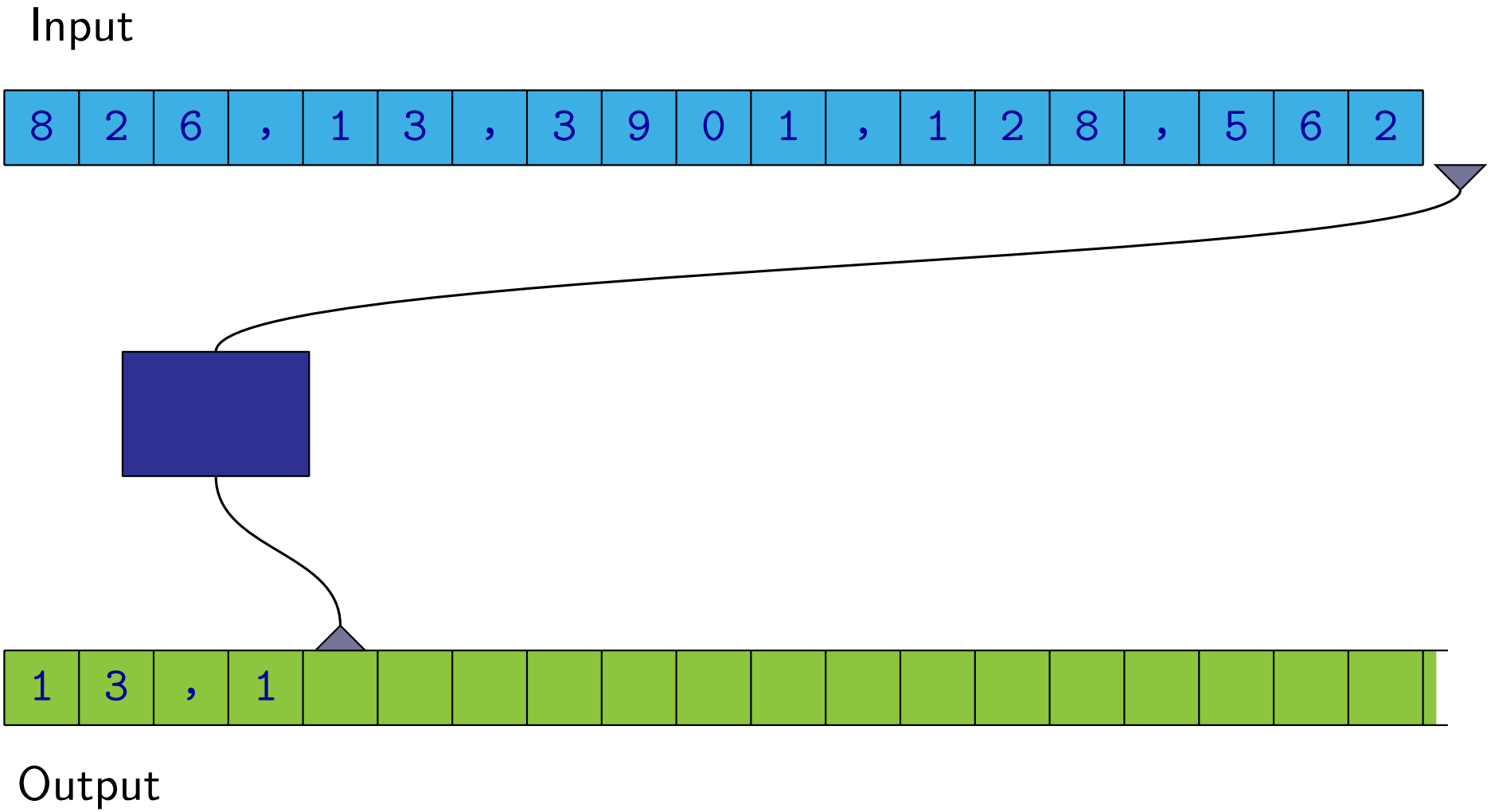
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



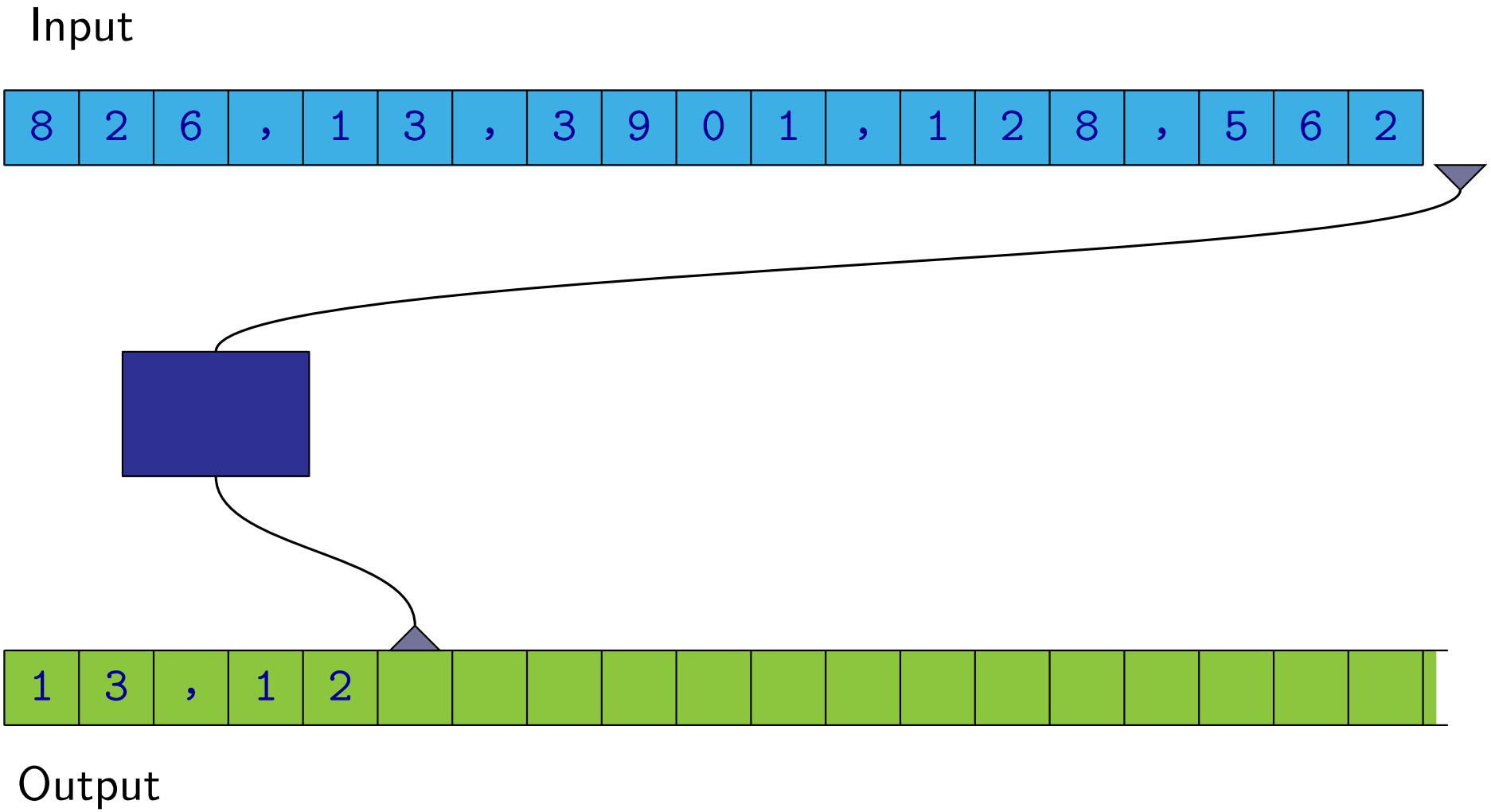
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



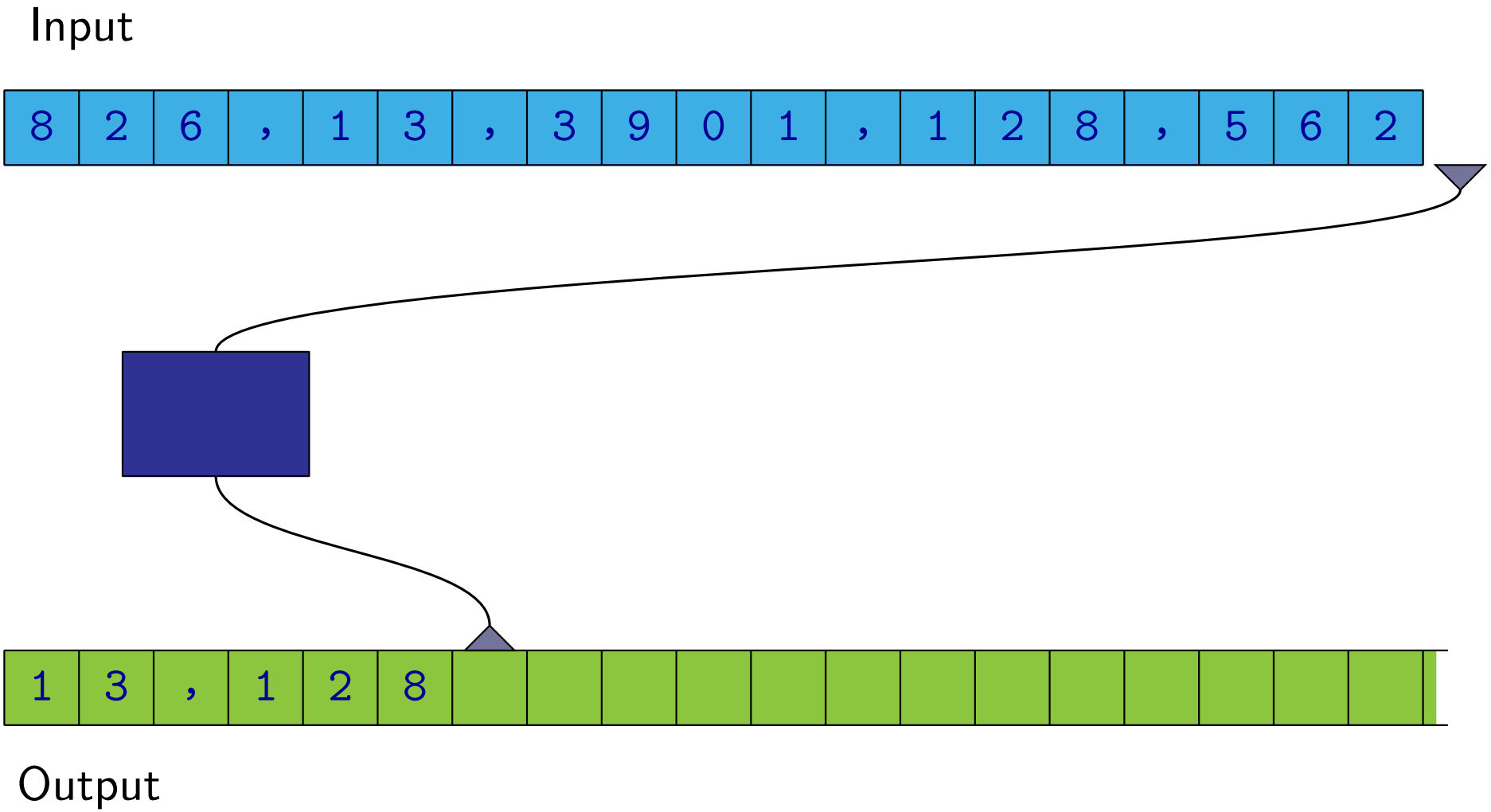
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



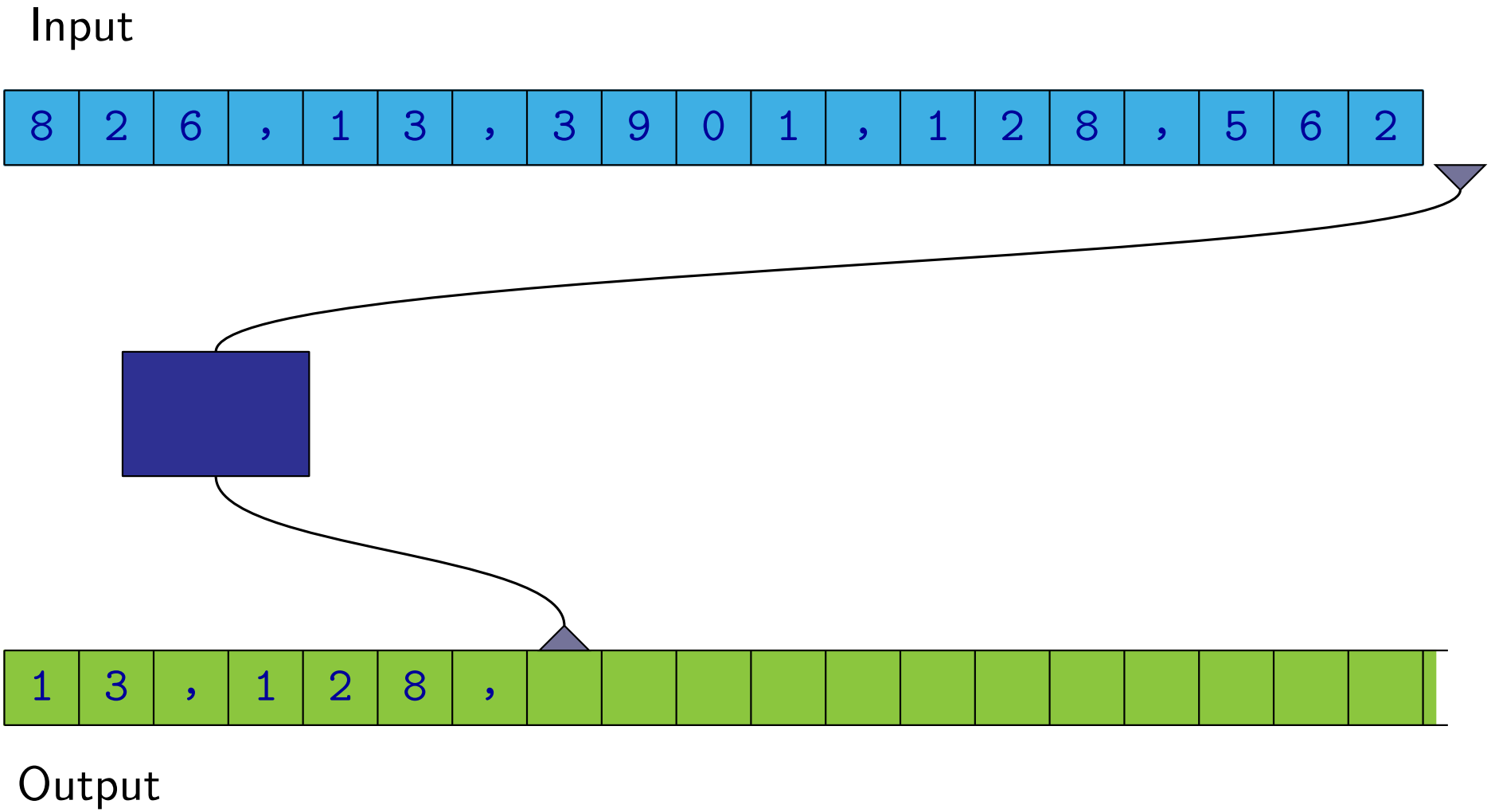
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



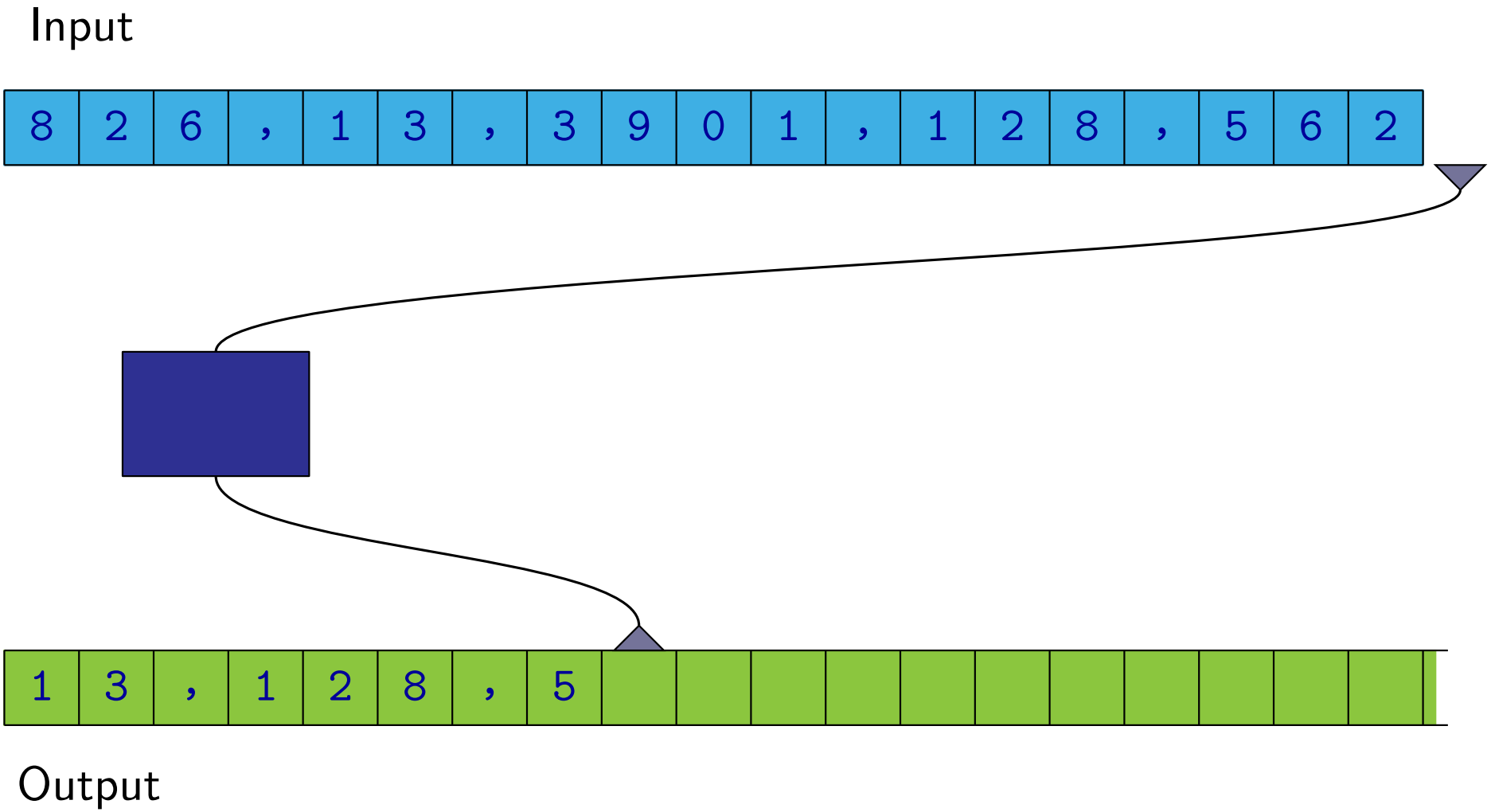
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



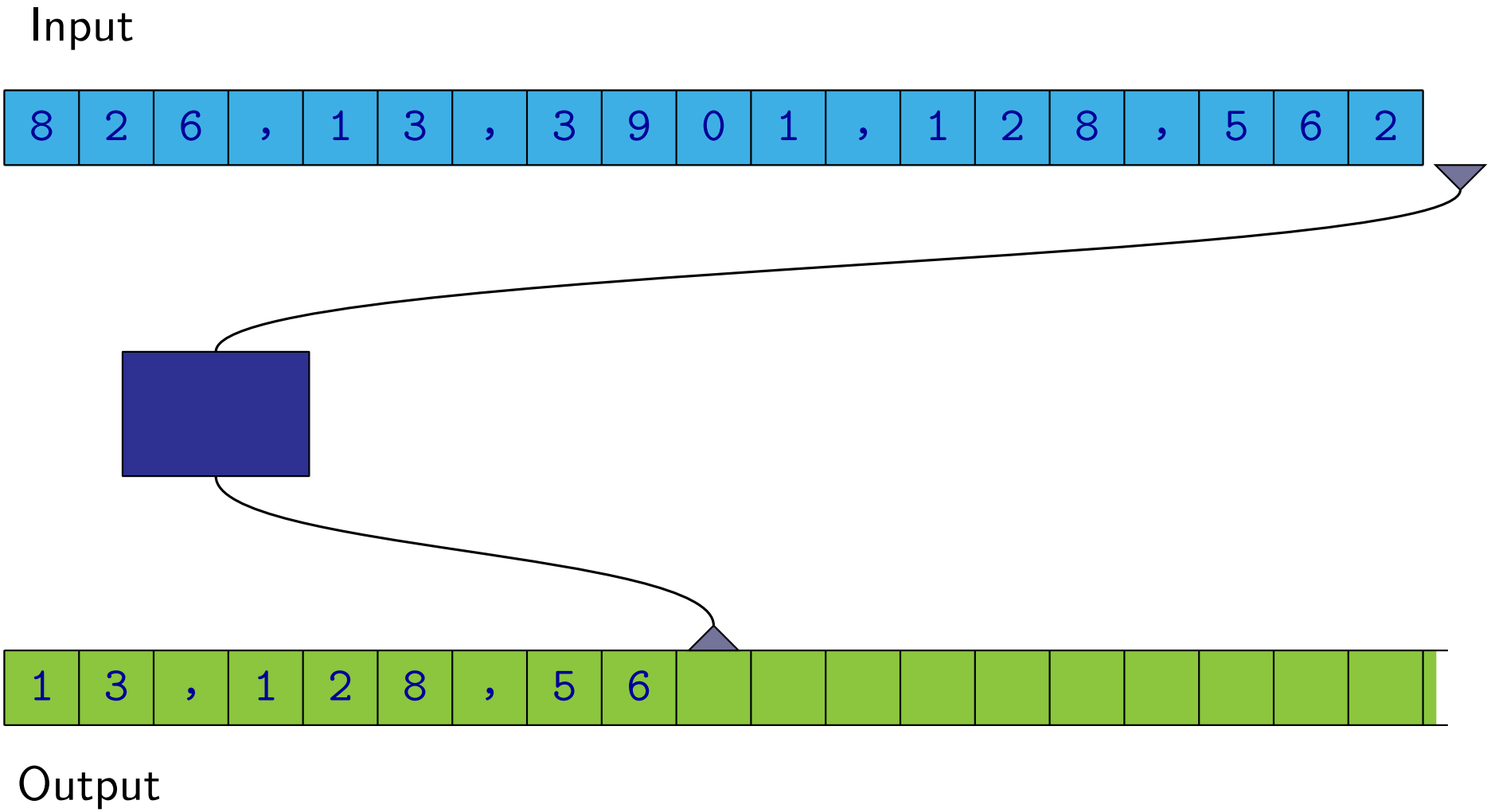
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



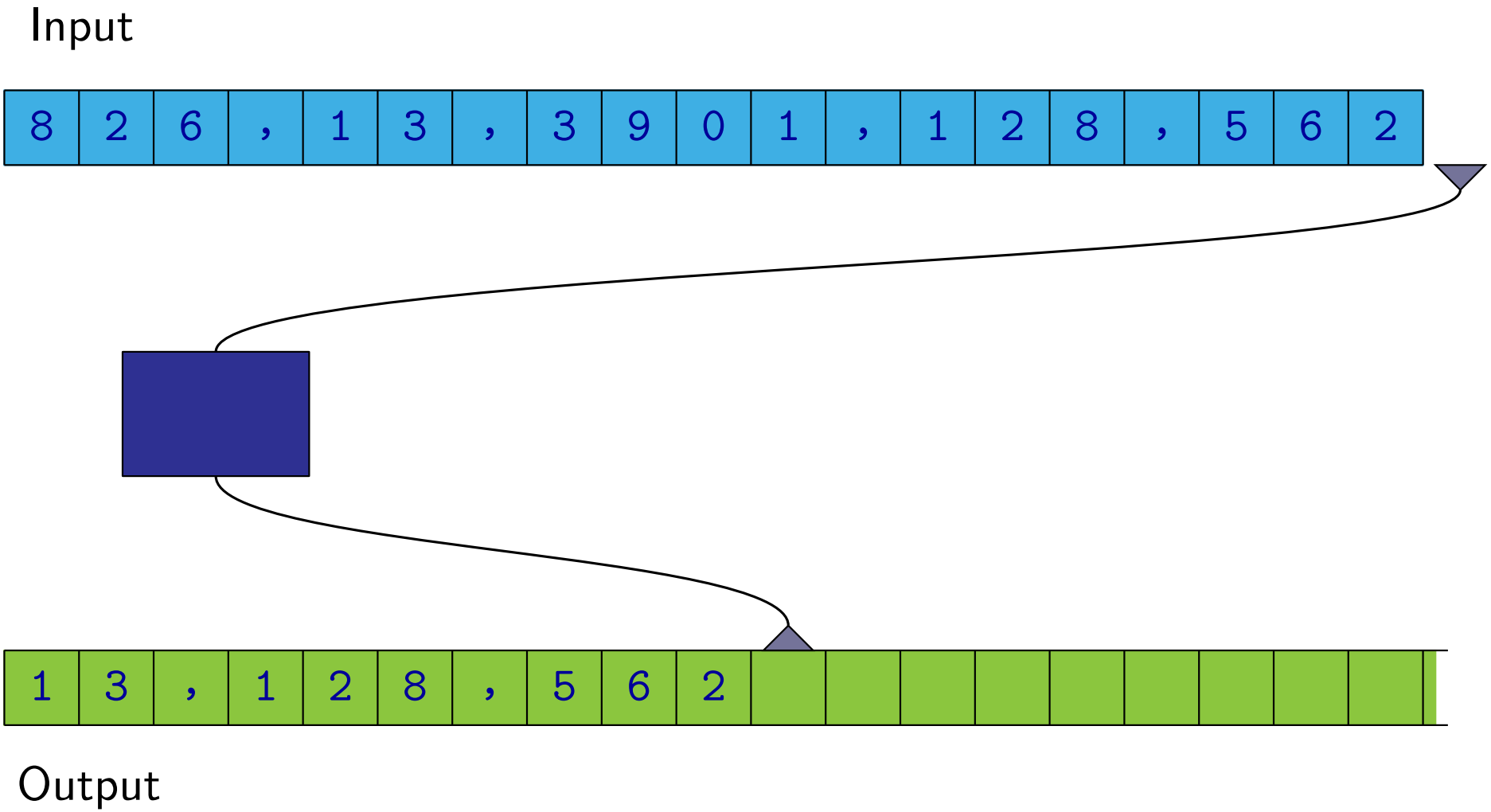
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



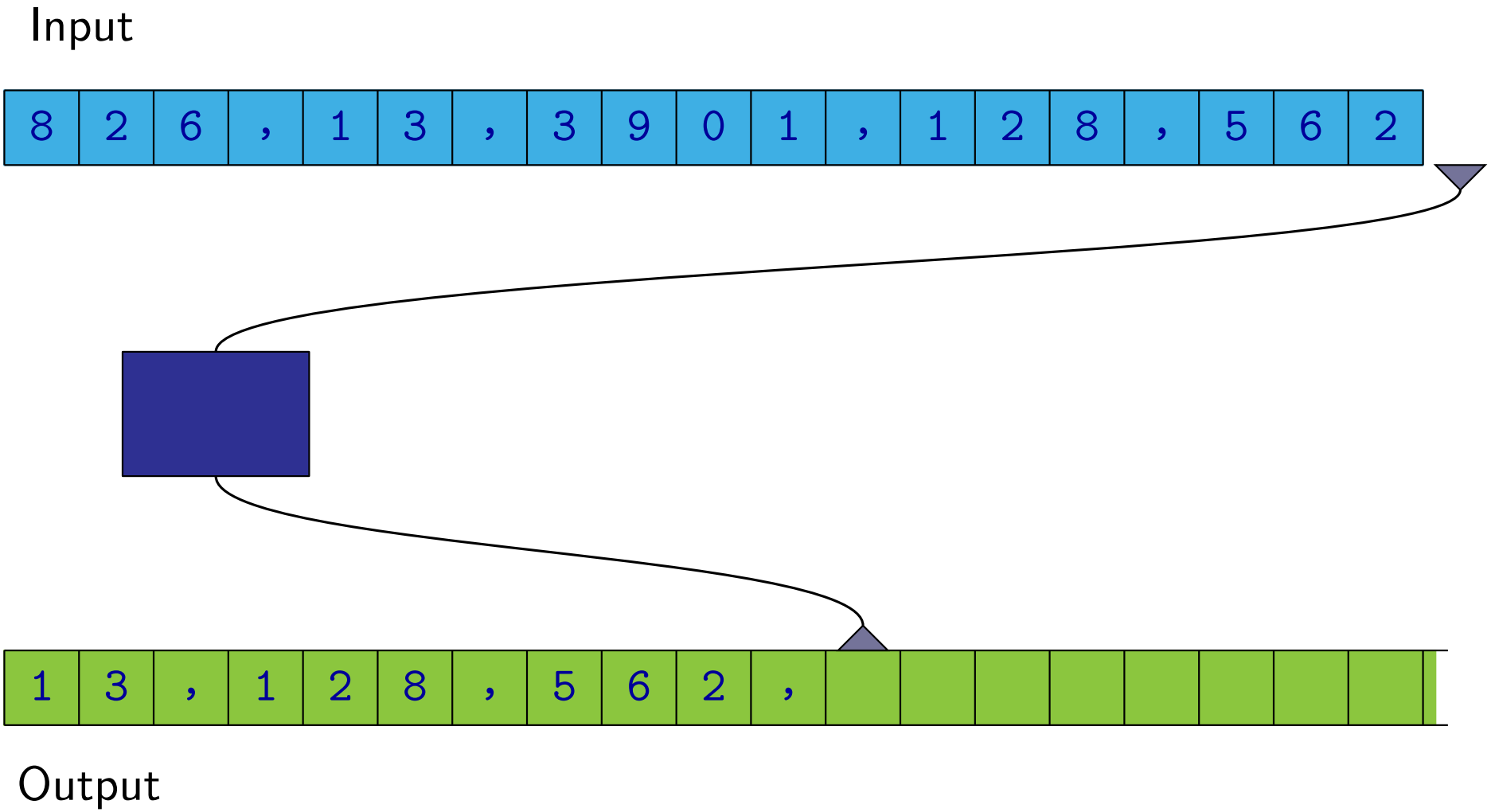
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



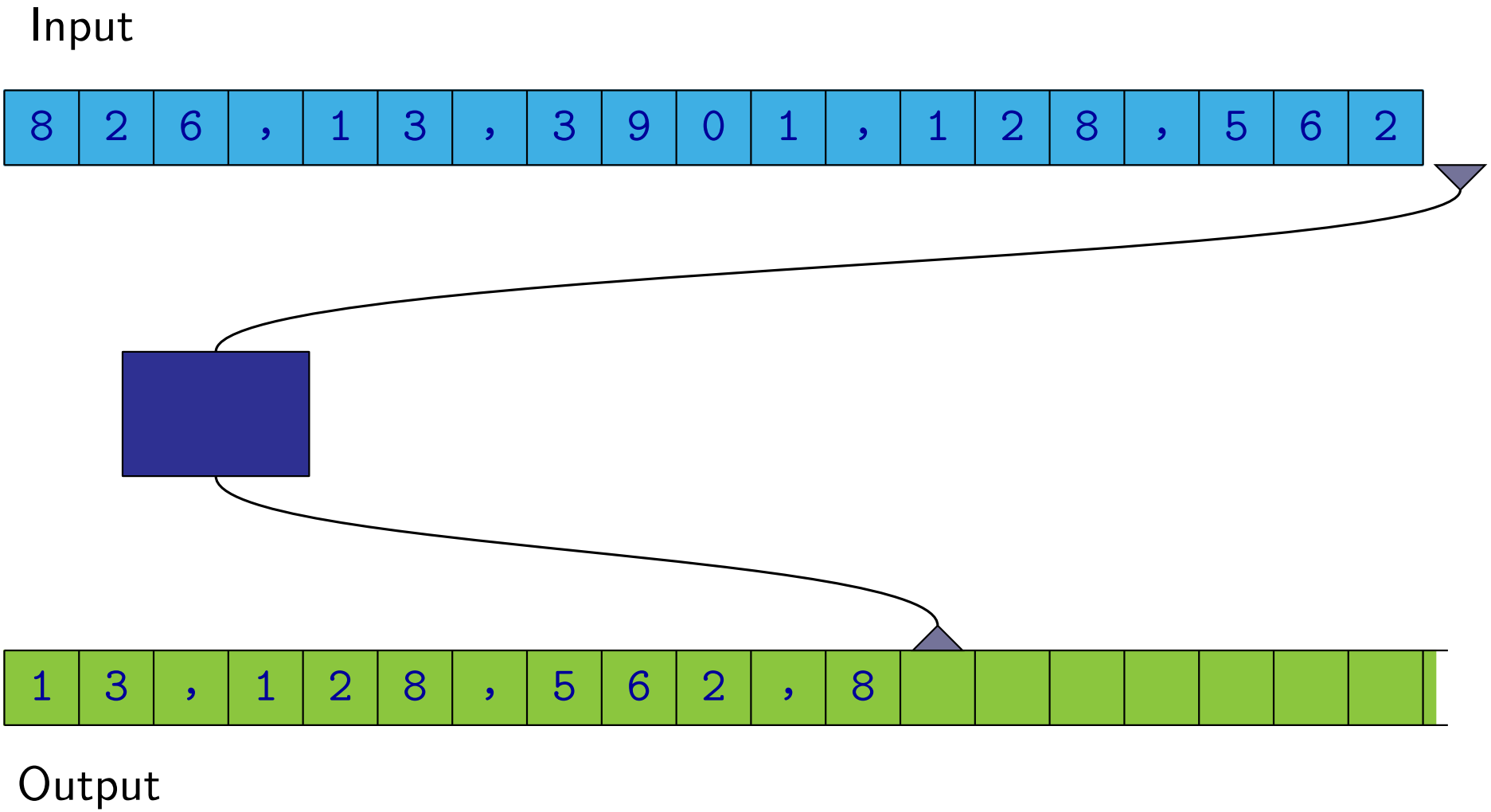
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



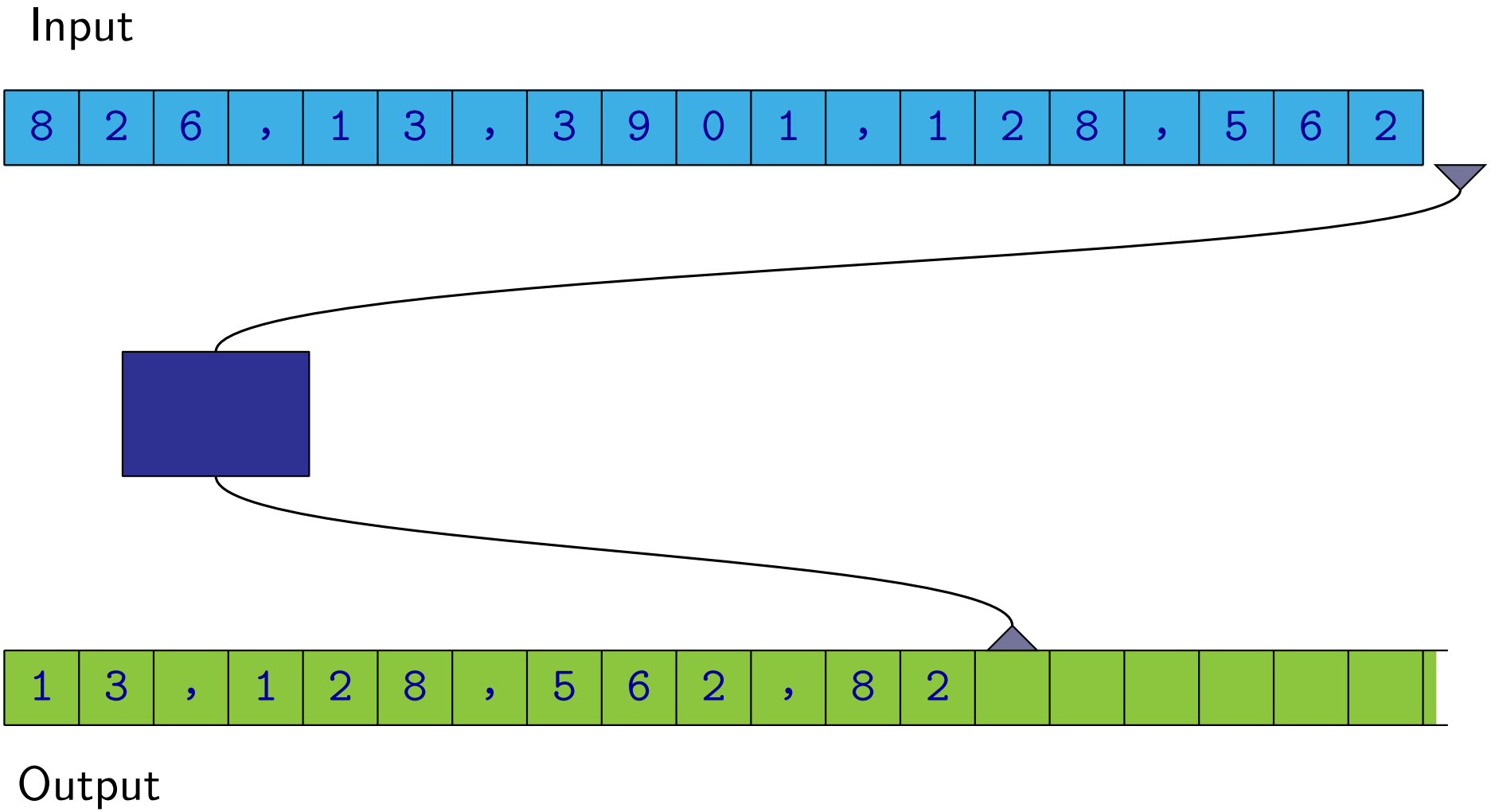
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



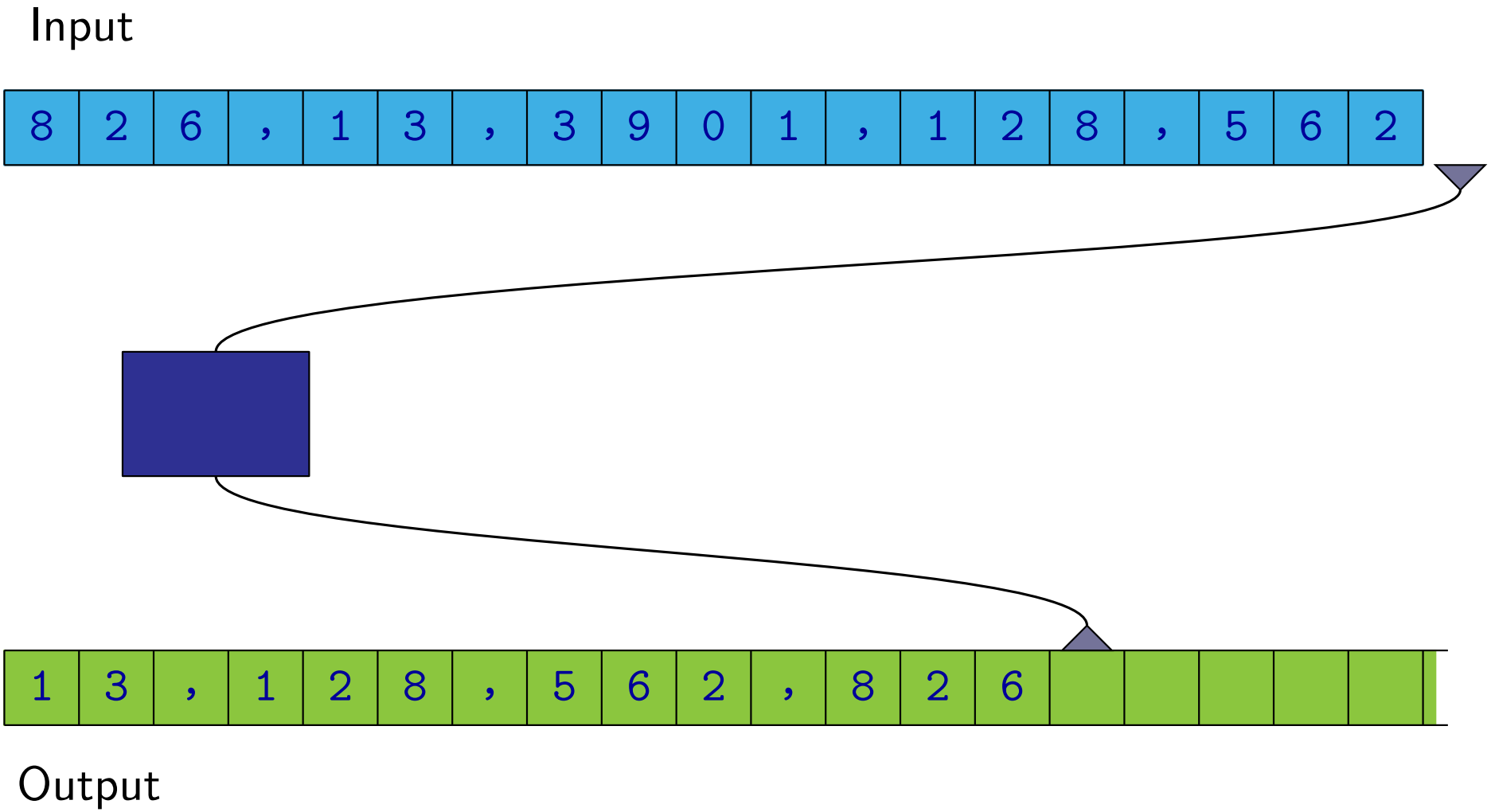
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



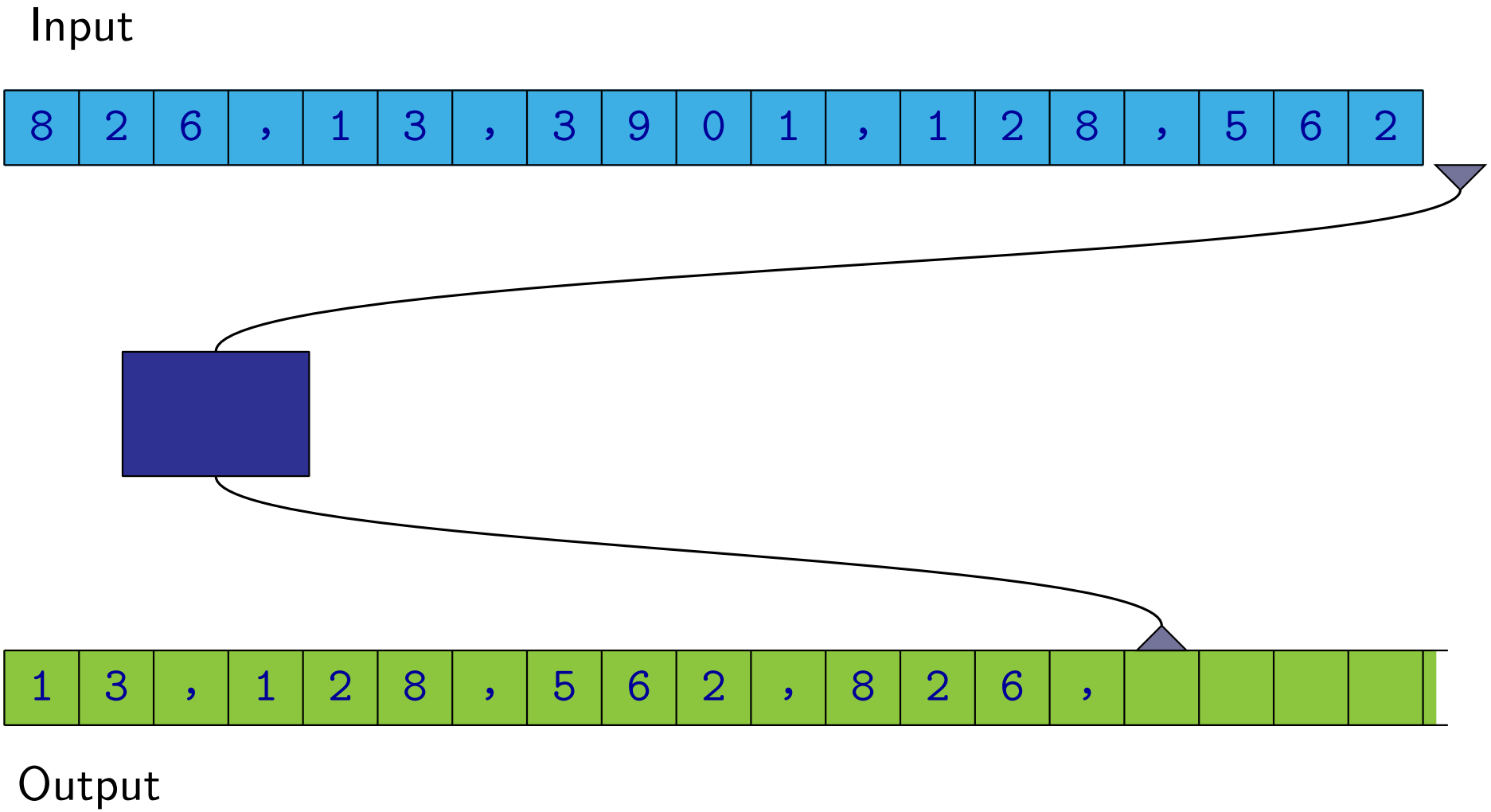
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



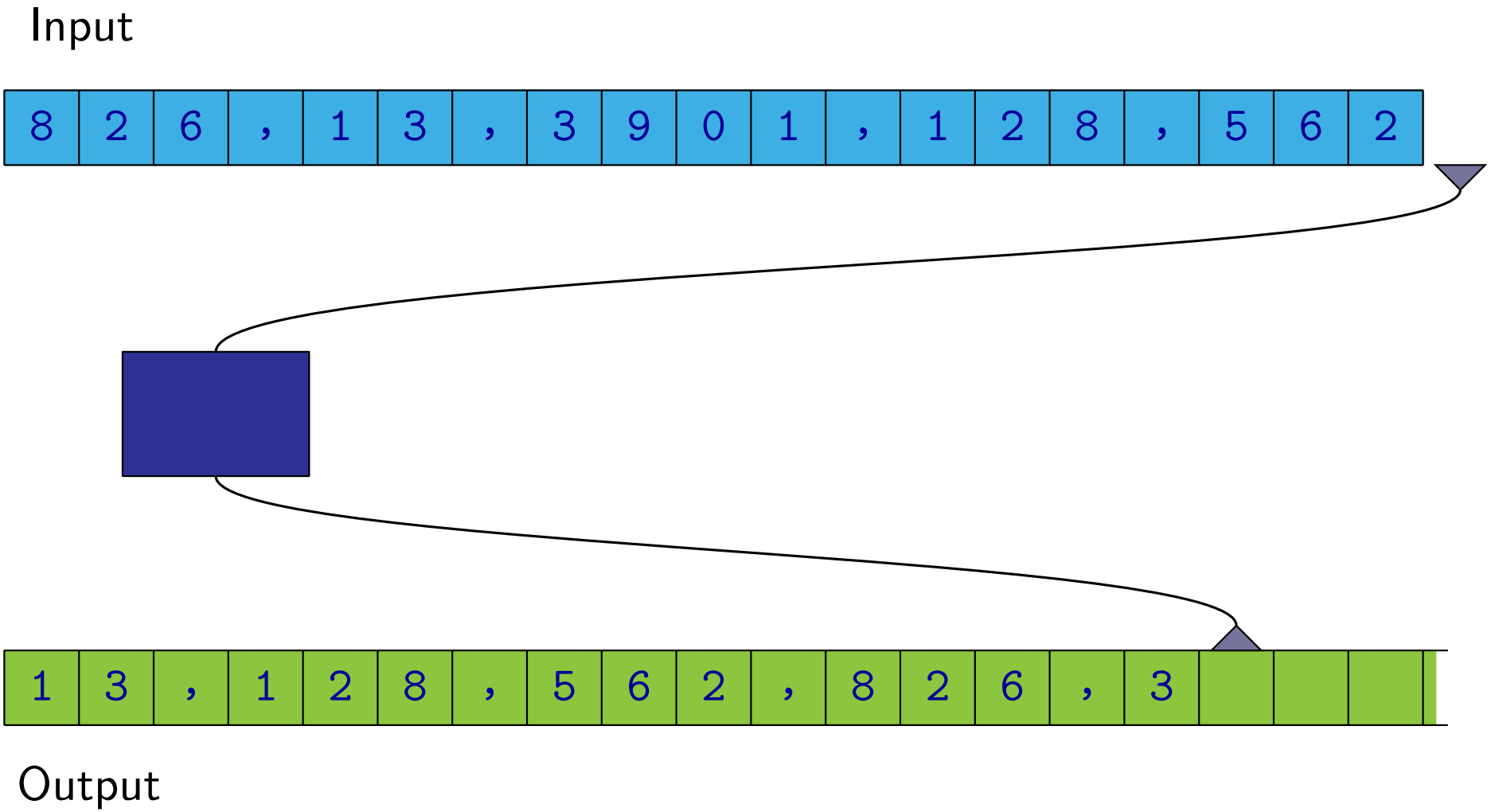
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



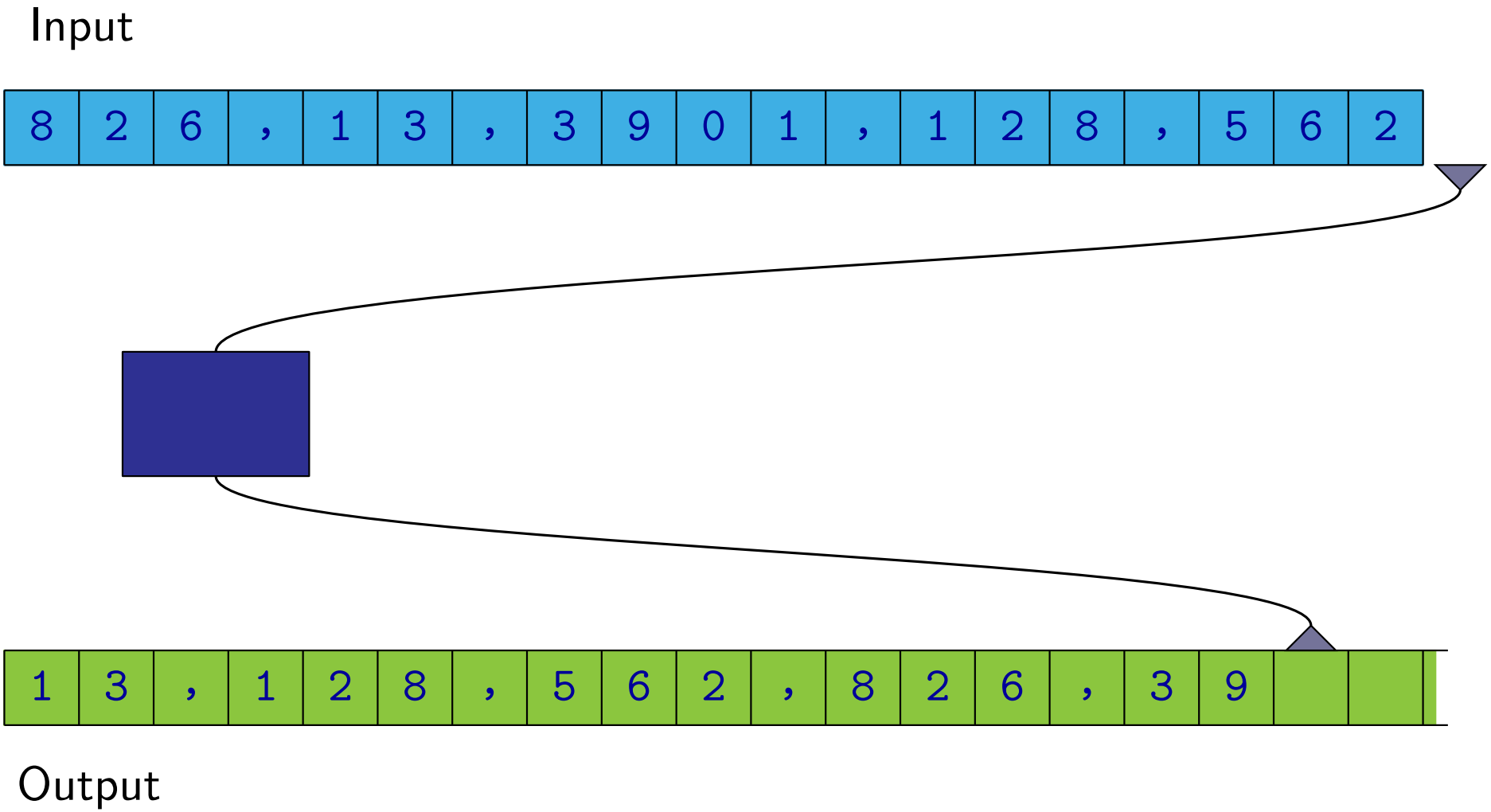
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



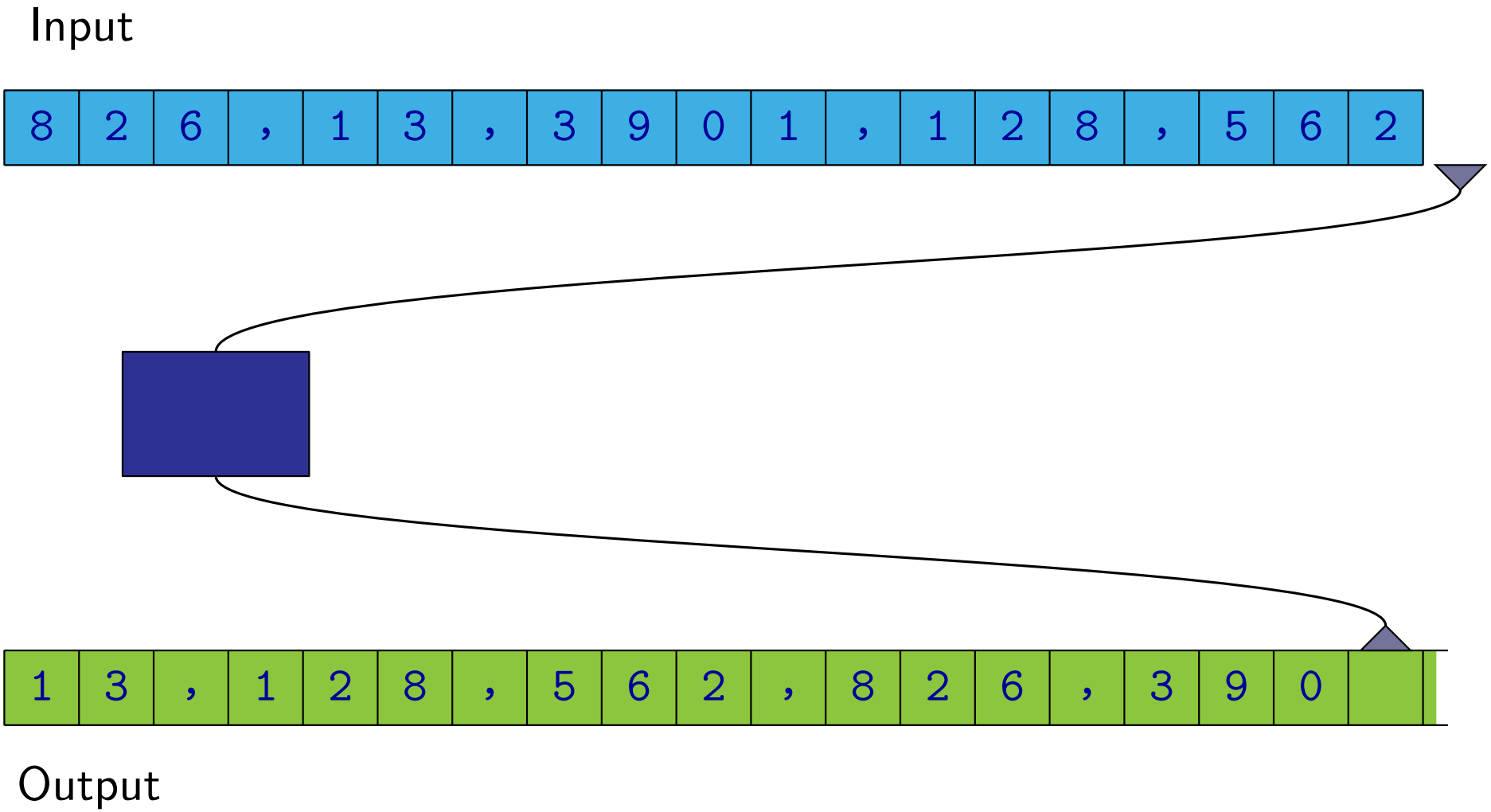
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



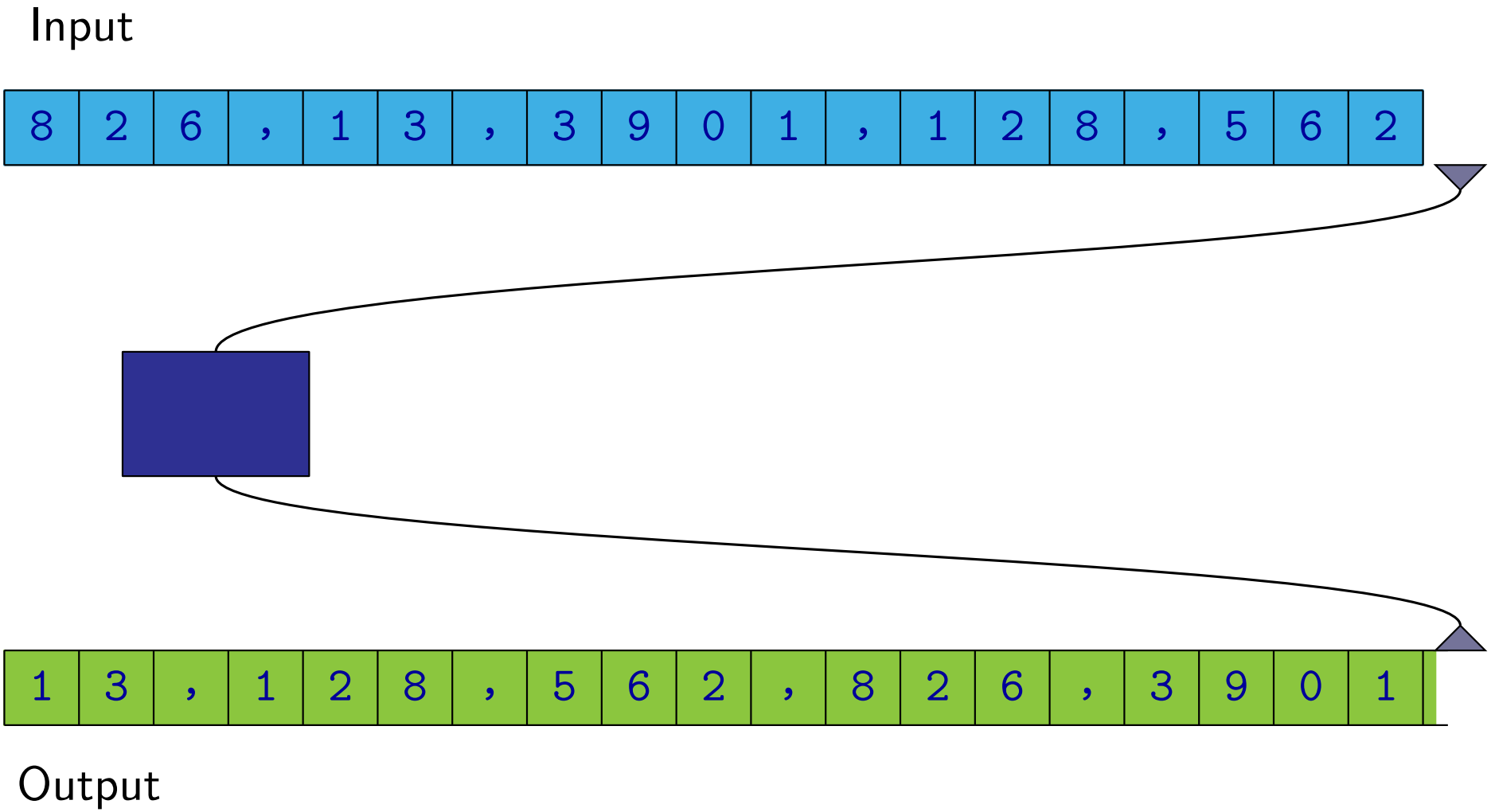
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



Činnost algoritmu

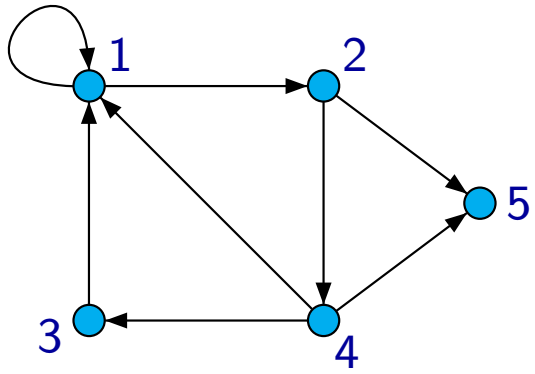
Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



Kódování vstupu a výstupu

Příklad: Pokud je vstupem nějakého problému například graf, můžeme ho reprezentovat jako seznam vrcholů a hran:

Například následující graf



můžeme reprezentovat jako slovo

$$(1,2,3,4,5),((1,2),(2,4),(4,3),(3,1),(1,1),(2,5),(4,5),(4,1))$$

v abecedě $\Sigma = \{0,1,2,3,4,5,6,7,8,9,,,(,)\}$.

Kódování vstupu a výstupu

Můžeme se omezit na případ, kdy jsou vstupy i výstupy kódovány jako slova v abecedě $\{0, 1\}$ (tj. jako sekvence bitů).

Symboly jakékoli jiné abecedy lze reprezentovat jako sekvence bitů.

Příklad: Abeceda $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$

$$
\begin{aligned}
\mathtt{a} &\leftrightarrow 001 \\
\mathtt{b} &\leftrightarrow 010 \\
\mathtt{c} &\leftrightarrow 011 \\
\mathtt{d} &\leftrightarrow 100 \\
\mathtt{e} &\leftrightarrow 101 \\
\mathtt{f} &\leftrightarrow 110 \\
\mathtt{g} &\leftrightarrow 111
\end{aligned}
$$

Slovo '$\mathtt{defb}$' můžeme reprezentovat jako '100101110010'.

Kódování vstupu a výstupu

Další alternativou je možnost, kdy vstupy a výstupy nejsou kódovány jako slova v abecedě, nýbrž jako posloupnosti čísel.

Stačí zvolit nějaké kódování, kdy symbolům dané abecedy přiřadíme číselné kódy.

Příklad: Abeceda $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$

$$
\begin{aligned}
\mathtt{a} &\leftrightarrow 1 \\
\mathtt{b} &\leftrightarrow 2 \\
\mathtt{c} &\leftrightarrow 3 \\
\mathtt{d} &\leftrightarrow 4 \\
\mathtt{e} &\leftrightarrow 5 \\
\mathtt{f} &\leftrightarrow 6 \\
\mathtt{g} &\leftrightarrow 7
\end{aligned}
$$

Slovo '$\mathtt{defb}$' můžeme reprezentovat jako posloupnost $[4, 5, 6, 2]$.

Kódování vstupu a výstupu

Na slova v abecedě $\Sigma = \{0, 1\}^*$ (tj. sekvence bitů) se můžeme dívat jako na zápisy čísel v binární soustavě.

Jedním číslem tak můžeme kódovat libovolně dlouhou posloupnost bitů.

Ještě další alternativou tak je to, že celý vstup, resp. výstup, je kódován jediným (většinou obrovským) číslem.

V této variantě tedy můžeme předpokládat, že

$$In = Out = \mathbb{N},$$

kde $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ je množina přirozených čísel.

Další příklady problémů

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Výstup: ANO pokud je n prvočíslo, NE v opačném případě.

Poznámka: Přirozené číslo n je **prvočíslo**, pokud je větší než 1 a je dělitelné beze zbytku pouze čísly 1 a n.

Prvních několik prvočísel: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

Rozhodovací problémy

Problémům, kde množina výstupů je $\{\text{Ano}, \text{Ne}\}$ se říká **rozhodovací problémy**.

Rozhodovací problémy jsou většinou specifikovány tak, že místo popisu toho, co je výstupem, je uvedena otázka.

Příklad:

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Otázka: Je n prvočíslo?

Rozhodovací problémy

Rozhodovací problém

Jednou z možností, jak formálně definovat **rozhodovací problém**, je definovat ho jako dvojici (In, T), kde:

- In je množina možných vstupů,
- $T \subseteq In$ je množina vstupů, pro které je odpověď ANO.

Rozhodovací problémy a jazyky

Pokud se omezíme na to, že vstupy jsou slova z nějaké abecedy Σ, můžeme na rozhodovací problémy pohlížet jako na jazyky.

Jazyk odpovídající danému rozhodovacímu problému je množina těch slov ze Σ^*, která reprezentují ty vstupy, pro něž je odpověď ANO.

Příklad: Jazyk L tvořený těmi slovy ze $\{0,1\}^*$, která jsou binárním zápisem nějakého prvočísla.

Například $101 \in L$, ale $110 \notin L$.

Příklad rozhodovacího problému

Problém SAT (splnitelnost booleovských formulí)

Vstup: Booleovská formule φ.

Otázka: Je φ splnitelná?

Příklad:

Formule $\varphi_1 = x_1 \wedge (\neg x_2 \vee x_3)$ je splnitelná:
např. při ohodnocení ν, kde $\nu(x_1) = 1$, $\nu(x_2) = 0$, $\nu(x_3) = 1$, platí $\nu(\varphi_1) = 1$.

Formule $\varphi_2 = (x_1 \wedge \neg x_1) \vee (\neg x_2 \wedge x_3 \wedge x_2)$ není splnitelná:
pro libovolné ohodnocení ν platí $\nu(\varphi_2) = 0$.

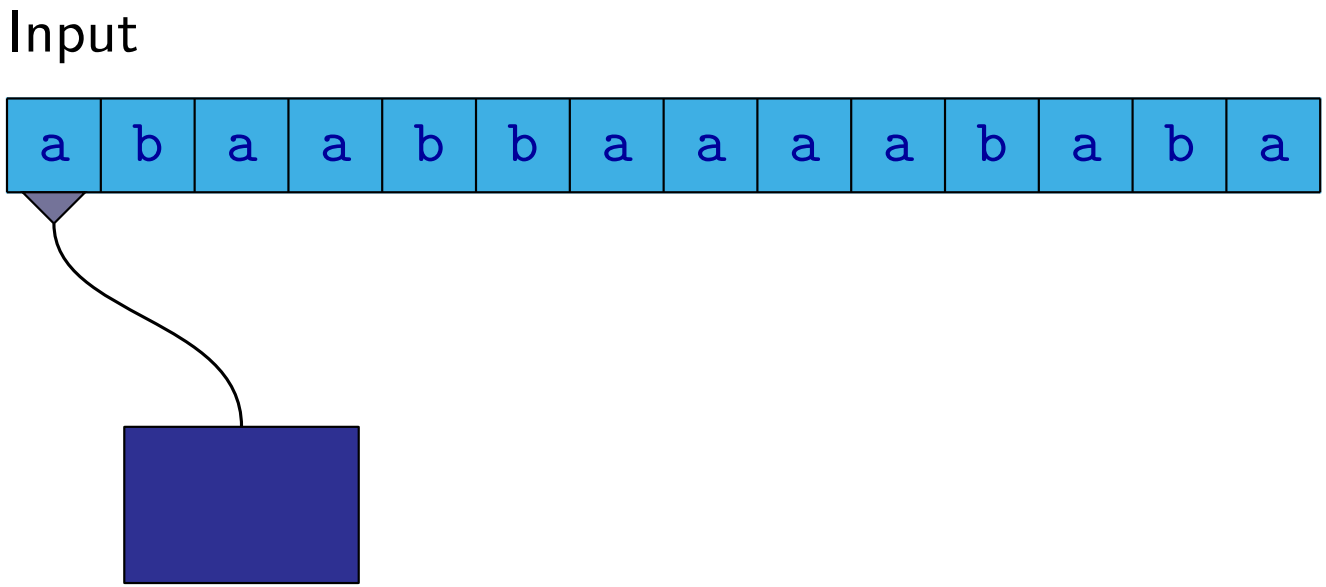
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



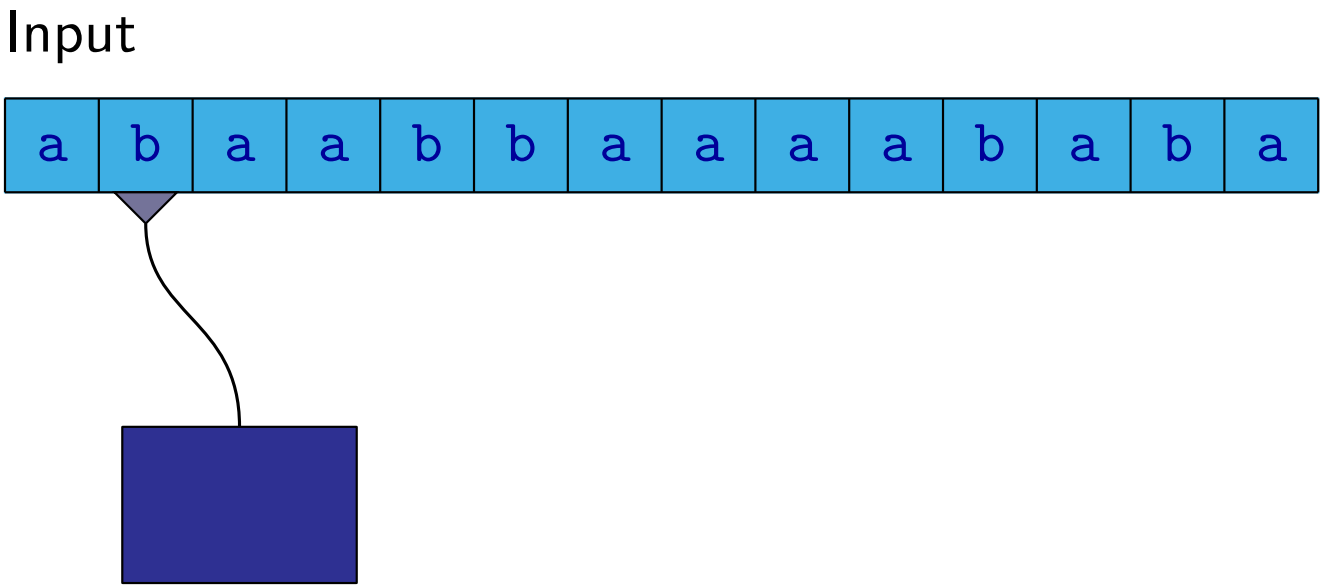
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



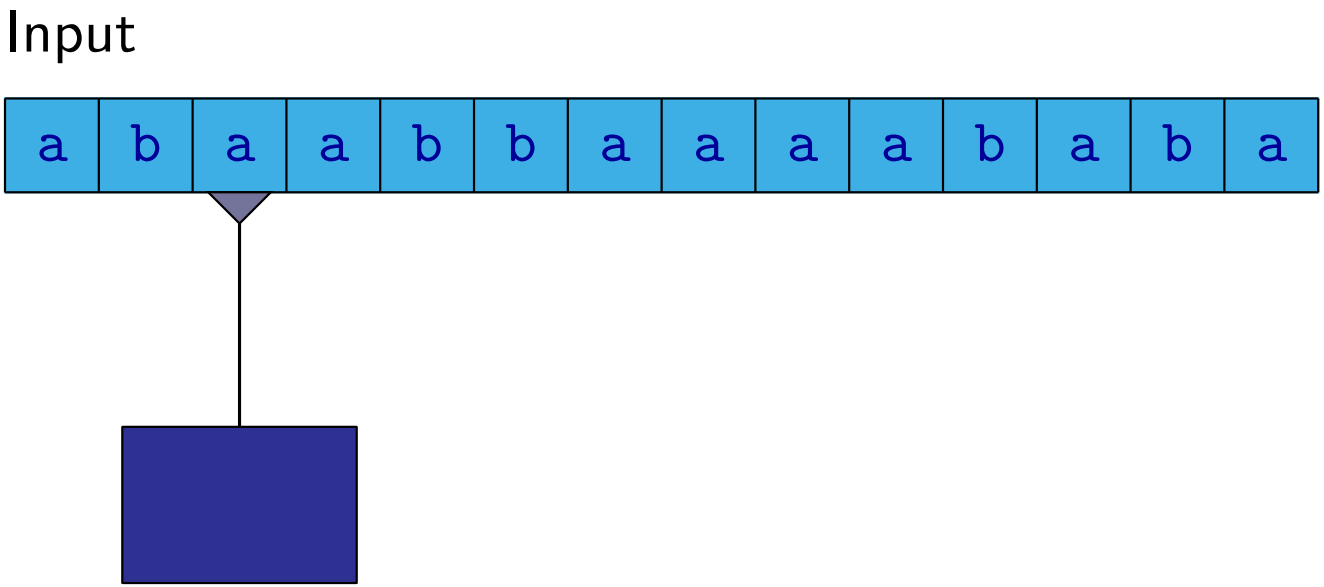
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



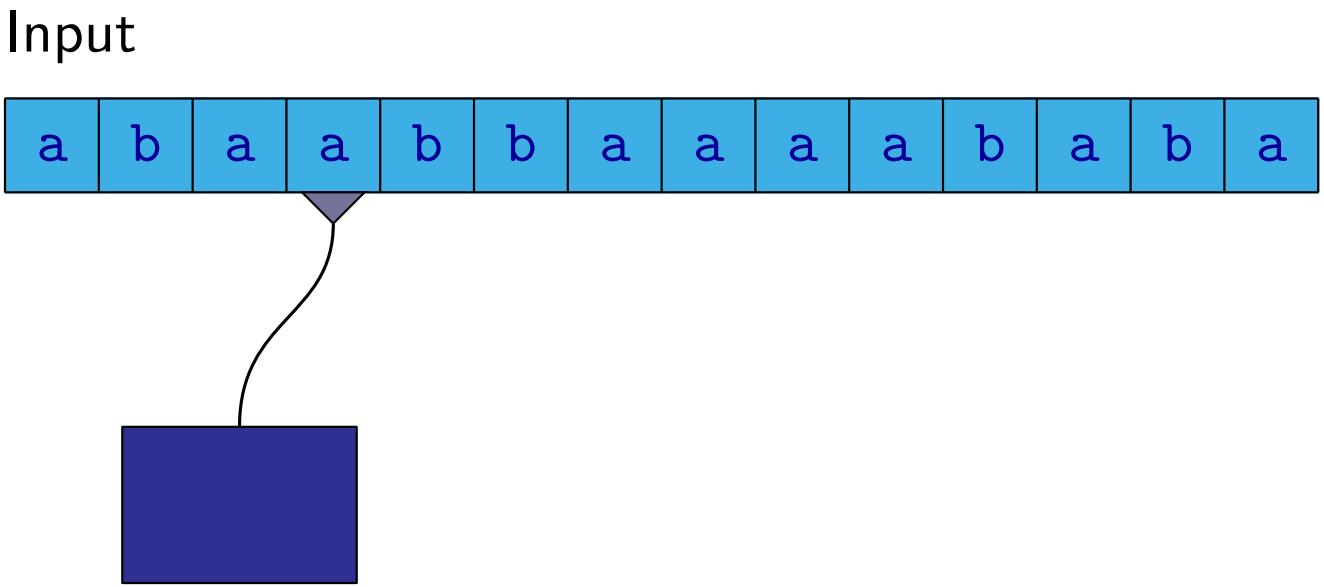
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



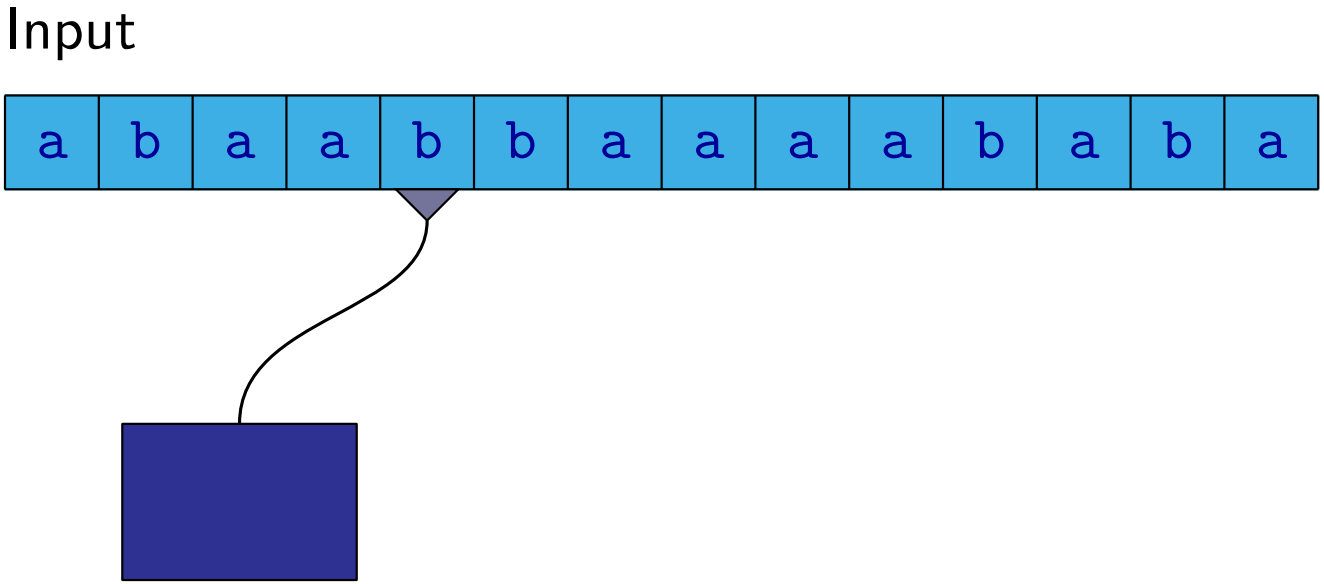
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



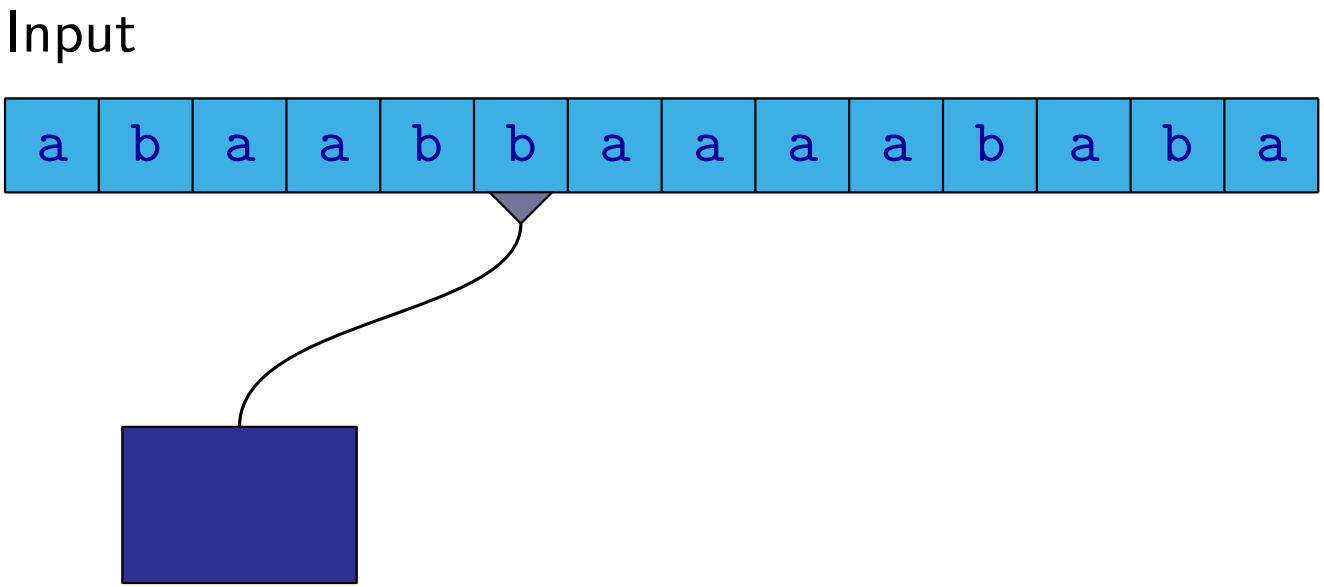
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



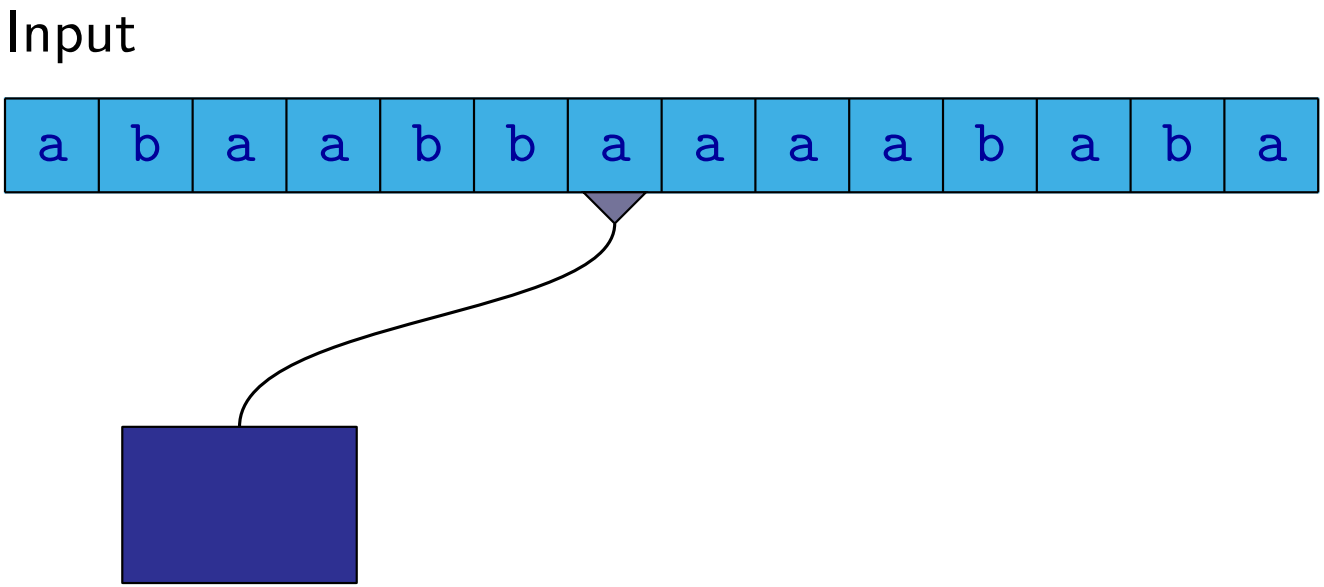
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



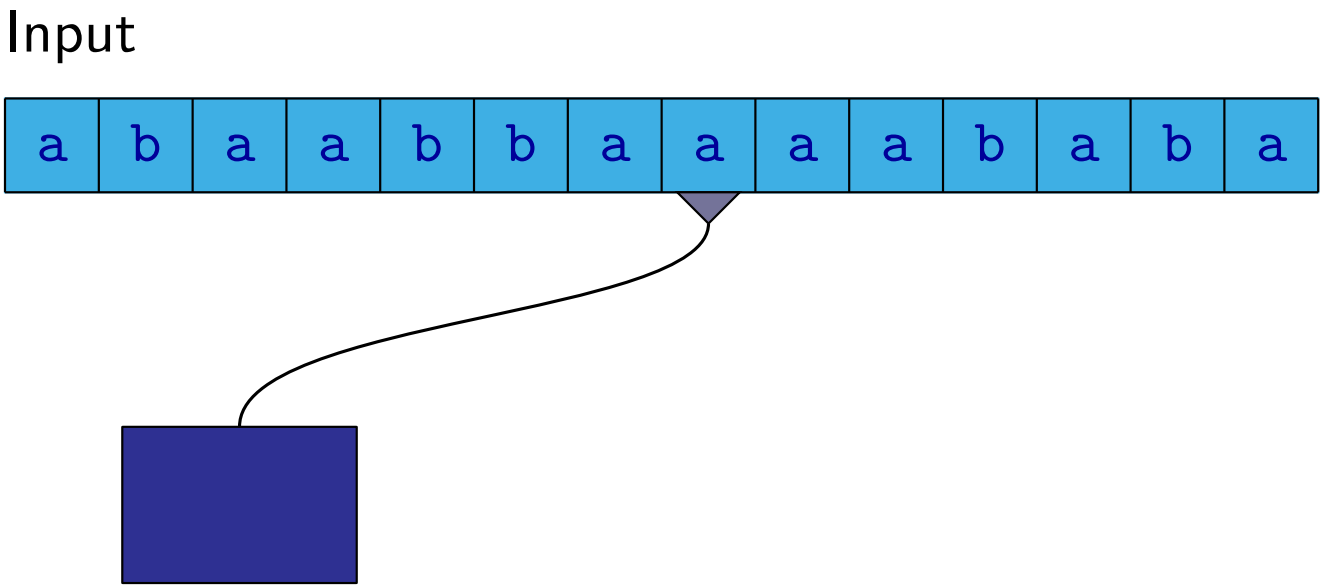
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



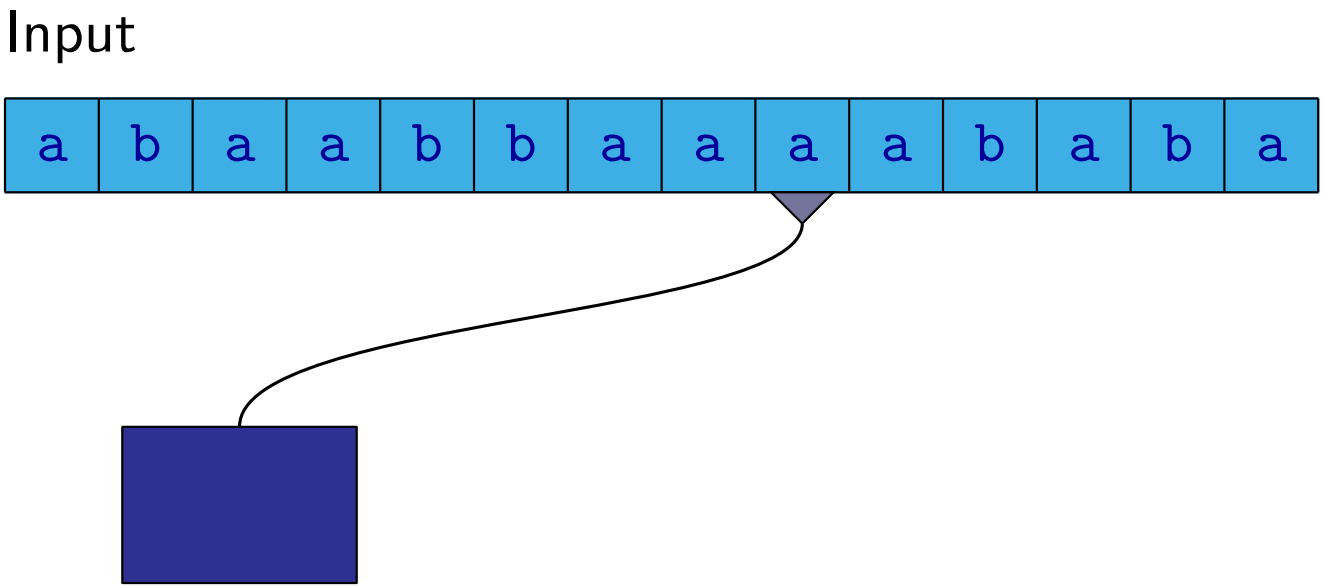
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



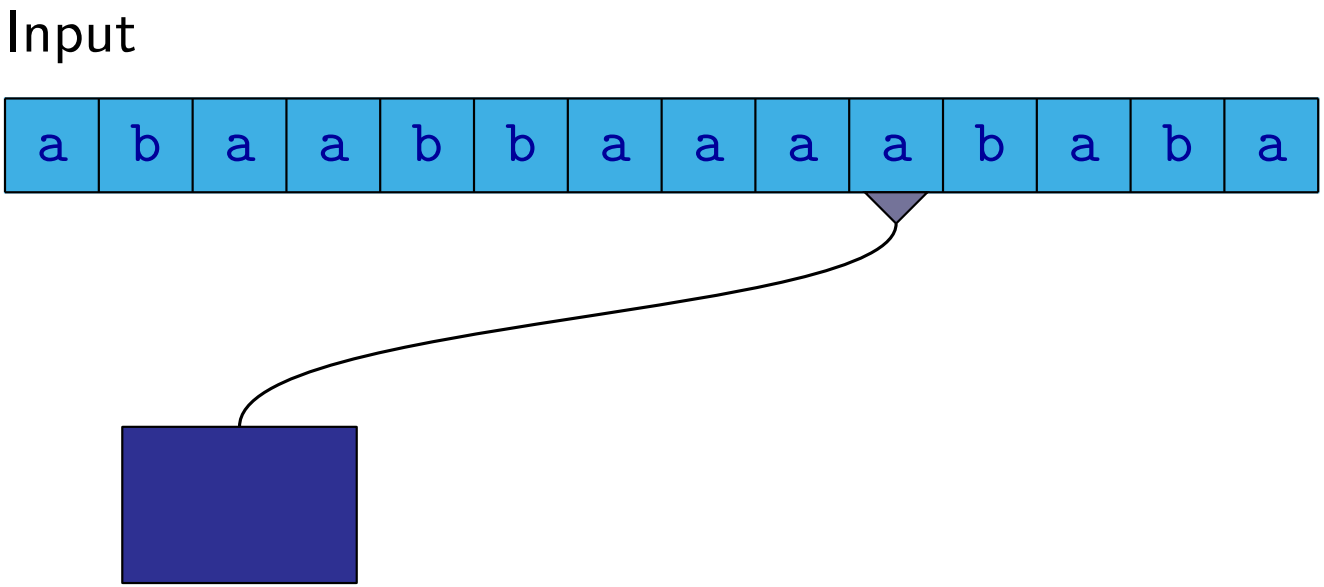
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



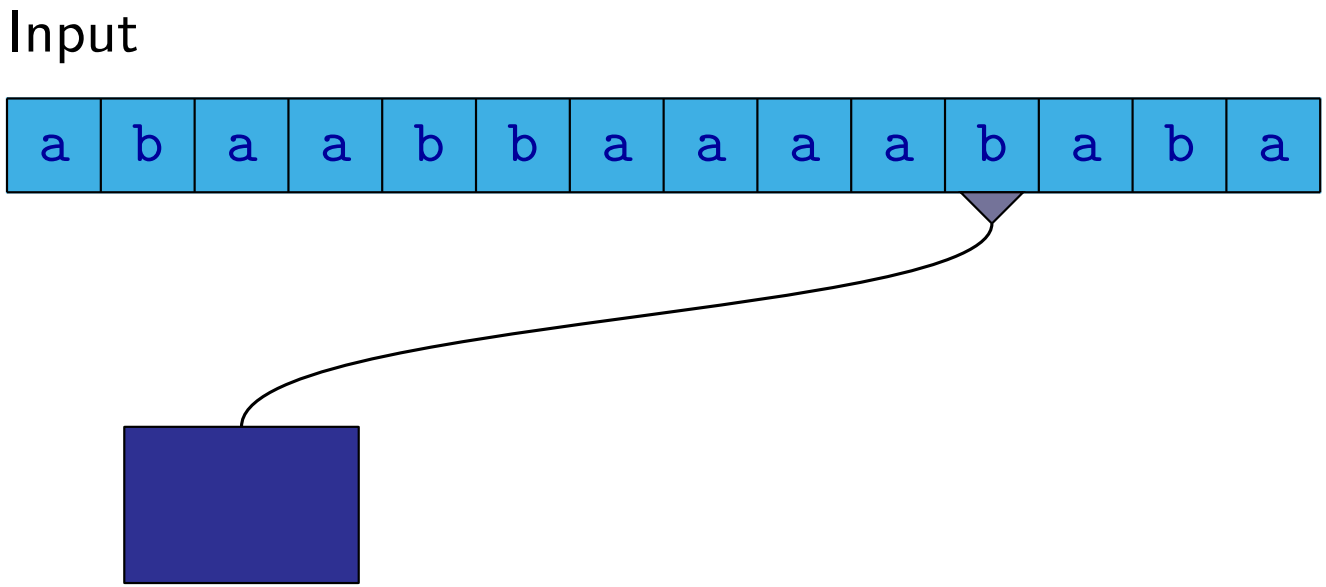
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



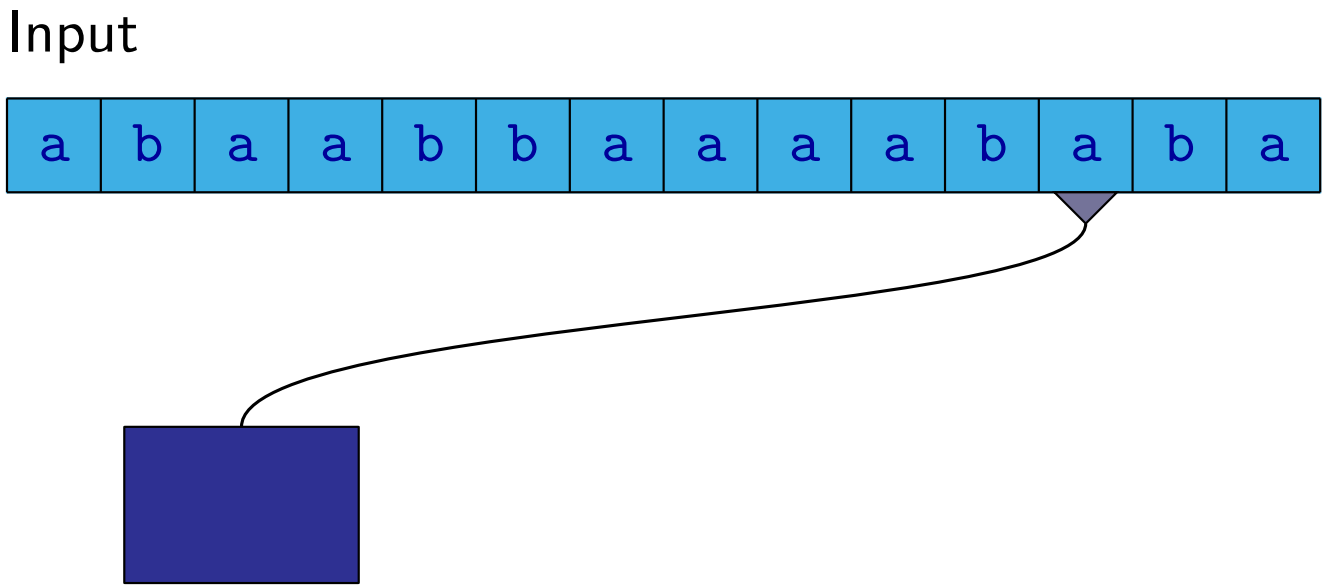
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



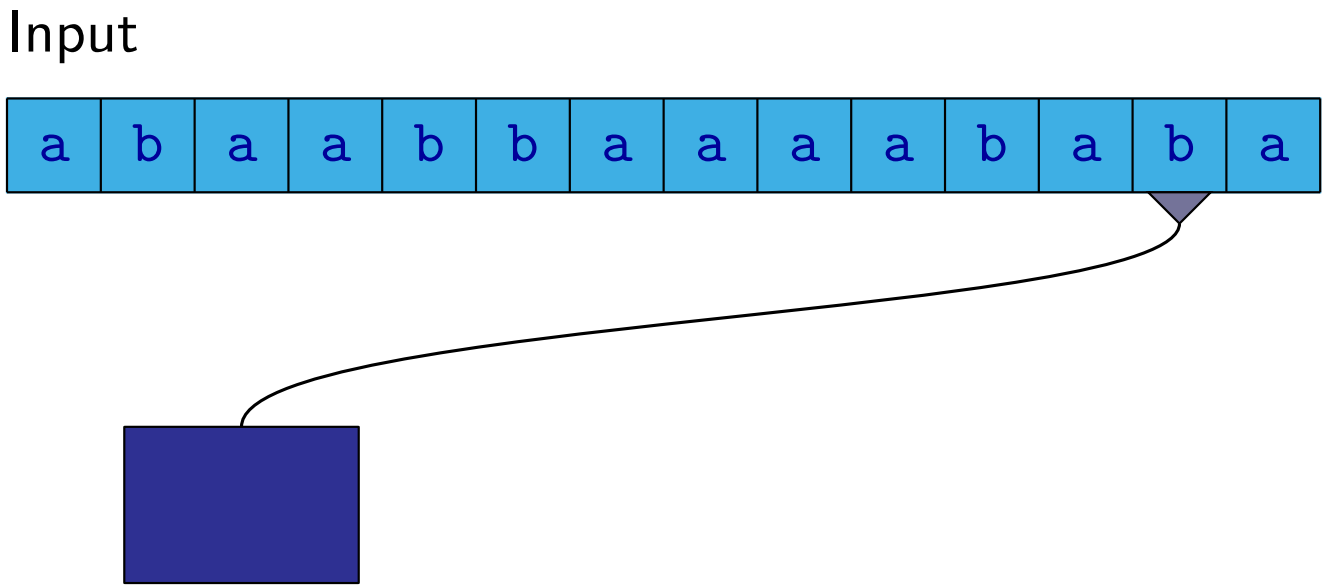
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



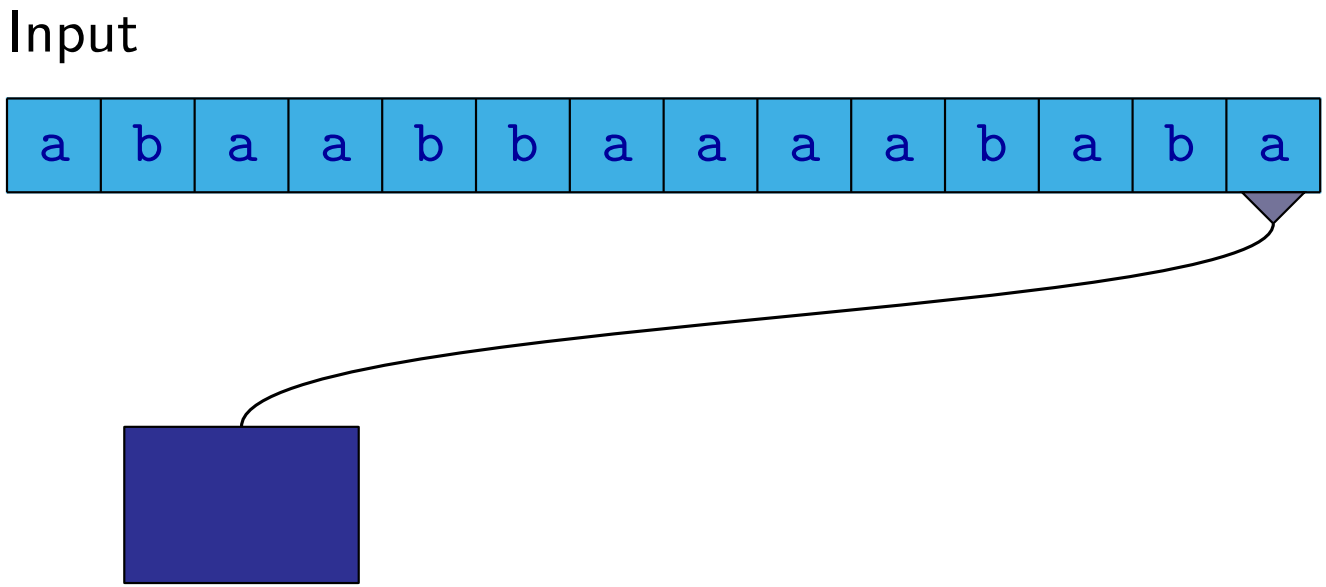
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



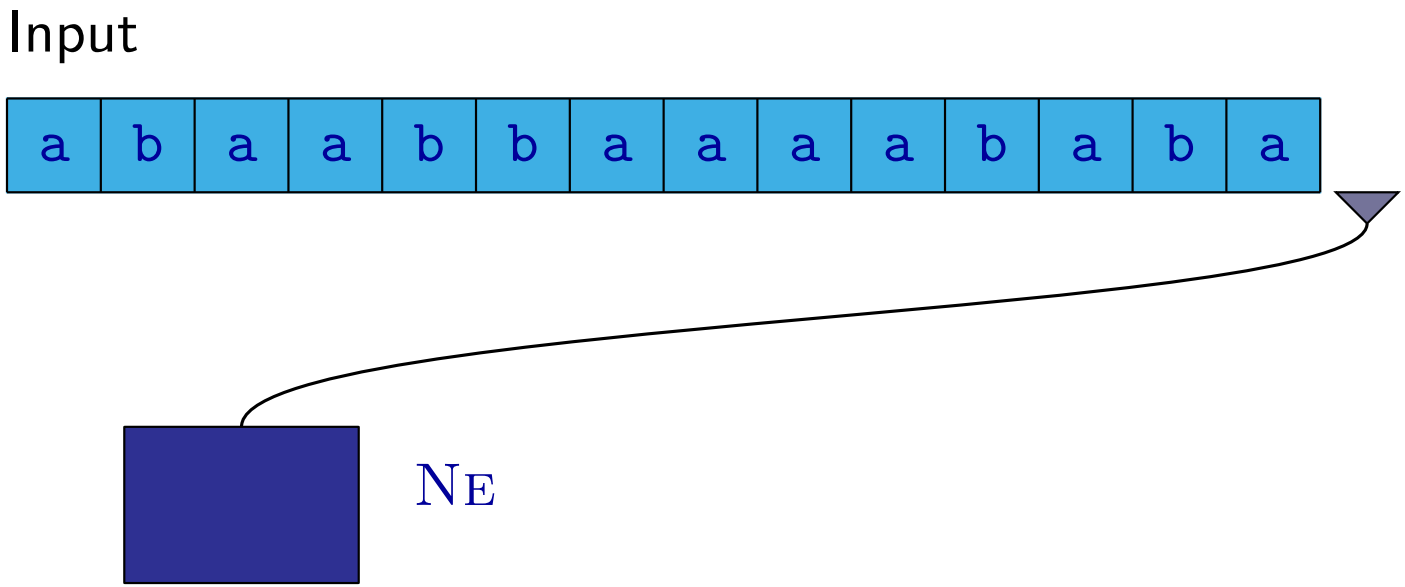
Činnost algoritmu řešícího rozhodovací problém

V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?



Vztah mezi rozpoznáváním formálních jazyků a rozhodovacími problémy

Mezi rozpoznáváním slov z daného jazyka a rozhodovacími problémy je úzký vztah:

- Každému jazyku L nad nějakou abecedou Σ odpovídá následující rozhodovací problém:

Vstup: Slovo w nad abecedou Σ.

Otázka: Patří slovo w do jazyka L?

- Ke každému rozhodovacímu problému P, jehož vstupy jsou kódovány jako slova nad abecedou Σ, existuje jemu odpovídající jazyk:

Jazyk L obsahující právě ta slova w nad abecedou Σ, pro která je odpověď na příslušnou otázku specifikovanou v zadání problému P "ANO".

Vztah mezi rozpoznáváním formálních jazyků a rozhodovacími problémy

Příklad: Na následující rozhodovací problém se můžeme dívat jako na níže uvedený jazyk L a naopak.

Problém

Vstup: Slovo w nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů $\mathtt{b}$?

Jazyk $L = \{ w \in \{\mathtt{a}, \mathtt{b}\}^* \mid \text{slovo } w \text{ obsahuje sudý počet výskytů symbolů } \mathtt{b} \}$

Rozhodovací problémy

Často se při zkoumání algoritmů a problému omezujeme jen na rozhodovací problémy.

Není to však na úkor obecnosti, neboť libovolný obecný problém je možné vhodným způsobem přeformulovat jako rozhodovací problém, s tím, že když najdeme algoritmus, který by řešil tento rozhodovací problém, tak bychom snadno sestrojili algoritmus, který by řešil původní problém, a naopak.

Rozhodovací problémy

Pokud například máme problém P, kde:

- vstupy jsou prvky z nějaké množiny In
- výstupy jsou slova z $\{0,1\}^*$

můžeme tento problém přeformulovat jako následující rozhodovací problém:

Problém

Vstup: Prvek $x \in In$ a číslo k.

Otázka: Když z je výstup, který odpovídá vstupu x problému P, má k-tý bit slova z hodnotu 1?

Optimalizační problémy

Problémům, kde je pro daný vstup určena nějaká množina **přípustných řešení** a kde je úkolem mezi těmito přípustnými řešeními vybrat takové, které je v nějakém ohledu minimální nebo maximální (případně zjistit, že žádné přípustné řešení neexistuje), se říká **optimalizační problémy**.

Příklad:

Problém „Hledání nejkratší cesty v (neorientovaném) grafu“

Vstup: Neorientovaný graf $G = (V, E)$ s ohodnocením hran, a dvojice vrcholů $u, v \in V$.

Výstup: Nejkratší cesta z vrcholu u do vrcholu v.

Optimalizační problémy

Formálně můžeme **optimalizační problém** definovat jako pětici (In, Out, f, m, g), kde:

- In je množina možných vstupů,
- Out je množina **řešení**,
- $f : In \to \mathcal{P}(Out)$ je funkce přiřazující každému vstupu x odpovídající množinu **přípustných řešení** $f(x)$,
- $m : \bigcup_{x \in In}(\{x\} \times f(x)) \to \mathbb{R}$ je **optimalizační funkce** (**kriteriální funkce**),
- g je $\min$ nebo $\max$.

Cílem je pro daný vstup $x \in In$ najít nějaké přípustné řešení $y \in f(x)$ takové, že

$$m(x, y) = g\{m(x, y') \mid y' \in f(x)\},$$

nebo zjistit, že pro daný vstup x žádné přípustné řešení neexistuje (tj. $f(x) = \emptyset$).

Optimalizační problémy

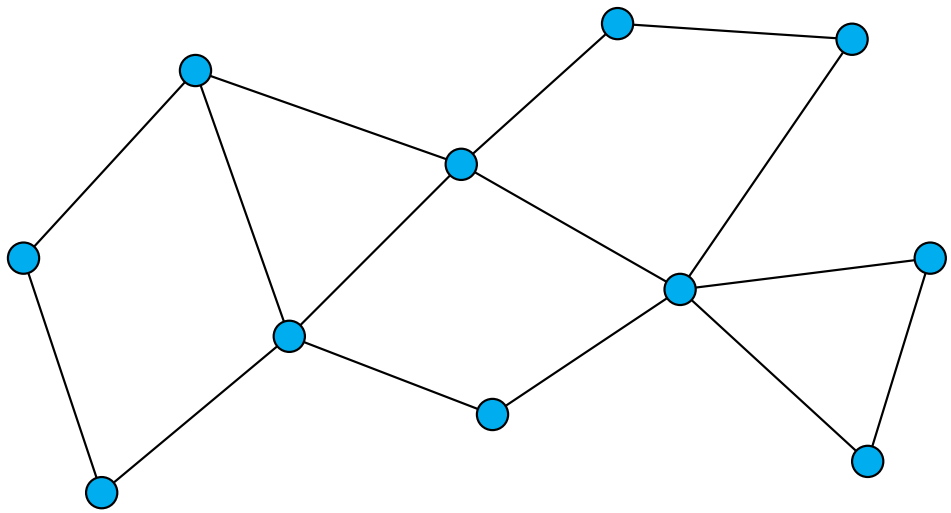
- Optimalizačním problémům, kde g je $\min$, se říká **minimalizační problémy**.
- Optimalizačním problémům, kde g je $\max$, se říká **maximalizační problémy**.

Optimalizační problémy

Problém „Barvení grafu“

Vstup: Neorientovaný graf G.

Výstup: Minimální počet barev, kterými je možné obarvit vrcholy grafu G tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu, a konkrétní příklad obarvení vrcholů používající tento minimální počet barev.

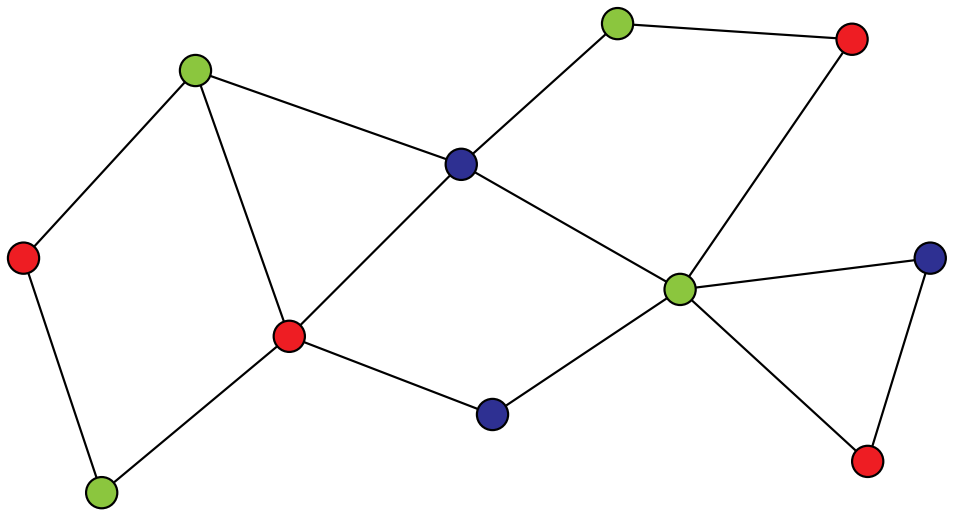


Optimalizační problémy

Problém „Barvení grafu“

Vstup: Neorientovaný graf G.

Výstup: Minimální počet barev, kterými je možné obarvit vrcholy grafu G tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu, a konkrétní příklad obarvení vrcholů používající tento minimální počet barev.



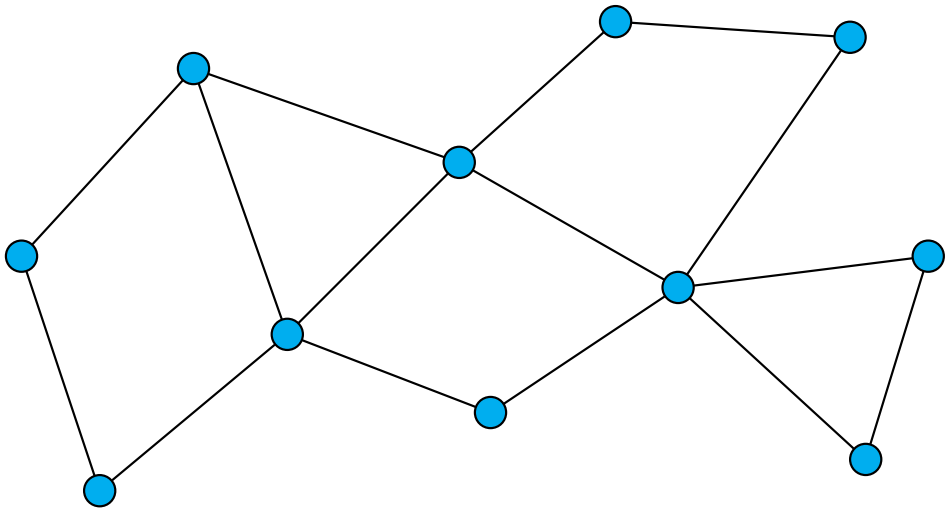
Barvy: 3

Optimalizační problémy přeformulované jako rozhodovací

Problém „Barvení grafu k barvami“

Vstup: Neorientovaný graf G a přirozené číslo k.

Otázka: Je možné obarvit vrcholy grafu G k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?



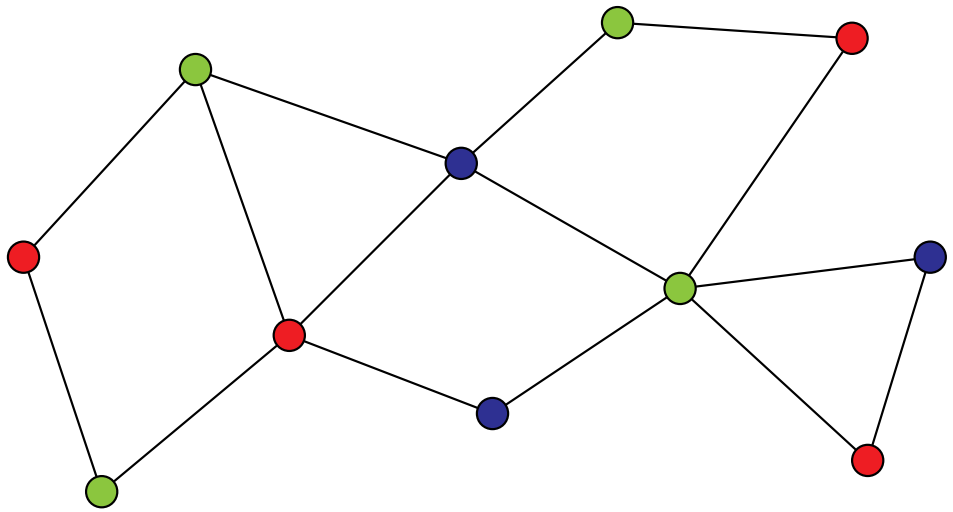
$k = 3$

Optimalizační problémy přeformulované jako rozhodovací

Problém „Barvení grafu k barvami“

Vstup: Neorientovaný graf G a přirozené číslo k.

Otázka: Je možné obarvit vrcholy grafu G k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?



$k = 3$

Řešení problémů

Řešení problému

Algoritmus **řeší** daný problém, když:

1. Se pro libovolný vstup daného problému (libovolnou vstupní instanci) po konečném počtu kroků zastaví.
2. Vyprodukuje výstup z množiny možných výstupů, který vyhovuje podmínkám uvedeným v zadání problému.

Pro jeden problém může existovat celá řada algoritmů, které jej korektně řeší.

Poznámka: **korektnost algoritmu** — algoritmus řeší daný problém

Řešení problémů

Každému algoritmu A můžeme přiřadit funkci

$$f_A : In \to Out$$

kde:

- In je množina vstupů pro algoritmus A,
- Out je množina výstupů generovaných algoritmem A,
- $f_A(x)$ je výstup, který algoritmus A vygeneruje pro vstup $x \in In$.

Funkce f_A nemusí být **totální** (tj. hodnota $f_A(x)$ nemusí být definovaná pro každé $x \in In$), ale může být **častečná** (**parciální**):

- hodnota $f_A(x)$ není definována, pokud se výpočet algoritmu A pro vstup x nikdy nezastaví, pokud během výpočtu dojde k chybě apod.

Řešení problémů

Pokud tedy máme dán nějaký problém $P = (In, Out, R)$ a nějaký algoritmus A realizující funkci $f_A : In \to Out$, pak řekneme, že

algoritmus A řeší problém P

jestliže:

- hodnota $f_A(x)$ je definovaná pro každé $x \in In$,
- pro každé $x \in In$ platí $(x, f_A(x)) \in R$

Algoritmicky řešitelné problémy

Předpokládejme, že máme dán nějaký problém P.

Jestliže existuje nějaký algoritmus, který řeší problém P, pak říkáme, že problém P je **algoritmicky řešitelný**.

Jestliže P je rozhodovací problém a jestliže existuje nějaký algoritmus, který problém P řeší, pak říkáme, že problém P je **(algoritmicky) rozhodnutelný**.

Když chceme ukázat, že problém P je algoritmicky řešitelný, stačí ukázat nějaký algoritmus, který ho řeší (a případně ukázat, že daný algoritmus problém P skutečně řeší).

Algoritmicky řešitelné problémy

U mnohých problémů je na první pohled zřejmé, že jsou algoritmicky řešitelné, jako třeba:

- Třídění
- Hledání nejkratší cesty v grafu
- Prvočíselnost

kde stačí probrat všechny možnosti (kterých je ve všech těchto případech konečně mnoho), i když takový triviální algoritmus založený na řešení **hrubou silou** nemusí být zrovna efektivní.

Na druhou stranu existuje celá řada problémů, u kterých to tak jasné není.

- Najít algoritmus a dokázat, že řeší daný problém, může být velmi netriviální úkol.
- Algoritmus, který by řešil daný problém, nemusí vůbec existovat.

ILP – celočíselné lineární programování

Problém ILP (celočíselné lineární programování)

Vstup: Celočíselná matice A a celočíselný vektor b.

Otázka: Existuje celočíselný vektor x, takový že $Ax \leq b$?

Příklad instance problému:

$$A = \begin{pmatrix} 3 & -2 & 5 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \qquad b = \begin{pmatrix} 8 \\ -3 \\ 5 \end{pmatrix}$$

Ptáme se tedy, zda existuje celočíselné řešení následující soustavy nerovnic:

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

ILP – celočíselné lineární programování

Jedním z řešení soustavy

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

je například $x_1 = -4$, $x_2 = 1$, $x_3 = 1$, tj.

$$x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}$$

neboť

$$\begin{aligned} 3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 &= -9 \leq 8 \\ -4 + 1 &= -3 \leq -3 \\ 2 \cdot (-4) + 1 &= -7 \leq 5 \end{aligned}$$

Pro tuto instanci je tedy odpověď Ano.

Příklady problémů

Problém

Vstup: Deterministické konečné automaty $\mathcal{A}_1$ a $\mathcal{A}_2$.

Otázka: Je $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$?

Problém

Vstup: Bezkontextové gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$.

Otázka: Je $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$?

Algoritmicky neřešitelné problémy

Problém, který není algoritmicky řešitelný, je **algoritmicky neřešitelný**.

Rozhodovací problém, který není rozhodnutelný, je **nerozhodnutelný**.

Kupodivu existuje řada algoritmických problémů (přesně definovaných), o kterých je dokázáno, že nejsou algoritmicky řešitelné.

Teorie vyčíslitelnosti — oblast teoretické informatiky, která se zabývá zkoumáním toho, které problémy jsou a které nejsou algoritmicky řešitelné.

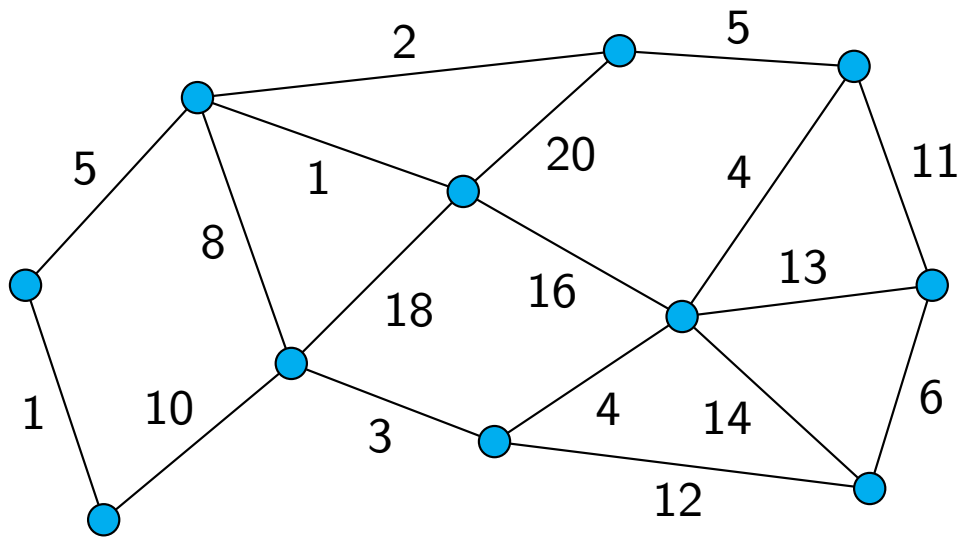
Teorie složitosti

Řada problémů je algoritmicky řešitelných, ale neexistují (nebo nejsou známy) efektivní algoritmy, které by je řešily:

TSP - Problém obchodního cestujícího

Vstup: Neorientovaný graf G s hranami ohodnocenými přirozenými čísly.

Výstup: Nejkratší uzavřená cesta, která projde všemi vrcholy a skončí v tom vrcholu, kde začíná.



Algoritmy

Co je to algoritmus?

Algoritmus

Algoritmus je mechanický postup skládající se z nějakých jednoduchých elementárních kroků, který pro nějaký zadaný **vstup** vyprodukuje nějaký **výstup**.

Algoritmus může být zadán:

- slovním popisem v přirozeném jazyce
- pseudokódem
- jako počítačový program v nějakém programovacím jazyce
- jako hardwarový obvod
- ...

Nutnost upřesnění pojmu „algoritmus“

Dosavadní definice pojmu algoritmus byla poněkud vágní.

Pokud bychom pro nějaký problém chtěli dokázat, že neexistuje algoritmus, který by daný problém řešil, tak by to s takovouto neurčitou definicí pojmu algoritmus asi nešlo.

Intuitivně chápeme, co by měl mít algoritmus za vlastnosti:

- Měl by se skládat z jednoduchých kroků, které je možno vykonávat „mechanicky“, bez porozumění problému.
- Objekty, se kterými algoritmus pracuje, i prováděné operace by měly být konečné.

Nějaká přesnější a konkrétnější definice pojmu algoritmus je také třeba k tomu, abychom mohli mluvit o výpočetní složitosti algoritmu:

- Pokud chceme např. mluvit o počtu operací, které musí algoritmus během výpočtu provést, je třeba nějak přesněji definovat, co přesně budeme považovat za tyto operace.

Výpočetní modely

Můžeme uvažovat různé druhy strojů, které mohou provádět nějaký algoritmus.

Tyto různé druhy strojů se mohou lišit v mnoha ohledech:

- jaké instrukce jsou schopny provádět
- jaký druh dat jsou schopny ukládat do své paměti a jak je tato paměť organizována
- ...

Různé druhy takovýchto strojů se označují jako různé **výpočetní modely**.

Výpočetní modely

Pro různé druhy výpočetních modelů můžeme zkoumat například:

- jaké algoritmické problémy jsou schopny řešit či jaké jazyky jsou schopny rozpoznávat.
- jak efektivně jsou schopny realizovat různé algoritmy
- jakým způsobem může určitý druh stroje simulovat činnost jiného druhu stroje
- jak při takové simulaci narůstá počet instrukcí provedených daným strojem
- ...

Příklad: Popis algoritmu pomocí **pseudokódu**:

Algoritmus: Algoritmus pro nalezení největšího prvku v poli

```
FIND-MAX (A, n):
    k := 0
    for i := 1 to n − 1 do
        if A[i] > A[k] then
            k := i
    return A[k]
```

Algoritmy

Algoritmus

- zpracovává **vstup**
- generuje **výstup**

Z hlediska analýzy toho, jak daný algoritmus funguje, většinou není příliš podstatný rozdíl v tom, jestli algoritmus:

- čte vstupní data z nějakého vstupního zařízení (např. ze souboru na disku, z klávesnice, apod.)
- zapisuje data na nějaké výstupní zařízení (např. do souboru, na obrazovku, apod.)

nebo

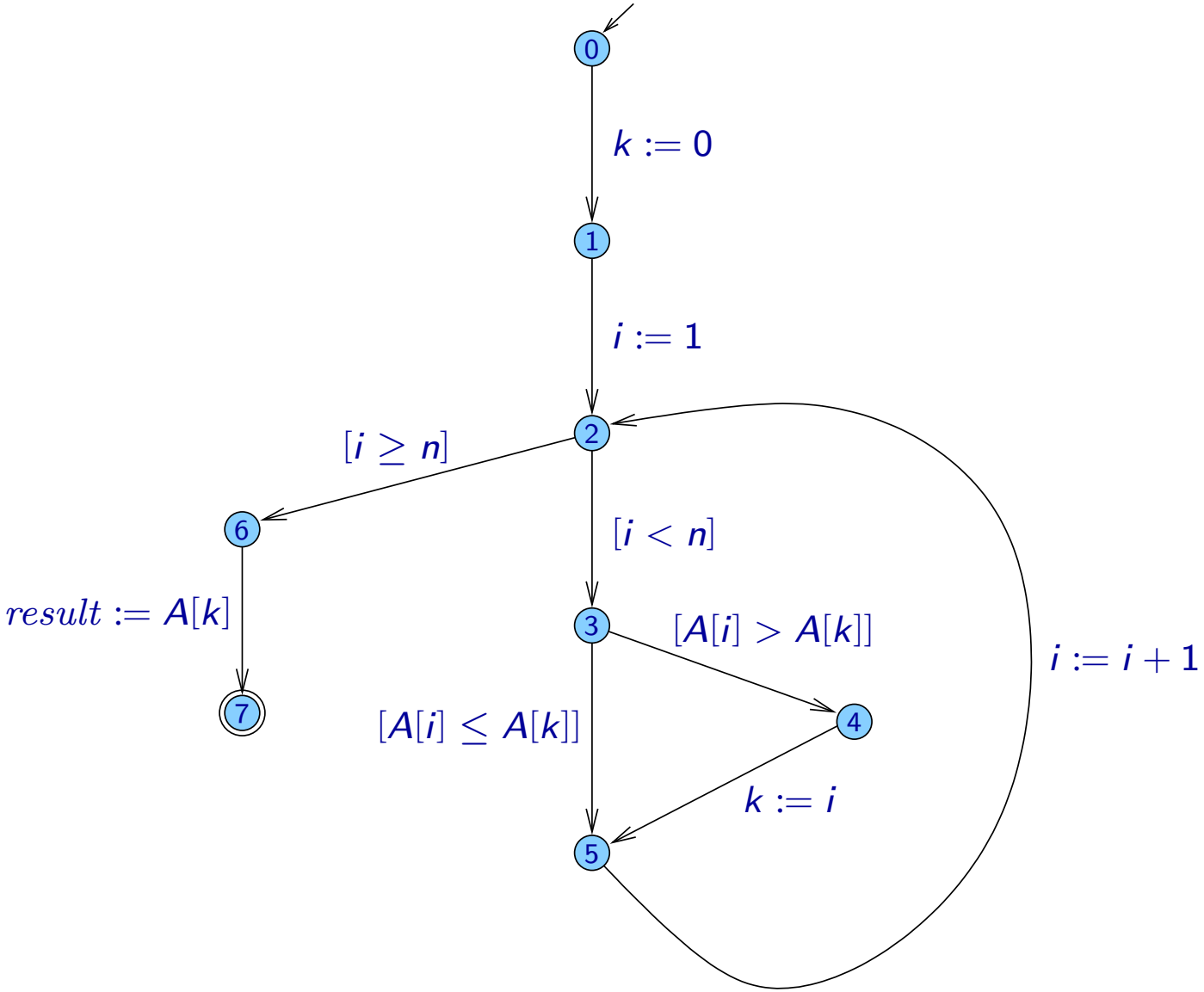
- čte vstupní data z paměti (např. jsou mu předány jako parametry)
- zapisuje data na do paměti (např. je vrátí jako návratovou hodnotu)

Řídící tok

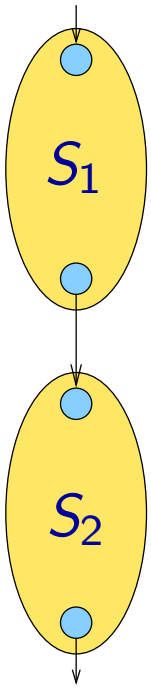
Instrukce lze zhruba rozdělit na dvě skupiny:

- instrukce přímo pracující s daty:
  - přiřazení
  - vyhodnocení hodnot výrazů v podmínkách
  - čtení vstupu, zápis na výstup
  - ...
- instrukce ovlivňující **řídící tok** — určují, které instrukce se budou provádět, v jakém pořadí, apod.:
  - větvení (if, switch, ...)
  - cykly (while, do .. while, for, ...)
  - uspořádání instrukcí do bloků
  - návraty z podpogramů (return, ...)
  - ...

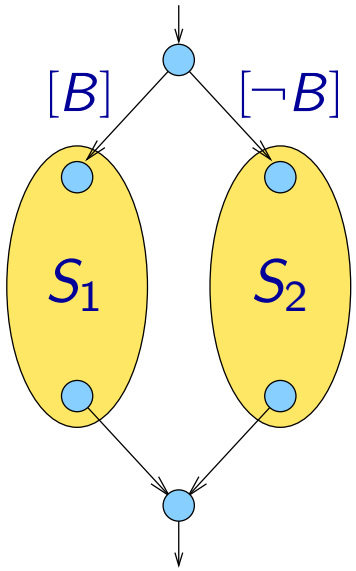
Graf řídícího toku



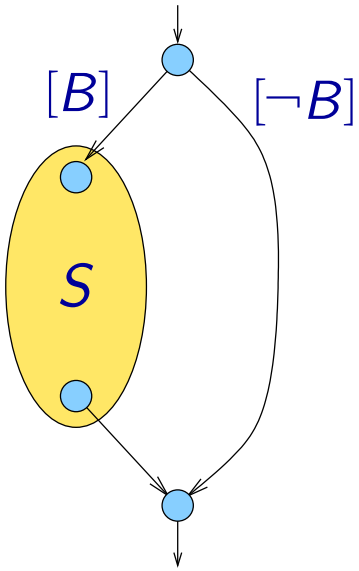
Některé základní konstrukce strukturovaného programování



$S_1; S_2$

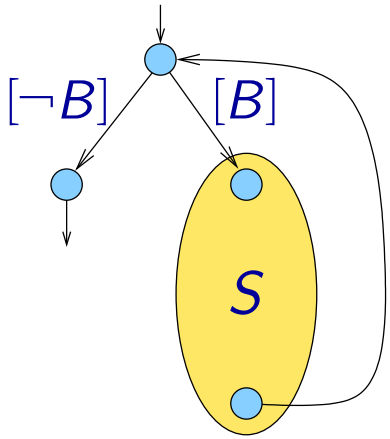


if B **then** S_1 **else** S_2

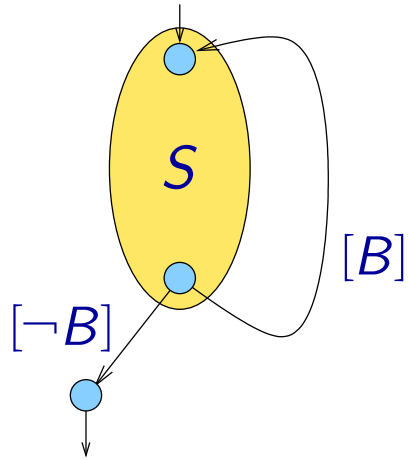


if B **then** S

Některé základní konstrukce strukturovaného programování

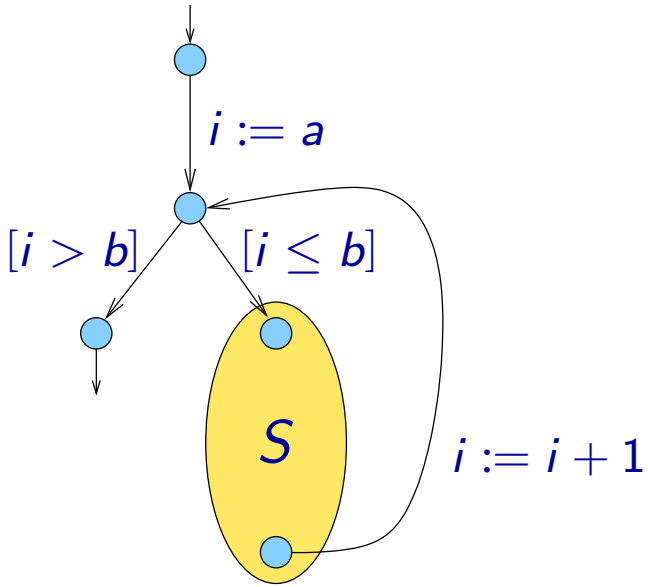


while B **do** S



do S **while** B

Některé základní konstrukce strukturovaného programování



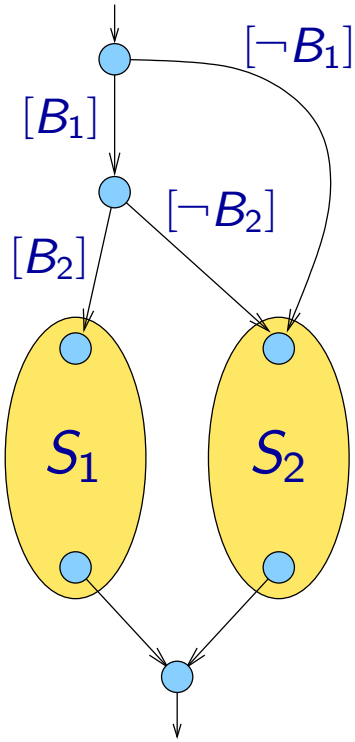
$i := a$
while $i \leq b$ **do**
  S
  $i := i + 1$

for $i := a$ **to** b **do** S

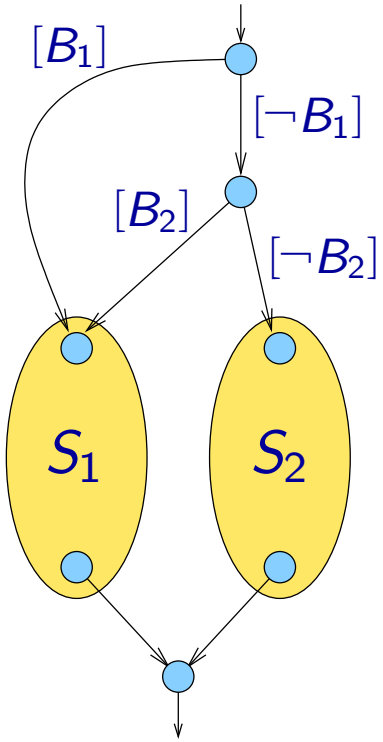
Některé základní konstrukce strukturovaného programování

Zkrácené vyhodnocování složených podmínek, např.:

while $i < n$ **and** $A[i] > x$ **do** ...



if B_1 **and** B_2 **then** S_1 **else** S_2



if B_1 **or** B_2 **then** S_1 **else** S_2

Řídící tok realizovaný pomocí goto

- **goto** ℓ — **nepodmíněný skok**
- **if** B **then goto** ℓ — **podmíněný skok**

Příklad:

0: $k := 0$
1: $i := 1$
2: **goto** *6*
3: **if** $A[i] \leq A[k]$ **then goto** *5*
4: $k := i$
5: $i := i + 1$
6: **if** $i < n$ **then goto** *3*
7: **return** $A[k]$

Řídící tok realizovaný pomocí goto

- **goto** ℓ — **nepodmíněný skok**
- **if** B **then goto** ℓ — **podmíněný skok**

Příklad:

$$
\begin{array}{rl}
\mathit{start}: & k := 0 \\
& i := 1 \\
& \textbf{goto } \mathit{L3} \\
\mathit{L1}: & \textbf{if } A[i] \leq A[k] \textbf{ then goto } \mathit{L2} \\
& k := i \\
\mathit{L2}: & i := i + 1 \\
\mathit{L3}: & \textbf{if } i < n \textbf{ then goto } \mathit{L1} \\
& \textbf{return } A[k]
\end{array}
$$

Vyhodnocení složitých výrazů

Vyhodnocení složitého výrazu, jako třeba

$$A[i + s] := (B[3 * j + 1] + x) * y + 8$$

může být na nižší úrovni nahrazeno posloupností jednodušších příkazů, jako třeba

$$\begin{aligned} t_1 &:= i + s \\ t_2 &:= 3 * j \\ t_2 &:= t_2 + 1 \\ t_3 &:= B[t_2] \\ t_3 &:= t_3 + x \\ t_3 &:= t_3 * y \\ t_3 &:= t_3 + 8 \\ A[t_1] &:= t_3 \end{aligned}$$

Výpočet algoritmu

Algoritmus je vykonáván strojem — může to být například:

- skutečný počítač — vykonává instrukce strojového kódu
- virtuální stroj — vykonává instrukce bytekódu
- nějaký idealizovaný matematický model počítače
- ...

Stroj může být:

- jednoúčelový — vykonává jen jeden algoritmus
- obecnější — algoritmus dostává ve formě **programu**

Stroj pracuje po **krocích**.

Algoritmus během výpočtu zpracovává konkrétní **vstup**.

Výpočet algoritmu

Během výpočtu si stroj musí pamatovat:

- která instrukce se právě provádí
- obsah pracovní paměti

Podle typu stroje je určeno:

- s jakým typem dat stroj pracuje
- jak jsou tato data v paměti organizována

Podle typu algoritmu a typu analýzy, kterou chceme provádět, se můžeme rozhodnout, zda má smysl mezi obsah paměti zahranout i místa

- odkud se čtou vstupní data
- kam se zapisují výstupní data

Výpočet algoritmu

Konfigurace — popis celkového stavu stroje v nějakém okamžiku během výpočtu

Příklad: Konfigurace tvaru

$$(q, mem)$$

kde

- q — aktuální řídící stav
- mem — představuje aktuální obsah paměti stroje — jaké hodnoty jsou momentálně přiřazeny jednotlivým proměnným.

Příklad obsahu paměti mem:

$$\langle A\colon [3,8,1,3,6],\quad n\colon 5,\quad i\colon 1,\quad k\colon 0,\quad result\colon ?\rangle$$

Výpočet algoritmu

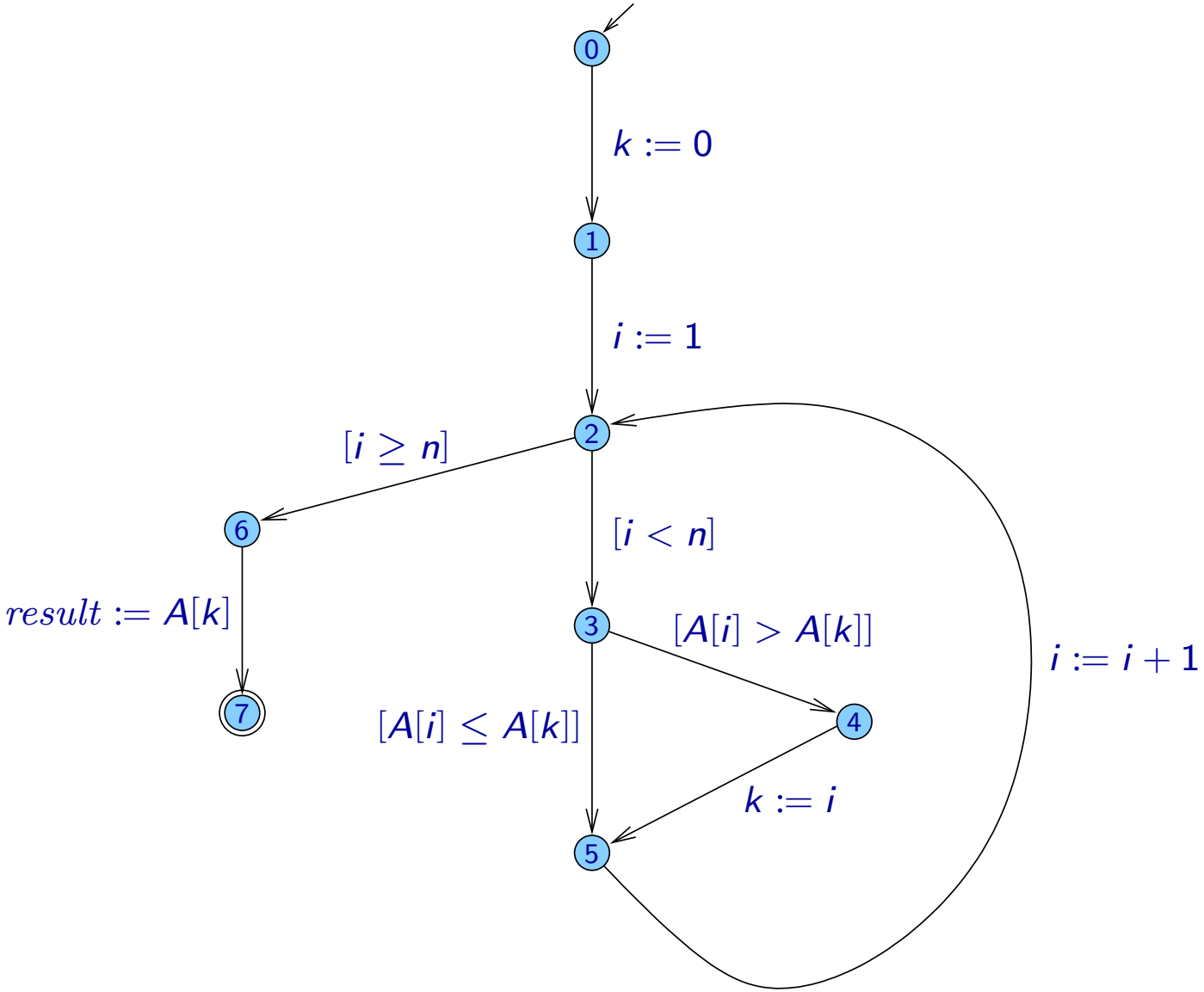
Příklad konfigurace:

$$(2,\ \langle A\colon [3,8,1,3,6],\quad n\colon 5,\quad i\colon 1,\quad k\colon 0,\quad result\colon ?\rangle)$$

Výpočet stroje $\mathcal{M}$ provádějícího algoritmus Alg, kde zpracovává vstup w, je posloupnost konfigurací.

- Začíná se v **počáteční konfiguraci**.
- Každým krokem stroj přechází z jedné konfigurace do další.
- Výpočet končí v **koncové konfiguraci**.

Výpočet algoritmu



Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3, 8, 1, 3, 6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0,\ \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1,\ \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ result\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle)
\end{array}
$$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ result\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{13}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle)
\end{array}
$$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ result\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{13}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{14}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle)
\end{array}
$$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ result\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{13}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{14}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{15}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ result\colon ?\rangle)
\end{array}
$$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ result\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{13}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{14}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{15}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{16}\colon & (6,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ result\colon ?\rangle)
\end{array}
$$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

$$
\begin{array}{rl}
\alpha_0: & (0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle) \\
\alpha_1: & (1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle) \\
\alpha_2: & (2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle) \\
\alpha_3: & (3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle) \\
\alpha_4: & (4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle) \\
\alpha_5: & (5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_6: & (2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_7: & (3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_8: & (5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_9: & (2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{10}: & (3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{11}: & (5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{12}: & (2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{13}: & (3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{14}: & (5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{15}: & (2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{16}: & (6, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } ?\rangle) \\
\alpha_{17}: & (7, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } 8\rangle)
\end{array}
$$

Výpočet algoritmu

Provedením instrukce I se přejde z konfigurace α do konfigurace α':

$$\alpha \xrightarrow{I} \alpha'$$

Výpočet může být:

- **Konečný**:

$$\alpha_0 \xrightarrow{I_0} \alpha_1 \xrightarrow{I_1} \alpha_2 \xrightarrow{I_2} \alpha_3 \xrightarrow{I_3} \alpha_4 \xrightarrow{I_4} \quad \cdots \quad \xrightarrow{I_{t-2}} \alpha_{t-1} \xrightarrow{I_{t-1}} \alpha_t$$

  kde α_t je buď koncová konfigurace nebo konfigurace, kde došlo k chybě a není možné pokračovat

- **Nekonečný**:

$$\alpha_0 \xrightarrow{I_0} \alpha_1 \xrightarrow{I_1} \alpha_2 \xrightarrow{I_2} \alpha_3 \xrightarrow{I_3} \alpha_4 \xrightarrow{I_4} \quad \cdots$$

Výpočet algoritmu

Výpočet je možné popsat dvěma různými způsoby:

- jako posloupnost konfigurací $\alpha_0, \alpha_1, \alpha_2, \ldots$
- jako posloupnost provedených instrukcí $I_0, I_1, I_2, \ldots$

Stroje RAM

Stroj RAM

Stroj RAM (Random Access Machine) je idealizovaný model počítače.

Skládá se z těchto částí:

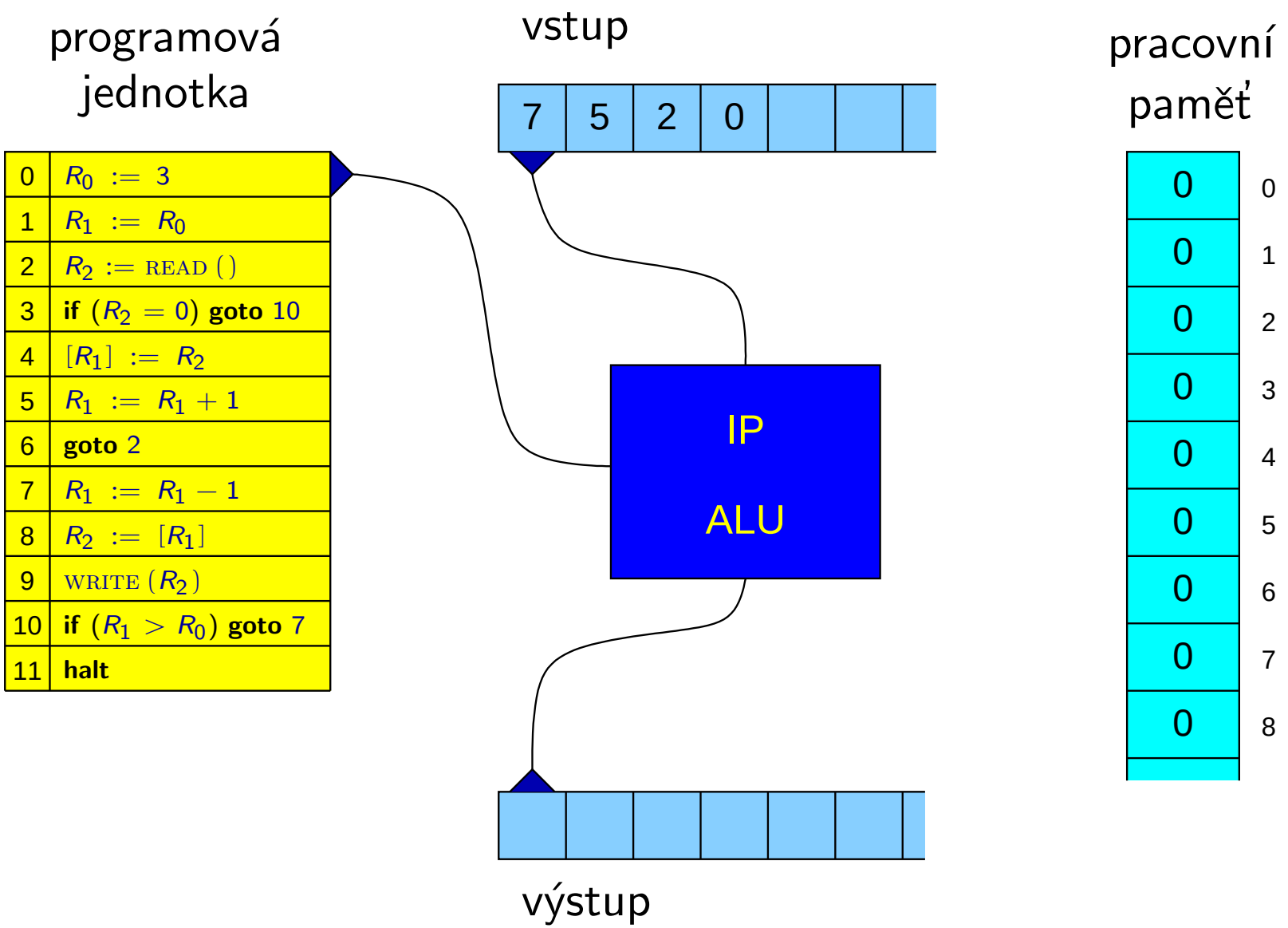
- **Programová jednotka** – obsahuje program stroje RAM a ukazatel na právě prováděnou instrukci
- **Pracovní paměť** tvořená buňkami očíslovanými $0, 1, 2, \ldots$

  Tyto buňky buňky budeme označovat $R_0, R_1, R_2, \ldots$

  Obsah buněk je možno číst i do nich zapisovat.
- **Vstupní páska** – je z ní možné pouze číst
- **Výstupní páska** – je na ni možno pouze zapisovat

Buňky paměti i vstupní a výstupní pásky obsahují jako hodnoty celá čísla (tj. prvky množiny $\mathbb{Z}$).

Stroj RAM


programová jednotka
vstup
pracovní paměť
0 $R_0 := 3$
1 $R_1 := R_0$
2 $R_2 := \text{READ}()$
3 **if** $(R_2 = 0)$ **goto** 10
4 $[R_1] := R_2$
5 $R_1 := R_1 + 1$
6 **goto** 2
7 $R_1 := R_1 - 1$
8 $R_2 := [R_1]$
9 $\text{WRITE}(R_2)$
10 **if** $(R_1 > R_0)$ **goto** 7
11 **halt**
7 5 2 0
IP
ALU
0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8
výstup

Stroj RAM

Přehled instrukcí:

$R_i := c$	– přiřazení konstanty
$R_i := R_j$	– přiřazení
$R_i := [R_j]$	– load (čtení z paměti)
$[R_i] := R_j$	– store (zápis do paměti)
$R_i := R_j \ op \ R_k$	– aritmetické instrukce, $op \in \{+, -, *, /\}$
nebo $R_i := R_j \ op \ c$	
if $(R_i \ rel \ R_j)$ **goto** ℓ	– podmíněný skok, $rel \in \{=, \neq, \leq, \geq, <, >\}$
nebo **if** $(R_i \ rel \ c)$ **goto** ℓ	
goto ℓ	– nepodmíněný skok
$R_i := \text{READ}()$	– čtení ze vstupu
$\text{WRITE}(R_i)$	– zápis na výstup
halt	– zastavení programu

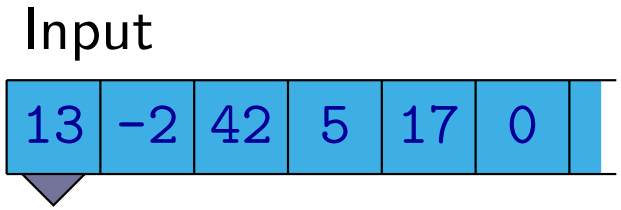
Stroj RAM

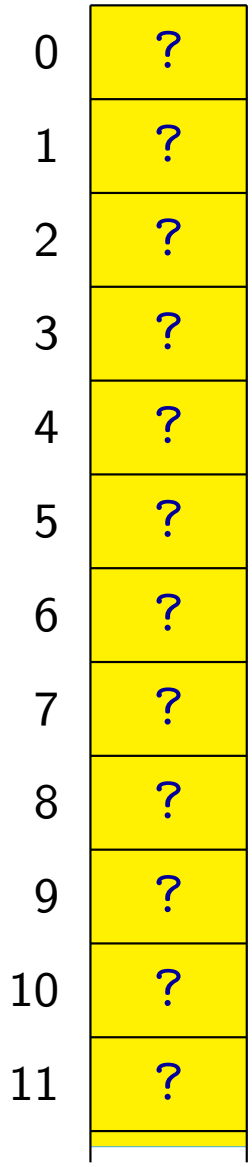
Příklady instrukcí:

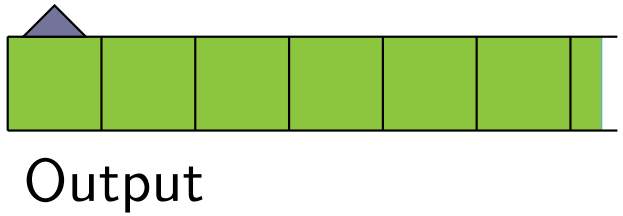
$R_5 := 42$	– přiřazení konstanty
$R_{12} := R_3$	– přiřazení
$R_8 := [R_2]$	– load (čtení z paměti)
$[R_{15}] := R_9$	– store (zápis do paměti)
$R_7 := R_3 + R_6$	– aritmetická instrukce
$R_{18} := R_{18} - 1$	– aritmetická instrukce
if $(R_4 \geq R_1)$ **goto** 2801	– podmíněný skok
if $(R_2 \neq 0)$ **goto** 3581	– podmíněný skok
goto 537	– nepodmíněný skok
$R_{23} := \text{READ}()$	– čtení ze vstupu
$\text{WRITE}(R_{17})$	– zápis na výstup
halt	– zastavení programu

Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt







Stroj RAM

$R_0 := 3$

$R_1 := R_0$

$L_1 : R_2 := \text{READ}()$

if $(R_2 = 0)$ **goto** L_3

$[R_1] := R_2$

$R_1 := R_1 + 1$

goto L_1

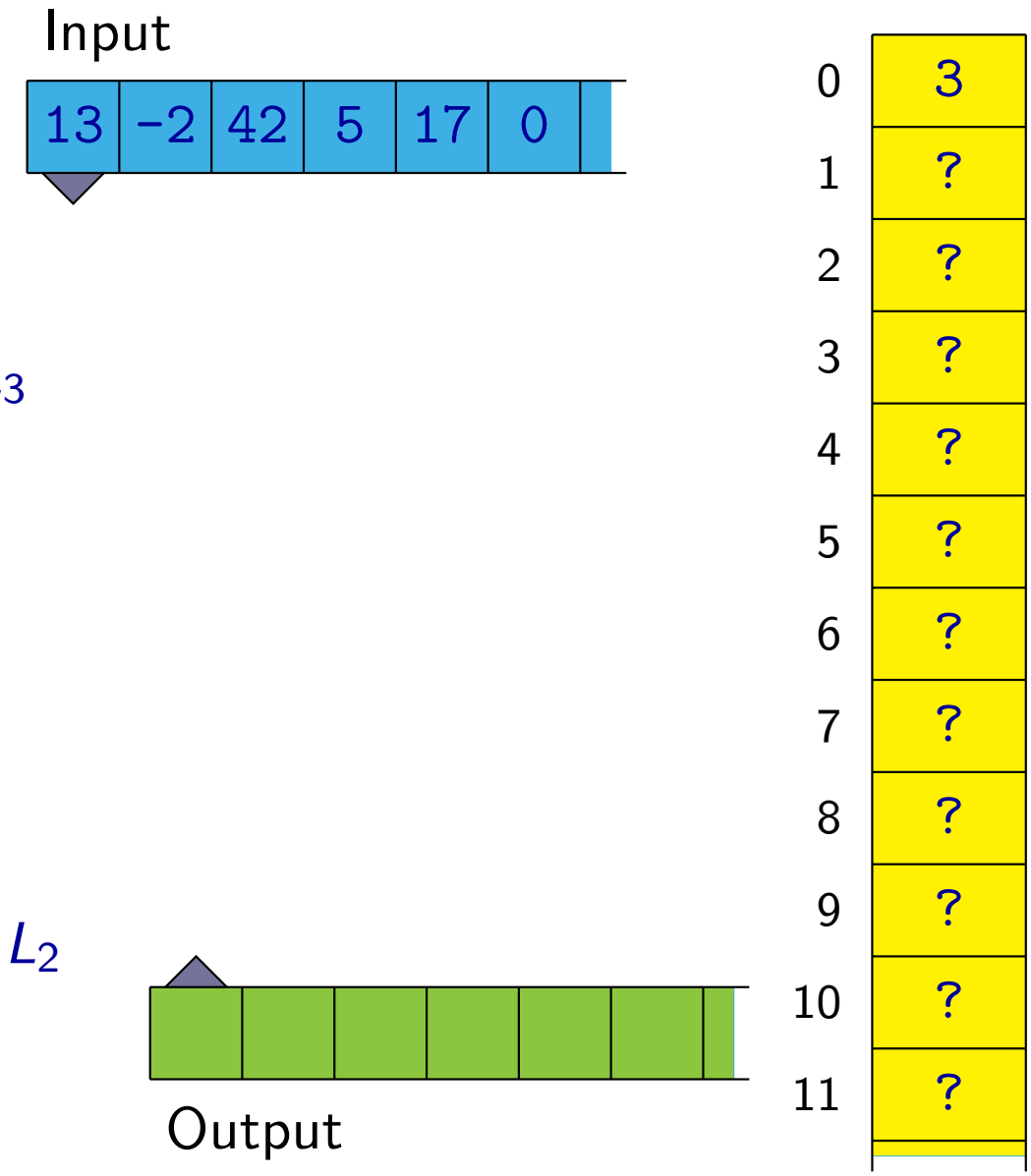
$L_2 : R_1 := R_1 - 1$

$R_2 := [R_1]$

$\text{WRITE}(R_2)$

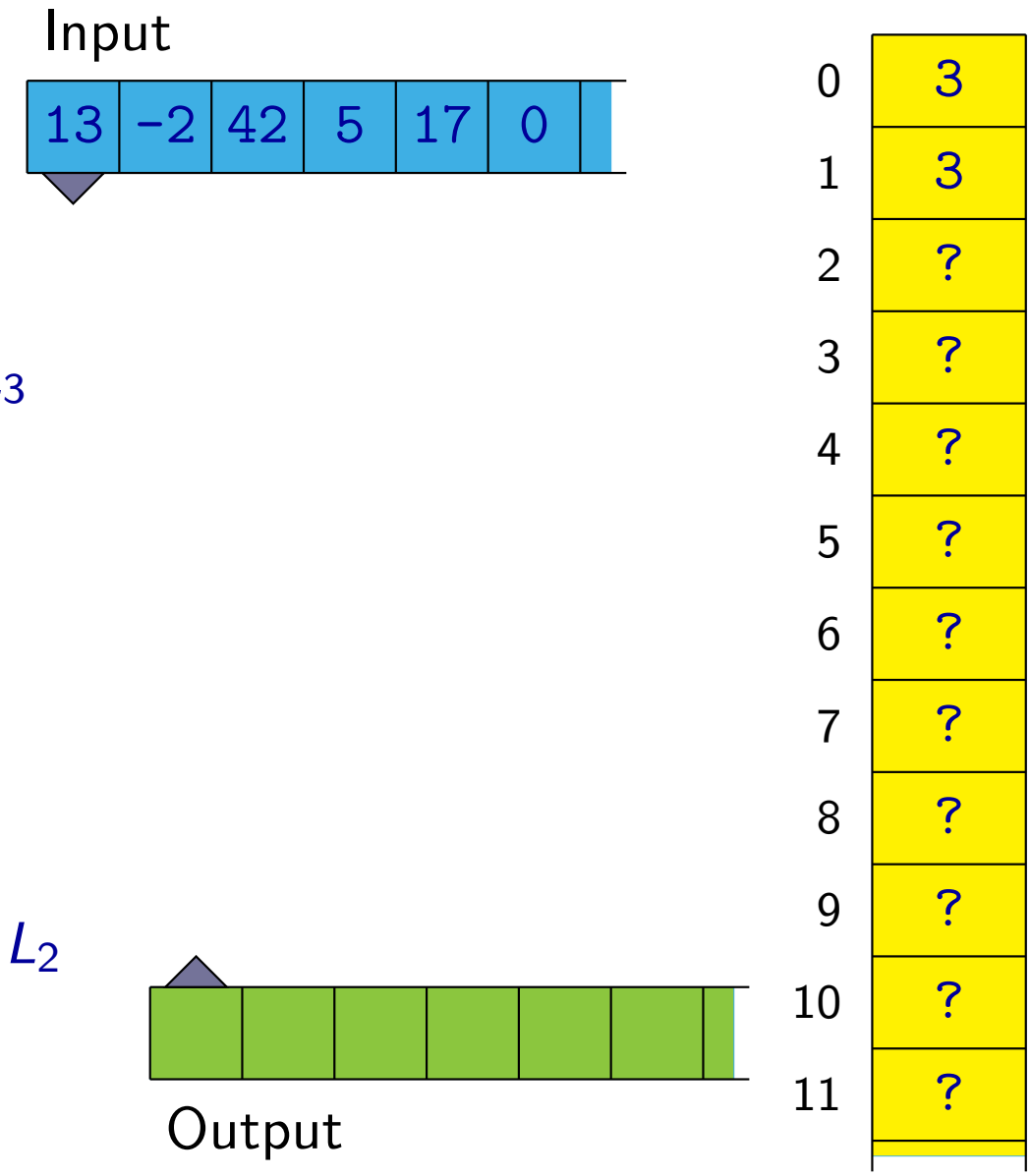
L_3 : **if** $(R_1 > R_0)$ **goto** L_2

halt



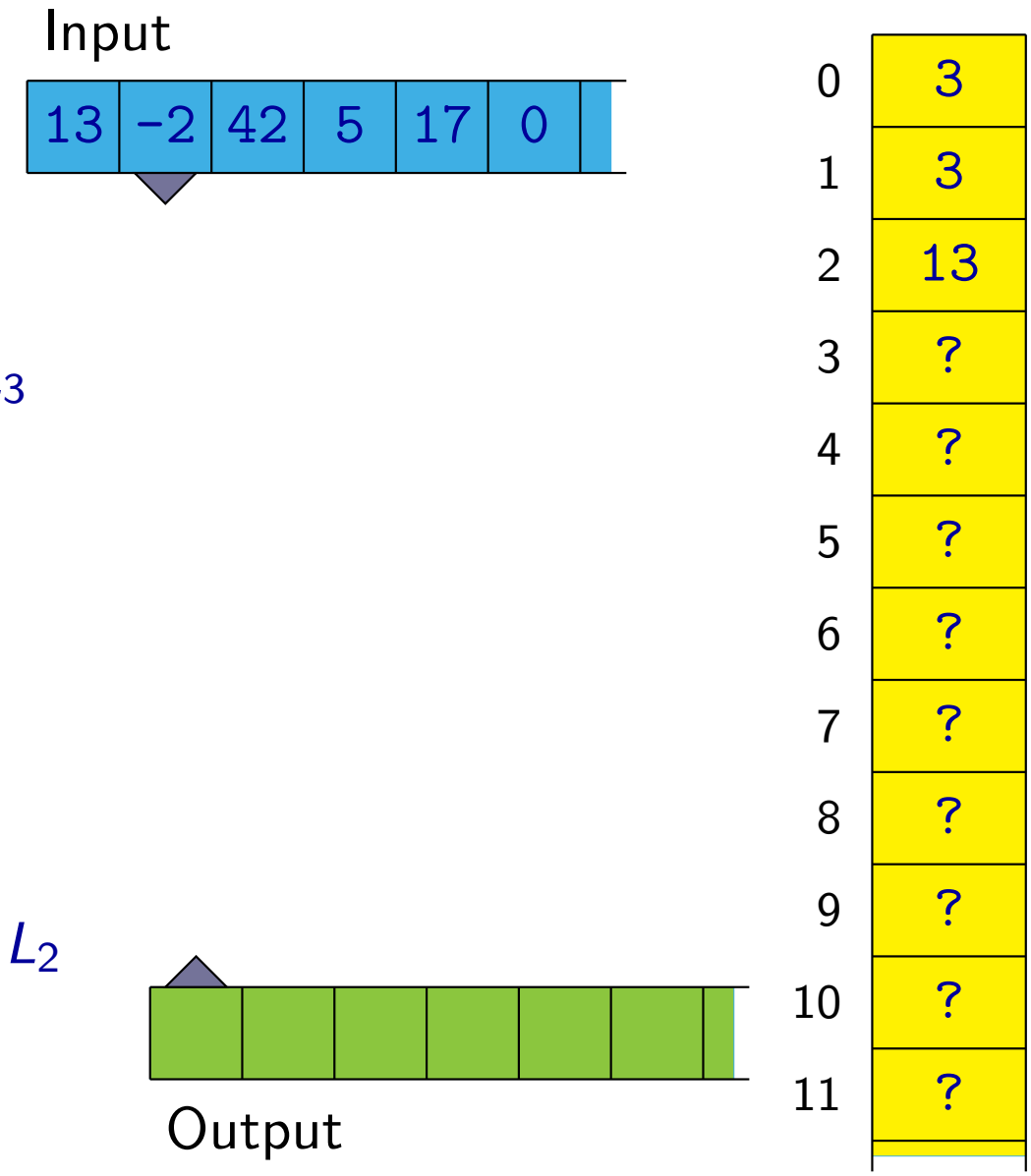
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



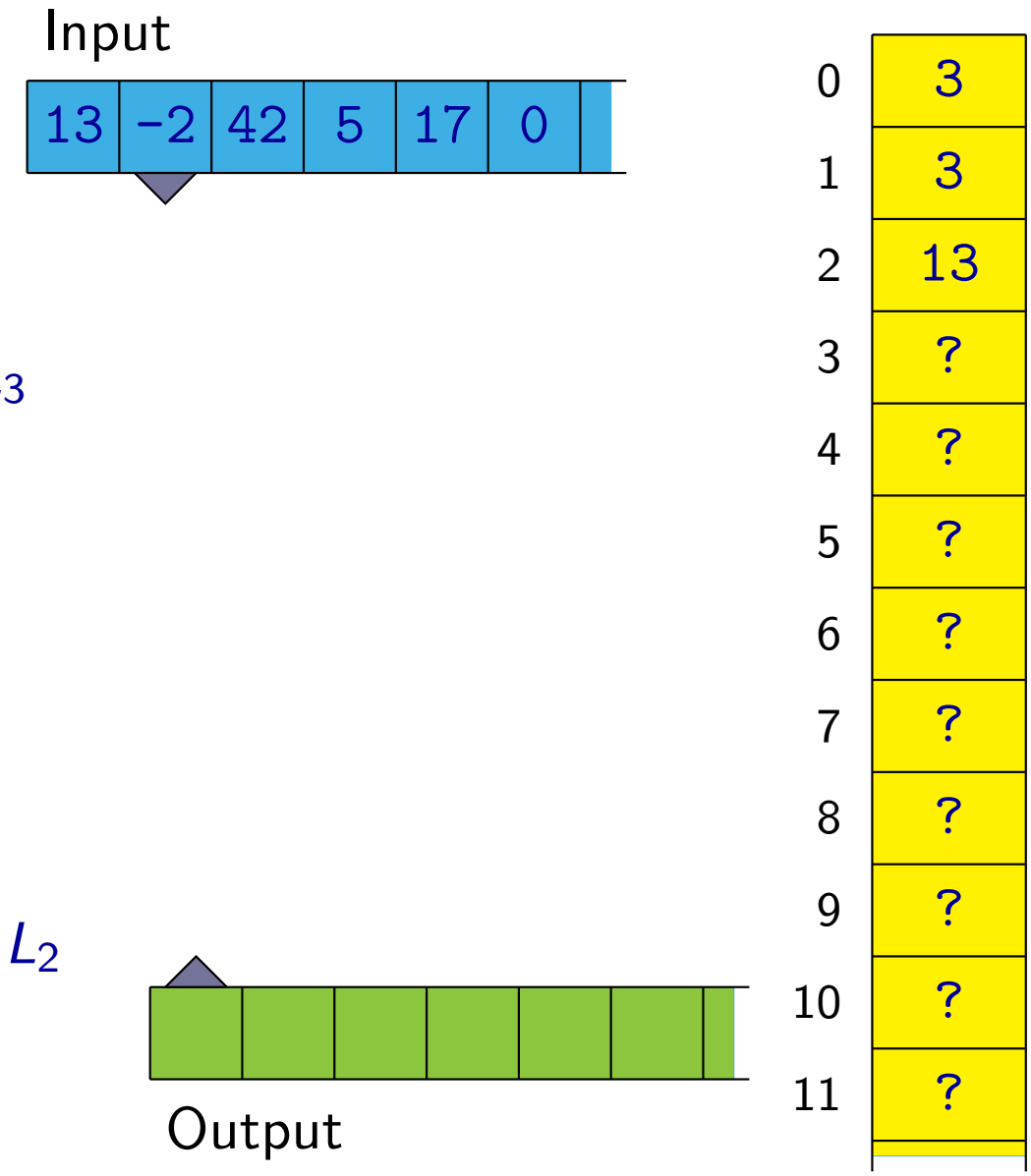
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



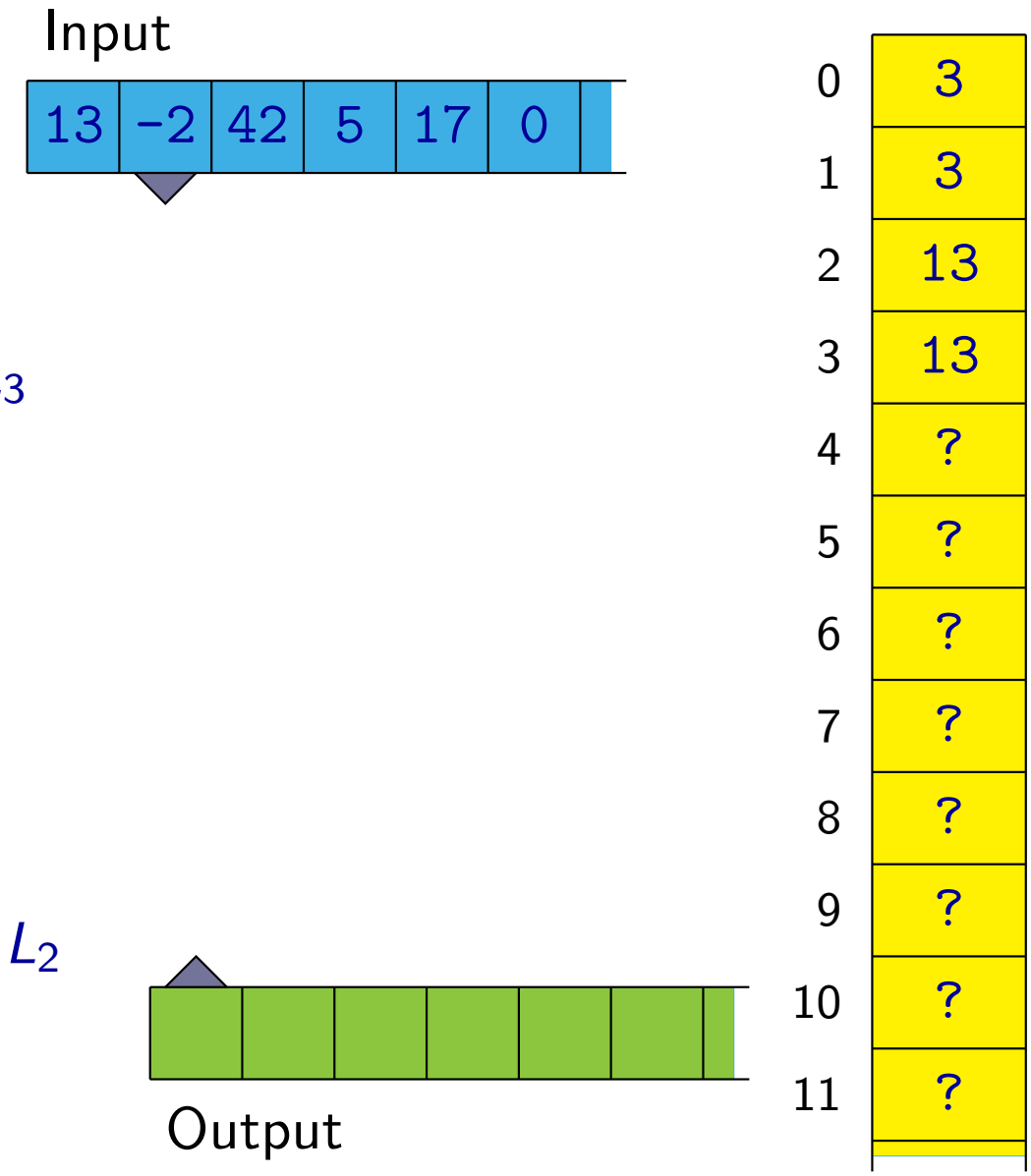
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



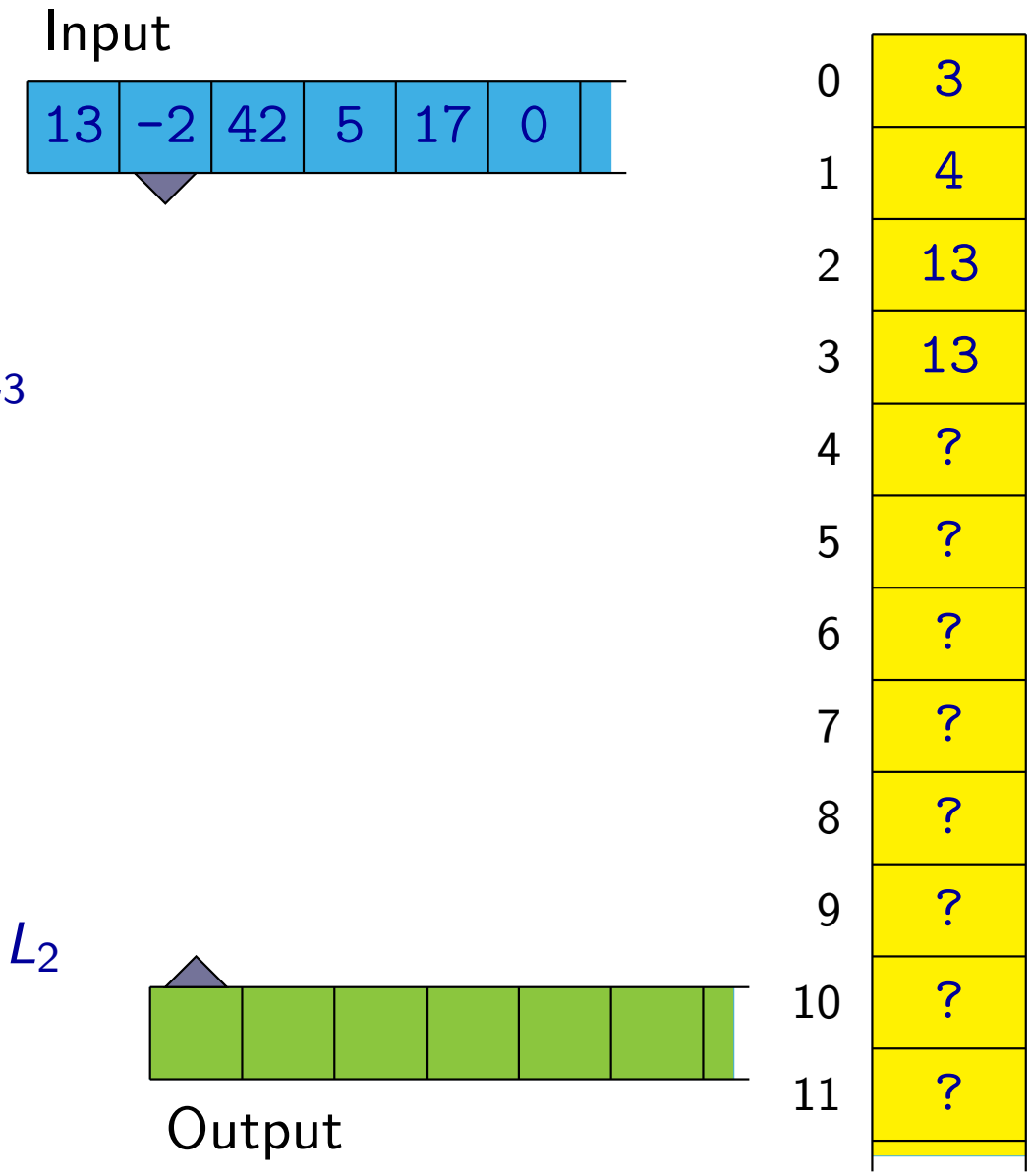
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt



Stroj RAM

$R_0 := 3$

$R_1 := R_0$

$L_1 : R_2 := \text{READ}()$

if $(R_2 = 0)$ **goto** L_3

$[R_1] := R_2$

$R_1 := R_1 + 1$

goto L_1

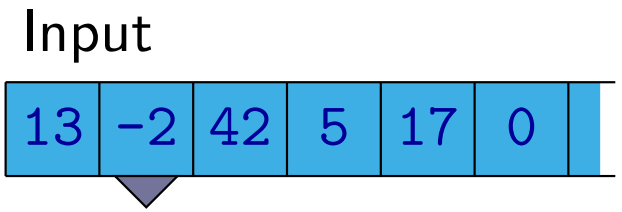
$L_2 : R_1 := R_1 - 1$

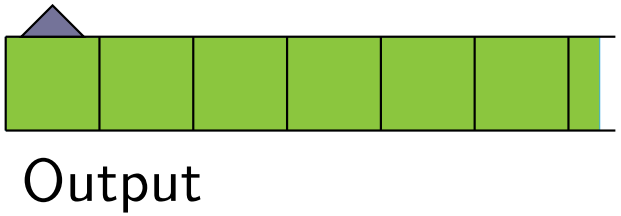
$R_2 := [R_1]$

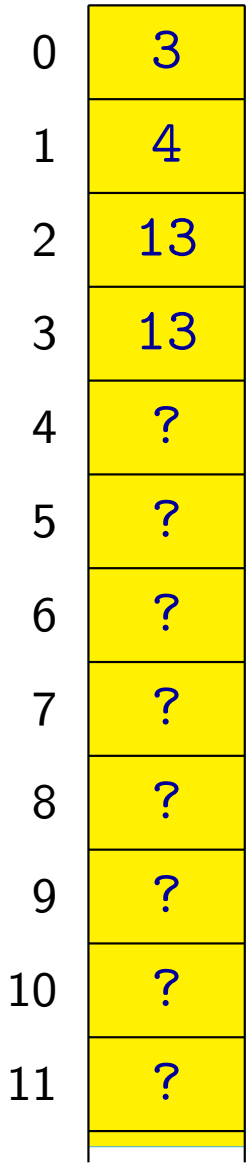
$\text{WRITE}(R_2)$

$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2

halt

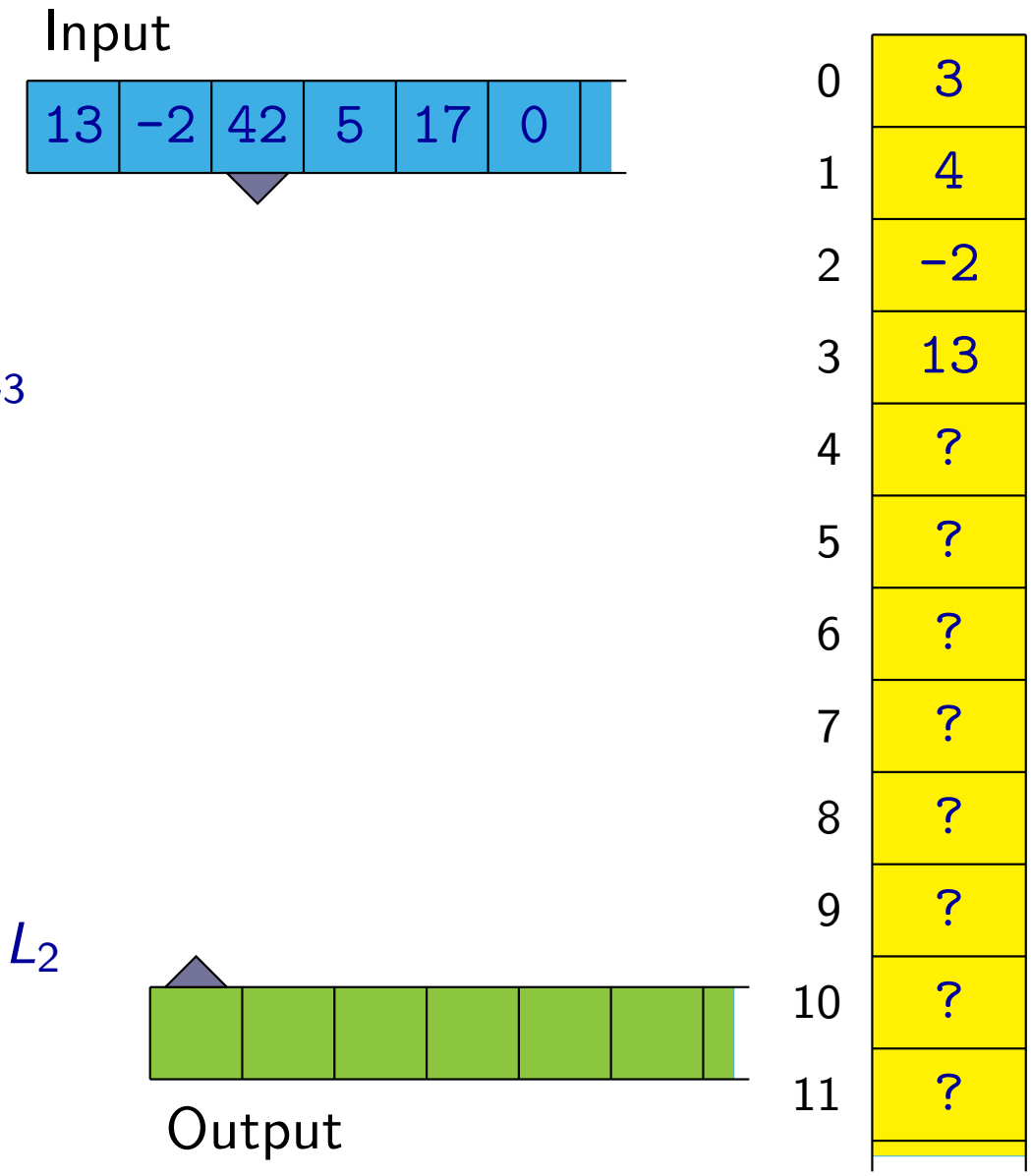






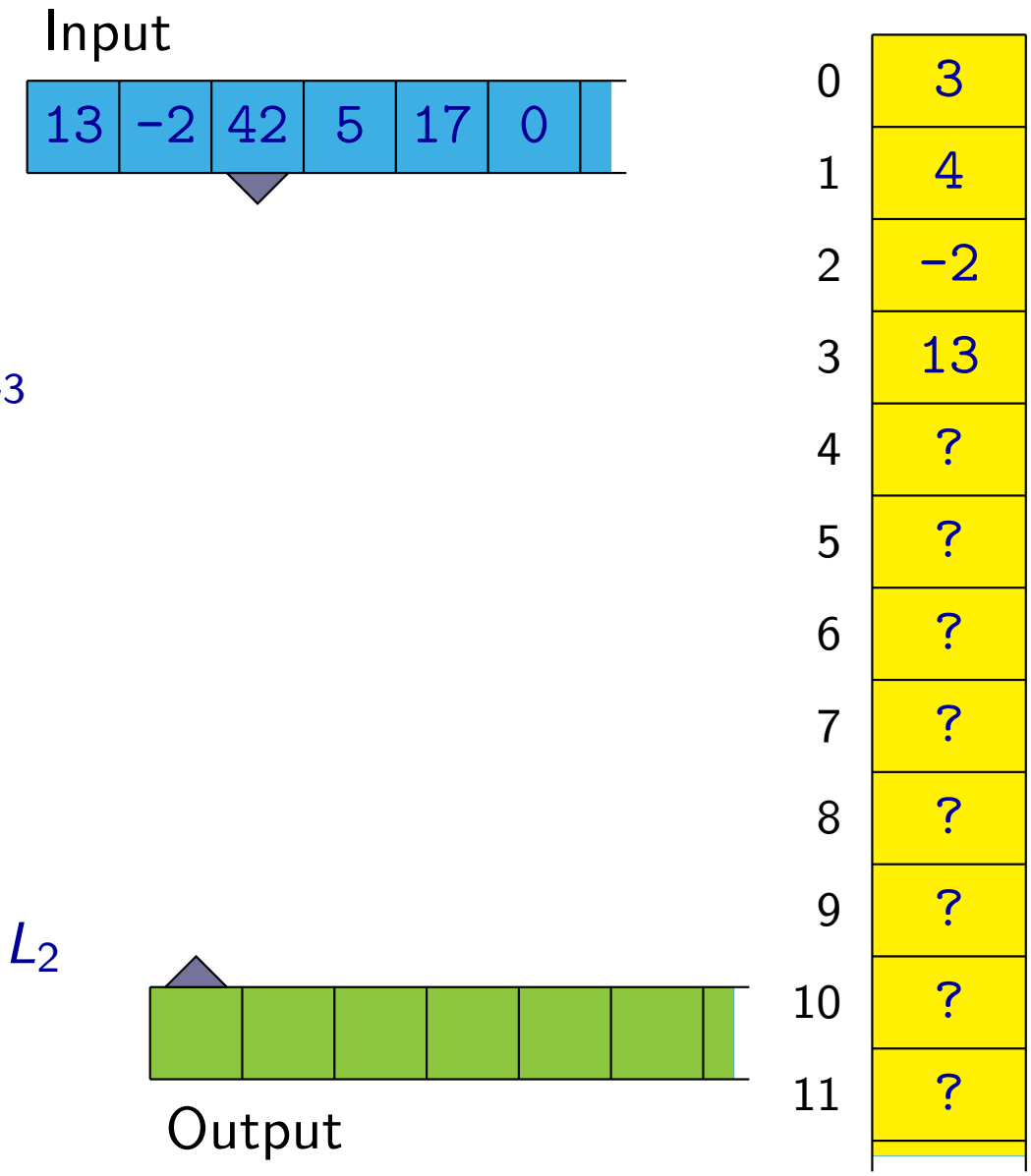
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



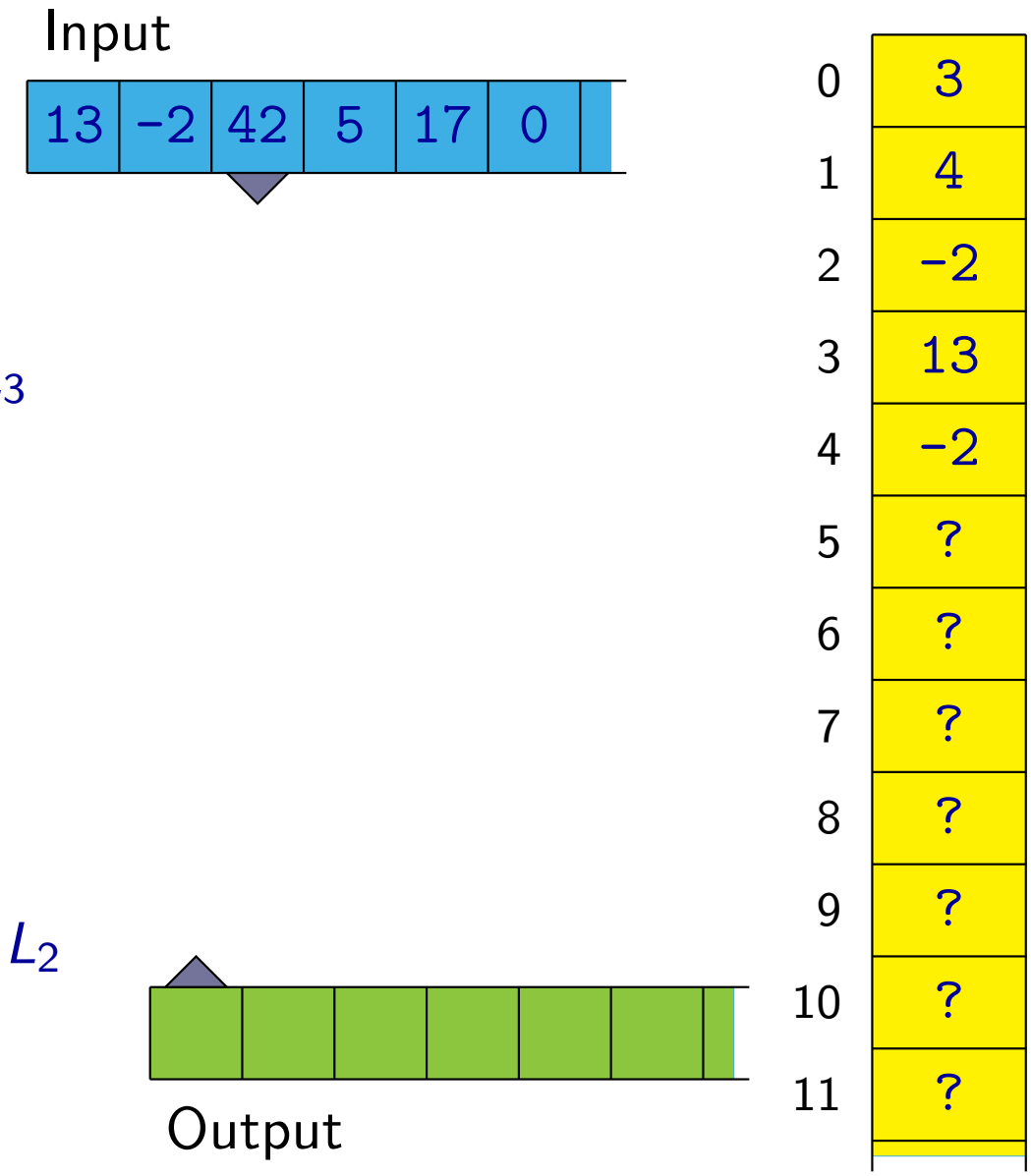
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



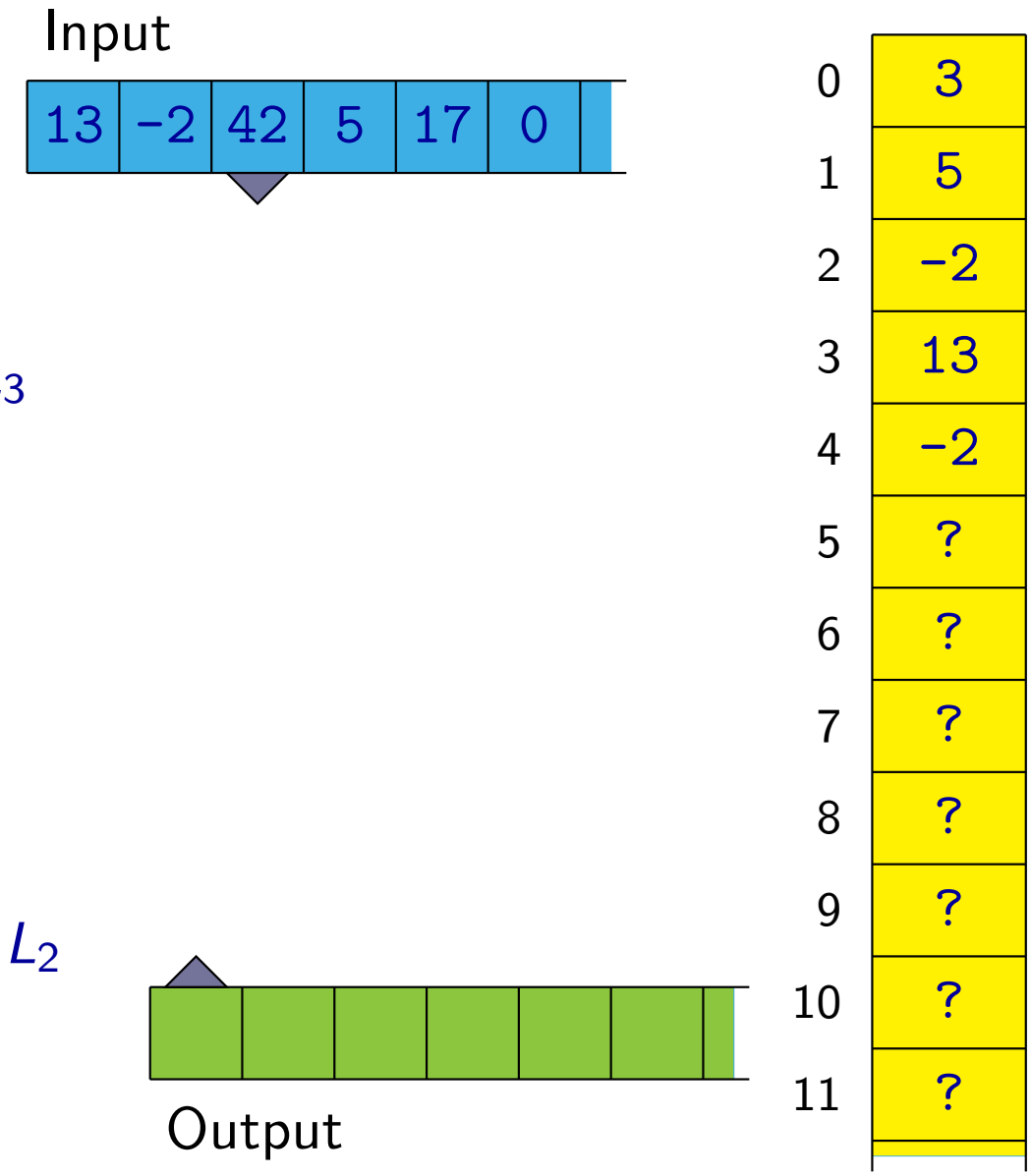
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt



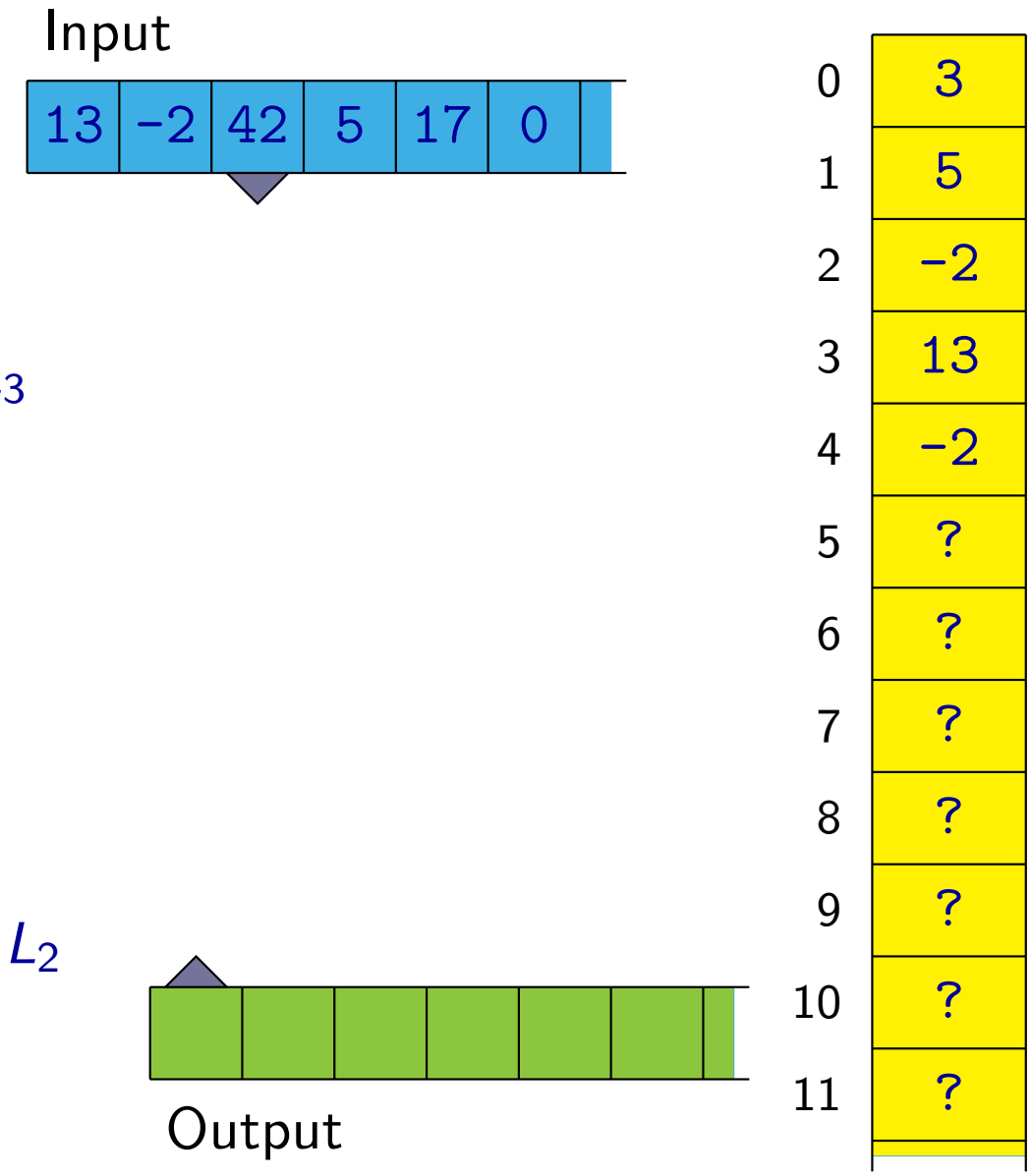
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt



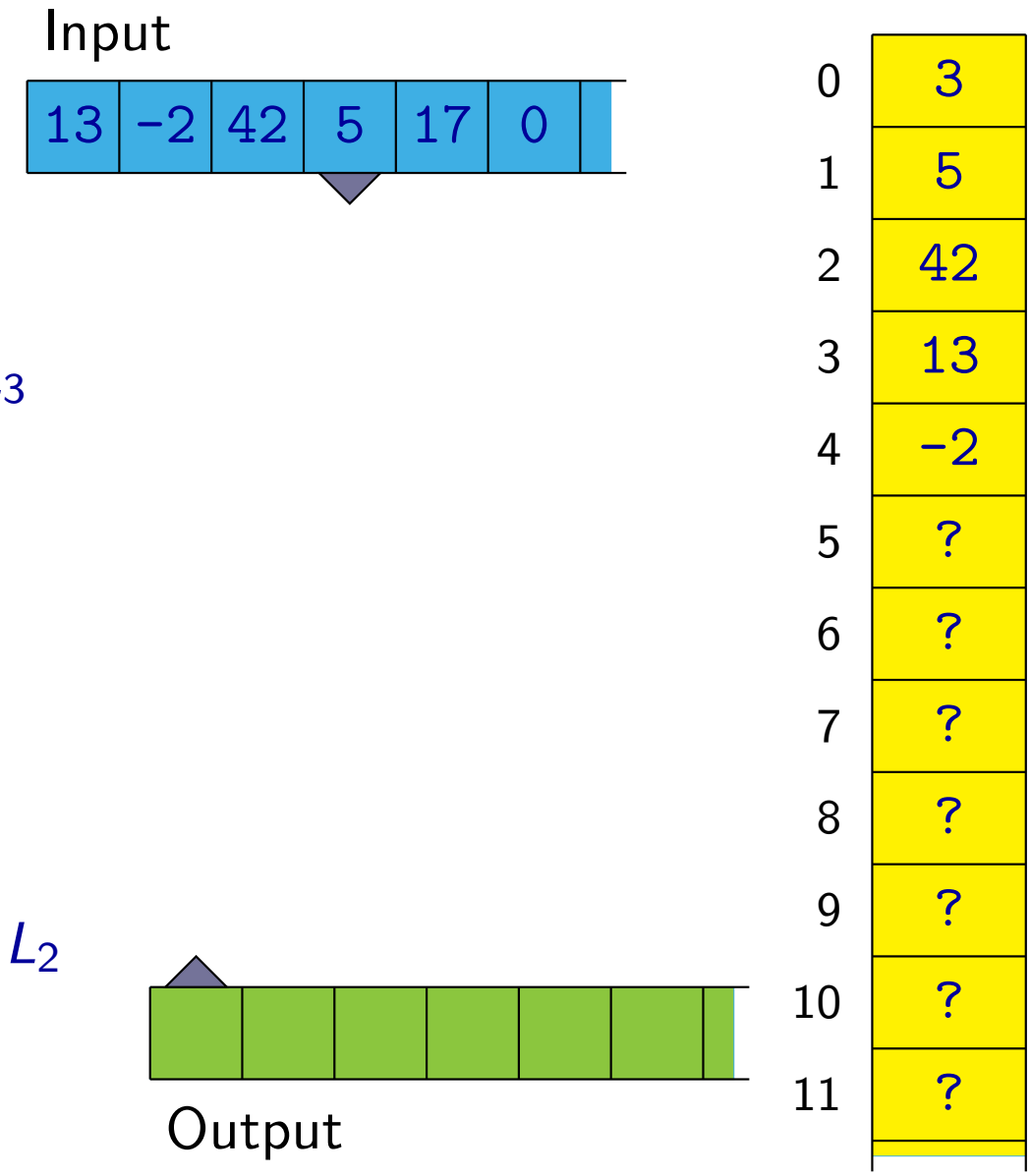
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



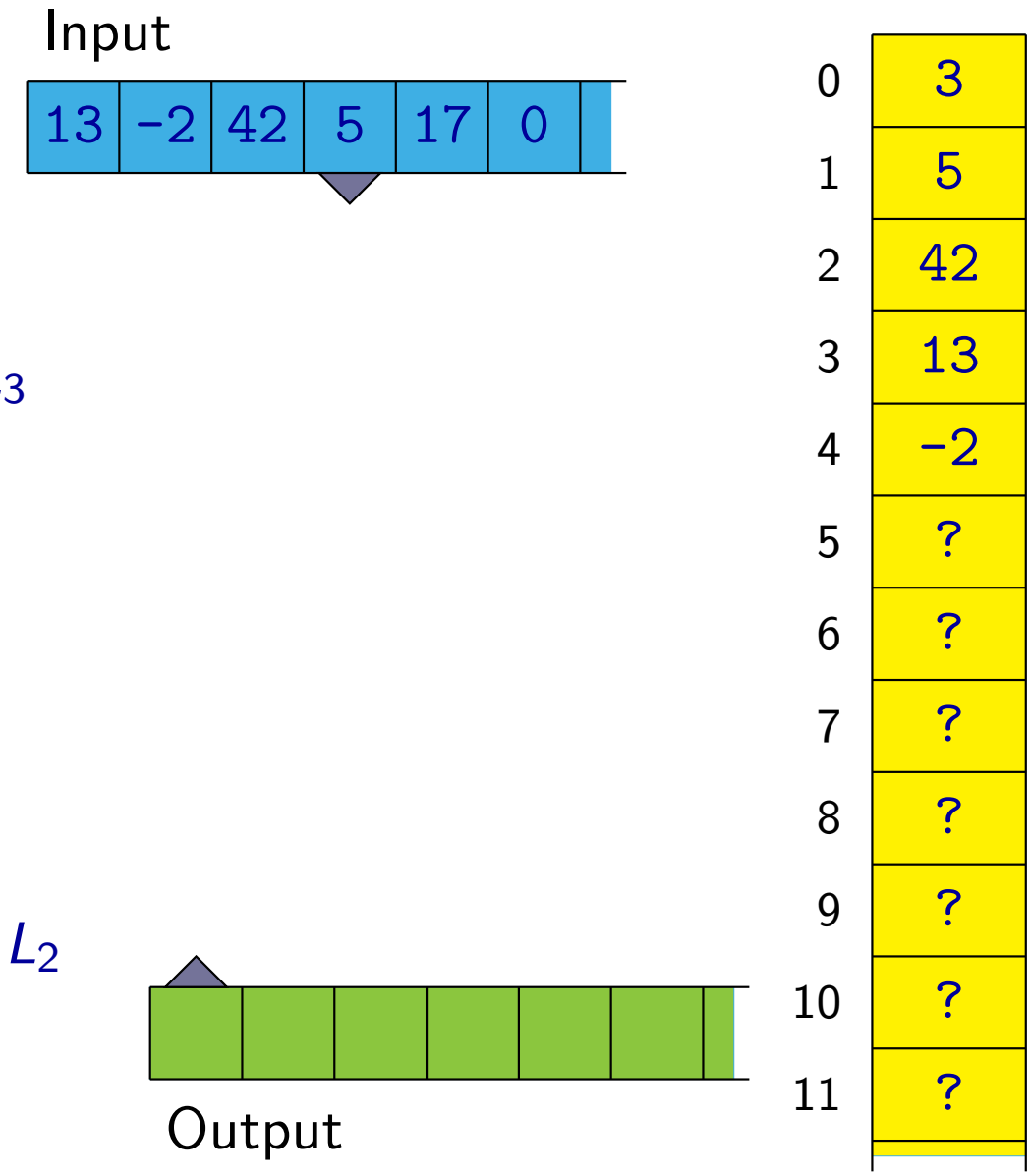
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



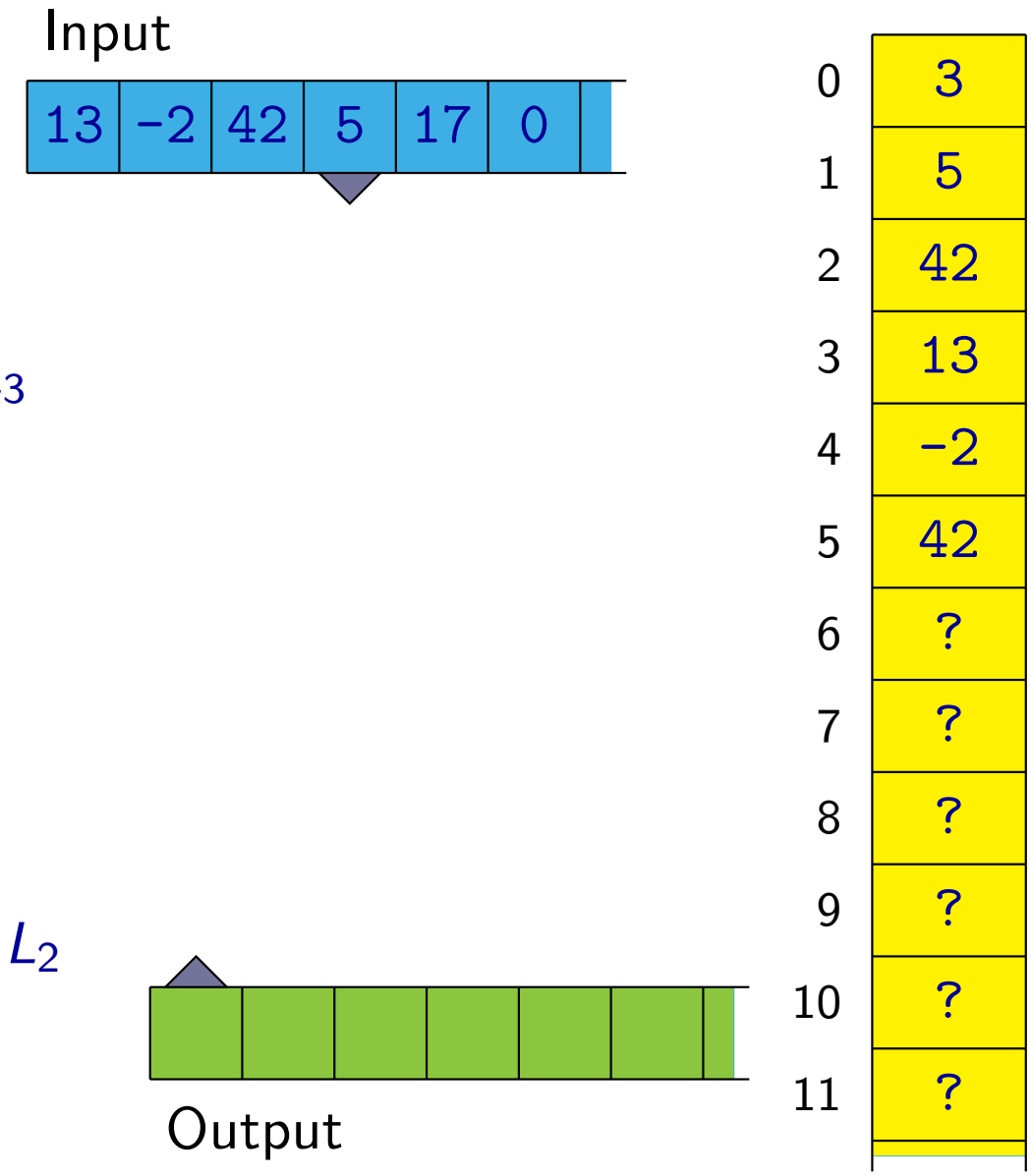
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



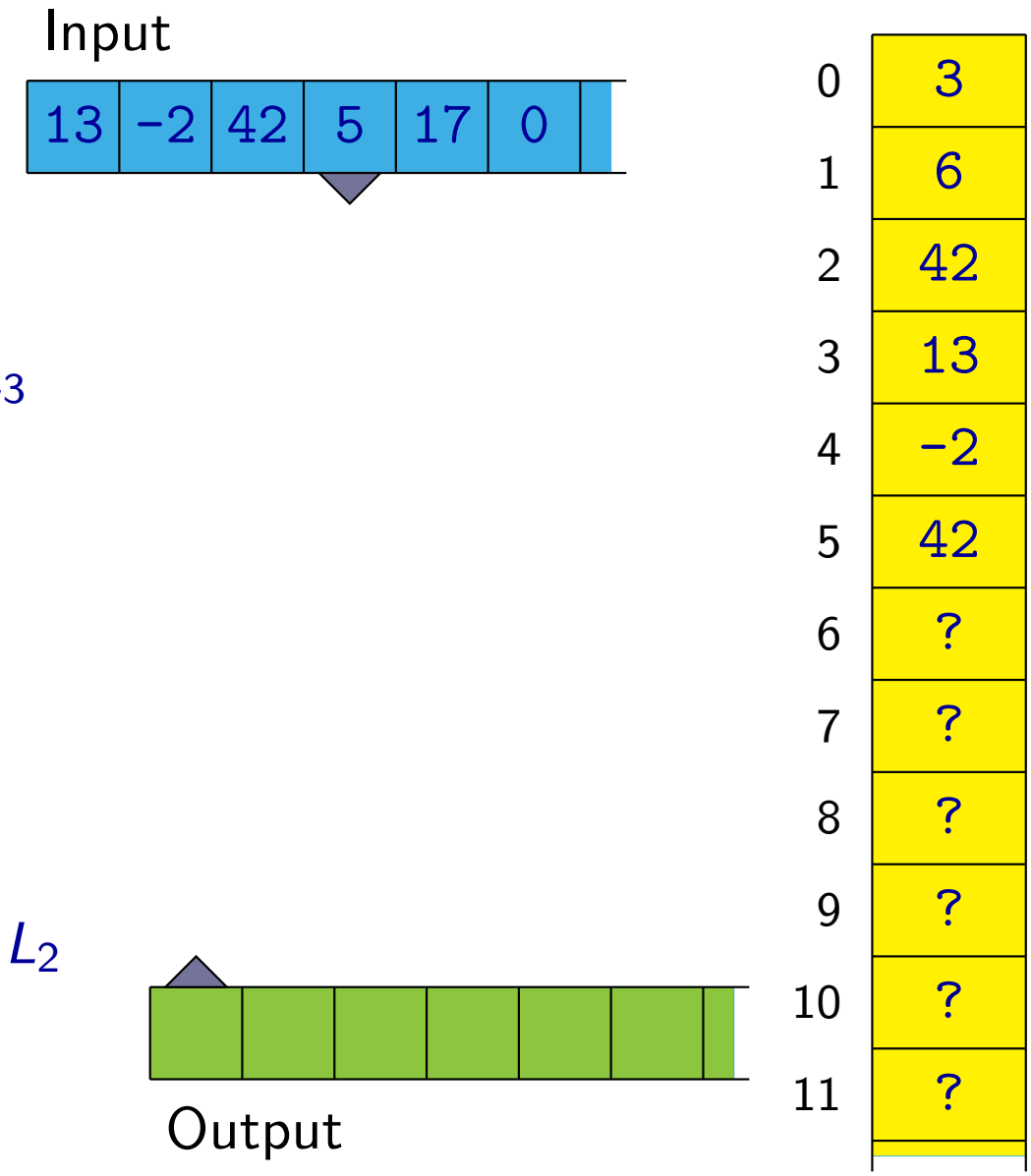
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt



Stroj RAM

$R_0 := 3$

$R_1 := R_0$

$L_1 : R_2 := \text{READ}()$

if $(R_2 = 0)$ **goto** L_3

$[R_1] := R_2$

$R_1 := R_1 + 1$

goto L_1

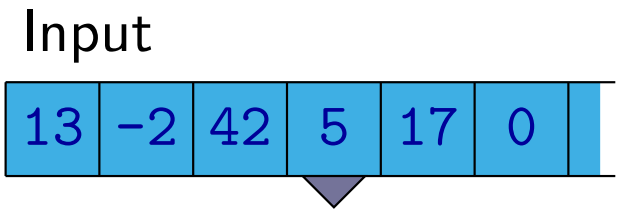
$L_2 : R_1 := R_1 - 1$

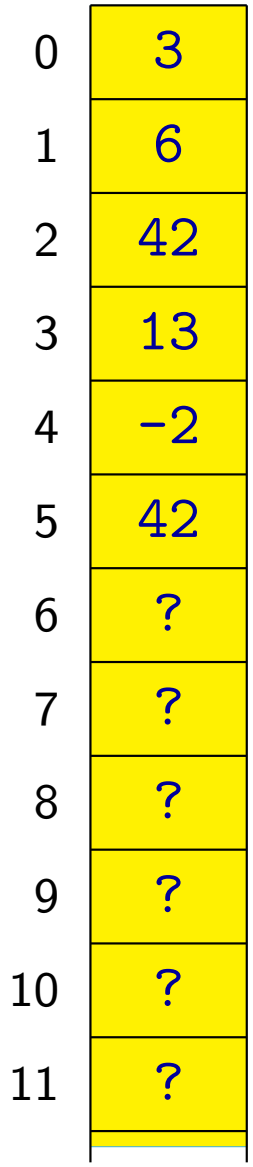
$R_2 := [R_1]$

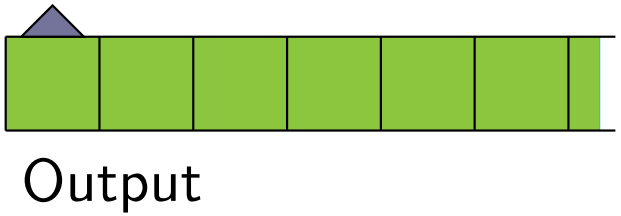
$\text{WRITE}(R_2)$

$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2

halt

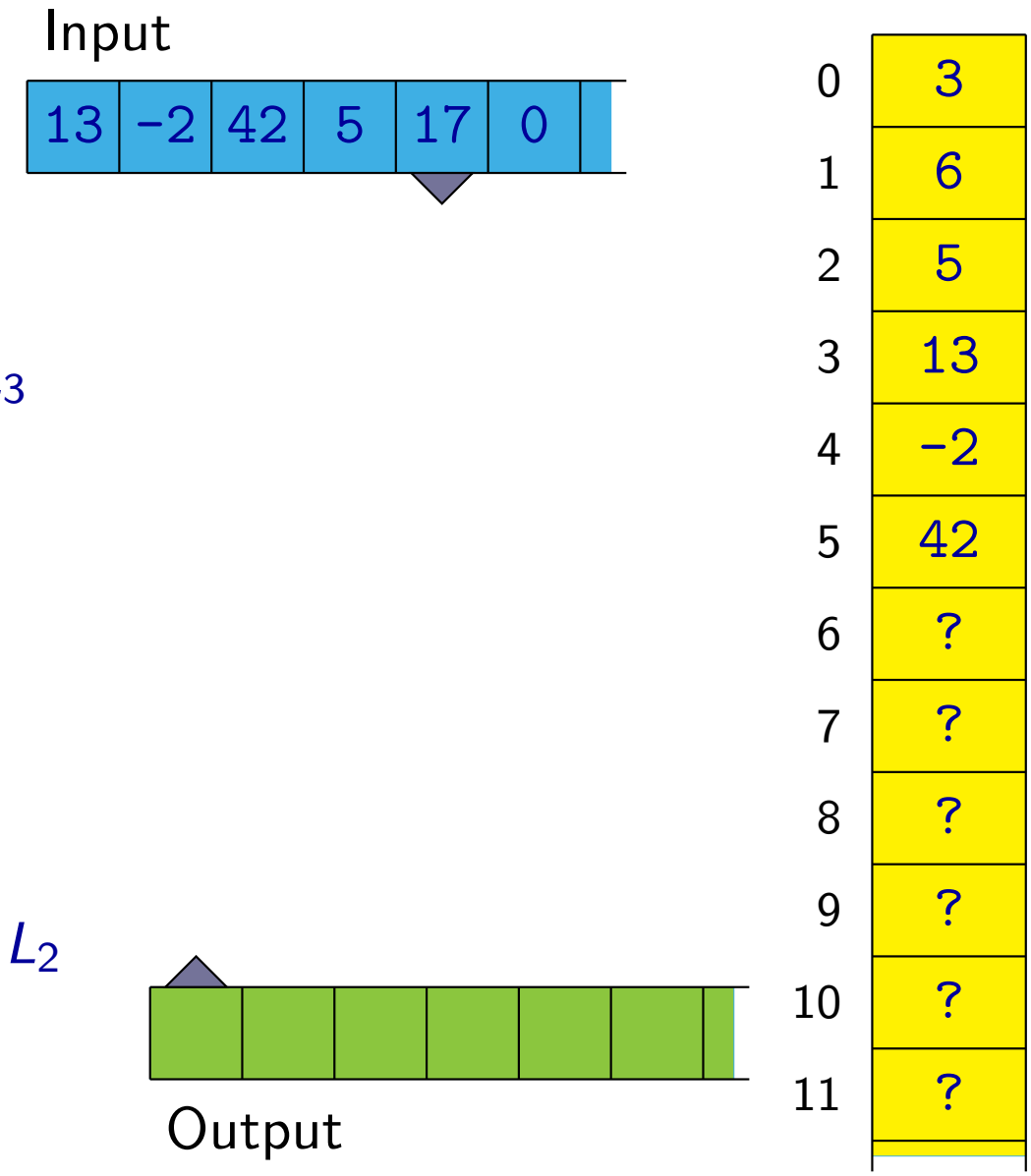






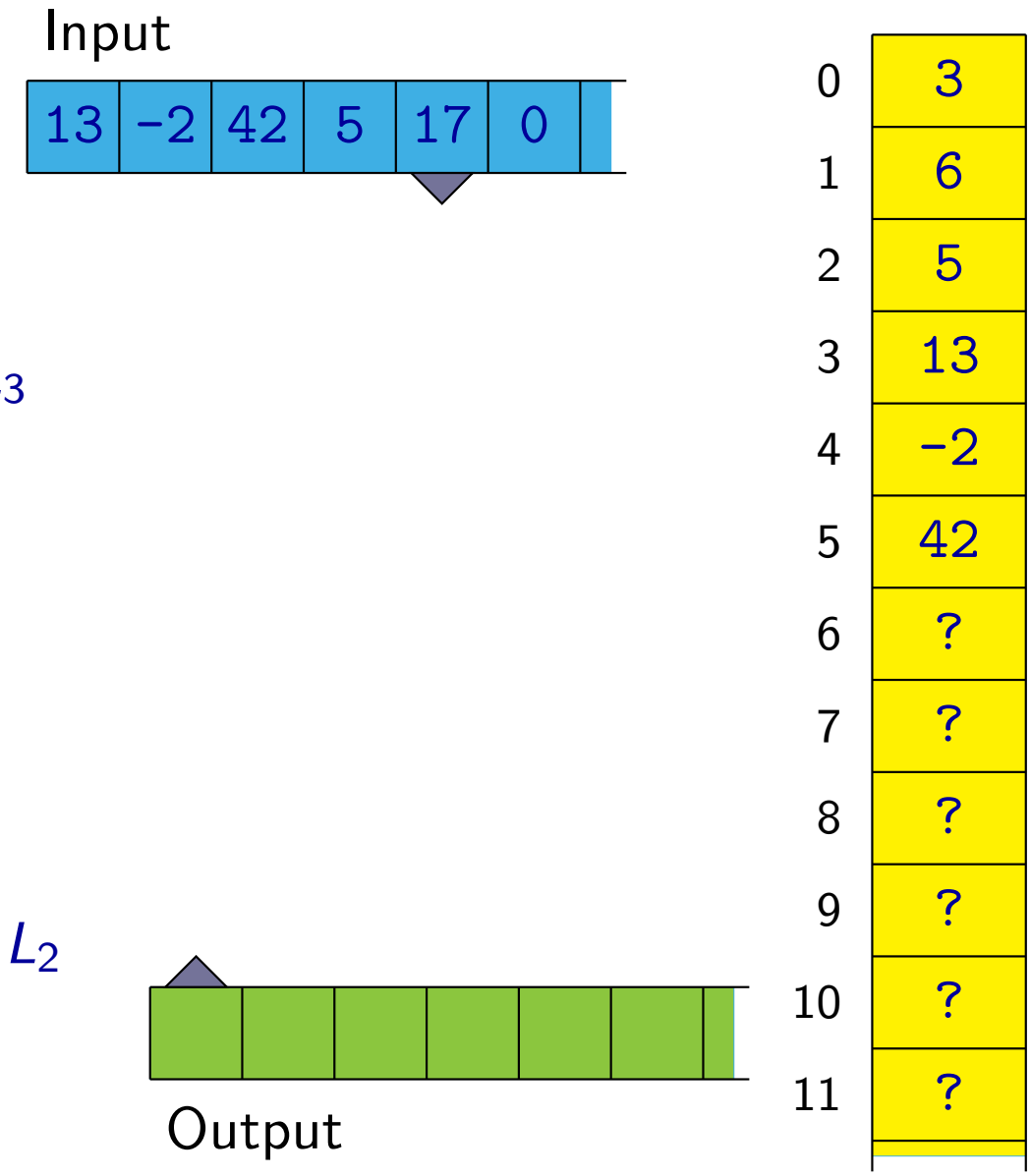
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



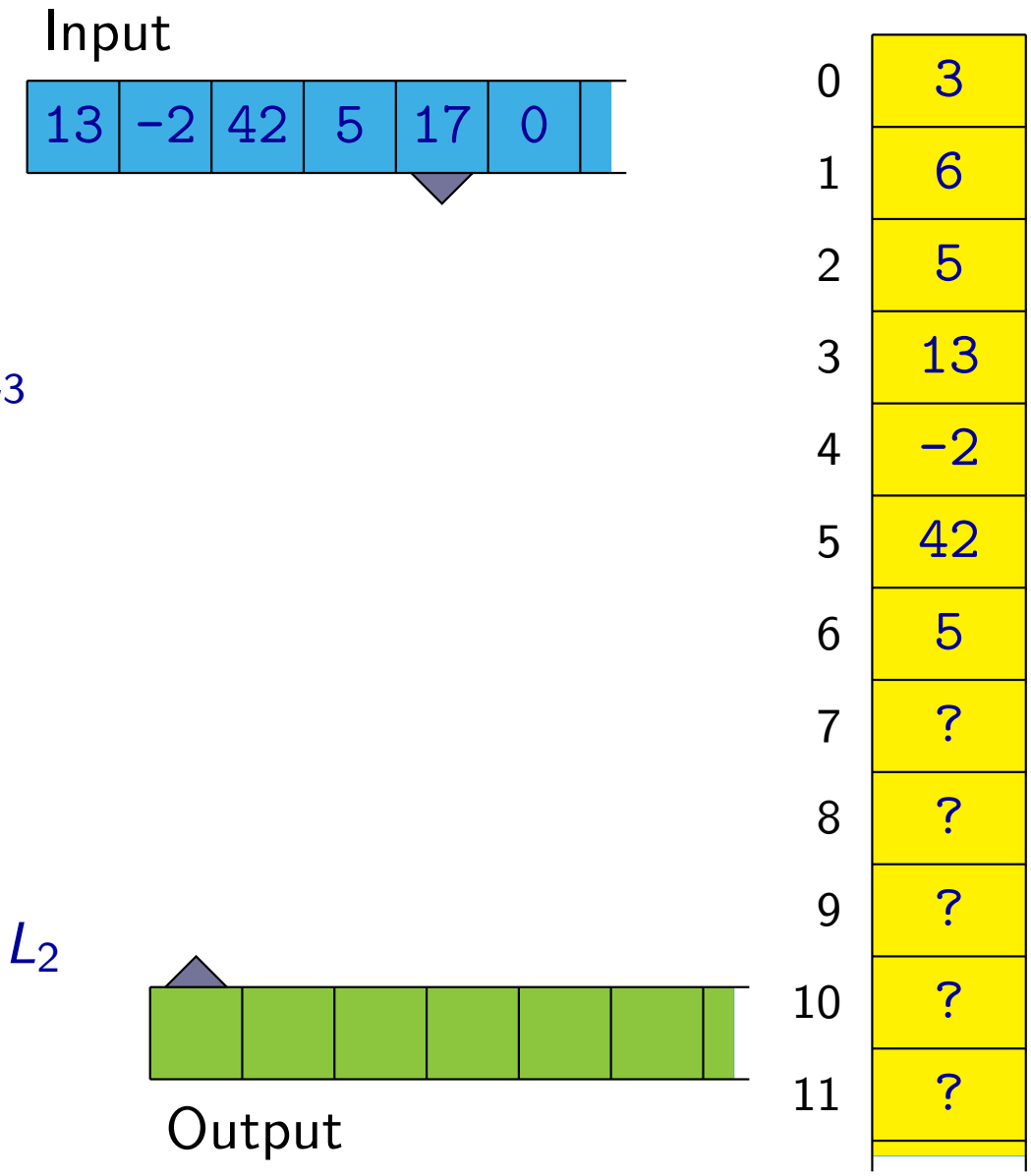
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



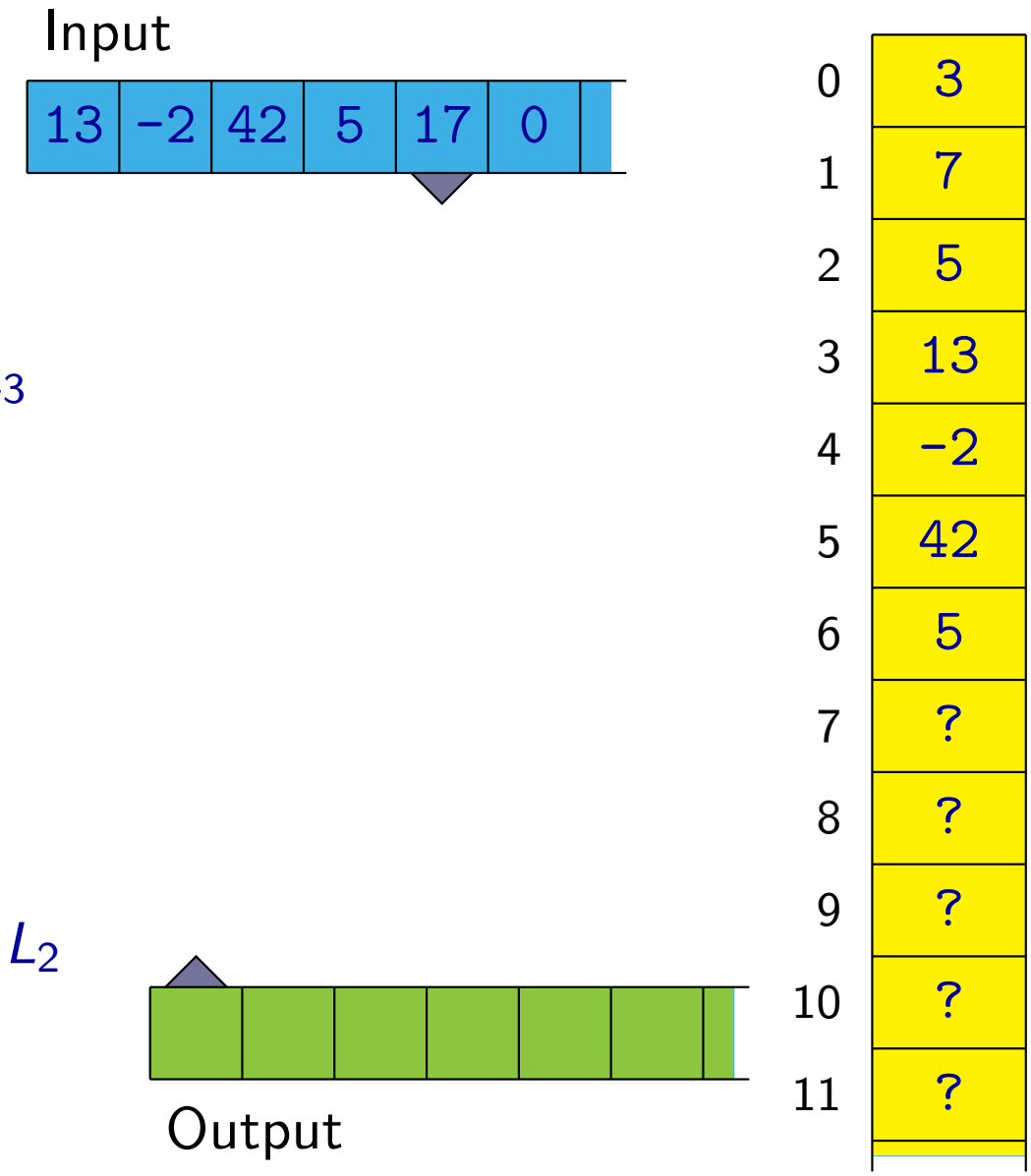
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



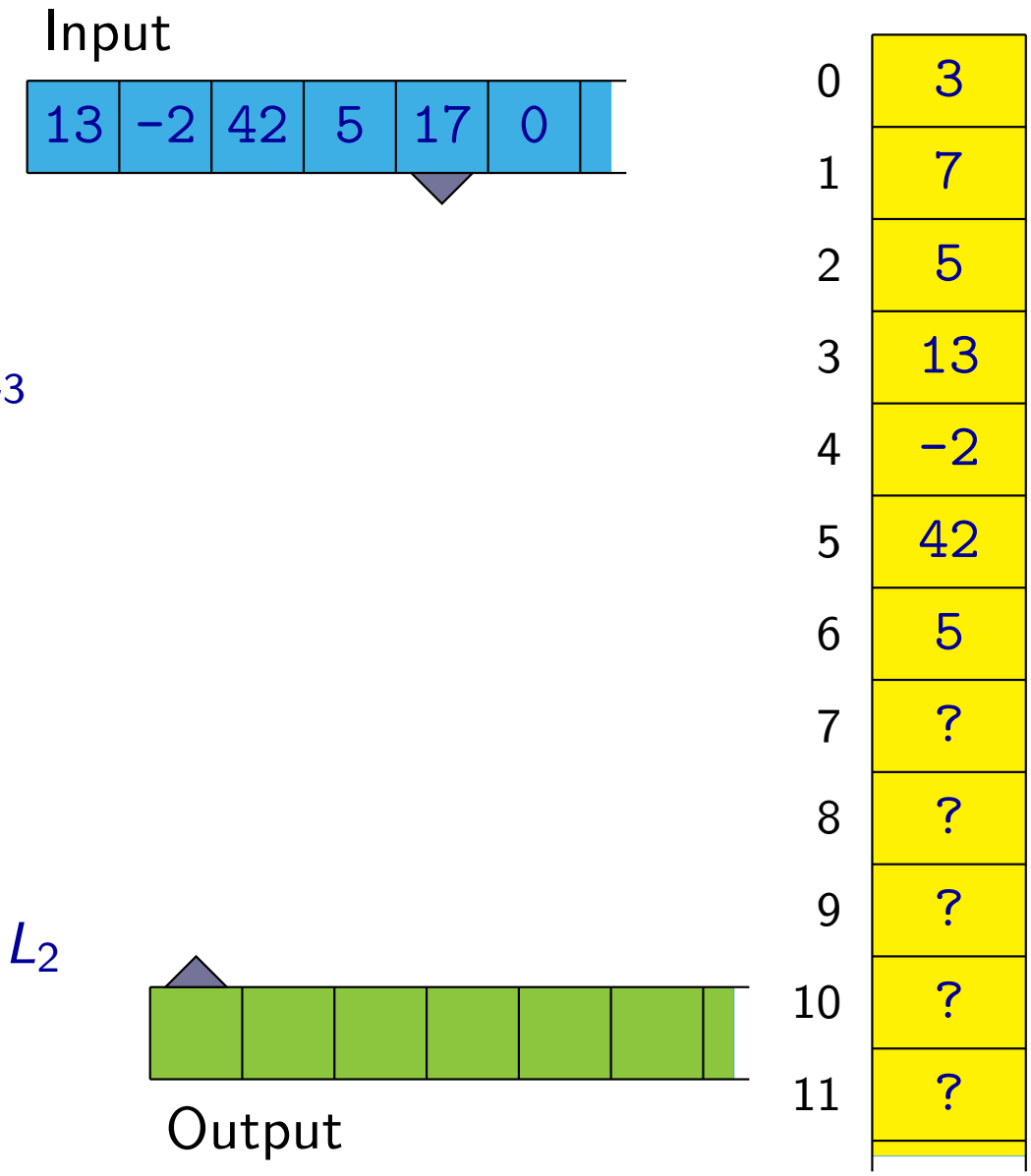
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



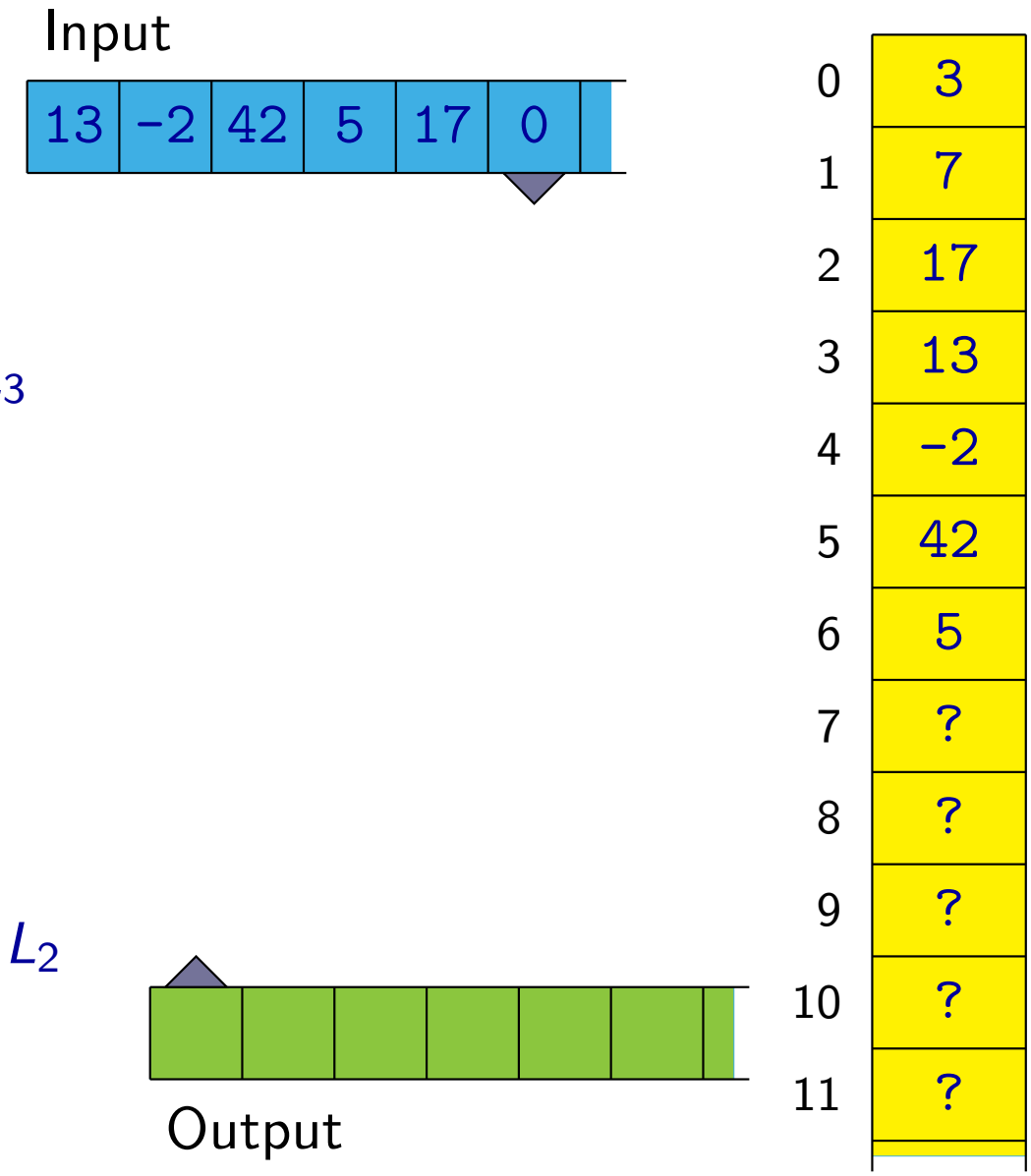
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



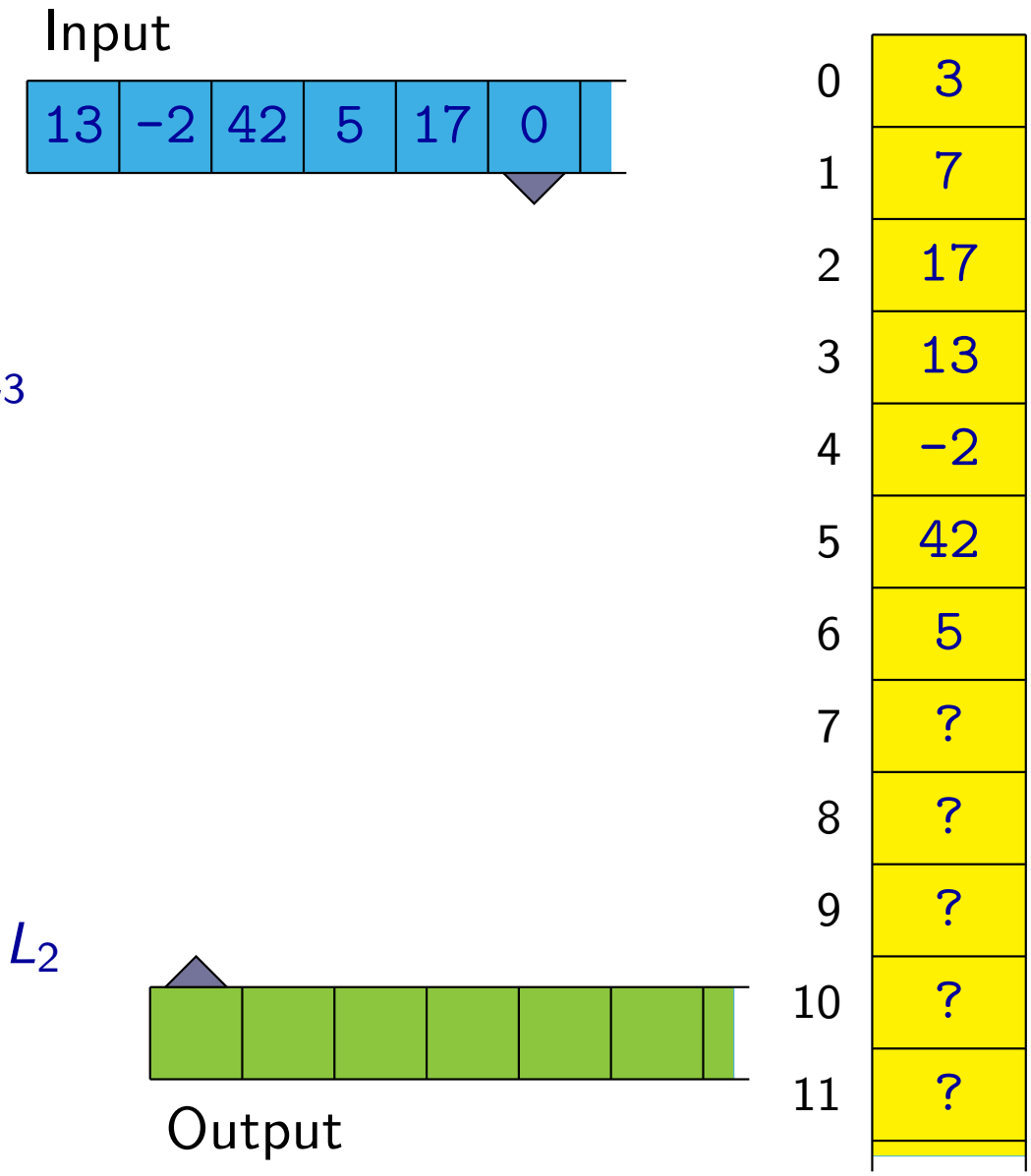
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



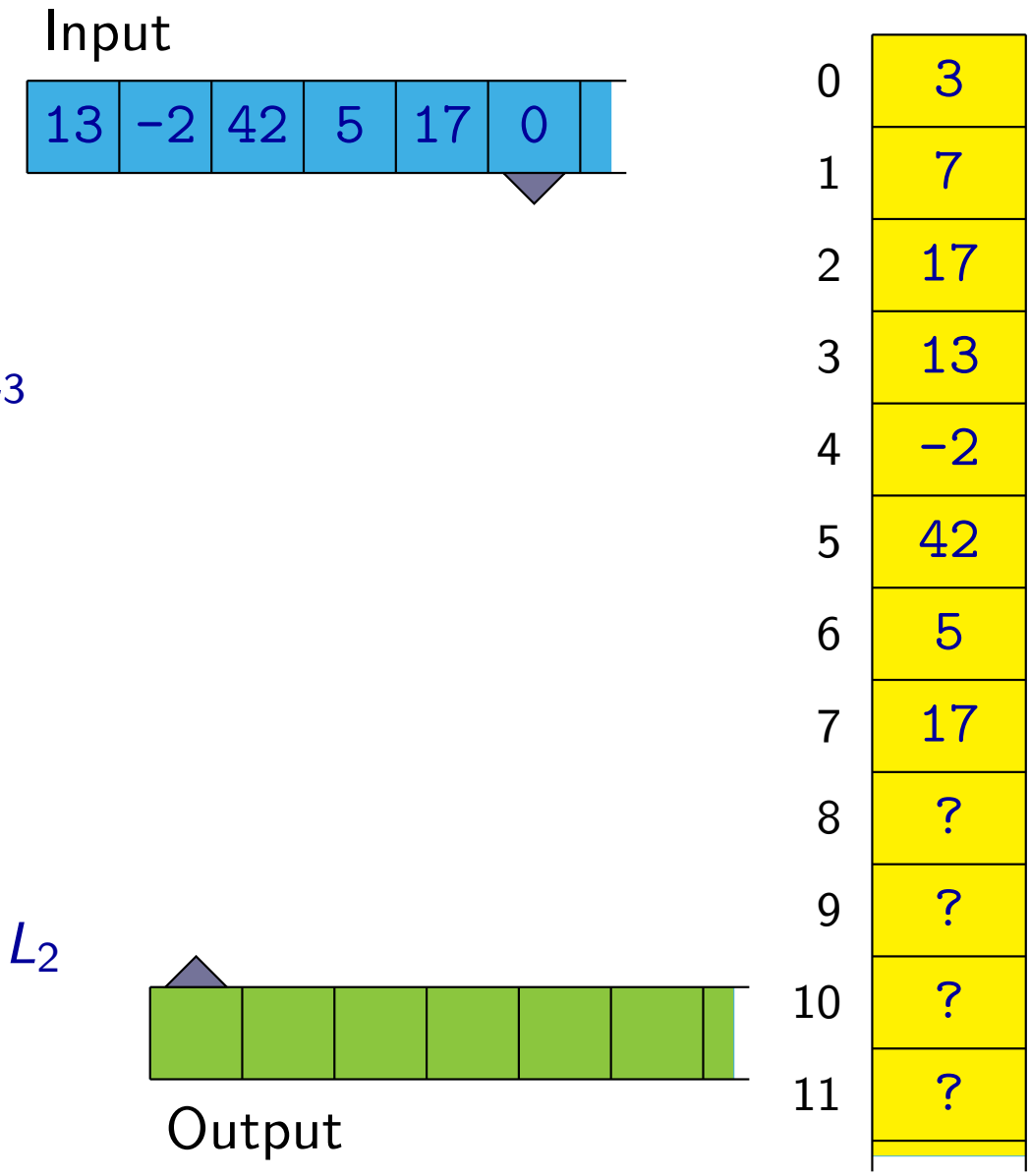
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



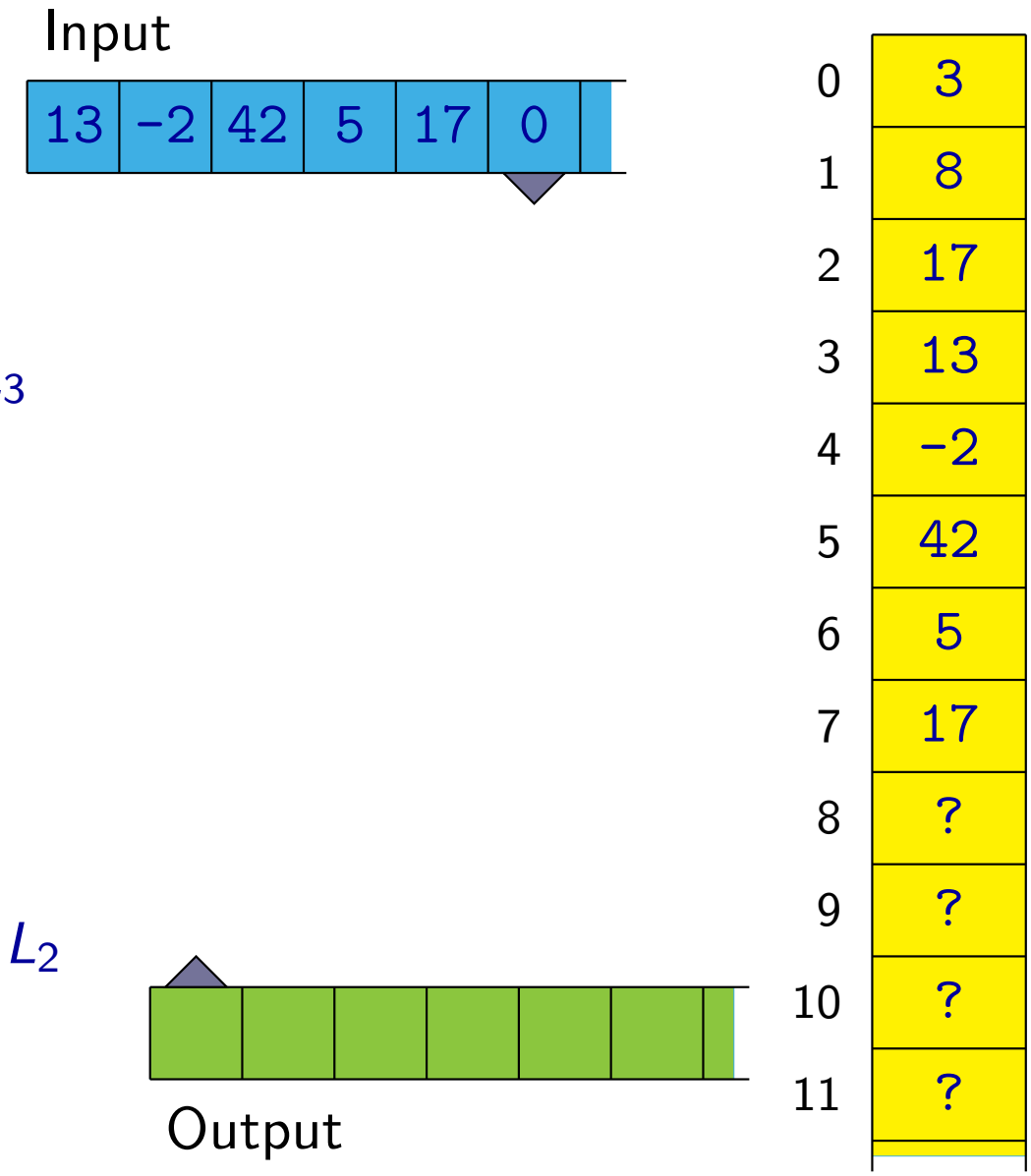
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



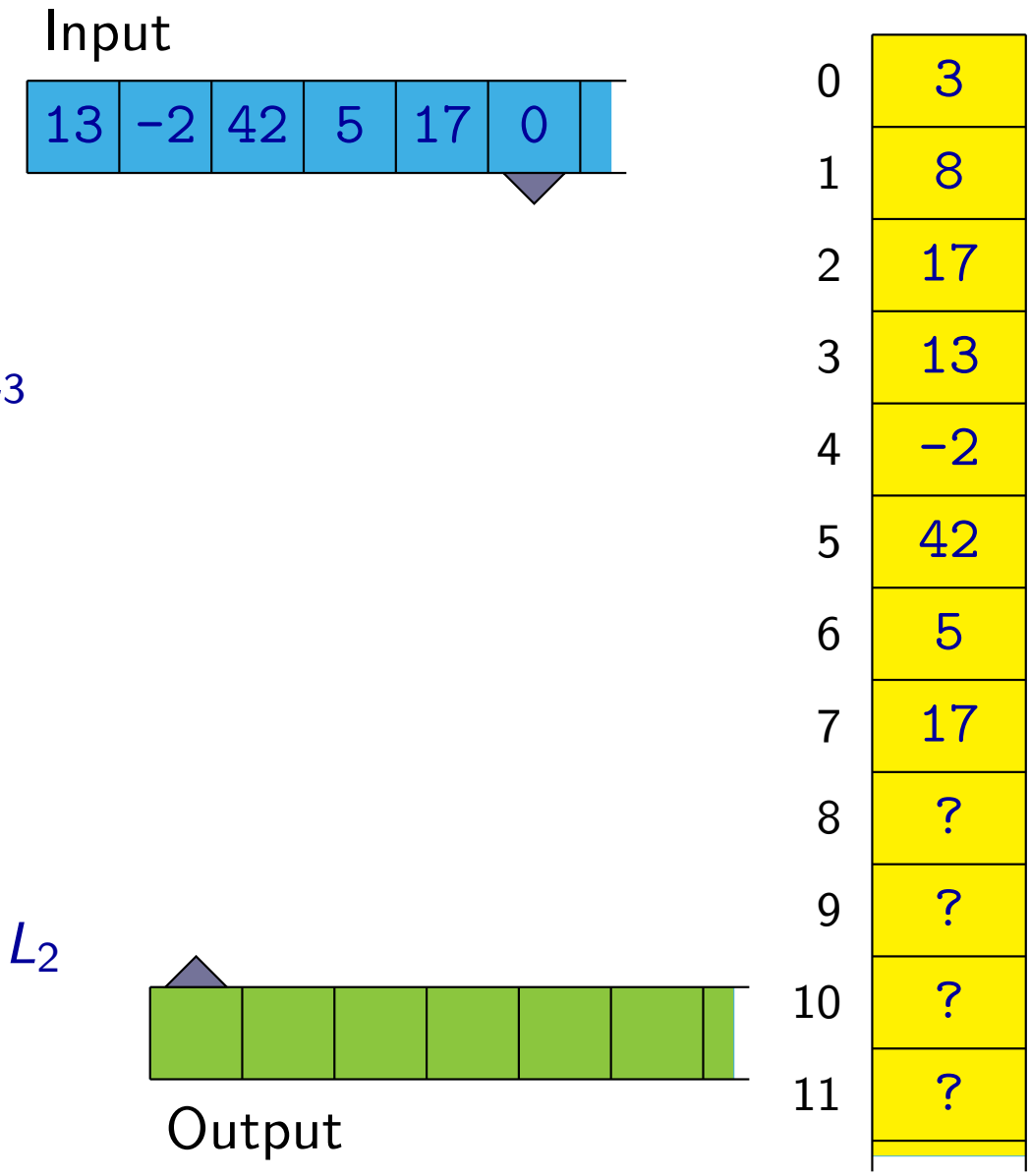
Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt



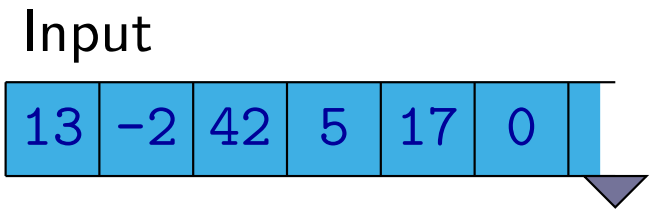
Stroj RAM

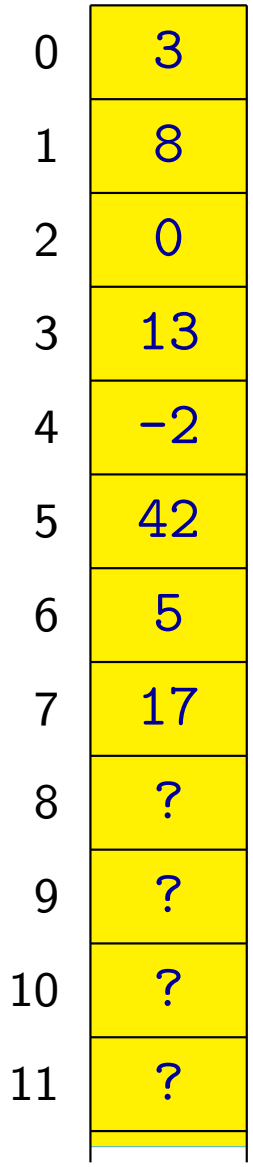
$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

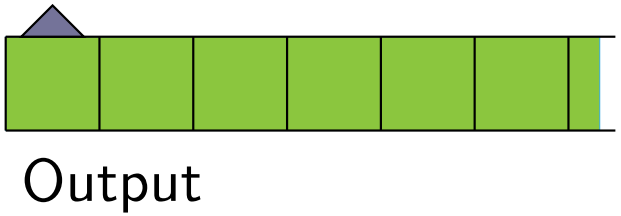


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

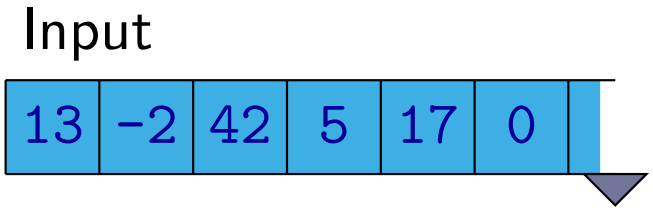


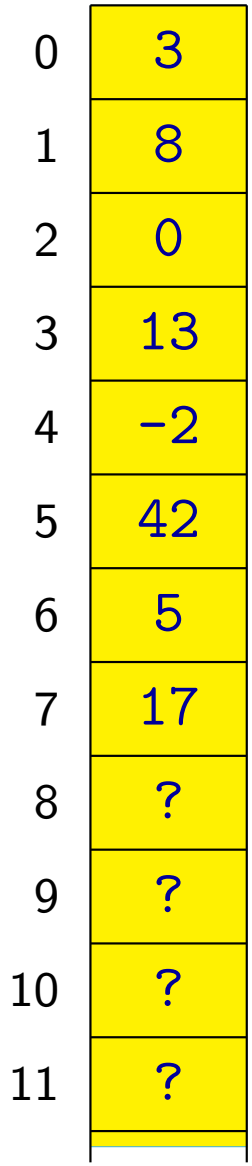


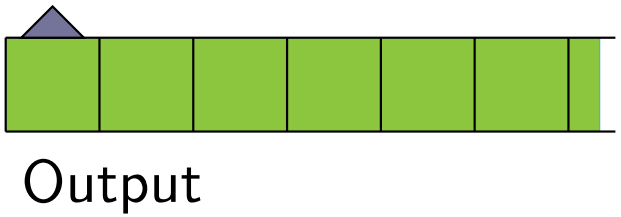


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

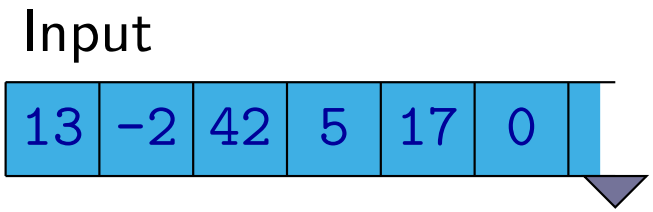


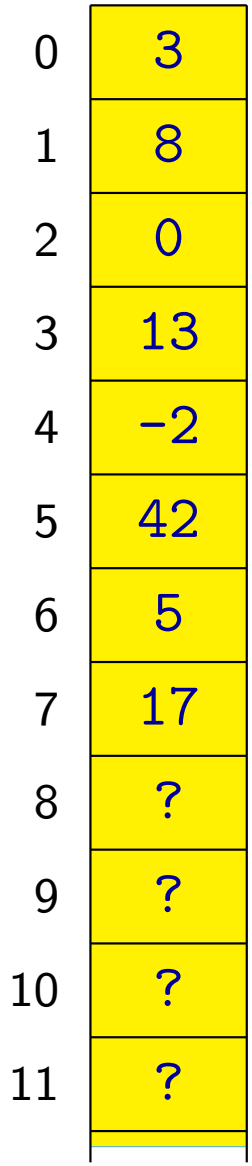


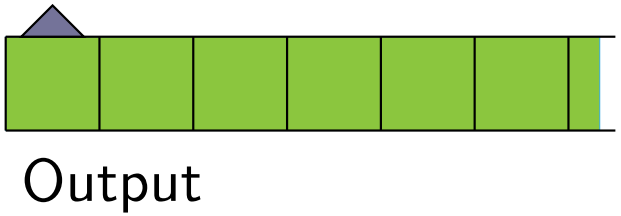


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

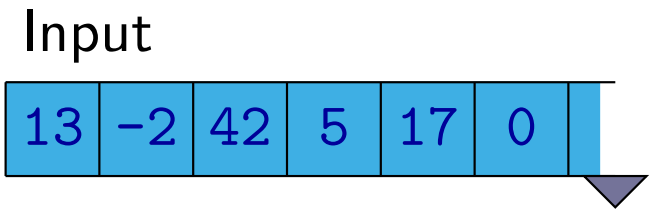


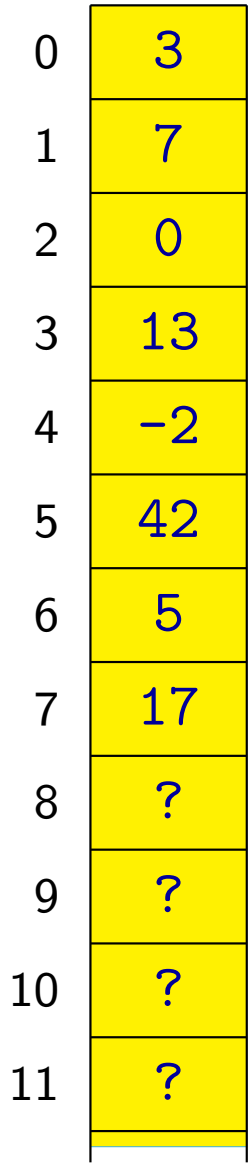


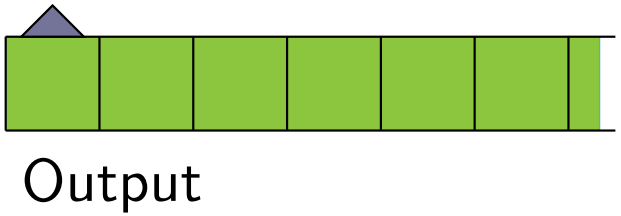


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

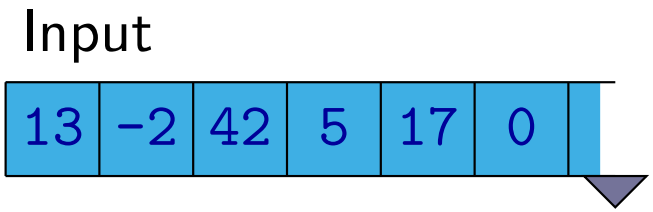


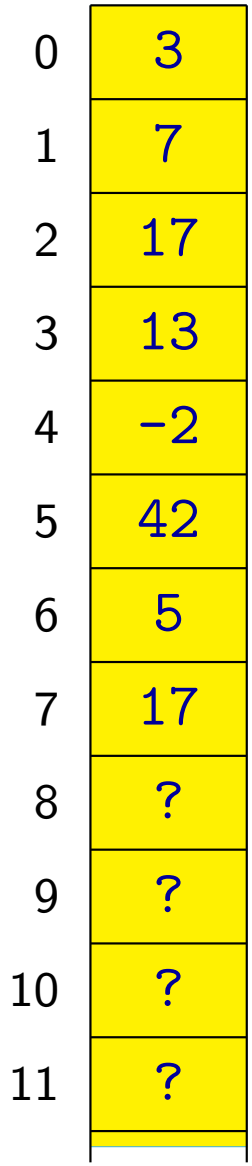


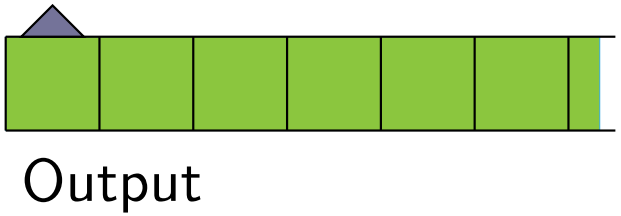


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt







Stroj RAM

$R_0 := 3$

$R_1 := R_0$

$L_1 : R_2 := \text{READ}()$

if $(R_2 = 0)$ **goto** L_3

$[R_1] := R_2$

$R_1 := R_1 + 1$

goto L_1

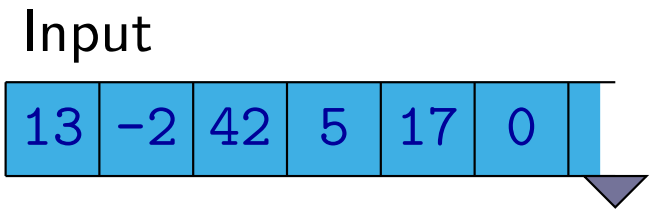
$L_2 : R_1 := R_1 - 1$

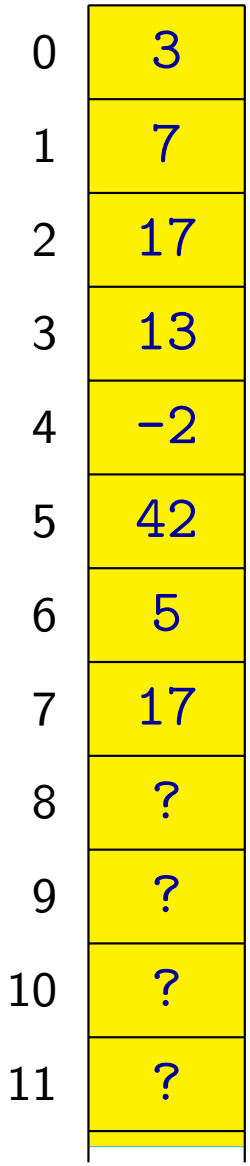
$R_2 := [R_1]$

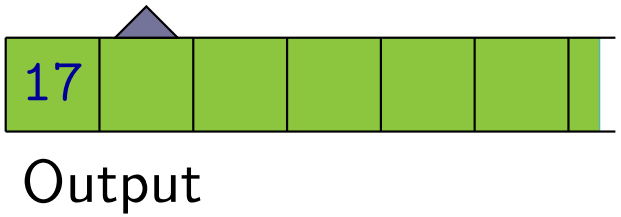
$\text{WRITE}(R_2)$

L_3 : **if** $(R_1 > R_0)$ **goto** L_2

halt

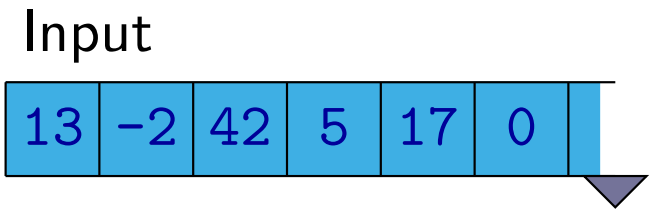


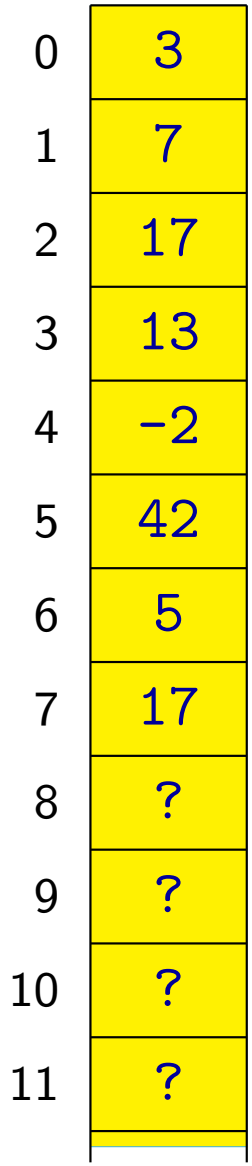


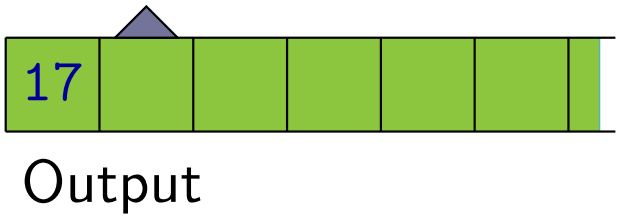


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

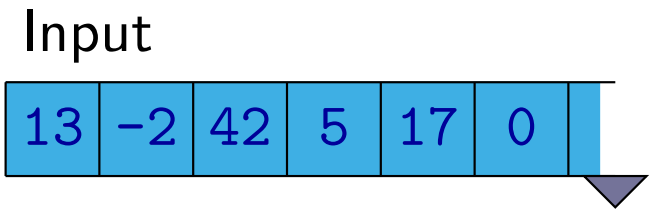


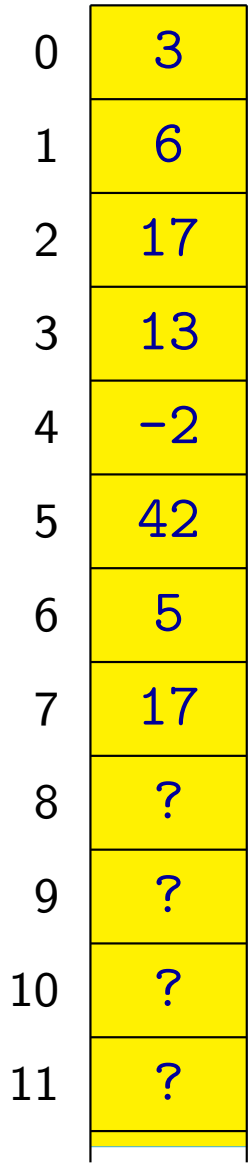


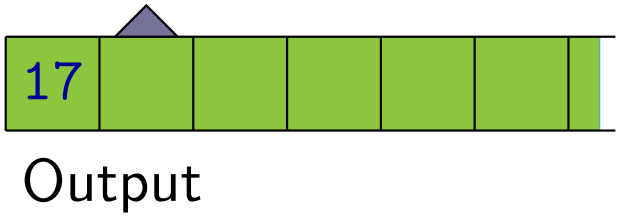


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt

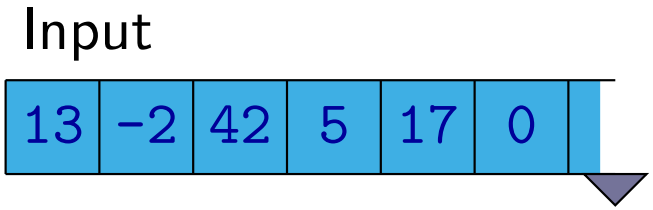


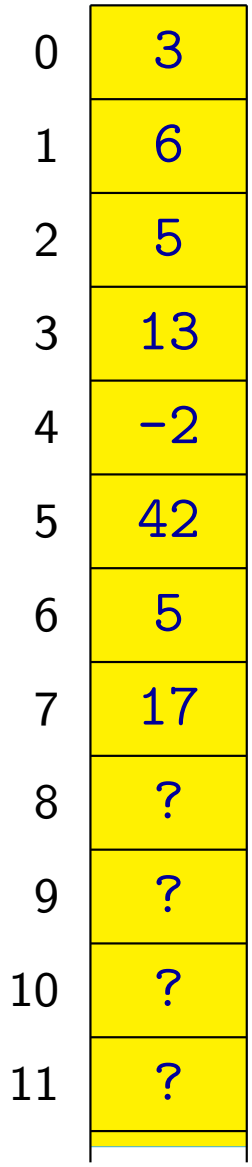


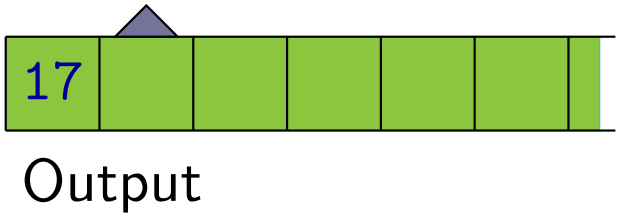


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

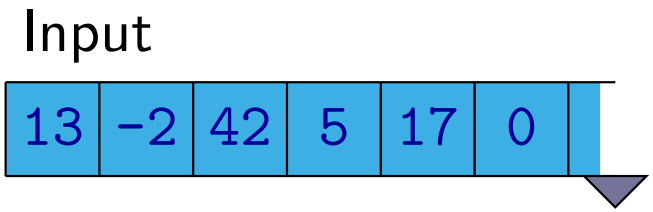


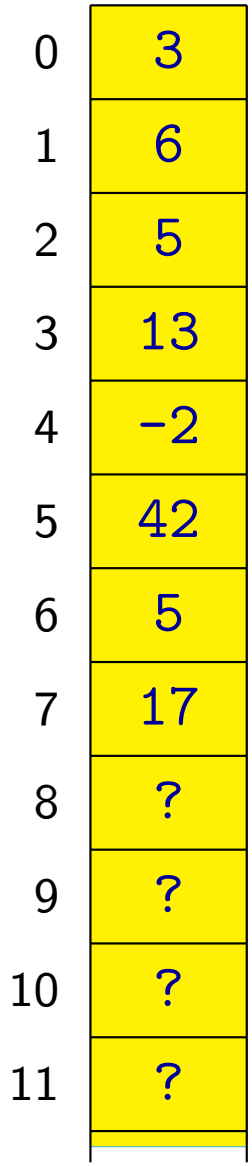


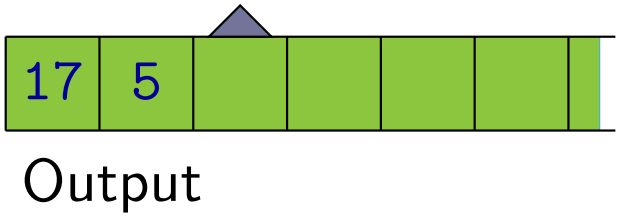


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

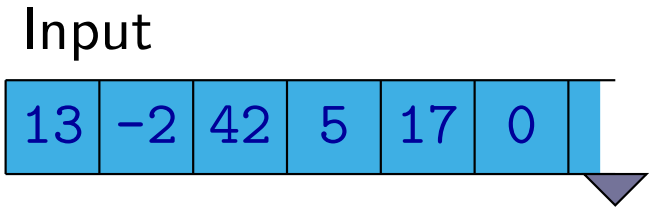


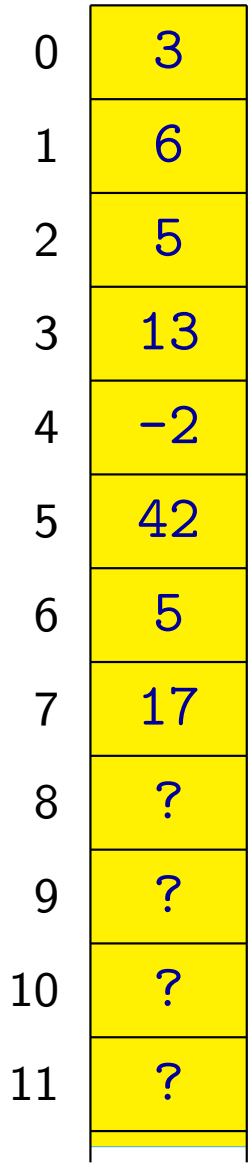


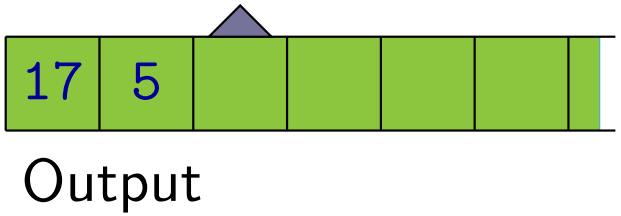


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

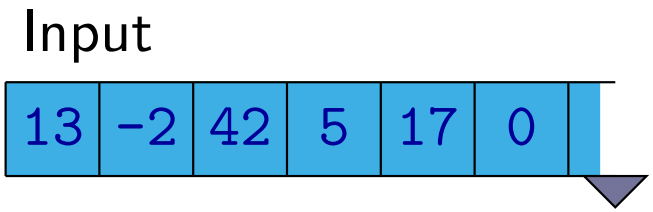


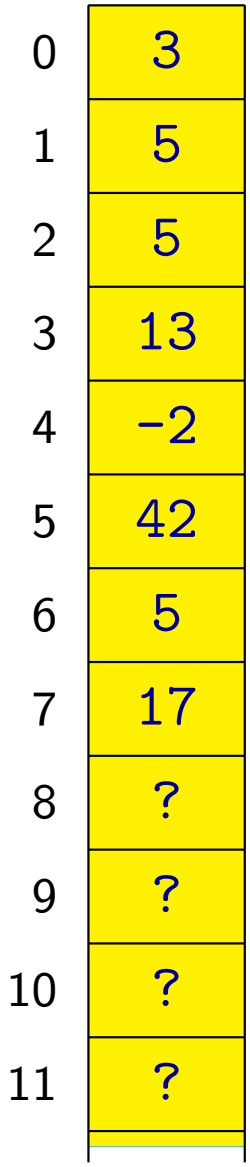


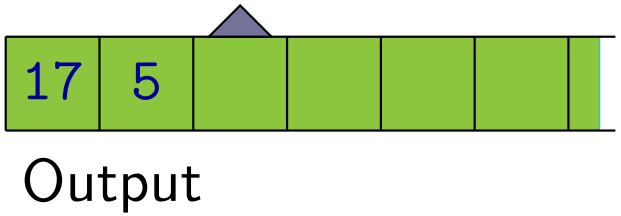


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt

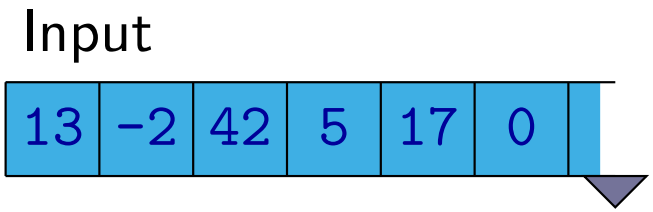


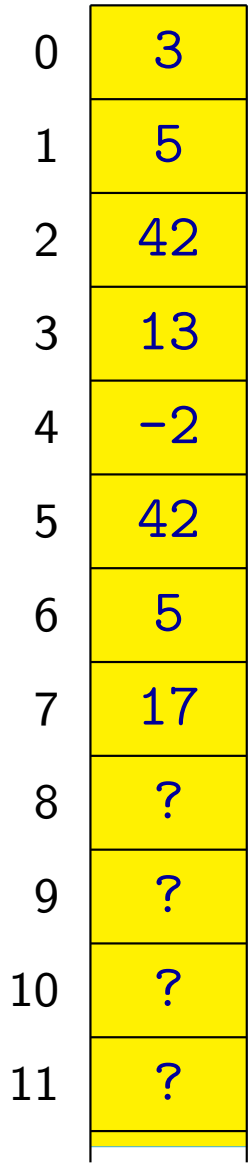


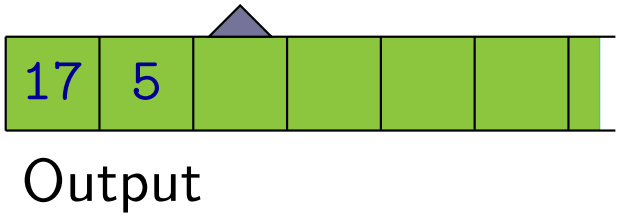


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

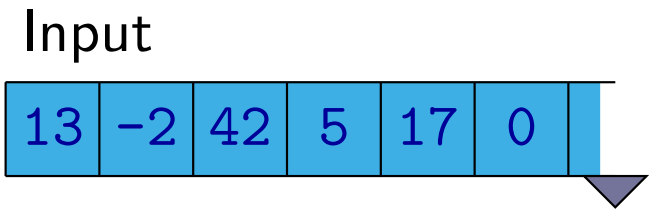


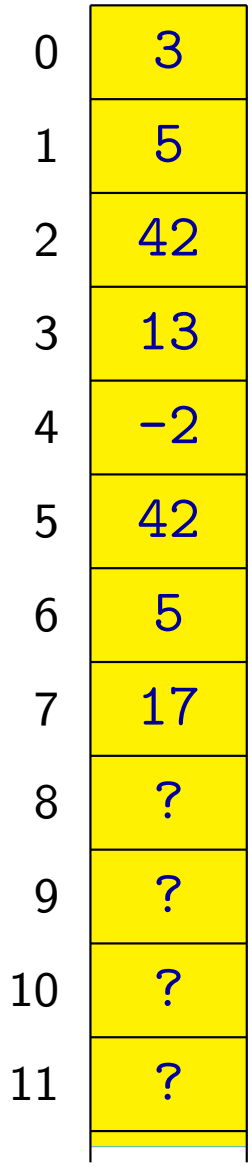


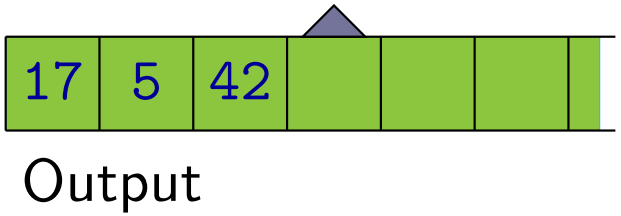


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

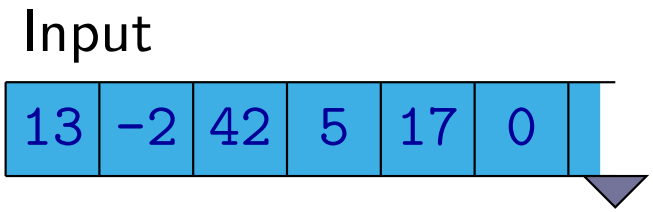


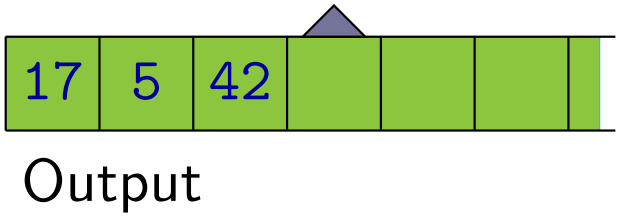


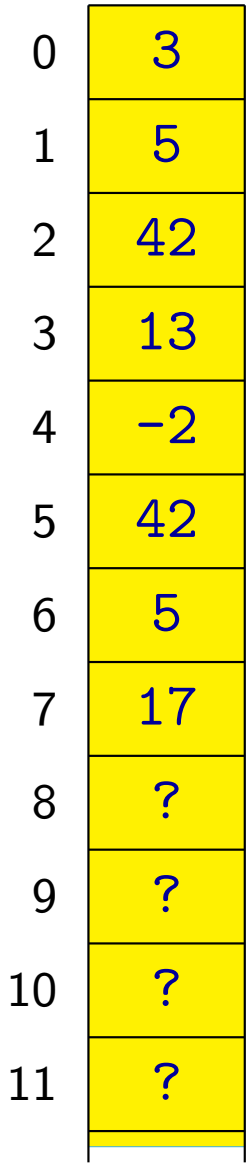


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

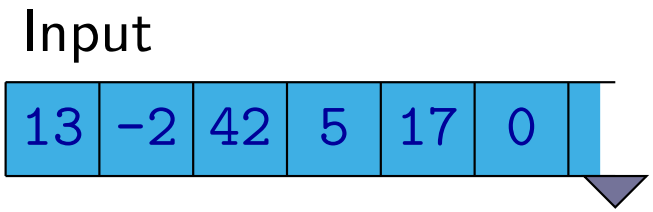


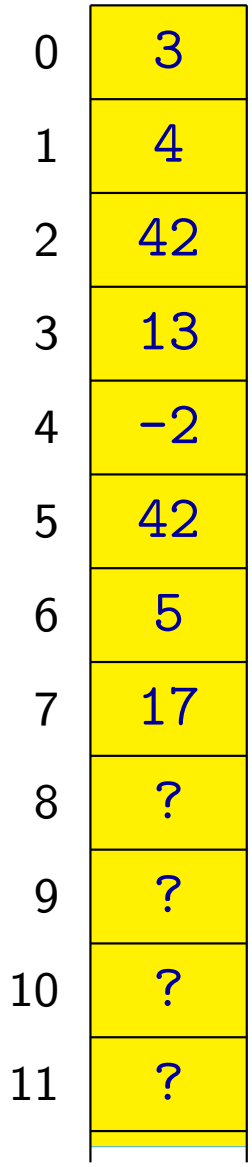


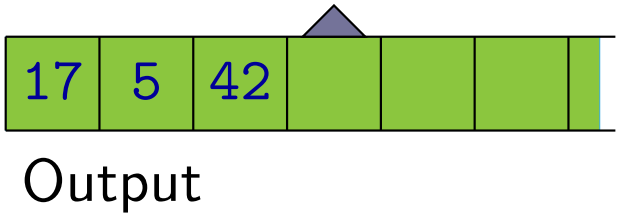


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt

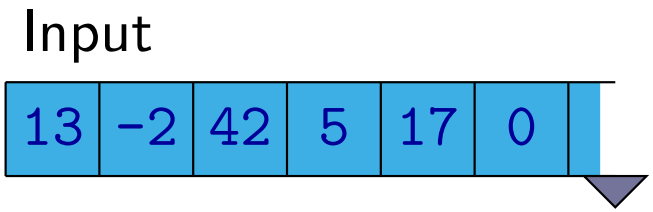


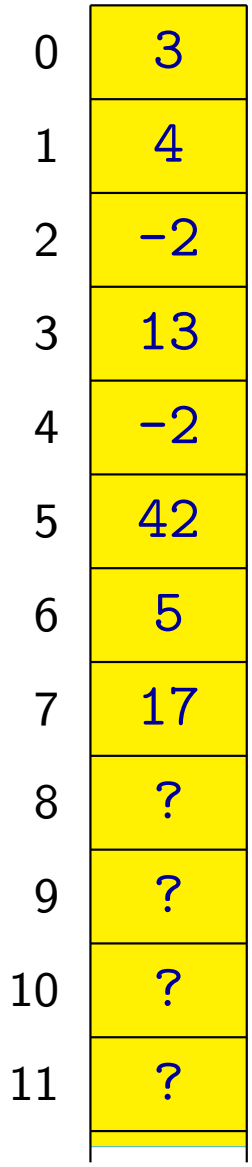


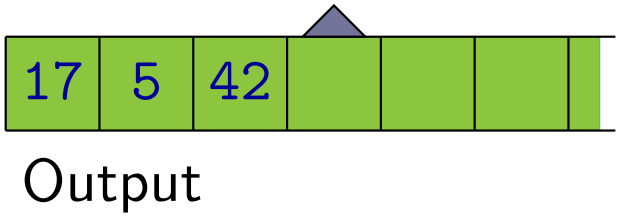


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

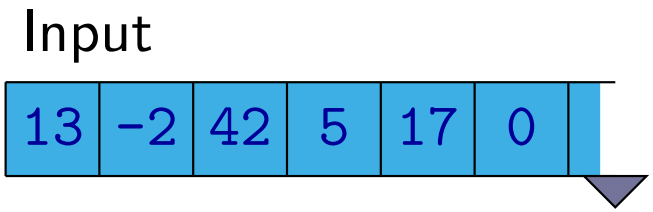


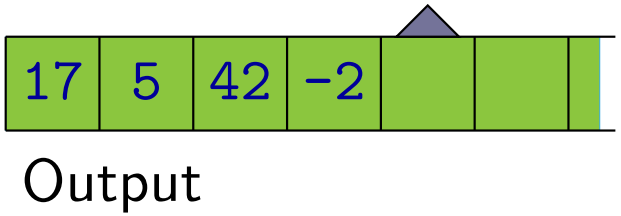


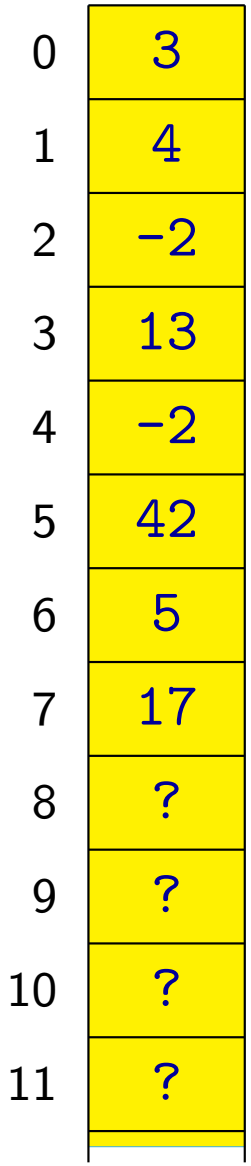


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

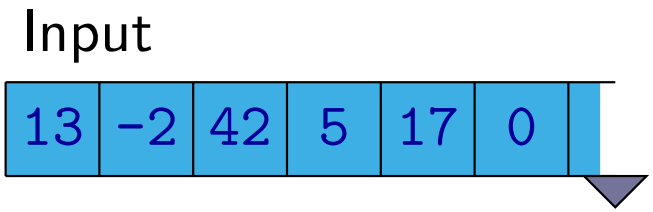


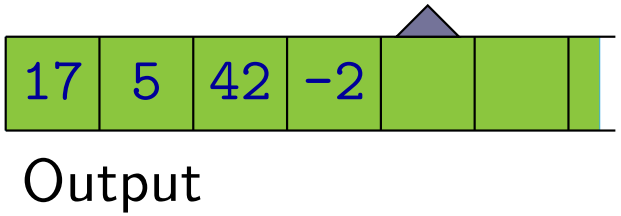


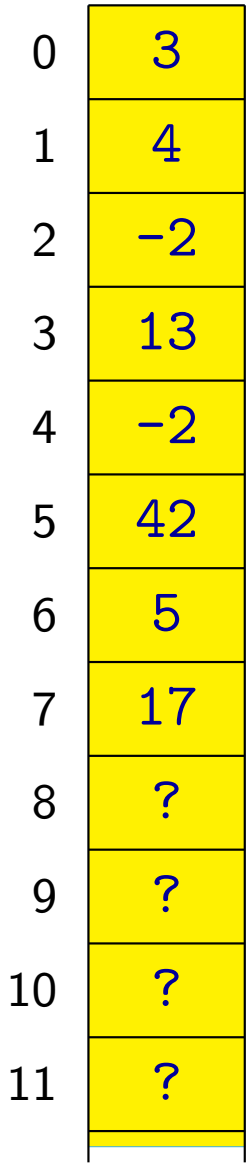


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

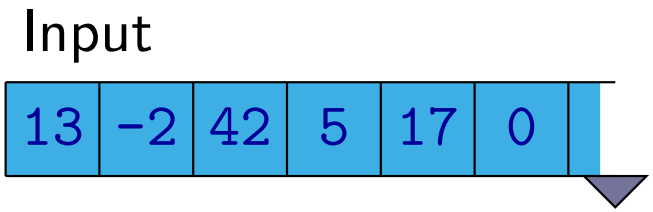


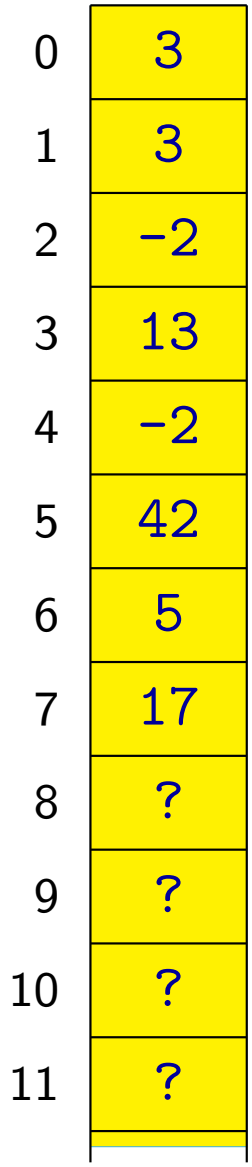


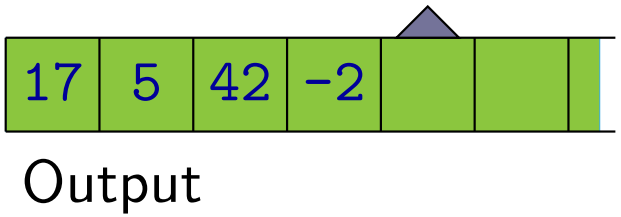


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
L_3 : **if** $(R_1 > R_0)$ **goto** L_2
halt

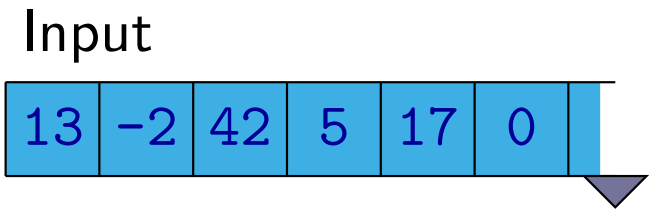


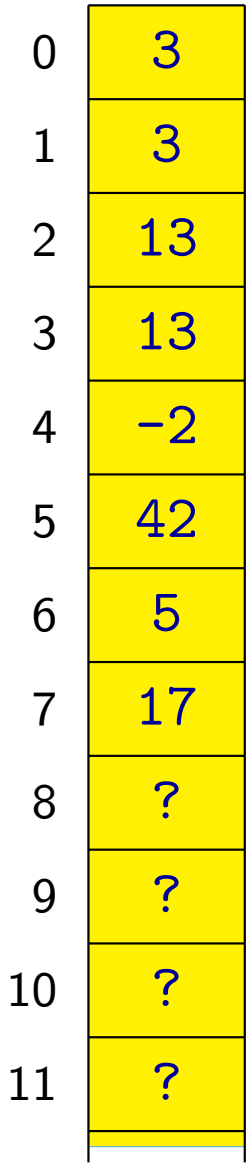


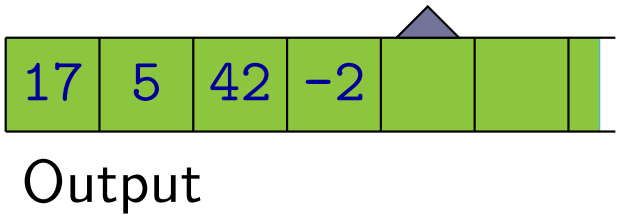


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

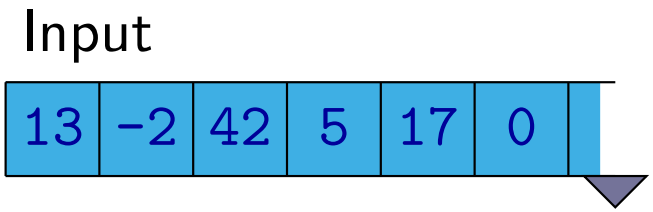


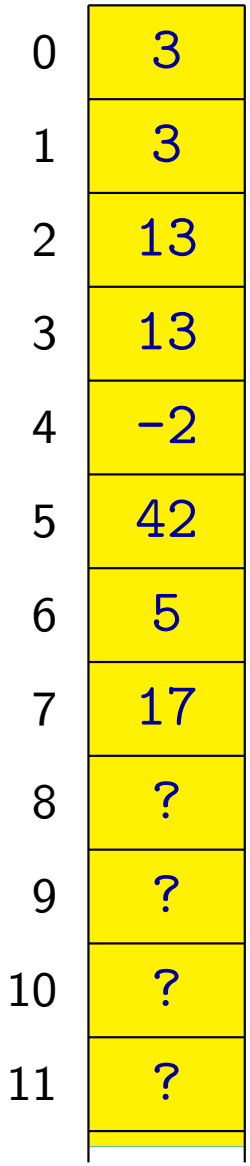


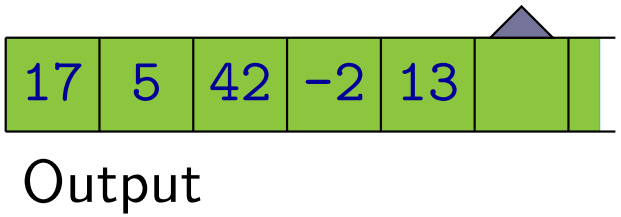


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt

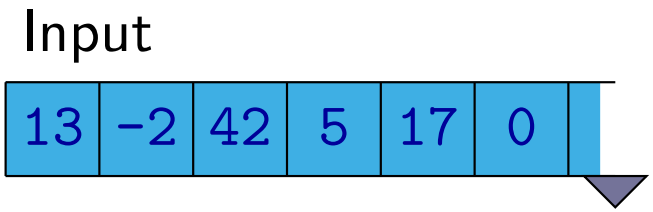


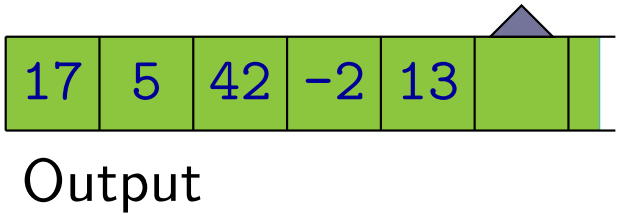


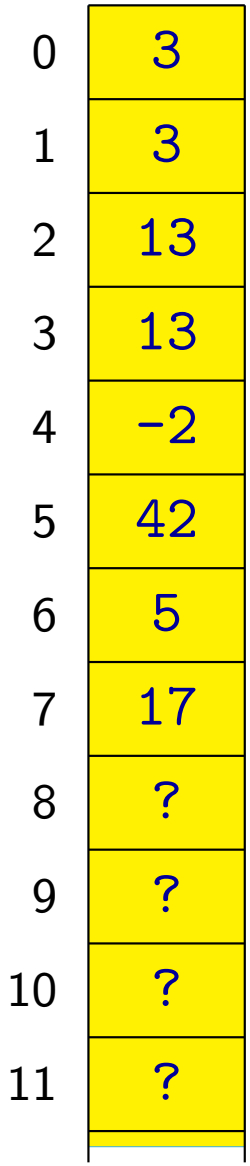


Stroj RAM

$R_0 := 3$
$R_1 := R_0$
$L_1 : R_2 := \text{READ}()$
if $(R_2 = 0)$ **goto** L_3
$[R_1] := R_2$
$R_1 := R_1 + 1$
goto L_1
$L_2 : R_1 := R_1 - 1$
$R_2 := [R_1]$
$\text{WRITE}(R_2)$
$L_3 :$ **if** $(R_1 > R_0)$ **goto** L_2
halt







Stroj RAM

Rozdíly oproti skutečnému počítači:

- Velikost paměti není omezena (adresa může být libovolné přirozené číslo).
- Velikost obsahu jednotlivých buněk není omezena (buňka může obsahovat libovolné celé číslo).
- Čte data sekvenčně ze vstupu, který je tvořen sekvencí celých čísel. Ze vstupu lze pouze číst.
- Zapisuje data sekvenčně na výstup, který je tvořen sekvencí celých čísel. Na výstup je možné pouze zapisovat.

Stroj RAM

- Operace jako přístup k buňce paměti na adrese menší než nula nebo dělení nulou vedou k chybě — výpočet se (neúspěšně) zastaví.

- Co se týká počátečního obsahu paměti, jsou dvě možnosti, jak ho definovat:
  - Všechny buňky jsou inicializovány hodnotou 0.
  - Čtení obsahu buňky, do které nebylo dosud nic zapsáno, způsobí chybu.

    Buňky na začátku obsahují speciální hodnotu (označenou zde symbolem '?'), která reprezentuje to, že buňka nebyla dosud inicializována.

- Uvažují se i varianty strojů RAM, kde buňky paměti (a vstupu a výstupu) neobsahují celá čísla (tj. prvky množiny $\mathbb{Z}$), ale mohou obsahovat jen přirozená čísla (tj. prvky množiny $\mathbb{N}$).

  Například operace odčítání ($R_i := R_j - R_k$) se pak chová tak, že pokud by výsledkem mělo být záporné číslo, je jako výsledek operace přiřazena hodnota 0.

Stroj RAM

- Různé varianty strojů RAM se mohou lišit tím, jaké konkrétní operace v aritmetických instrukcích podporují nebo naopak nepodporují.

  Například:

  - podpora bitových operací (and, or, not, xor, ...), bitový posunů, ...
  - varianta stroje RAM, která nemá operace násobení a dělení

- Mohli bychom také uvažovat variantu stroje RAM, kde místo instrukcí tvaru

  $$\textbf{if } (R_i \ rel \ R_j) \textbf{ goto } \ell \qquad \text{nebo} \qquad \textbf{if } (R_i \ rel \ c) \textbf{ goto } \ell$$

  jsou všechny podmíněné skoky jen tvaru

  $$\textbf{if } (R_i \ rel \ 0) \textbf{ goto } \ell$$

  Místo všech relací $\{=, \neq, \leq, \geq, <, >\}$ může být podporována jen nějaká podmnožina z nich, např. $\{=, >\}$.

Stroj RAM

- V některých variantách stroje RAM nemají vstup a výstup podobu sekvence čísel.

  Místo toho pracuje stroj z hlediska vstupu a výstupu s páskami obsahujícími sekvence symbolů z nějaké dané abecedy, např. $\{0, 1\}$.

  Stroj má pak například instrukce, které mu umožňují větvit výpočet podle symbolu přečteného ze vstupu.

  Vnitřní paměť ovšem i v této variantě pracuje s čísly.

- Pokud má stroj jako výsledek dávat jen odpověď Ano/Ne (tj. přijmout nebo nepřijmout daný vstup), nemusí mít výstupní pásku.

  Instrukce **halt** je pak nahrazena instrukcemi **accept** a **reject**.

Stroj RAM

- Ve standardní definici stroje RAM se většinou neuvažují instrukce skoku na adresu instrukce uloženou v buňce paměti, tj. instrukce typu

  goto R_i

  Stroj RAM bychom mohli rozšířit o tento druh instrukcí.

- Jako standardní se u stroje RAM bere to, že kód programu není uložen v pracovní paměti, ale má zvláštní samostatnou paměť, která je jen pro čtení.

  V průběhu výpočtu se tedy kód programu nemůže měnit.

Stroj RASP

- Druh stroje podobný stroji RAM, kde je ovšem program uložen v pracovní paměti (instrukce jsou kódovány čísly) a je možné ho průběhu výpočtu měnit, se označuje jako stroj **RASP** (**random-access stored program**).

  Stroj RASP tak umožňuje provádět činnost sebemodifikujících se programů.

Stroj RAM

Dobu výpočtu stroje RAM je možno počítat dvěma různými způsoby:

- **jednotková míra** — počet provedených instrukcí
- **logaritmická míra** — součet doby trvání jednotlivých instrukcí; doba trvání každé instrukce závisí na počtu bitů hodnot, se kterými se v této instrukci pracuje.

  Například:

  - Doba provádění instrukcí sčítání a odčítání je rovna součtu počtů bitů jejich operandů.
  - Doba provádění instrukcí násobení a dělení je rovna součinu počtů bitů jejich operandů.
  - Doba provádění instrukce přístupu do paměti (load, store) je dána jako součet počtu bitů adresy a počtu bitů čteného či zapisovaného čísla.

  Poznámka: Při počítání počtu bitů daného čísla se počítá, že hodnota 0 má 1 bit.

Stroj RAM

Rovněž množství paměti použité během výpočtu strojem RAM je možno počítat dvěma různými způsoby:

- **jednotková míra** — množství použitých buněk paměti, tj. počet buněk, do kterých stroj během výpočtu zapisoval nebo z nich četl.
- **logaritmická míra** — maximální množství paměti, které bylo během výpočtu potřeba pro nějakou konfiguraci.

  Množství paměti pro konfiguraci je dáno jako součet počtu bitů všech použitých buněk paměti a počtu bitů jejich adres.

Stroj RAM

Jednotková míra realisticky odráží množství provedené práce či množství použité paměti pouze v případech, kdy jsou hodnoty čísel uložených v buňkách paměti „malé“, tj. pokud by je ve skutečné implementaci pro „rozumně“ velké vstupy bylo možné reprezentovat např. 32-bitovými či 64-bitovými čísly.

Stroj RAM

- U strojů, které nemají instrukci násobení, se dá snadno ukázat, že každá jednotlivá instrukce může vytvořit číslo, které má nanejvýš o jeden bit více než (v absolutní hodnotě) větší z jejích operandů.

  U takovýchto strojů obsahuje po t krocích výpočtu každá buňka číslo, které má nejvýše $t + m + n$ bitů, kde m je počet bitů největší konstanty, která se vyskutuje v programu a n je největší počet bitů, jaké má libovolné číslo na vstupu.

- Pokud má stroj instrukci násobení, může nějaká buňka paměti obsahovat po t krocích číslo, které má až zhruba 2^t bitů.

Churchova-Turingova teze

Churchova-Turingova teze

Dalším příkladem výpočetního modelu jsou **Turingovy stroje**.

Turingovy stroje jsou ještě jednodušším modelem než stroje RAM.

Příliš se nepodobají skutečným počítačům.

Turingovy stroje byly zavedeny jako teoretický model zachycující formálně význam pojmu algoritmus.

Churchova-Turingova teze

Každý algoritmus je možné realizovat nějakým Turingovým strojem.

Není to věta, kterou by bylo možno dokázat v matematickém smyslu – není formálně definováno, co je to algoritmus.

Tezi formulovali nezávisle na sobě v polovině 30. let 20. století Alan Turing a Alonzo Church.

Churchova-Turingova teze

Příklady matematických formalismů zachycujících pojem algoritmus:

- stroje RAM
- Turingovy stroje
- lambda kalkulus
- rekurzivní funkce
- ...

Dále můžeme uvést:

- Libovolný (obecný) programovací jazyk (jako např. C, Java, Lisp, Haskell, Prolog apod.).

Všechny tyto modely jsou ekvivalentní z hlediska algoritmů, které jsou schopny realizovat.

Výpočetní modely

Takovým jazykům (resp. strojům), které jsou dostatečně obecné na to, aby se do nich (resp. do jejich instrukcí) daly přeložit programy napsané v libovolném jiném programovacím jazyce, se říká **Turingovsky úplné**.

Zatím jsme si popsali stroj RAM.

Později si podrobně popíšeme nejen Turingovy stroje, ale i některé další příklady Turingovsky úplných modelů.