Úvod do teoretické informatiky

Zdeněk Sawa

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

12. února 2025

Přednášející

Jméno: doc. Ing. Zdeněk Sawa, Ph.D.

E-mail: zdenek.sawa@vsb.cz

Místnost: EA413

Web: https://www.cs.vsb.cz/sawa/uti

Na těchto stránkách najdete:

- Informace o předmětu
- Učební texty
- Slidy z přednášek
- Zadání příkladů na cvičení
- Aktuální informace
- Odkaz na stránku s animacemi

Požadavky

- **Zápočet** (30 bodů):
  - Zápočtová písemka (24 bodů) — bude se psát na cvičení
    - Minimum pro získání zápočtu je 12 bodů.
    - Možnost opravy za 20 bodů.
  - Aktivita na cvičení (6 bodů)
    - Minimum pro získání zápočtu jsou 3 body.
- **Zkouška** (70 bodů)
  - Písemná zkouška skládající se ze dvou částí po 35 bodech, přičemž z každé části je nutné získat nejméně 12 bodů.
  - Celkově je třeba získat minimálně 30 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 x a y.

Výstup: Přirozené číslo z takové, že $z = x + y$.

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 x a y.

Výstup: Přirozené číslo z takové, že $z = x + y$.

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

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

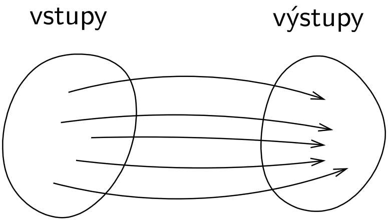
Výstupem pro tuto instanci je číslo 762.

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



Příklady 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:

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

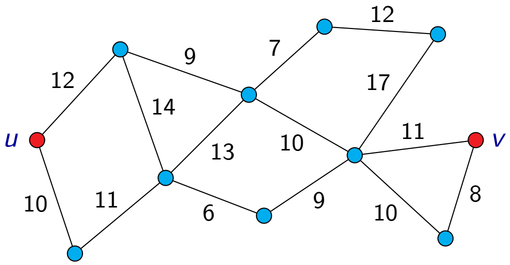
Příklad algoritmického problému

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

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

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: $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots$

Rozhodovací problémy

Problémům, kde množina výstupů je $\{\textsc{Ano}, \textsc{Ne}\}$ 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?

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 $u, v \in V$.

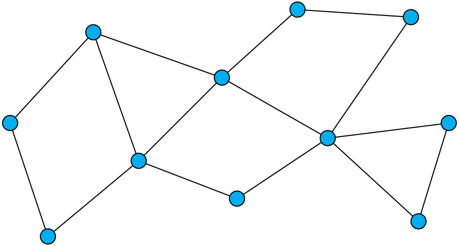
Výstup: Nejkratší cesta z vrcholu u do vrcholu v.

Optimalizační problémy

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.

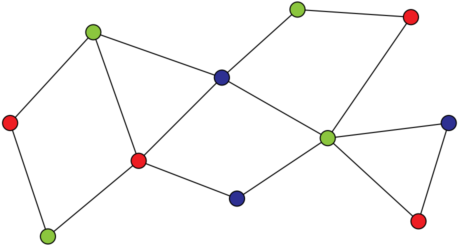


Optimalizační problémy

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.



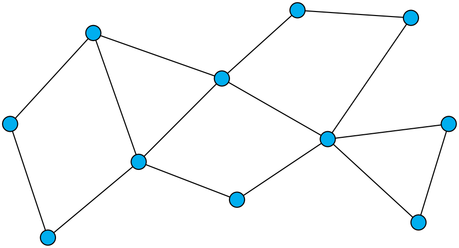
Barvy: 3

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?



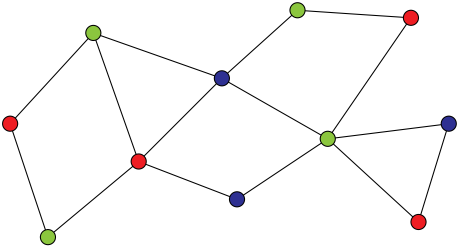
$k = 3$

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?



$k = 3$

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 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.

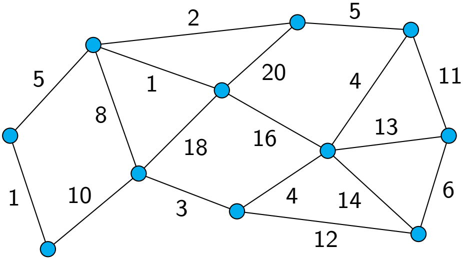
Teorie vyčíslitelnosti — oblast teoretické informatiky, která se zabývá zkoumáním toho, které problémy jsou a které nejsou algoritmicky řešitelné.

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.



Některé další oblasti teoretické informatiky:

- teorie složitosti
- teorie formálních jazyků
- výpočetní modely
- paralelní a distribuované algoritmy
- ...

Teorie formálních jazyků

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

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

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
- ...

Teorie formálních jazyků – motivace

Příklady typů problémů, při jejichž řešení se využívá poznatků z teorie formálních jazyků:

- Tvorba překladačů:
    - lexikální analýza
    - syntaktická analýza
- Vyhledávání v textu:
    - hledání zadaného vzorku
    - hledání textu zadaného regulárním výrazem

Abeceda, slovo

- **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^*.

  Pro proměnné, jejichž hodnoty jsou slova, budeme používat názvy w, u, v, x, y, z, apod., případně s indexy (např. w_1, w_2)

  Zápis $w = \mathtt{cabcbba}$ tedy znamená, že hodnotou proměnné w je slovo $\mathtt{cabcbba}$.

  Podobně zápis $w \in \Sigma^*$ znamená, že hodnotou proměnné w je nějaké slovo tvořené symboly z abecedy Σ.

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 $L_2 = \{\, w \in \Sigma^* \mid \text{počet výskytů symbolů } \mathtt{b} \text{ ve slově } w \text{ je sudý} \,\}$

Příklad:

Abeceda Σ je množina všech ASCII znaků.

Příklad slova:

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

`#include␣<stdio.h>` ↩↩ `int␣main()` ↩ `{` ↩ `␣␣␣␣printf("He`···

Formální jazyky

Prostředky používané pro popis formálních jazyků:

- automaty
- gramatiky
- regulární výrazy

Kódování vstupu a výstupu

U algoritmických problémů často předpokládáme, že vstupy i výstupy jsou kódovány jako slova v nějaké abecedě Σ.

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`

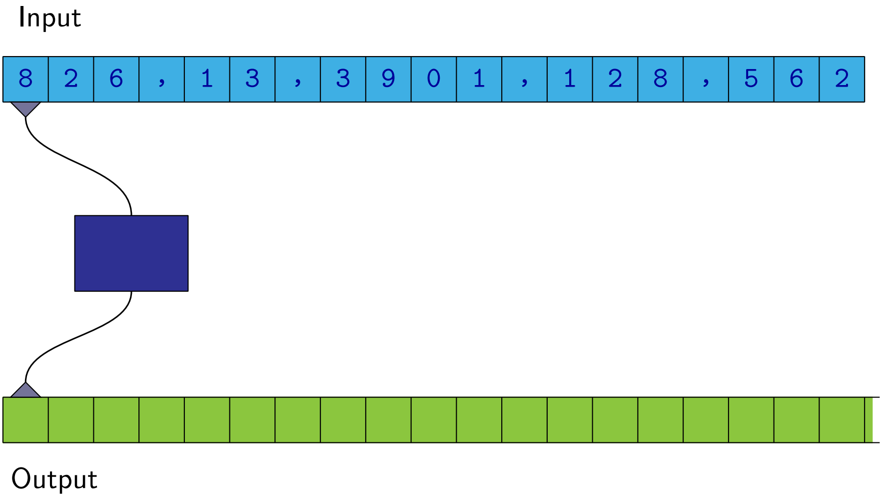
a výstupem slovo

`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.

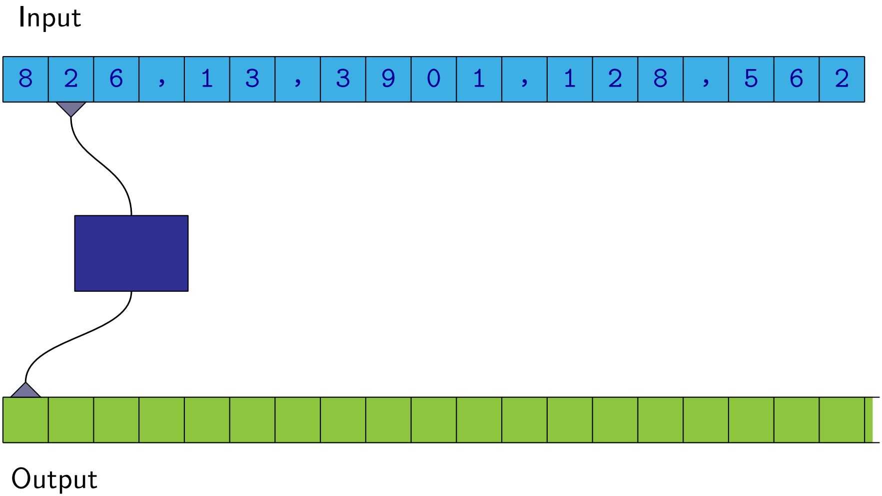
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



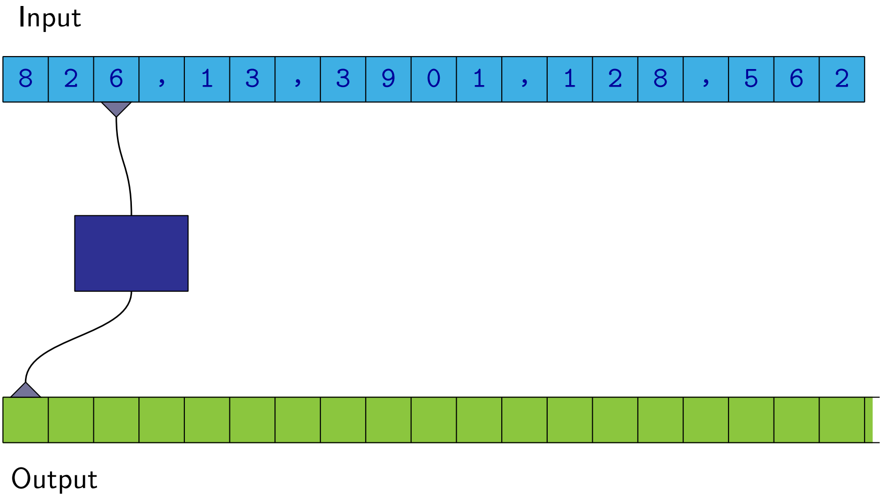
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



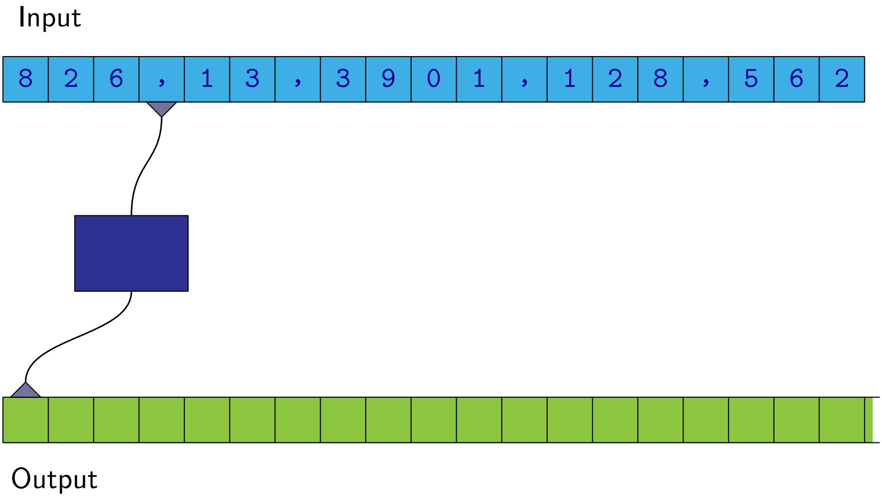
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



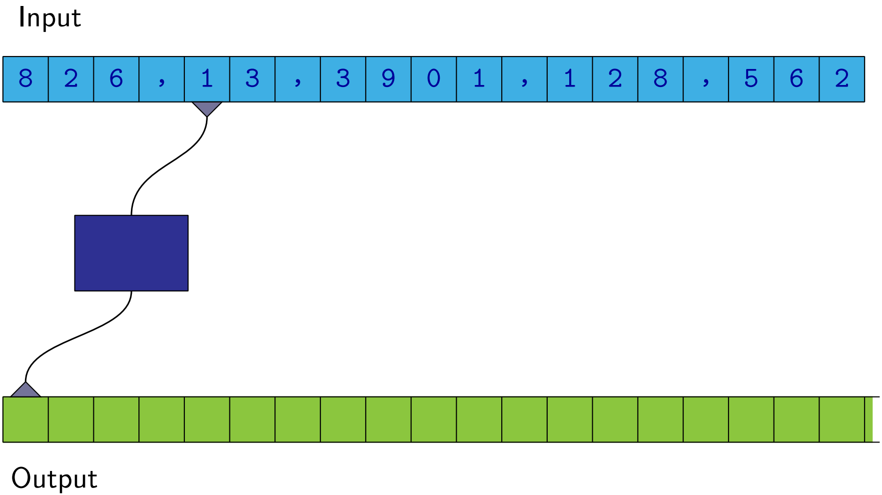
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



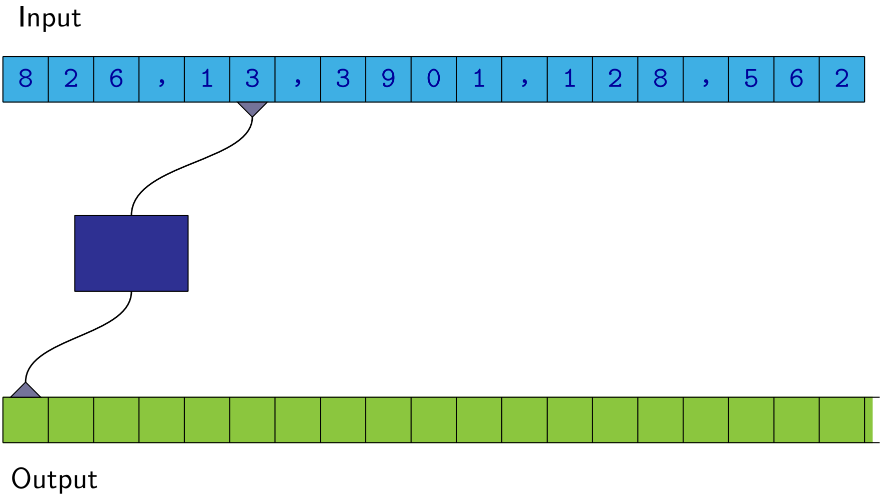
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



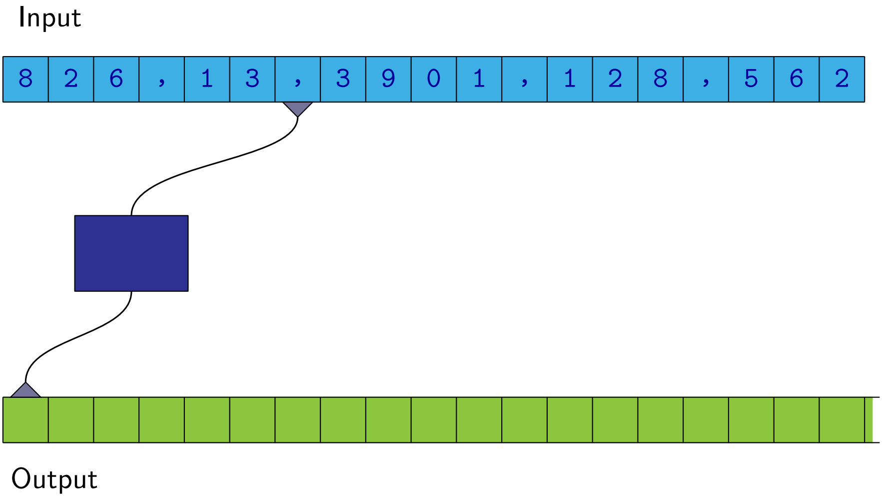
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



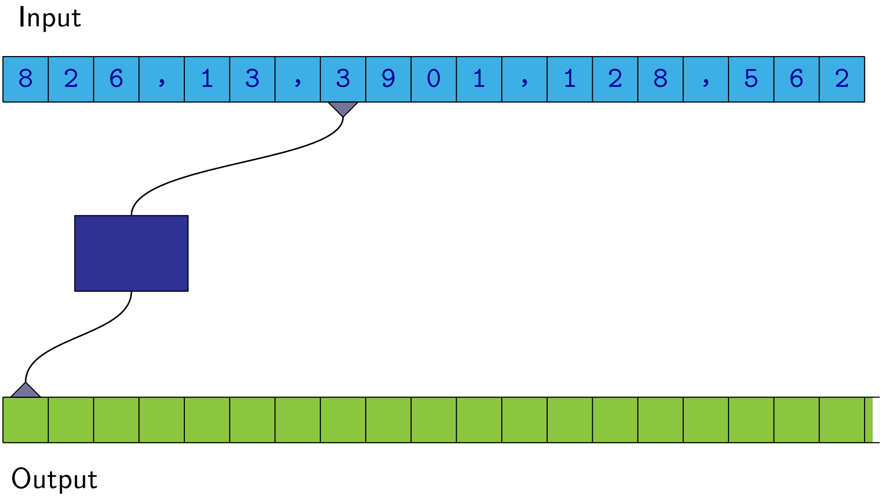
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



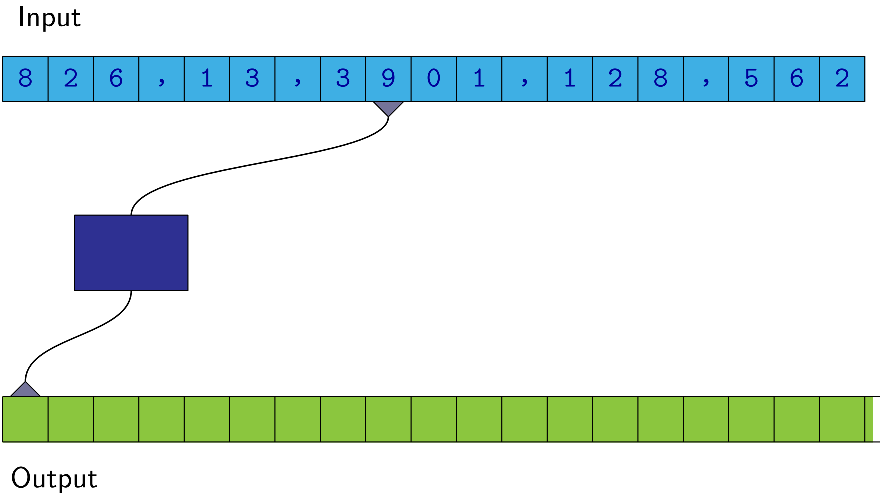
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



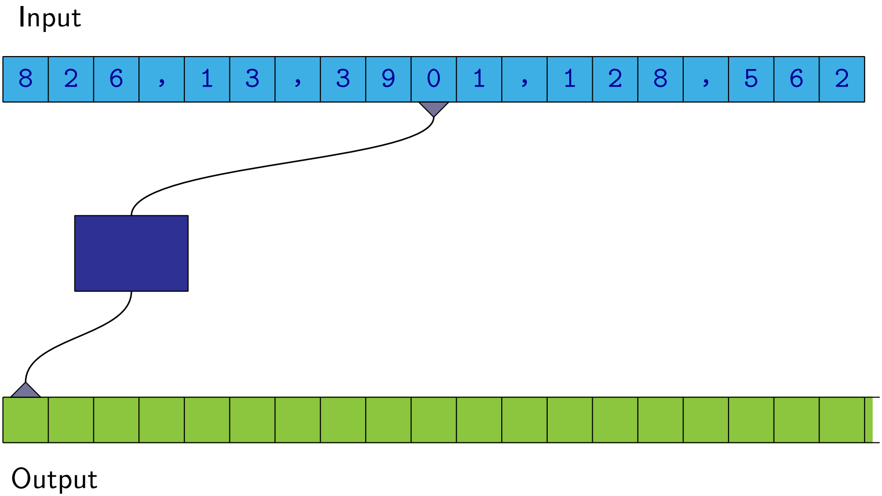
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



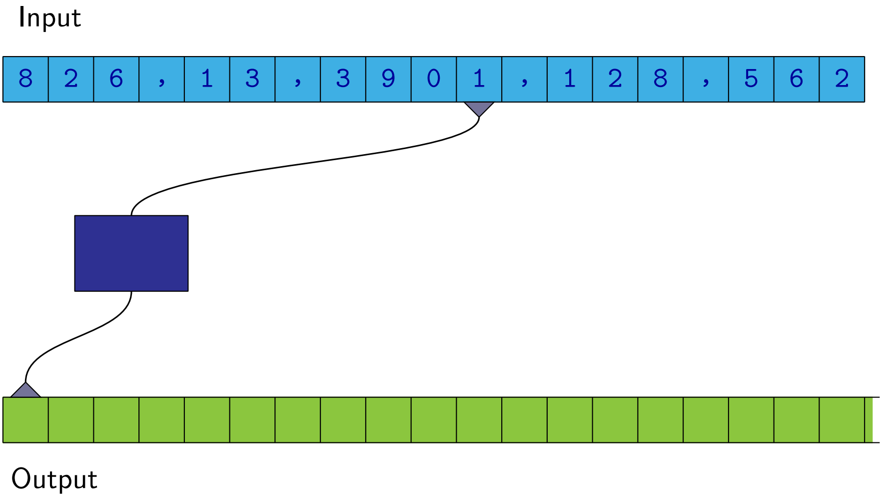
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



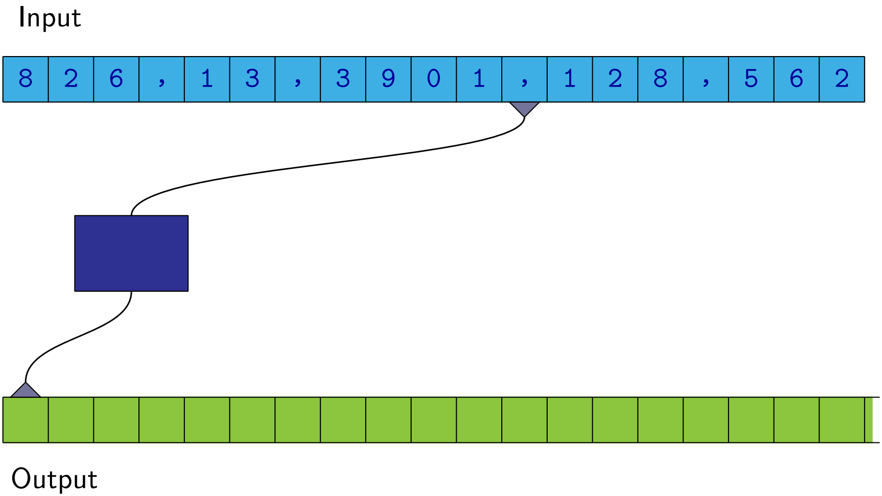
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



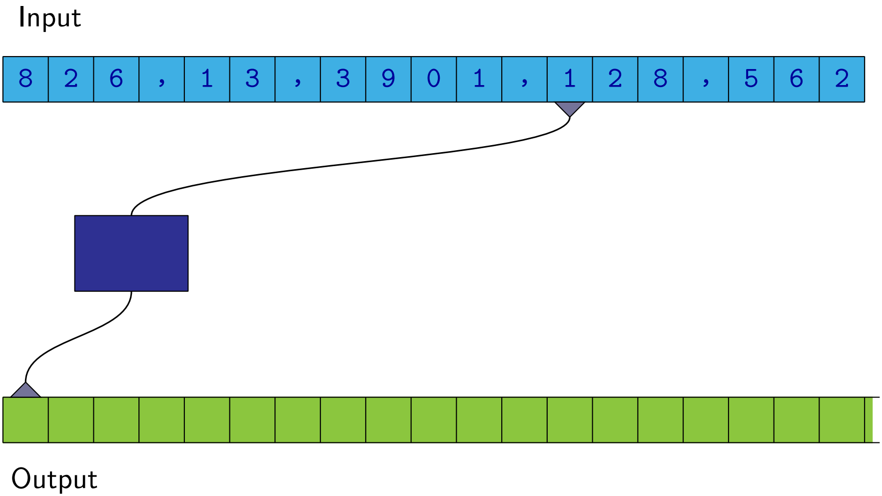
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



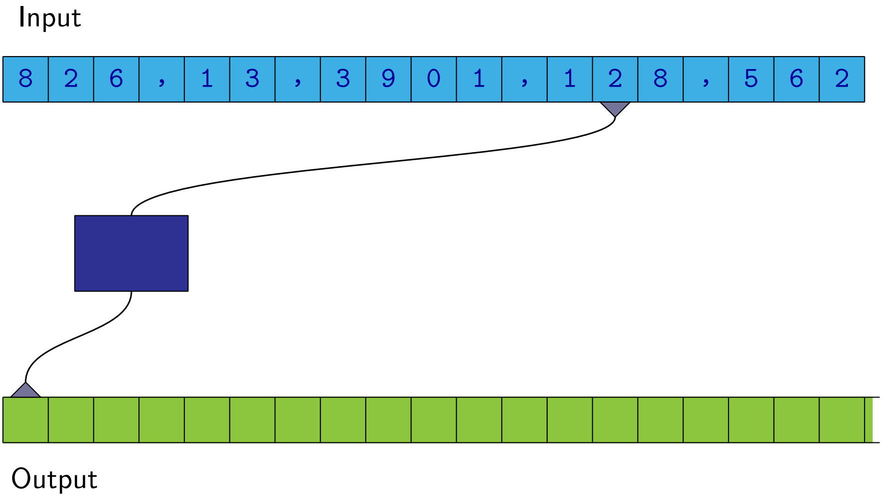
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



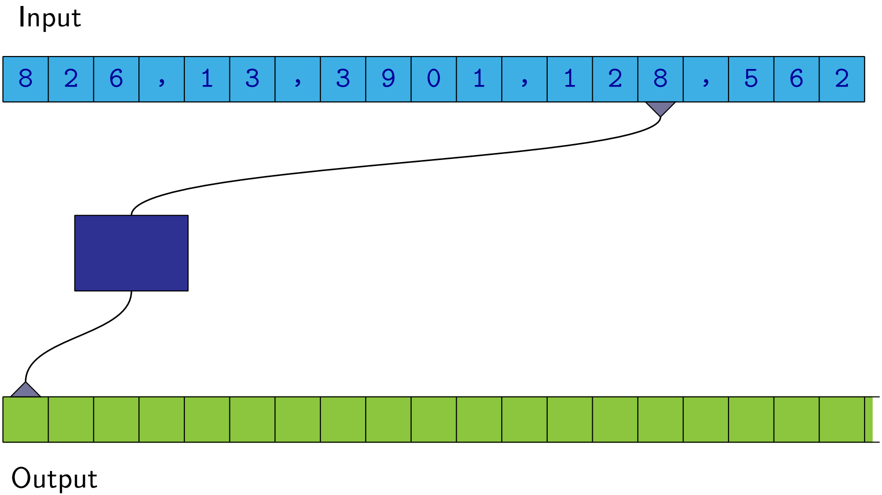
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



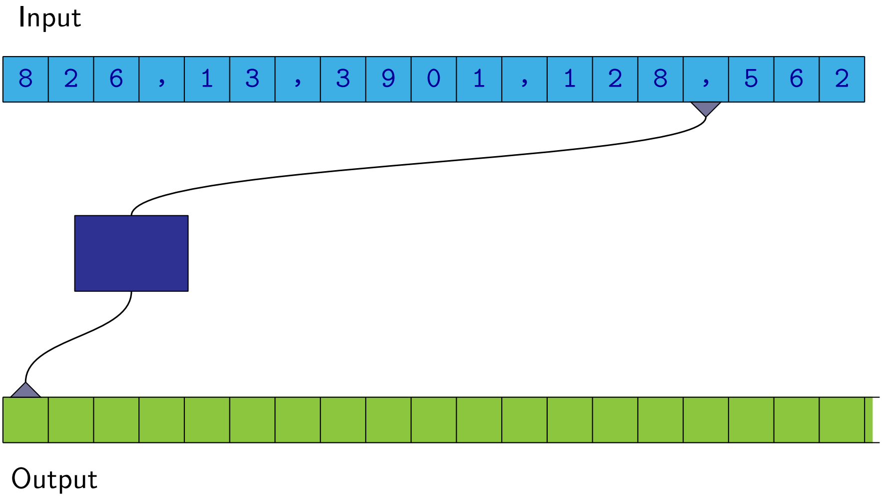
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



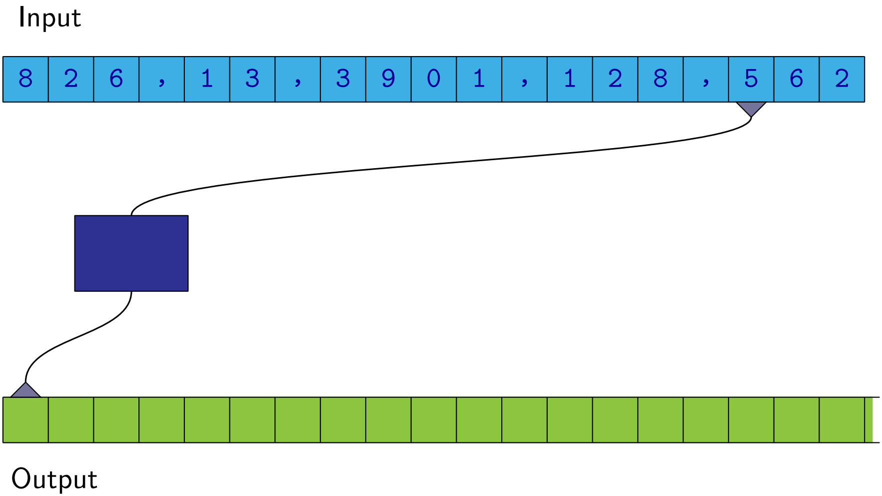
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



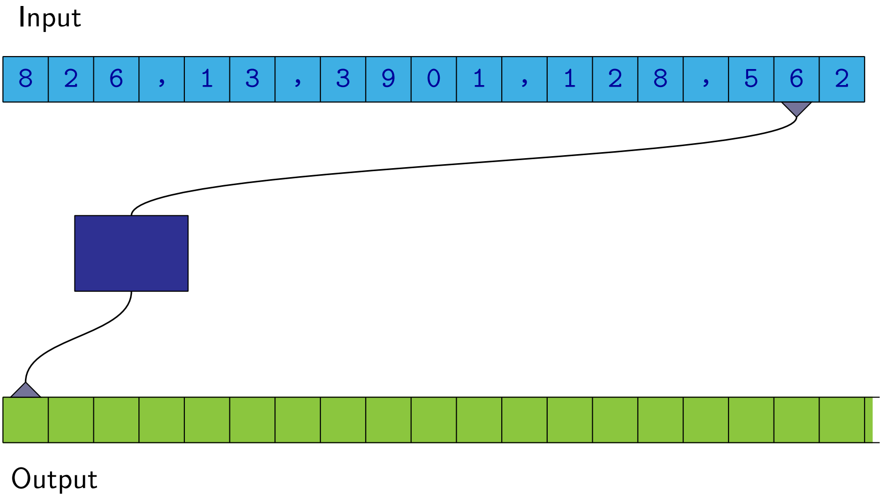
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



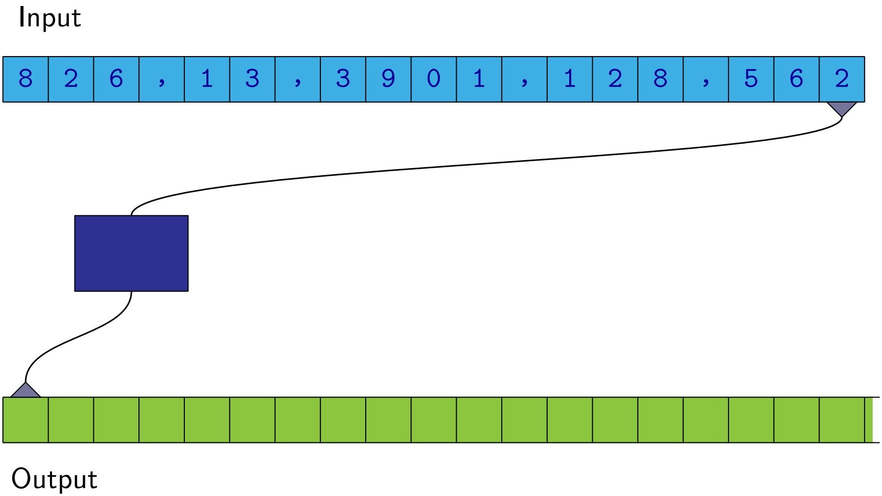
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



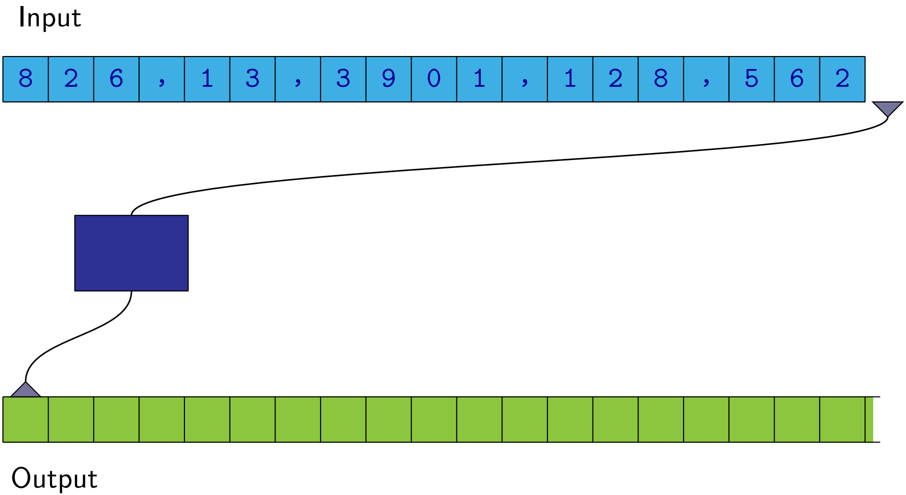
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



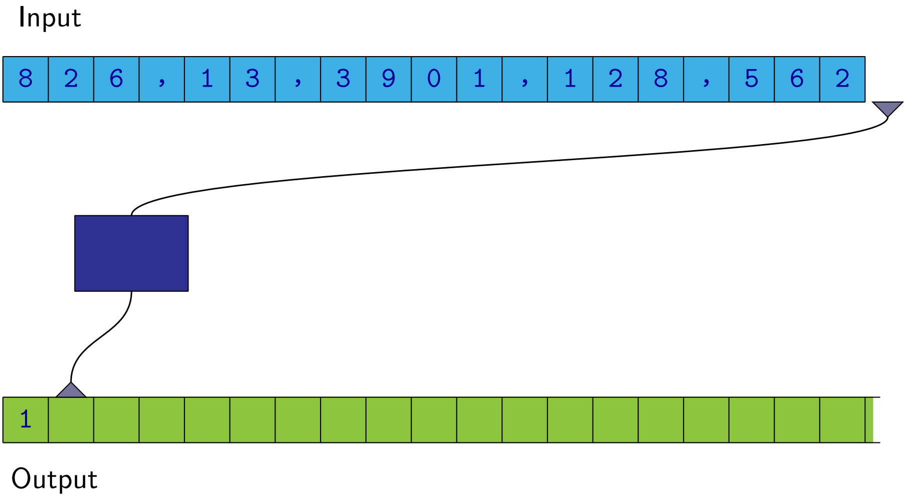
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



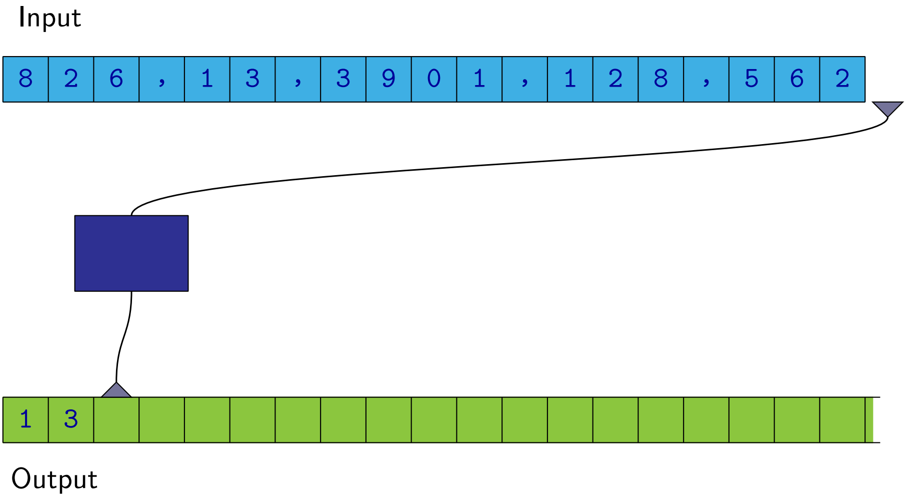
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



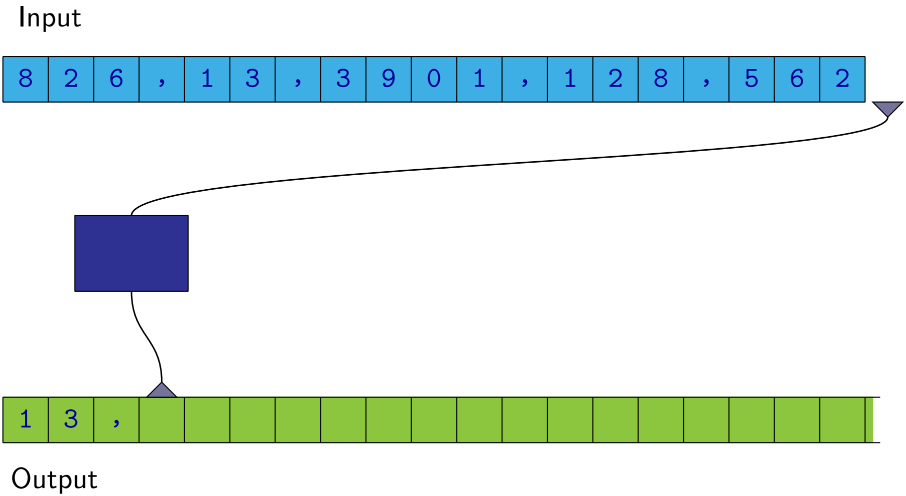
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



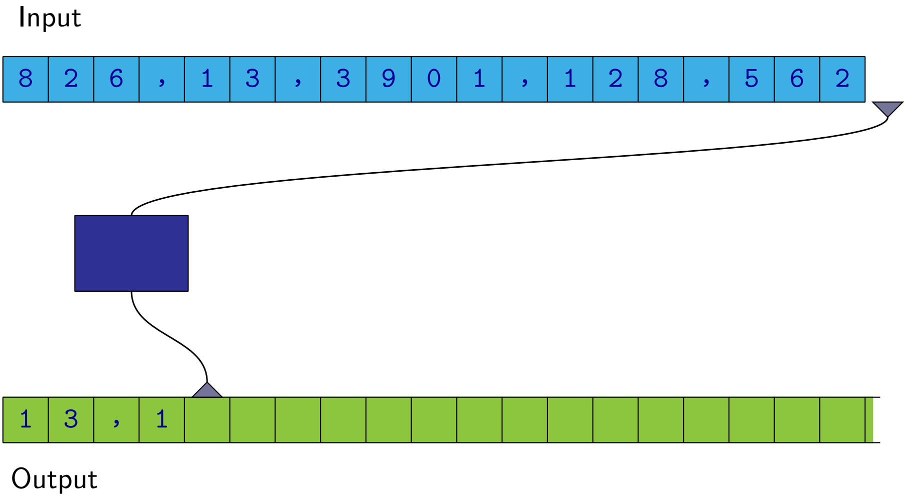
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



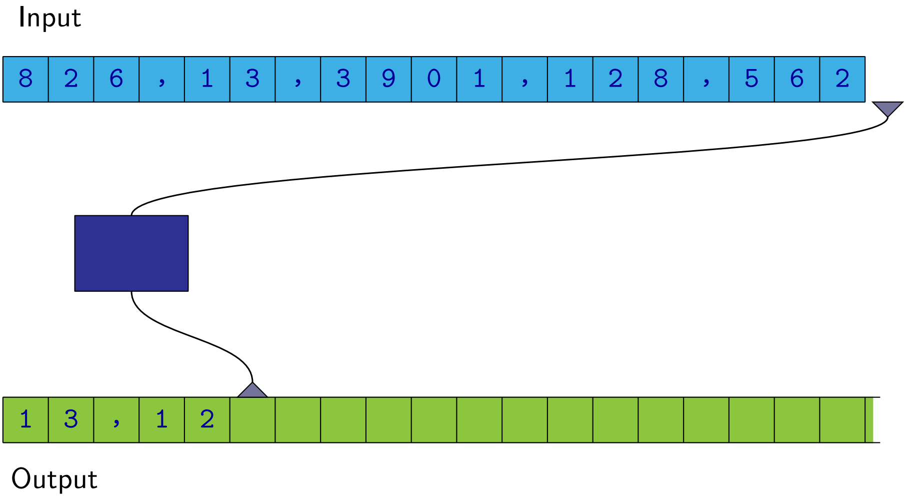
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



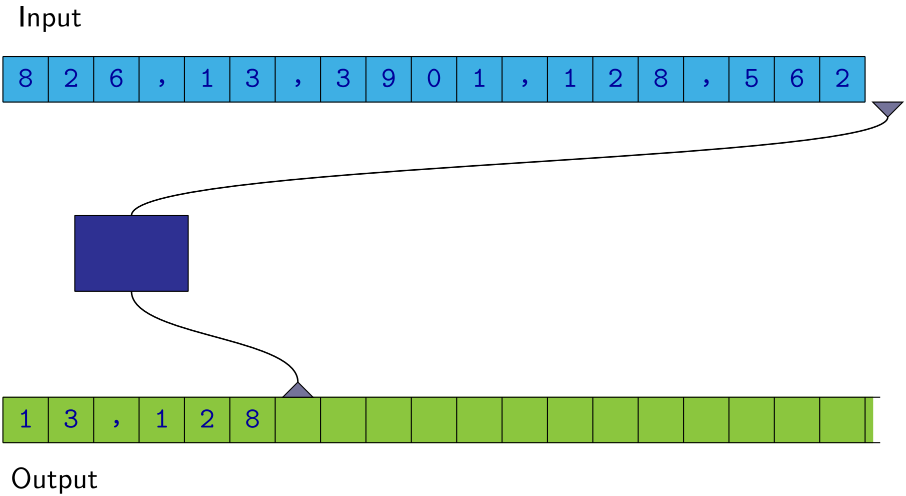
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



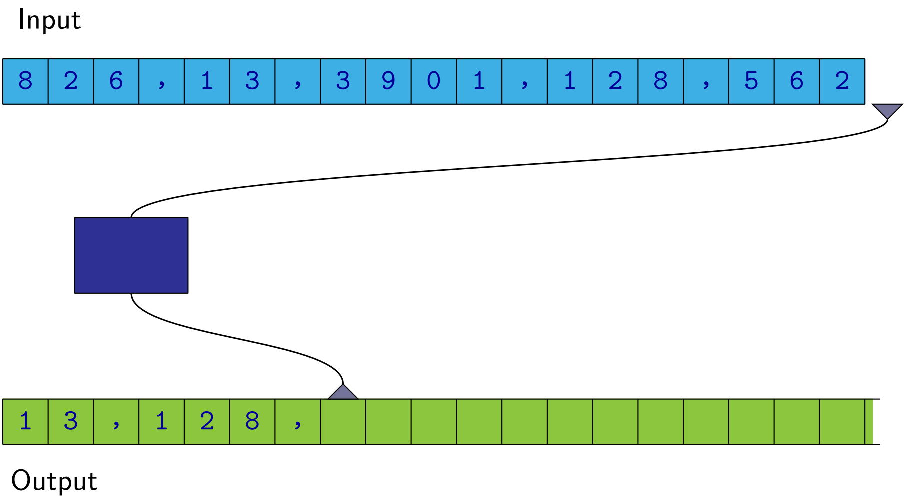
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



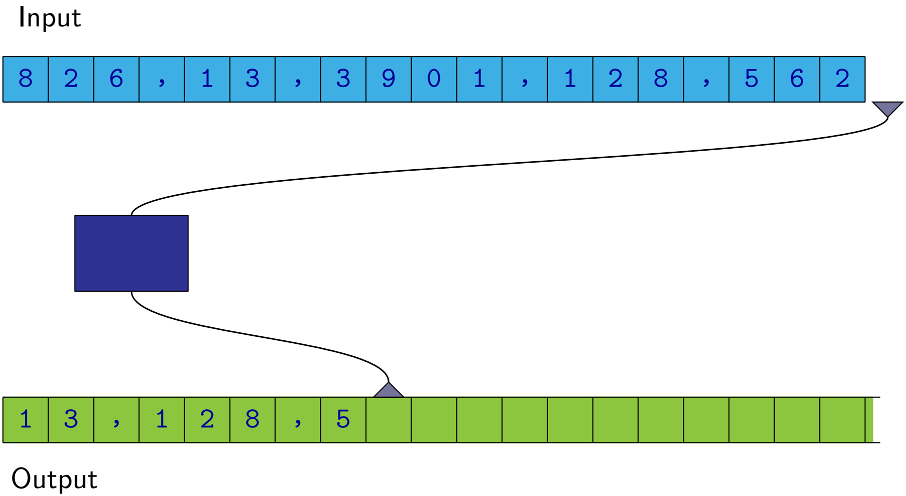
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



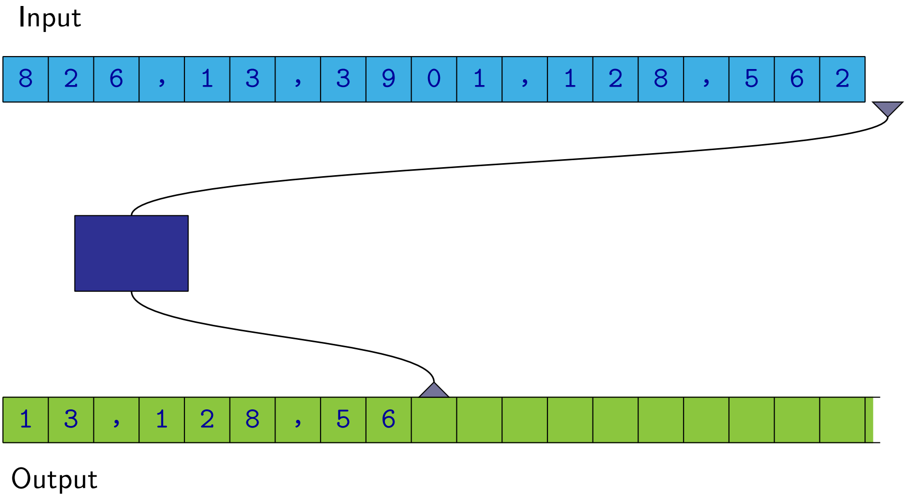
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



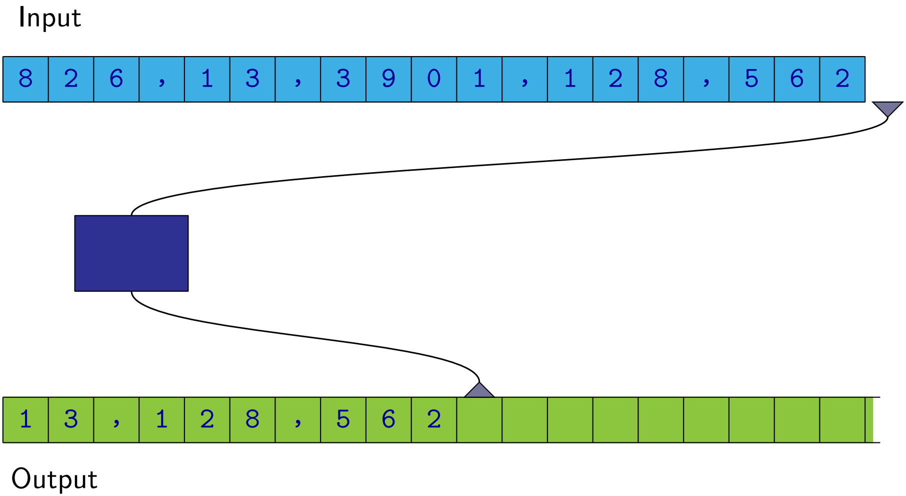
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



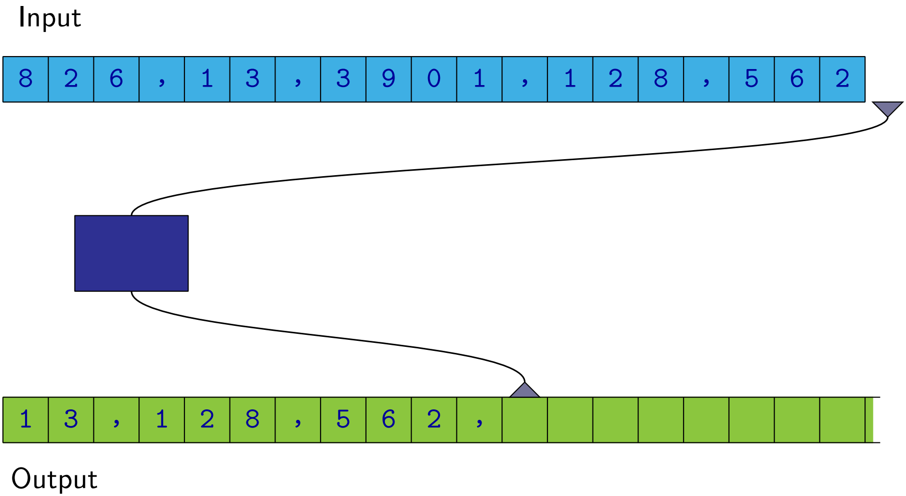
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



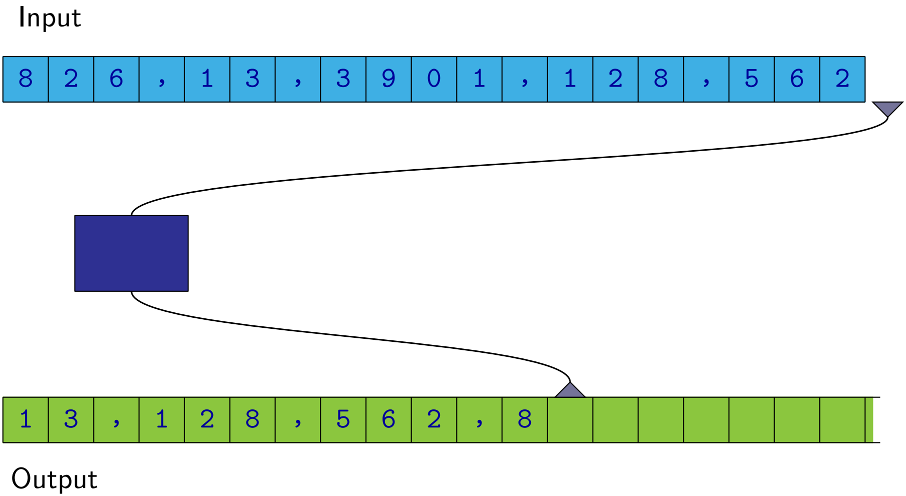
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



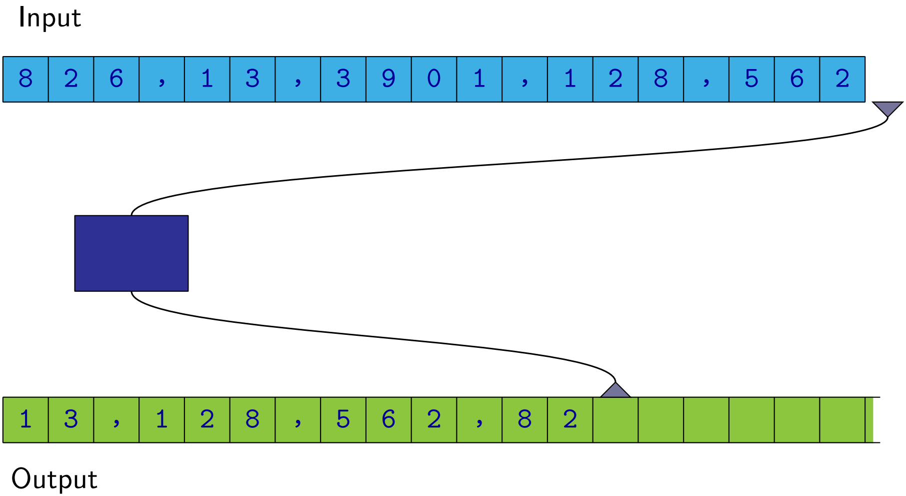
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



Činnost algoritmu

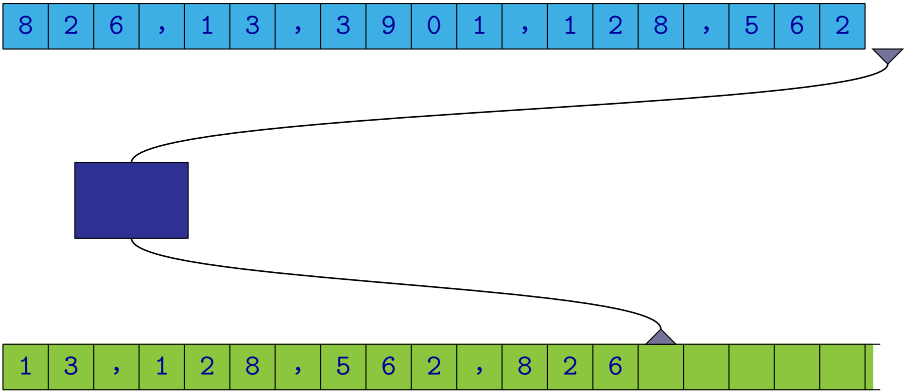
Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.

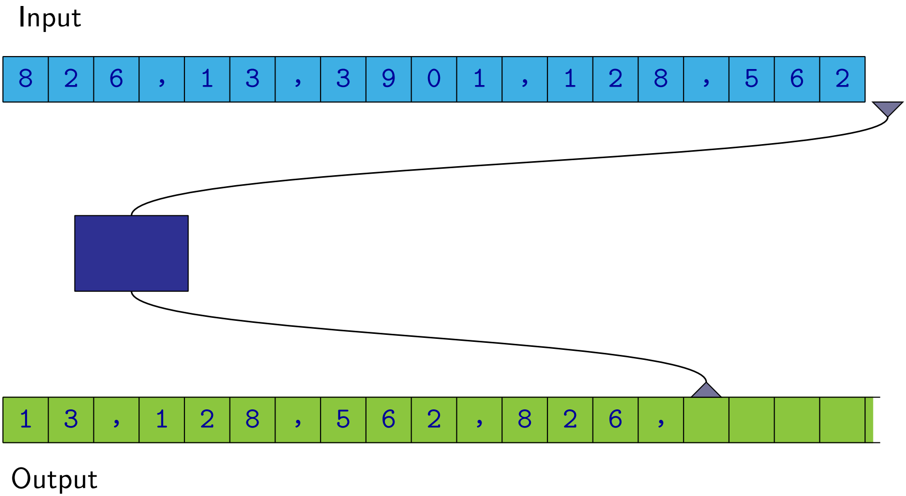
Input



Output

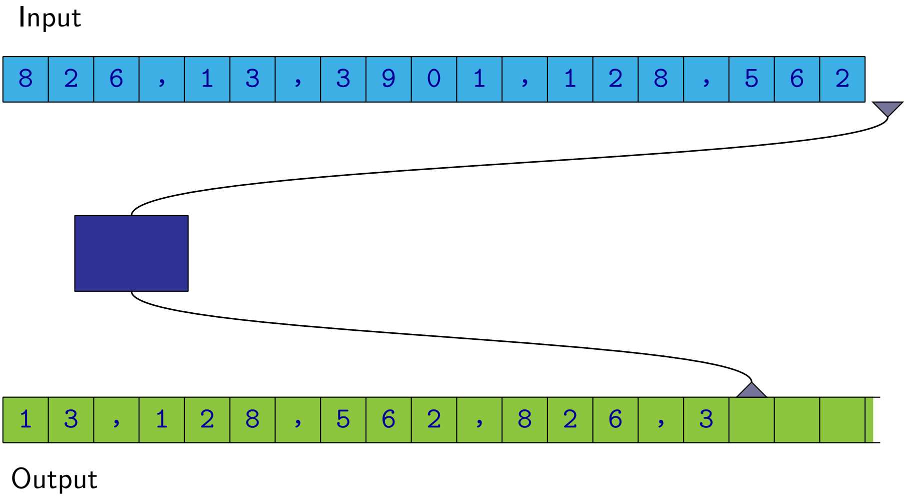
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



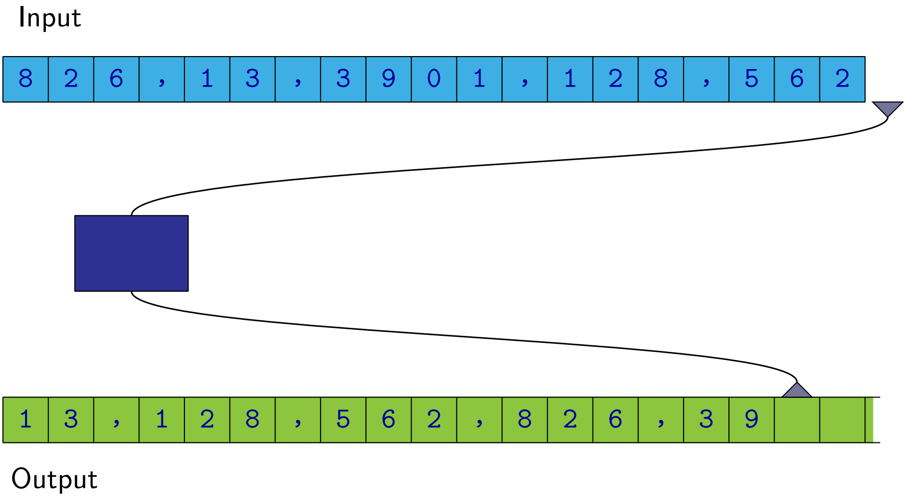
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



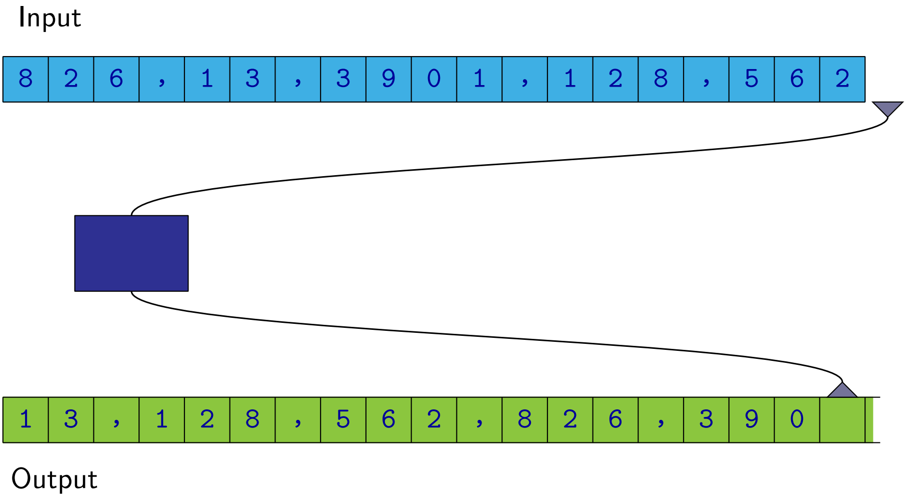
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



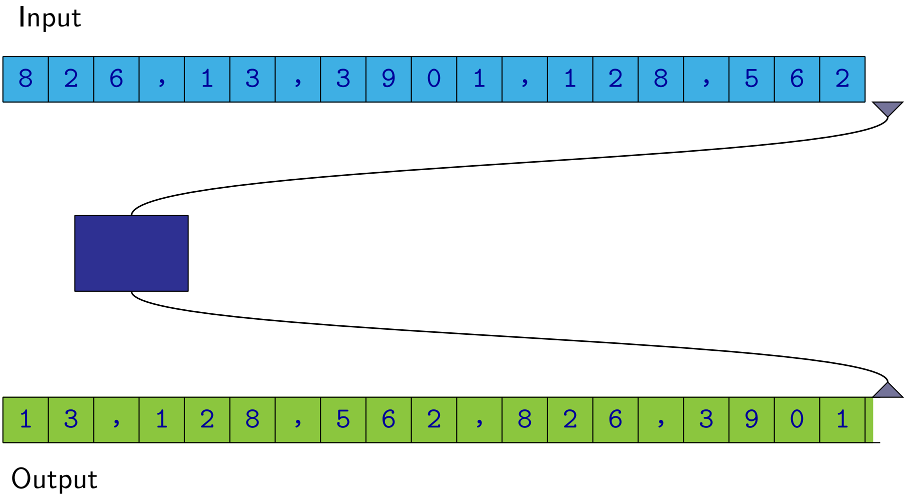
Činnost algoritmu

Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



Činnost algoritmu

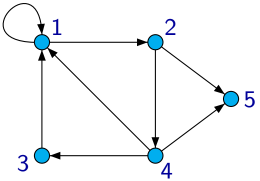
Předpokládáme, že algoritmus je vykonáván nějakým druhem stroje.



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

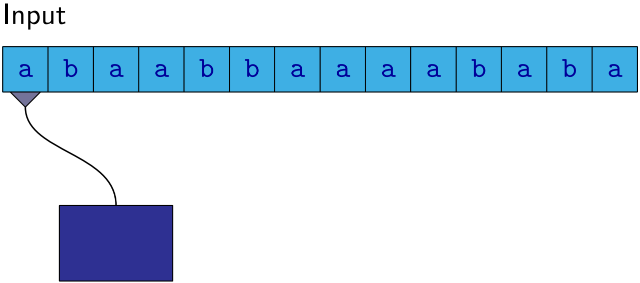
Činnost algoritmu řešícího rozhodovací problém

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

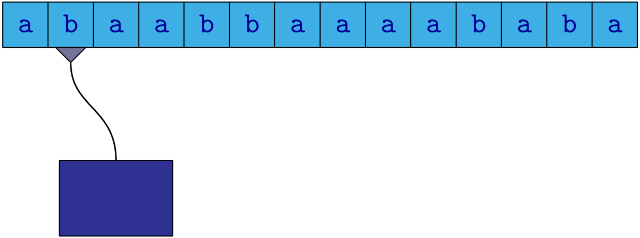
V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

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

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

Input



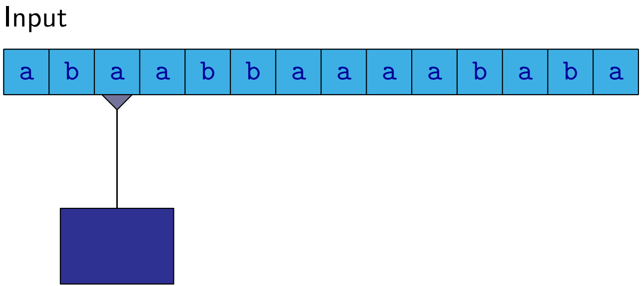
Činnost algoritmu řešícího rozhodovací problém

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 $\mathtt{b}$?



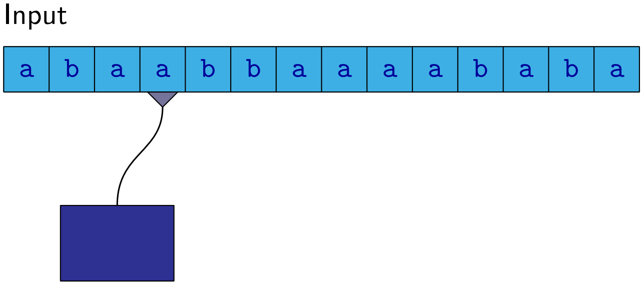
Činnost algoritmu řešícího rozhodovací problém

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 $\{a, b\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů b ?



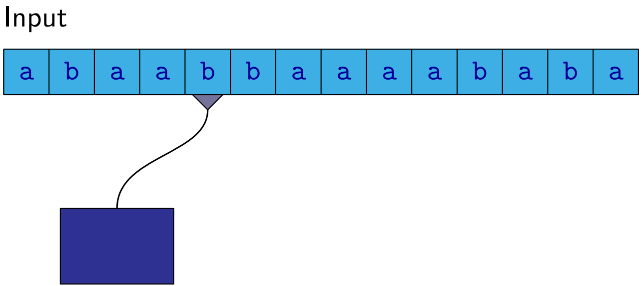
Činnost algoritmu řešícího rozhodovací problém

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

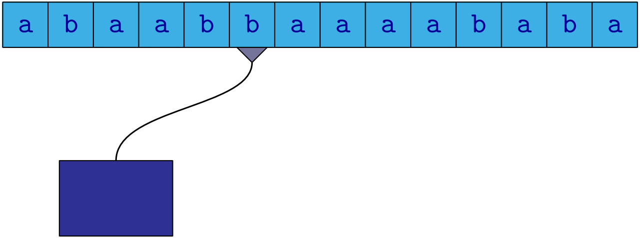
V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

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

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

Input



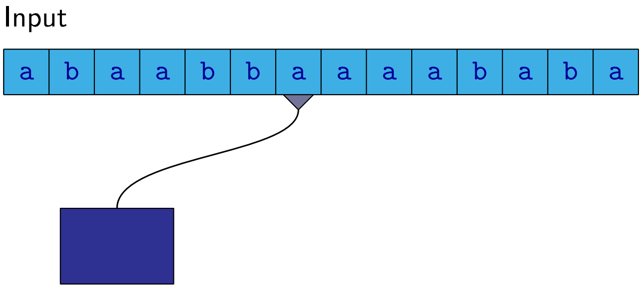
Činnost algoritmu řešícího rozhodovací problém

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 $\mathtt{b}$?



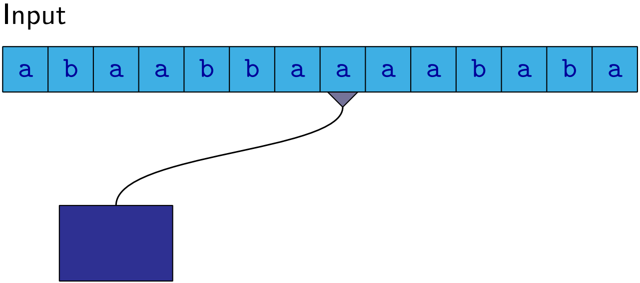
Činnost algoritmu řešícího rozhodovací problém

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 $\mathtt{b}$?



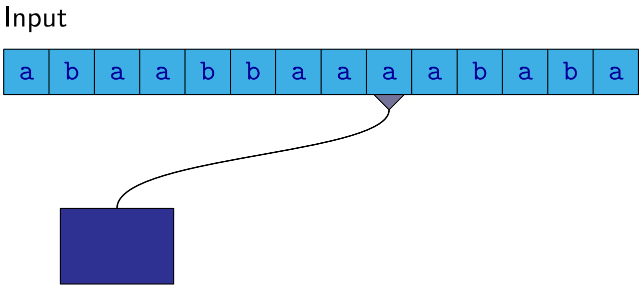
Činnost algoritmu řešícího rozhodovací problém

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 $\{a, b\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů b ?



Činnost algoritmu řešícího rozhodovací problém

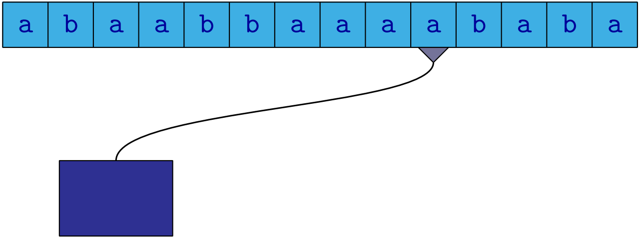
V případě algoritmu, který řeší nějaký **rozhodovací** problém stačí, když algoritmus vydá jako výstup vždy jen ANO nebo NE.

Problém

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

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

Input



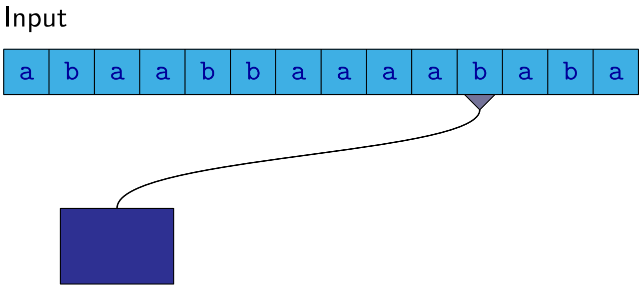
Činnost algoritmu řešícího rozhodovací problém

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 $\mathtt{b}$?



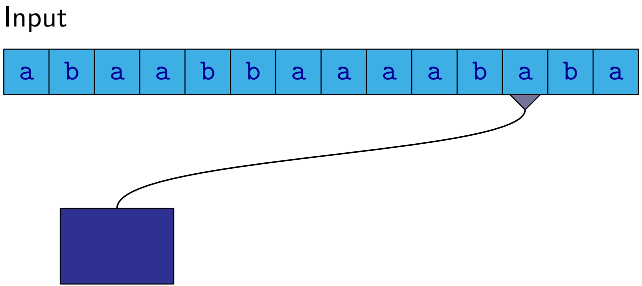
Činnost algoritmu řešícího rozhodovací problém

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 $\mathtt{b}$?



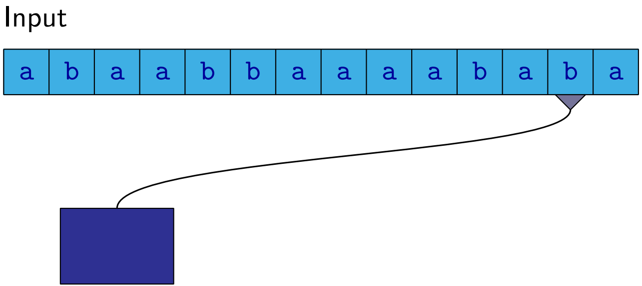
Činnost algoritmu řešícího rozhodovací problém

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 $\{a, b\}$.

Otázka: Obsahuje slovo w sudý počet výskytů symbolů b ?



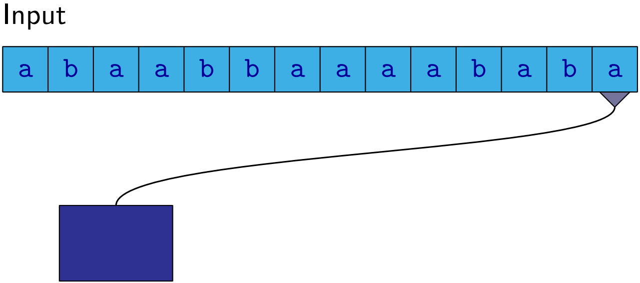
Činnost algoritmu řešícího rozhodovací problém

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 $\mathtt{b}$?



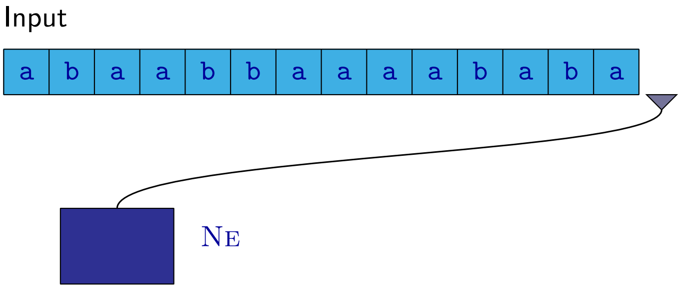
Činnost algoritmu řešícího rozhodovací problém

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

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**.

V případě velmi jednoduchých druhů strojů se běžně tyto stroje v teorii formálních jazyků označují jako **automaty**.

V tomto předmětu se seznámíme s několika druhy takovýchto automatů.

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
- ...

Formální jazyky

Abeceda a slovo

Definice

Abeceda je libovolná neprázdná konečná množina **symbolů** (**znaků**).

Poznámka: Abeceda se často označuje řeckým písmenem Σ (velké sigma).

Definice

Slovo v dané abecedě je libovolná konečná posloupnost symbolů z této abecedy.

Příklad 1:

$\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{J}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{O}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{U}, \mathtt{V}, \mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$

Slova v abecedě Σ: $\mathtt{AHOJ}$ $\mathtt{XYZZY}$ $\mathtt{COMPUTER}$

Abeceda a slovo

Příklad 2:

$\Sigma_2 = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{J}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{O}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{U}, \mathtt{V}, \mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}, \texttt{␣}\}$

Slovo v abecedě Σ_2: $\mathtt{HELLO␣WORLD}$

Příklad 3:

$\Sigma_3 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Slova v abecedě Σ_3: 0, 31415926536, 65536

Příklad 4:

Slova v abecedě $\Sigma_4 = \{0, 1\}$: 011010001, 111, 1010101010101010

Příklad 5:

Slova v abecedě $\Sigma_5 = \{\mathtt{a}, \mathtt{b}\}$: aababb, abbabbba, aaab

Jazyk

Množina všech slov tvořených symboly z abecedy Σ se označuje Σ^*.

Definice

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

Příklad 1: Množina $\{00, 01001, 1101\}$ je jazyk v abecedě $\{0, 1\}$.

Příklad 2: Množina všech syntakticky správných programů v jazyce C je jazyk v abecedě tvořené množinou všech ASCII znaků.

Příklad 3: Množina všech textů obsahujících sekvenci znaků `ahoj` je jazyk v abecedě tvořené množinou všech ASCII znaků.

Některé základní pojmy

Délka slova je počet znaků ve slově.

Například délka slova $\texttt{abaab}$ je 5.

Délku slova w označujeme $|w|$.

Pokud tedy např. $w = \texttt{abaab}$, pak $|w| = 5$.

Počet výskytů znaku a ve slově w označujeme $|w|_a$.

Příklad: Pokud $w = \texttt{cabcbba}$, pak $|w| = 7$, $|w|_\texttt{a} = 2$, $|w|_\texttt{b} = 3$, $|w|_\texttt{c} = 2$, $|w|_\texttt{d} = 0$.

Prázdné slovo je slovo délky 0, tj. slovo neobsahující žádné znaky.

Prázdné slovo se označuje řeckým písmenem ε (epsilon).

$$|\varepsilon| = 0$$

Zřetězení slov

Se slovy je možné provádět operaci **zřetězení**:

Například zřetězením slov `cabc` a `bba` vznikne slovo `cabcbba`.

Operace zřetězení se označuje symbolem $\cdot$ (podobně jako násobení). Tento symbol je možné vypouštět.

Pokud $u, v \in \Sigma^*$, pak zřetězení slov u a v tedy zapisujeme buď jako $u \cdot v$ nebo jen jako uv.

Příklad: Pokud $u = \mathtt{cabc}$ a $v = \mathtt{bba}$, pak

$$u \cdot v = \mathtt{cabcbba}$$

Poznámka: Z formálního hlediska je zřetězení slov nad abecedou Σ funkcí typu

$$\Sigma^* \times \Sigma^* \to \Sigma^*$$

Zřetězení slov

Zřetězení je **asociativní**, tj. pro libovolná tři slova u, v a w platí

$$(u \cdot v) \cdot w = u \cdot (v \cdot w)$$

Díky tomu můžeme při zápisu více zřetězení vypouštět závorky a psát například $w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5$ místo $(w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)$.

Slovo ε je pro operaci zřetězení neutrálním prvkem, pro libovolné slovo w tedy platí:

$$\varepsilon \cdot w = w \cdot \varepsilon = w$$

Poznámka: Je zjevné, že pokud daná abeceda obsahuje alespoň dva různé symboly, tak operace zřetězení není komutativní, např.

$$\mathtt{a} \cdot \mathtt{b} \neq \mathtt{b} \cdot \mathtt{a}$$

Mocnina slova

Pro libovolné slovo $w \in \Sigma^*$ a libovolné $k \in \mathbb{N}$ můžeme definovat slovo w^k jako slovo, které vznikne zřetězením k kopií slova w.

Příklad: Pro $w = \mathtt{abb}$ je $w^4 = \mathtt{abbabbabbabb}$.

Příklad: Zápis $\mathtt{a}^5\mathtt{b}^3\mathtt{a}^4$ označuje slovo $\mathtt{aaaaabbbaaaa}$.

Poněkud formálnější induktivní definice vypadá takto:

$$w^0 = \varepsilon, \qquad w^{k+1} = w^k \cdot w \quad \text{pro } k \in \mathbb{N}$$

To znamená

$$\begin{array}{rcl} w^0 & = & \varepsilon \\ w^1 & = & w \\ w^2 & = & w \cdot w \\ w^3 & = & w \cdot w \cdot w \\ w^4 & = & w \cdot w \cdot w \cdot w \\ w^5 & = & w \cdot w \cdot w \cdot w \cdot w \\ & & \cdots \end{array}$$

Zrcadlový obraz slova

Zrcadlový obraz slova w je slovo w zapsané „pozpátku".
Zrcadlový obraz slova w značíme w^R.

Příklad: $w = \mathtt{abbab}$ $\qquad w^R = \mathtt{babba}$

Pokud tedy $w = a_1 a_2 \cdots a_n$ (kde $a_i \in \Sigma$), pak $w^R = a_n a_{n-1} \cdots a_1$.

Formálně můžeme definovat w^R pomocí následující induktivně definované funkce $rev : \Sigma^* \to \Sigma^*$ jako hodnotu $rev(w)$.

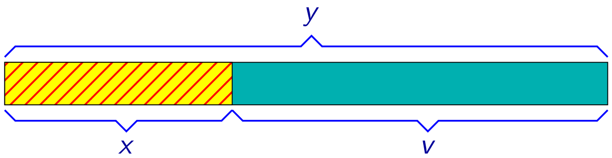
Funkce rev je definována následovně:

- $rev(\varepsilon) = \varepsilon$
- pro $a \in \Sigma$ a $w \in \Sigma^*$ je $rev(a \cdot w) = rev(w) \cdot a$

Prefix slova

Definice

Slovo x je **prefixem** slova y, jestliže existuje slovo v takové, že $y = xv$.

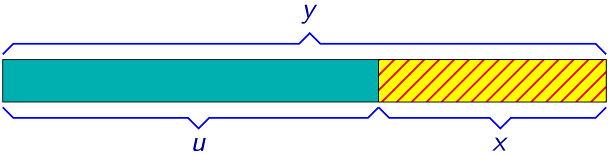


Příklad: Prefixy slova `abaab` jsou ε, `a`, `ab`, `aba`, `abaa`, `abaab`.

Sufix slova

Definice

Slovo x je **sufixem** slova y, jestliže existuje slovo u takové, že $y = ux$.

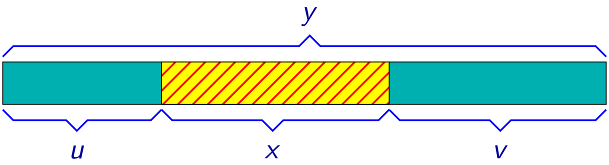


Příklad: Sufixy slova $\mathtt{abaab}$ jsou ε, $\mathtt{b}$, $\mathtt{ab}$, $\mathtt{aab}$, $\mathtt{baab}$, $\mathtt{abaab}$.

Podslovo

Definice

Slovo x je **podslovem** slova y, jestliže existují slova u a v taková, že $y = uxv$.

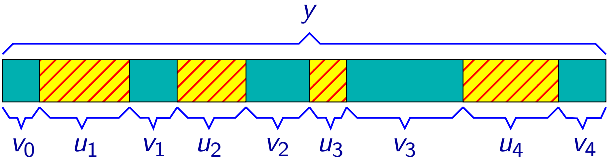


Příklad: Podslova slova $\mathtt{abaab}$ jsou ε, $\mathtt{a}$, $\mathtt{b}$, $\mathtt{ab}$, $\mathtt{ba}$, $\mathtt{aa}$, $\mathtt{aba}$, $\mathtt{baa}$, $\mathtt{aab}$, $\mathtt{abaa}$, $\mathtt{baab}$, $\mathtt{abaab}$.

Podsekvence

Definice

Slovo x je **podsekvencí** slova y, jestliže existuje číslo n a slova $u_1, u_2, \ldots, u_n$ a $v_0, v_1, \ldots, v_n$ taková, že $x = u_1 u_2 \cdots u_n$ a $y = v_0 u_1 v_1 u_2 v_2 \cdots u_n v_n$.



Příklad: Slovo `cbab` je podsekvencí slova `acabccabbaa`.

Uspořádání na slovech

Předpokládejme určité (lineární) uspořádání $<$ symbolů abecedy Σ,
tj. pokud $\Sigma = \{a_1, a_2, \ldots, a_n\}$, tak platí

$$a_1 < a_2 < \ldots < a_n \, .$$

Příklad: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, přičemž $\mathtt{a} < \mathtt{b} < \mathtt{c}$.

Na množině Σ^* můžeme definovat následující (lineární) uspořádání $<_L$:
$x <_L y$ právě tehdy, když:

- $|x| < |y|$, nebo
- $|x| = |y|$ a existují slova $u, v, w \in \Sigma^*$ a symboly $a, b \in \Sigma$ takové, že platí

$$x = uav \qquad y = ubw \qquad a < b$$

Neformálně můžeme říct v uspořádání $<_L$ řadíme slova podle délky a v rámci stejné délky lexikograficky (podle abecedy).

Uspořádání na slovech

Všechna slova nad abecedou Σ můžeme pomocí uspořádání $<_L$ seřadit do posloupnosti

$$w_0, w_1, w_2, \ldots$$

ve které se každé slovo $w \in \Sigma^*$ vyskytuje právě jednou a kde pro libovolná $i, j \in \mathbb{N}$ platí, že $w_i <_L w_j$ právě tehdy, když $i < j$.

Příklad: Pro abecedu $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ (kde $\mathtt{a} < \mathtt{b} < \mathtt{c}$) bude začátek posloupnosti vypadat následovně:

$$\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{ab}, \mathtt{ac}, \mathtt{ba}, \mathtt{bb}, \mathtt{bc}, \mathtt{ca}, \mathtt{cb}, \mathtt{cc}, \mathtt{aaa}, \mathtt{aab}, \mathtt{aac}, \mathtt{aba}, \mathtt{abb}, \mathtt{abc}, \ldots$$

Pokud budeme mluvit například o prvních deseti slovech jazyka $L \subseteq \Sigma^*$, máme tím na mysli deset slov, která patří do jazyka L a jsou mezi všemi slovy z jazyka L nejmenší vzhledem k uspořádání $<_L$.

Uspořádání na slovech

ε
a
b
c
aa
ab
ac
ba
bb
bc
ca
cb
cc
aaa
aab
aac
aba
abb
abc
$\vdots$

Příklad:

Jazyk
$L = \{ w \in \{a, b, c\}^* \mid |w|_b \bmod 2 = 0 \}$

Uspořádání na slovech

ε	1
a	2
b	
c	3
aa	4
ab	
ac	5
ba	
bb	6
bc	
ca	7
cb	
cc	8
aaa	9
aab	
aac	10
aba	
abb	11
abc	
$\vdots$	

Příklad:

Jazyk

$L = \{ w \in \{a, b, c\}^* \mid |w|_b \bmod 2 = 0 \}$

Operace na jazycích

Řekněme, že jsme nějaké jazyky již popsali. Z těchto jazyků můžeme vytvářet další nové jazyky pomocí nejrůznějších **operací na jazycích**.

Popis nějakého komplikovaného jazyka můžeme tedy „dekomponovat“ tím způsobem, že tento jazyk vyjádříme jako výsledek aplikování nějakých operací na nějaké jednodušší jazyky.

Příklady důležitých operací na jazycích:

- sjednocení
- průnik
- doplněk
- zřetězení
- iterace
- ...

Poznámka: Při operacích nad jazyky předpokládáme, že jazyky, se kterými operaci provádíme, používají tutéž abecedu Σ.

Množinové operace na jazycích

Vzhledem k tomu, že jazyky jsou množiny, můžeme s nimi provádět množinové operace:

Sjednocení – $L_1 \cup L_2$ je jazyk tvořený slovy, která patří buď do jazyka L_1 nebo do jazyka L_2 (nebo do obou).

Průnik – $L_1 \cap L_2$ je jazyk tvořený slovy, která patří současně do jazyka L_1 i do jazyka L_2.

Doplněk – $\overline{L_1}$ je jazyk tvořený těmi slovy ze Σ^*, která nepatří do L_1.

Rozdíl – $L_1 - L_2$ je jazyk tvořený slovy, která patří do L_1, ale nepatří do L_2.

Poznámka: Předpokládáme, že $L_1, L_2 \subseteq \Sigma^*$ pro nějakou danou abecedu Σ.

Množinové operace na jazycích

Formálně:

Sjednocení: $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$

Průnik: $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$

Doplněk: $\overline{L_1} = \{w \in \Sigma^* \mid w \notin L_1\}$

Rozdíl: $L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \notin L_2\}$

Množinové operace na jazycích

Příklad:

Uvažujme jazyky nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

- L_1 — množina všech slov obsahujících podslovo $\mathtt{baa}$
- L_2 — množina všech slov se sudým počtem výskytů symbolu $\mathtt{b}$

Pak

- $L_1 \cup L_2$ — množina všech slov obsahujících podslovo $\mathtt{baa}$ nebo sudý počet symbolů $\mathtt{b}$
- $L_1 \cap L_2$ — množina všech slov obsahujících podslovo $\mathtt{baa}$ a sudý počet symbolů $\mathtt{b}$
- $\overline{L_1}$ — množina všech slov, která neobsahují podslovo $\mathtt{baa}$
- $L_1 - L_2$ — množina všech slov, ve kterých se vyskytuje podslovo $\mathtt{baa}$, ale kde počet symbolů $\mathtt{b}$ není sudý

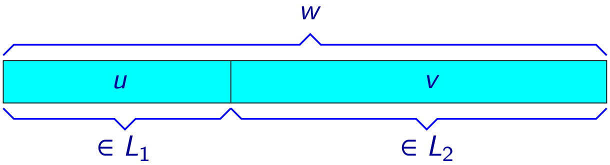
Zřetězení jazyků

Definice

Zřetězení jazyků L_1 a L_2, kde $L_1, L_2 \subseteq \Sigma^*$, je jazyk $L \subseteq \Sigma^*$ takový, že pro každé $w \in \Sigma^*$ platí

$$w \in L \iff (\exists u \in L_1)(\exists v \in L_2)(w = u \cdot v)$$

Zřetězení jazyků L_1 a L_2 označujeme zápisem $L_1 \cdot L_2$.



Zřetězení jazyků

Příklad:

$$\begin{aligned} L_1 &= \{\mathtt{abb}, \mathtt{ba}\} \\ L_2 &= \{\mathtt{a}, \mathtt{ab}, \mathtt{bbb}\} \end{aligned}$$

Jazyk $L_1 \cdot L_2$ obsahuje slova:

abba abbab abbbbb baa baab babbb

Poznámka: Všimněte si, že zřetězení jazyků je asociativní, tj. pro libovolné jazyky L_1, L_2, L_3 platí:

$$L_1 \cdot (L_2 \cdot L_3) = (L_1 \cdot L_2) \cdot L_3$$

Mocnina jazyka

Zápis L^k, kde $L \subseteq \Sigma^*$ a $k \in \mathbb{N}$, označuje zřetězení tvaru

$$L \cdot L \cdot \ \cdots \ \cdot L$$

kde se jazyk L vyskytuje k-krát, tj.

$$\begin{array}{rcl}
L^0 & = & \{\varepsilon\} \\
L^1 & = & L \\
L^2 & = & L \cdot L \\
L^3 & = & L \cdot L \cdot L \\
L^4 & = & L \cdot L \cdot L \cdot L \\
L^5 & = & L \cdot L \cdot L \cdot L \cdot L \\
 & & \cdots
\end{array}$$

Příklad: Pokud $L = \{\mathtt{aa}, \mathtt{b}\}$, pak jazyk L^3 obsahuje následující slova:

$\mathtt{aaaaaa} \quad \mathtt{aaaab} \quad \mathtt{aabaa} \quad \mathtt{aabb} \quad \mathtt{baaaa} \quad \mathtt{baab} \quad \mathtt{bbaa} \quad \mathtt{bbb}$

Mocnina jazyka

Příklad: Slovo w patřící do jazyka L^5 vznikne zřetězením pěti slov z jazyka L:

$$\overbrace{\underbrace{w_1}_{\in L}\,\underbrace{w_2}_{\in L}\,\underbrace{w_3}_{\in L}\,\underbrace{w_4}_{\in L}\,\underbrace{w_5}_{\in L}}^{w}$$

Formálně můžeme **mocninu jazyka** L^k definovat pomocí následující induktivní definice:

$$L^0 = \{\varepsilon\}, \qquad L^{k+1} = L^k \cdot L \quad \text{pro } k \in \mathbb{N}$$

Iterace jazyka

Iterace jazyka L, označovaná zápisem L^*, je jazyk tvořený slovy vzniklými zřetězením libovolného počtu slov z jazyka L.

Tj., slovo w patří do L^* právě tehdy, když existuje posloupnost $w_1, w_2, \ldots, w_n$ slov z jazyka L taková, že

$$w = w_1 w_2 \cdots w_n .$$

Příklad: $L = \{\mathtt{aa}, \mathtt{b}\}$

$$L^* = \{\varepsilon, \mathtt{aa}, \mathtt{b}, \mathtt{aaaa}, \mathtt{aab}, \mathtt{baa}, \mathtt{bb}, \mathtt{aaaaaa}, \mathtt{aaaab}, \mathtt{aabaa}, \mathtt{aabb}, \ldots\}$$

Poznámka: Počet slov, která zřetězujeme, může být i 0, což znamená, že vždy platí $\varepsilon \in L^*$ (bez ohledu na to, zda $\varepsilon \in L$ nebo ne).

Iterace jazyka

Formálně můžeme definovat jazyk L^* jako sjednocení všech mocnin jazyka L. Tj. slovo w patří do jazyka L^* právě tehdy, když existuje $k \in \mathbb{N}$ takové, že $w \in L^k$:

Definice

Iterace jazyka L je jazyk

$$L^* = \bigcup_{k \geq 0} L^k$$

Poznámka:

$$\bigcup_{k \geq 0} L^k = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$

Iterace jazyka

Zápis L^+ označuje jazyk tvořený právě těmi slovy, která vzniknou zřetězením nějakého nenulového počtu slov z jazyka L.

Platí tedy

$$L^+ = \bigcup_{k \geq 1} L^k$$

tj.

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots$$

Formálně můžeme jazyk L^+ definovat též následujícím způsobem:

$$L^+ = L \cdot L^*$$

Zrcadlový obraz jazyka

Zrcadlový obraz jazyka L je jazyk tvořený zrcadlovými obrazy všech slov z jazyka L.

Zrcadlový obraz jazyka L značíme L^R.

$$L^R = \{w^R \mid w \in L\}$$

Příklad: $L = \{\mathtt{ab}, \mathtt{baaba}, \mathtt{aaab}\}$

$L^R = \{\mathtt{ba}, \mathtt{abaab}, \mathtt{baaa}\}$

Některé vlastnosti operací na jazycích

$$
\begin{aligned}
L_1 \cup (L_2 \cup L_3) &= (L_1 \cup L_2) \cup L_3 \\
L_1 \cup L_2 &= L_2 \cup L_1 \\
L_1 \cup L_1 &= L_1 \\
L_1 \cup \emptyset &= L_1 \\
\\
L_1 \cap (L_2 \cap L_3) &= (L_1 \cap L_2) \cap L_3 \\
L_1 \cap L_2 &= L_2 \cap L_1 \\
L_1 \cap L_1 &= L_1 \\
L_1 \cap \emptyset &= \emptyset \\
\\
L_1 \cdot (L_2 \cdot L_3) &= (L_1 \cdot L_2) \cdot L_3 \\
L_1 \cdot \{\varepsilon\} &= L_1 \\
\{\varepsilon\} \cdot L_1 &= L_1 \\
L_1 \cdot \emptyset &= \emptyset \\
\emptyset \cdot L_1 &= \emptyset
\end{aligned}
$$

Některé vlastnosti operací na jazycích

$$
\begin{aligned}
L_1 \cdot (L_2 \cup L_3) &= (L_1 \cdot L_2) \cup (L_1 \cdot L_3) \\
(L_1 \cup L_2) \cdot L_3 &= (L_1 \cdot L_3) \cup (L_2 \cdot L_3) \\
(L_1^*)^* &= L_1^* \\
\emptyset^* &= \{\varepsilon\} \\
L_1^* &= \{\varepsilon\} \cup (L_1 \cdot L_1^*) \\
L_1^* &= \{\varepsilon\} \cup (L_1^* \cdot L_1) \\
(L_1 \cup L_2)^* &= L_1^* \cdot (L_2 \cdot L_1^*)^* \\
(L_1 \cdot L_2)^R &= L_2^R \cdot L_1^R
\end{aligned}
$$