Formální jazyky

Motivace 1: Vyhledávání v textu

Potřebujeme řešit následující problém:

- Máme řadu různých textů (např. soubory na disku nebo webové stránky apod.).
- Potřebujeme zjistit, které z těchto textů obsahují nějaké dané slovo či frázi, případně nějakou kombinaci slov apod.

Požadujeme, aby řešení bylo:

- **Rychlé** – můžeme prohledávat mnoho MB dat
- **Dostatečně obecné** – chceme mít možnost formulovat dostatečně obecné dotazy (např. mít možnost používat booleovské spojky)

Motivace 2: Lexikální a syntaktická analýza

Při popisu libovolného programovacího jazyka (Java, C, C++, Pascal, ...) musí být řečeno:

1. Co jsou jeho **lexikální elementy** (**tokeny**) –
   - identifikátory
   - klíčová slova
   - literály (číselné a řetězcové konstanty)
   - operátory
   - oddělovače
   - komentáře
   - ...

   a jak přesně vypadají.
2. Které sekvence těchto lexikálních elementů tvoří (syntakticky) dobře utvořené programy.

Motivace 2: Lexikální a syntaktická analýza

Chceme řešit následující problémy:

- Jak přesně (jednoznačně) popsat jednotlivé typy lexikálních elementů?
- Jak implementovat v překladači rozpoznávání těchto jednotlivých typů?
- Jak přesně popsat všechny možné způsoby jakými je možné z lexikálních elementů „poskládat“ syntakticky správně napsaný program?
- Jak implementovat v překladači rozpoznání dobře utvořených výrazů, příkazů, procedur, metod apod.?

Motivace 2: Lexikální a syntaktická analýza

Lexikální analýza – činnost překladače, kdy rozpoznává v textu jednotlivé lexikální elementy.

Syntaktická analýza – činnost překladače, kdy rozpoznává v dané sekvenci lexikálních elementů např. aritmetické výrazy, příkazy, podprogramy nebo i celé programy.

Základní pojmy

Co mají všechny dosud zmíněné problémy společného?

- Pracujeme se sekvencemi **znaků** (říkáme též **symbolů**).
- Znaky patří do nějaké konečné **abecedy** (typicky např. ASCII nebo Unicode).
- Musíme rozpoznávat ty sekvence znaků, které nějakou vlastnost mají a ty co ji nemají.

Poznámka: V teorii formálních jazyků se sekvencím znaků říká **slova**. Při programování máme na mysli například řetězce (stringy) nebo třeba soubory na disku apod.

Množině slov z nějaké abecedy se říká **jazyk**.

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 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: `AHOJ` `ABRACADABRA` `ERROR`

Abeceda a slovo

Příklad 2:

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

Slovo v abecedě Σ: $\mathtt{HELLO}\sqcup\mathtt{WORLD}$

Příklad 3:

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

Slova v abecedě Σ: 0, 314159, 666, 65536

Příklad 4:

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

Příklad 5:

Slova v abecedě $\Sigma = \{a, b\}$: *aababb*, *abbabbba*, *aaab*

Abeceda a slovo

Příklad 6:

Abeceda Σ je množina všech ASCII znaků.

Příklad slova:

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

```
class␣HelloWorld␣{ ↩ ␣␣␣␣public␣static␣void␣main(Str···
```

Délka slova

Délka slova je počet znaků ve slově.

Například délka slova $abaab$ je 5.

Délku slova w označujeme $|w|$.

Pokud tedy např. $w = abaab$, pak $|w| = 5$.

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

Prázdné slovo se označuje řeckým písmenem ε (epsilon).

(Pozn.: Někteří autoři používají pro označení prázdného slova místo ε řecké písmeno λ (lambda).)

$$|\varepsilon| = 0$$

Zřetězení slov

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

Například zřetězením slov OST a RAVA vznikne slovo OSTRAVA.

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

$$\mathtt{OST} \cdot \mathtt{RAVA} = \mathtt{OSTRAVA}$$

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

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

což znamená, že při zápisu více zřetězení můžeme vypouštět závorky a psát například $w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5$ místo $(w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)$

Zřetězení slov

Zřetězení není **komutativní**, tj. obecně pro dvojici slov u a v neplatí rovnost

$$u \cdot v = v \cdot u$$

Příklad:

$$\texttt{OST} \cdot \texttt{RAVA} \neq \texttt{RAVA} \cdot \texttt{OST}$$

Zjevně pro libovolná slova v a w platí:

$$|v \cdot w| = |v| + |w|$$

Pro libovolné slovo w také platí:

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

Jazyk

Množinu všech slov tvořených symboly z abecedy Σ označujeme Σ^*.

Definice

(Formální) jazyk v abecedě Σ je nějaká libovolná podmnožina množiny Σ^*.

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

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

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

Jazyk

Příklad 4: Uvažujme abecedu Σ tvořenou množinou všech Unicode znaků.

Množina všech komentářů v jazyce Java tvoří jazyk:

- jednořádkové komentáře začínající dvojicí znaků `//` a končící znakem konce řádku (nebo koncem souboru).
- víceřádkové komentáře začínající dvojicí znaků `/*` a končící dvojicí znaků `*/`, přičemž uvnitř se nesmí nacházet žádná další dvojice znaků `*/`.

Pokud bychom množinu všech jednořádkových komentářů označili jako jazyk L_1 a množinu všech víceřádkových komentářů jako jazyk L_2, můžeme množinu všech komentářů označit jako jazyk L, definovaný jako

$$L = L_1 \cup L_2$$

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

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

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

Množinové operace na jazycích

Příklad:

Uvažujme množinu všech textů tvořených ASCII znaky. Jestliže:

- L_1 je množina všech textů, ve kterých se vyskytuje sekvence znaků `FOO`, a
- L_2 je množina všech textů, ve kterých se vyskytuje sekvence znaků `BAR`,

pak

- $L_1 \cup L_2$ jsou všechny texty, ve kterých se vyskytuje `FOO` nebo `BAR`,
- $L_1 \cap L_2$ jsou všechny texty, ve kterých se vyskytuje `FOO` i `BAR`,
- $\overline{L_1}$ jsou všechny texty, ve kterých se nevyskytuje `FOO`,
- $L_1 - L_2$ jsou všechny texty, ve kterých se vyskytuje `FOO`, ale nevyskytuje `BAR`.

Zřetězení jazyků

Definice

Zřetězení jazyků L_1 a L_2 je jazyk

$$L = \{uv \mid u \in L_1, v \in L_2\}$$

tj. jazyk všech slov, která začínají slovem z L_1 a pokračují slovem z L_2.
Zřetězení jazyků L_1 a L_2 označujeme $L_1 \cdot L_2$.

Příklad:

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

Jazyk $L_1 \cdot L_2$ obsahuje slova:

$$abba \qquad abbab \qquad abbbbb \qquad baa \qquad baab \qquad babbb$$

Zřetězení jazyků

Příklad:
Pro nějaký programovací jazyk chceme definovat, jak mohou vypadat konstanty reprezentující čísla v plovoucí řádové čárce (floating-point), např.:

`1e1` `2.0` `.3` `0.0` `3.14` `1E-9` `4.5e137`

Abeceda $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., \mathtt{e}, \mathtt{E}, +, -\}$

Zřetězení jazyků

Příklad:
Pro nějaký programovací jazyk chceme definovat, jak mohou vypadat konstanty reprezentující čísla v plovoucí řádové čárce (floating-point), např.:

`1e1` `2.0` `.3` `0.0` `3.14` `1E-9` `4.5e137`

Abeceda $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., \mathtt{e}, \mathtt{E}, +, -\}$

Pokud zvolíme L_{num} jako množinu všech (neprázdných) slov tvořených pouze číslicemi, a $L_{dot} = \{.\}$, můžeme konstanty jako například `3467.982`, `3.141592` nebo `0.0` popsat takto:

$$L_{num} \cdot L_{dot} \cdot L_{num}$$

Zřetězení jazyků

Definice jazyka L všech možných konstant v plovoucí řádové čárce by pak mohla vypadat například takto:

$$\begin{aligned} L \quad = \quad & L_{num} \cdot L_{dot} \cdot (L_{num} \cup \{\varepsilon\}) \cdot (L_{exp} \cup \{\varepsilon\}) \quad \cup \\ & L_{dot} \cdot L_{num} \cdot (L_{exp} \cup \{\varepsilon\}) \quad \cup \\ & L_{num} \cdot L_{exp} \end{aligned}$$

kde

$$\begin{aligned} L_{num} \quad & - \quad \text{neprázdné sekvence číslic} \\ L_{dot} \quad & = \quad \{\,.\,\} \\ L_{exp} \quad & = \quad L_E \cdot L_{sign} \cdot L_{num} \\ L_E \quad & = \quad \{\mathtt{E}, \mathtt{e}\} \\ L_{sign} \quad & = \quad \{\varepsilon, +, -\} \end{aligned}$$

Iterace jazyka

Používáme následující zápis:

$$\begin{aligned} L^2 &= L \cdot L \\ L^3 &= L \cdot L \cdot L \\ L^4 &= L \cdot L \cdot L \cdot L \\ L^5 &= L \cdot L \cdot L \cdot L \cdot L \\ &\dots \end{aligned}$$

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

$$aaaaaa \quad aaaab \quad aabaa \quad aabb \quad baaaa \quad baab \quad bbaa \quad bbb$$

Definujeme

$$\begin{aligned} L^1 &= L \\ L^0 &= \{\varepsilon\} \end{aligned}$$

Iterace jazyka

Induktivní definice pro libovolné $k \geq 0$:

$$L^0 = \{\varepsilon\}, \qquad L^{k+1} = L^k \cdot L \quad \text{pro } k \geq 0.$$

Definice

Iterace jazyka L je jazyk

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$

tj. jazyk tvořený slovy vzniklými zřetězením libovolného počtu slov z jazyka L.

Příklad: $L = \{aa, b\}$

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

Iterace jazyka

Příklad: $L_{dig} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

$$L_{num} = L_{dig} \cdot L_{dig}^*$$

Poznámka: Používá se také následující značení:

$$L^+ = L^1 \cup L^2 \cup L^3 \cup L^4 \cup \cdots$$

Řešení předchozího příkladu bychom tedy také mohli zapsat stručněji:

$$L_{num} = L_{dig}^+$$

Zrcadlový obraz

Zrcadlový obraz slova w je slovo w zapsané „pozpátku“.

Zrcadlový obraz slova w značíme w^R.

Příklad: $w = \texttt{AHOJ}$ $\quad$ $w^R = \texttt{JOHA}$

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

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

Práce s formálními jazyky

Když chceme nějaký jazyk popsat, máme několik možností:

- Můžeme vyjmenovat všechna jeho slova (což je ale použitelné jen pro konečné jazyky).

  Příklad: $L = \{aab, babba, aaaaaa\}$

- Můžeme specifikovat nějakou vlastnost, kterou mají právě ta slova, která do tohoto jazyka patří:

  Příklad: Jazyk nad abecedou $\{0, 1\}$, obsahující všechna slova, ve kterých je počet výskytů symbolu 1 sudý.

Práce s formálními jazyky

V teorii formálních jazyků se používají především následující dva přístupy:

- Popsat (idealizovaný) stoj, zařízení, algoritmus, který rozpozná slova patřící do daného jazyka – vede k použití tzv. **automatů**.

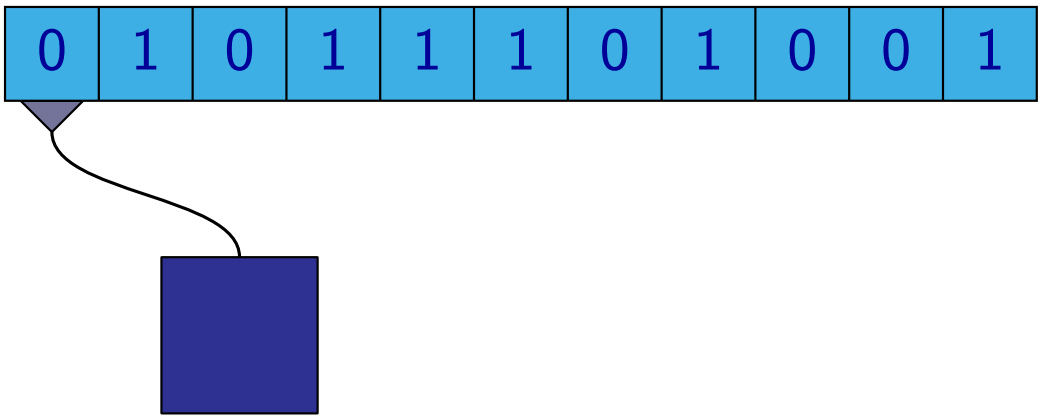
- Popsat postup, jak mechanicky generovat všechna možná slova patřící do daného jazyka – vede k tzv. **gramatikám** a **regulárním výrazům**. (budeme se jim věnovat později)

Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

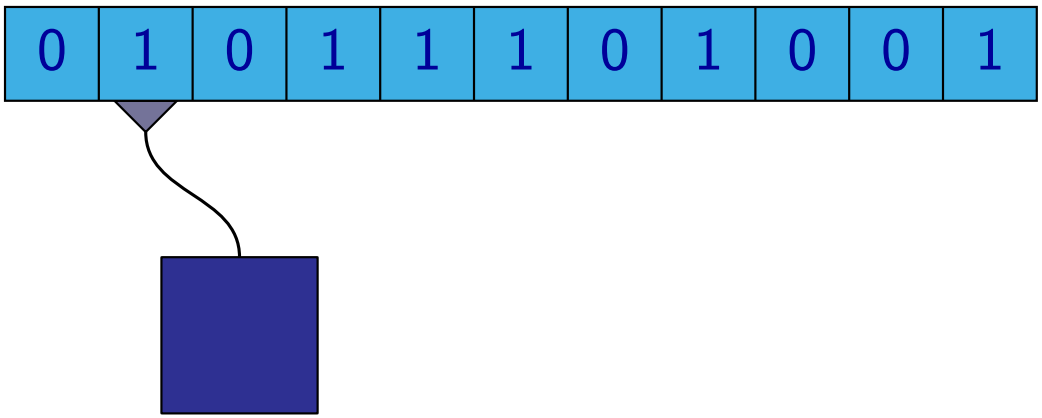


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

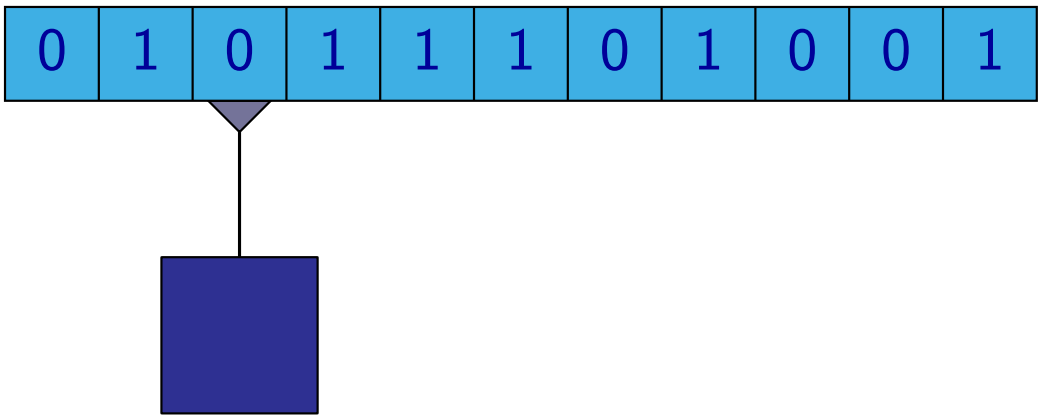


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

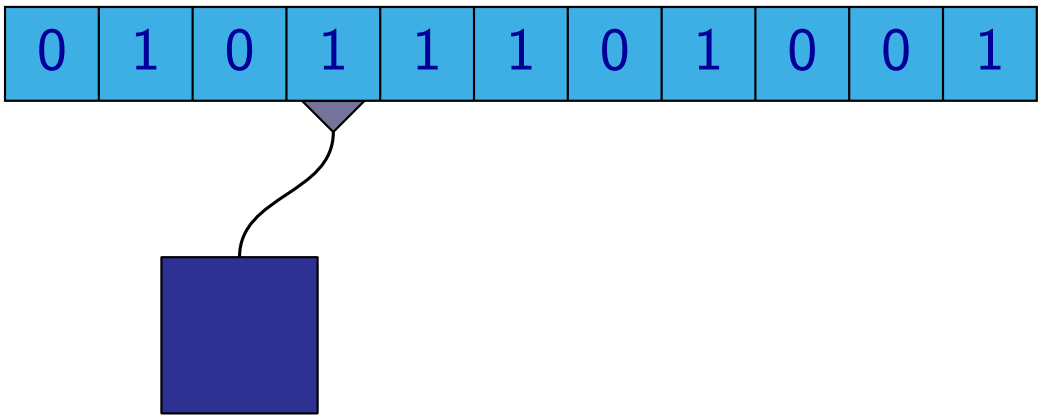


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

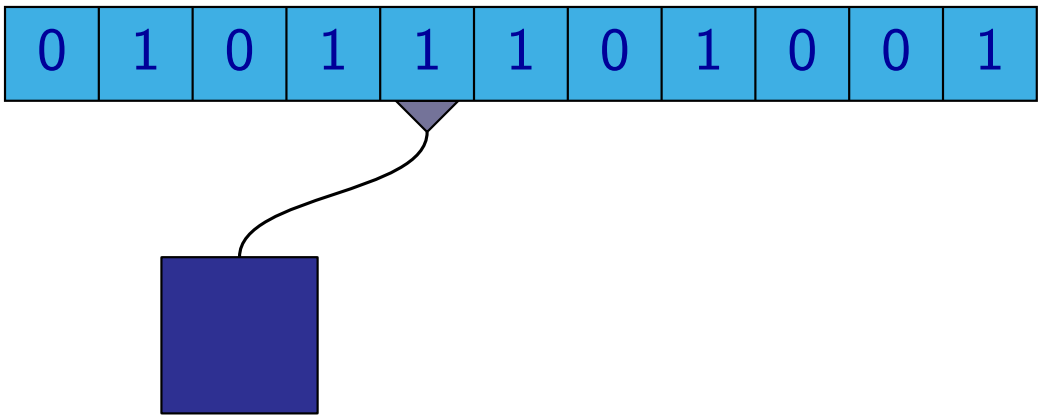


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

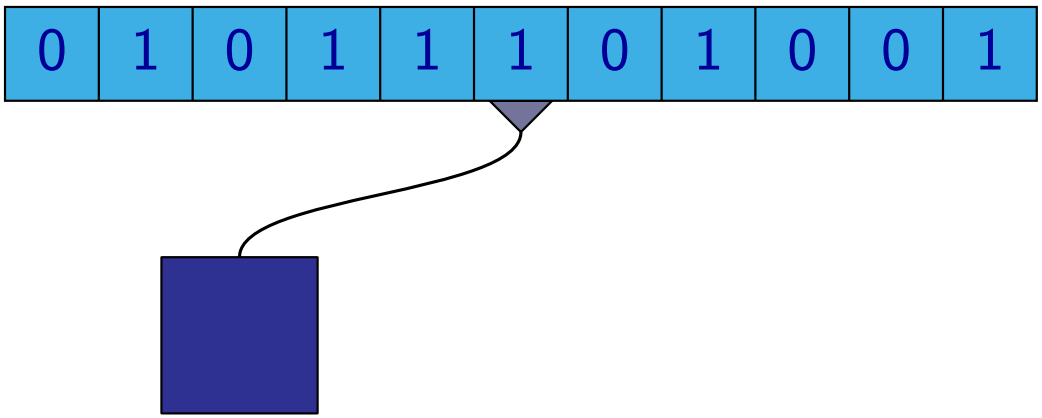


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

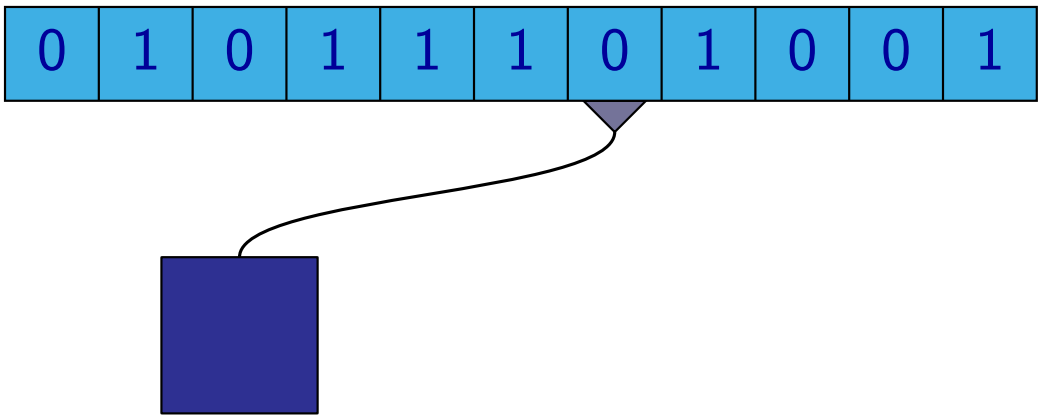


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

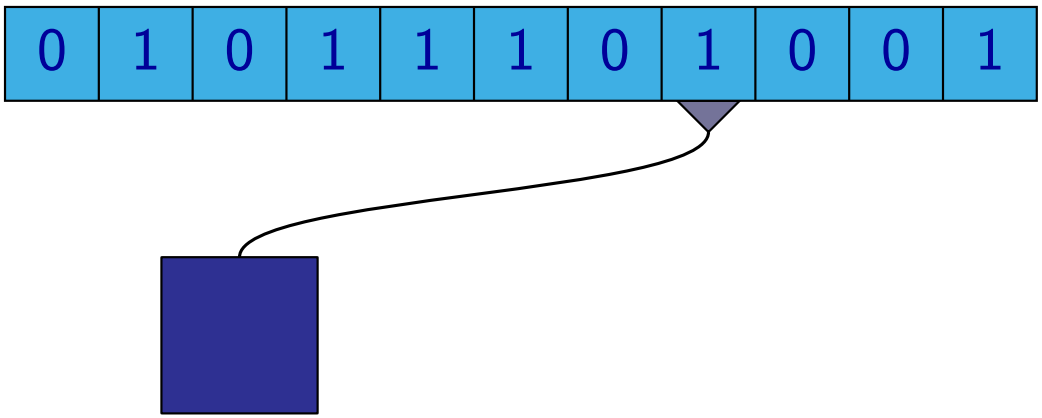


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

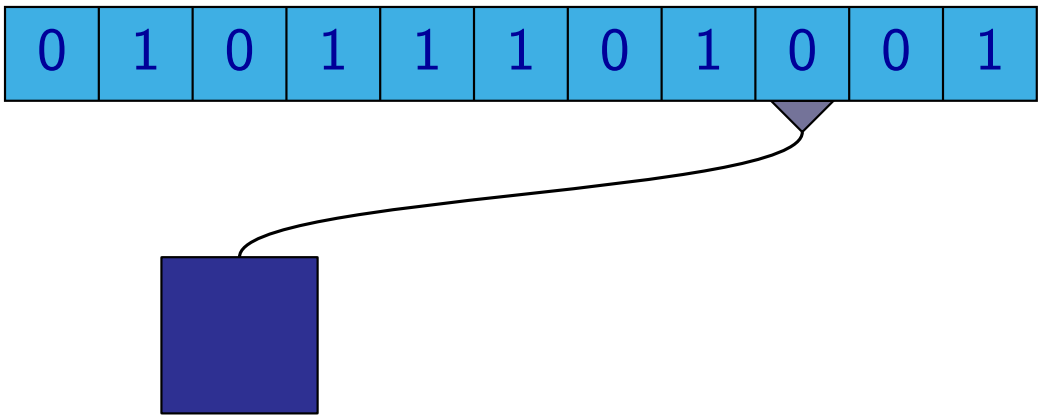


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

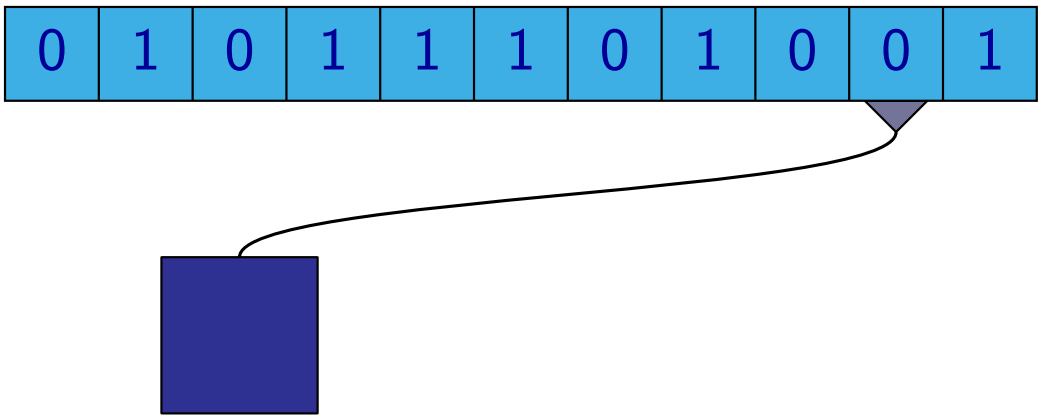


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

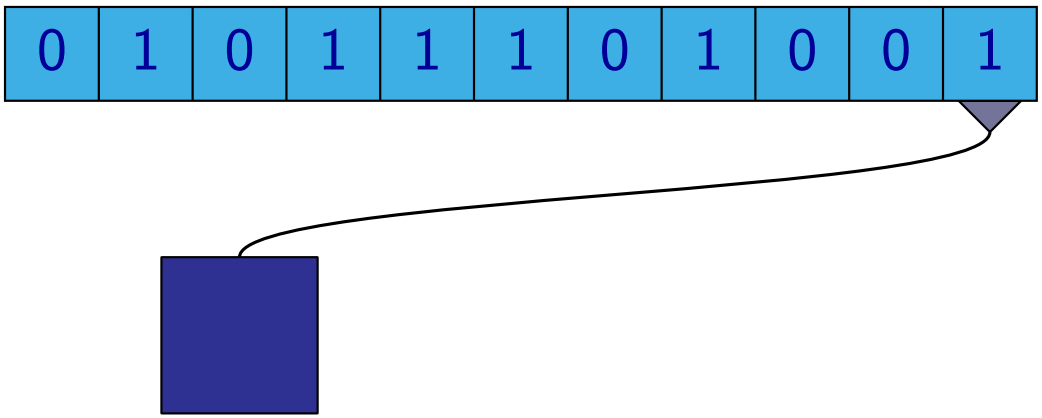


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

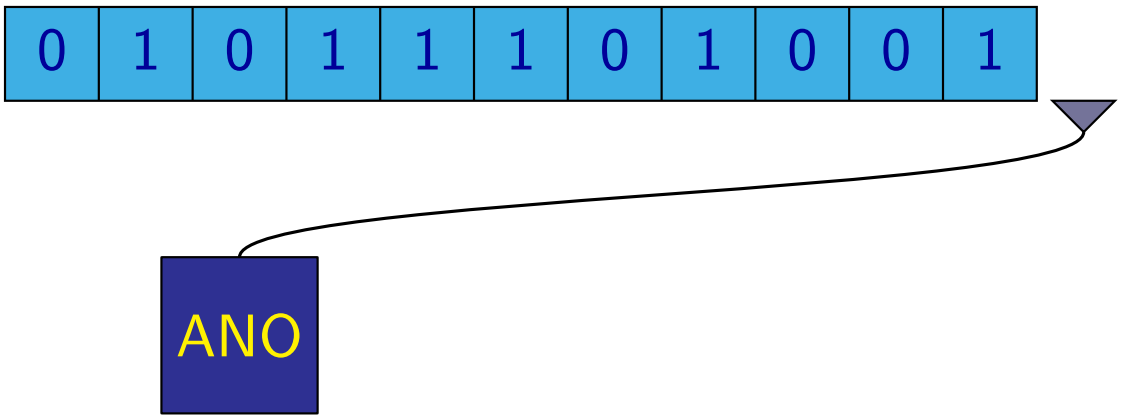


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

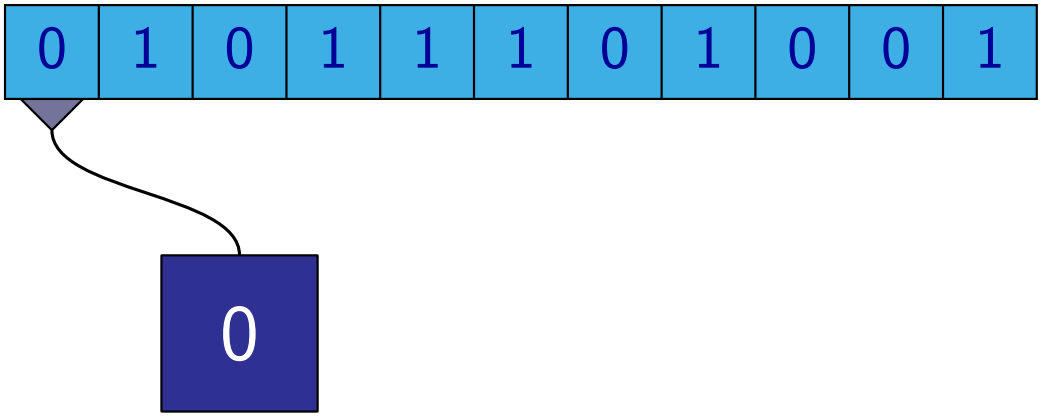
Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.



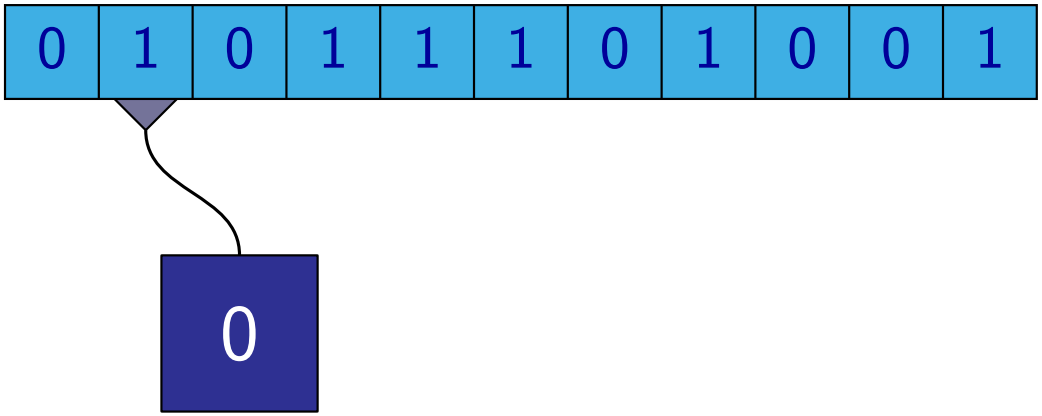
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



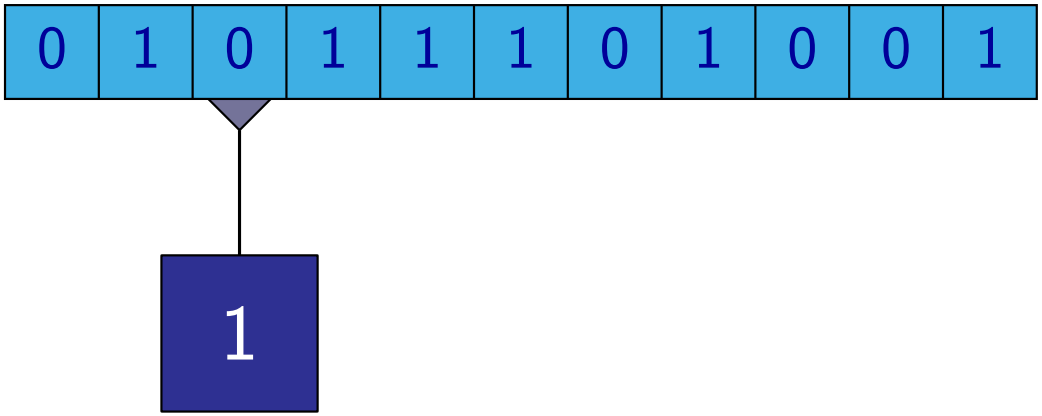
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



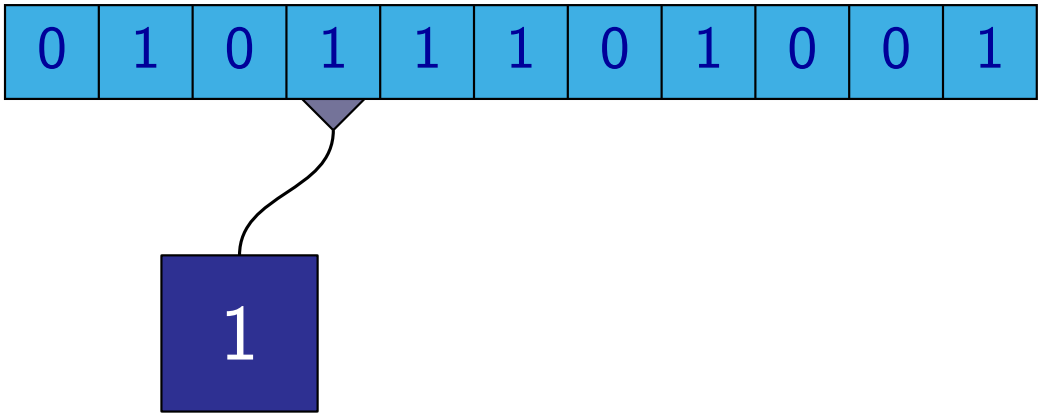
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



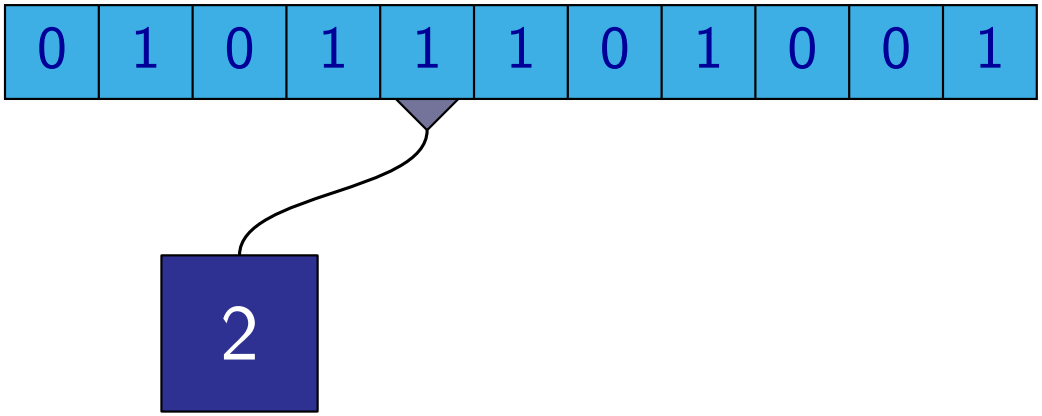
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



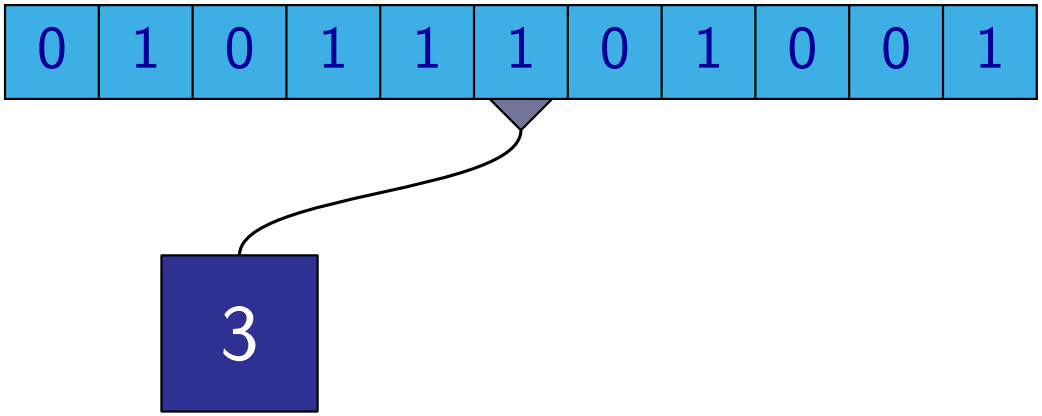
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



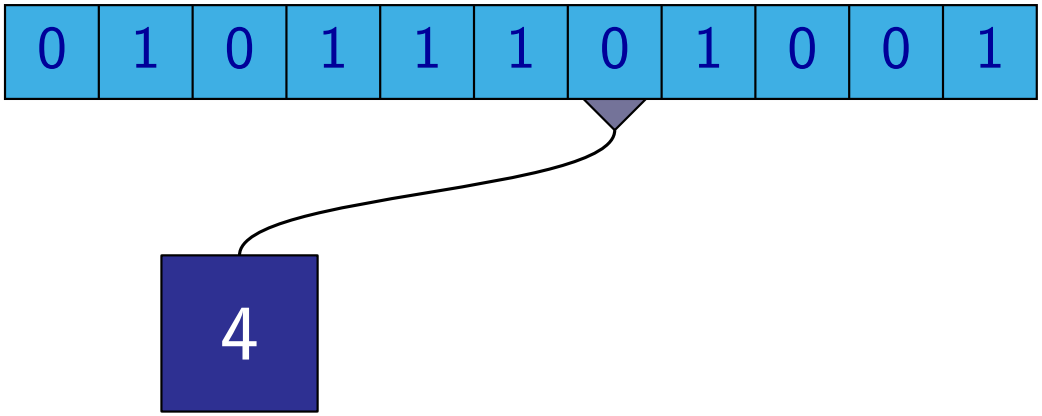
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



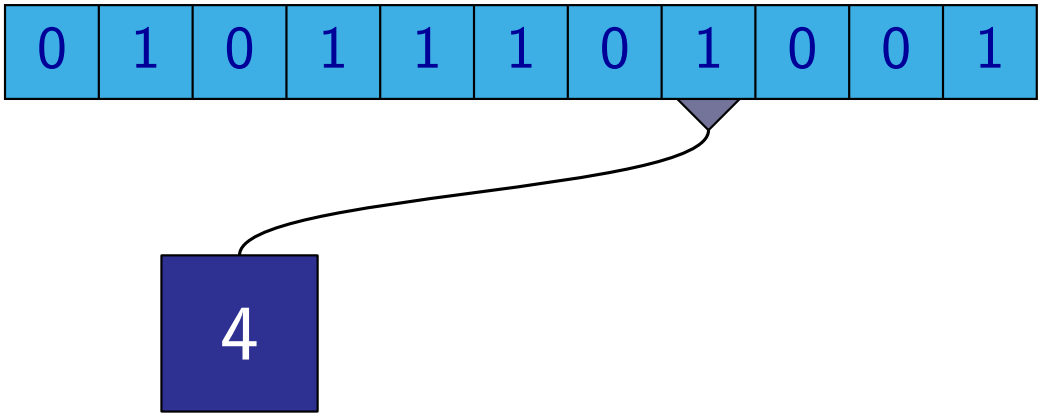
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



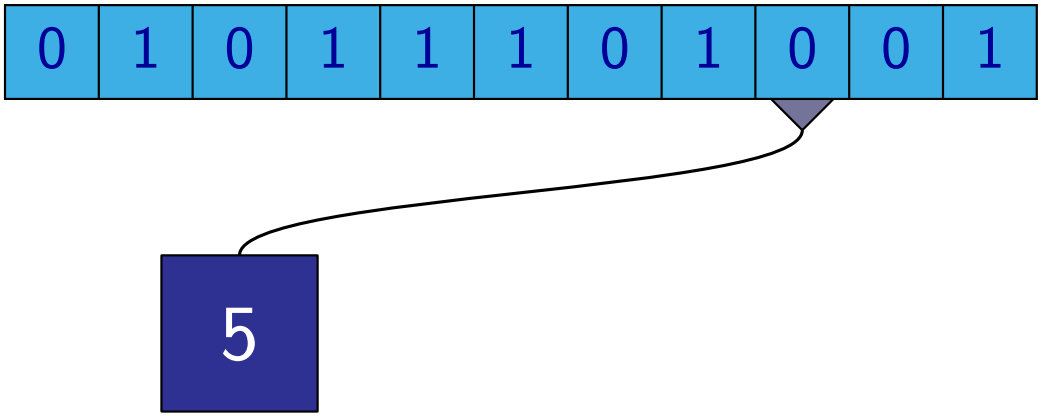
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



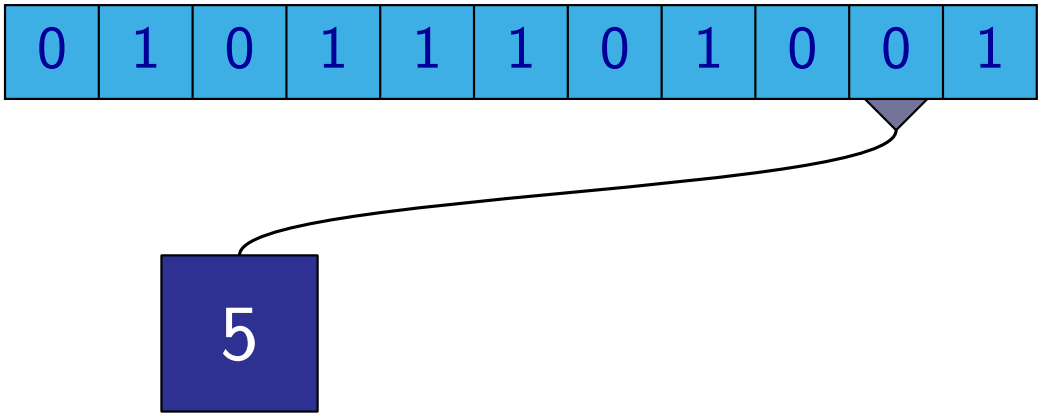
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



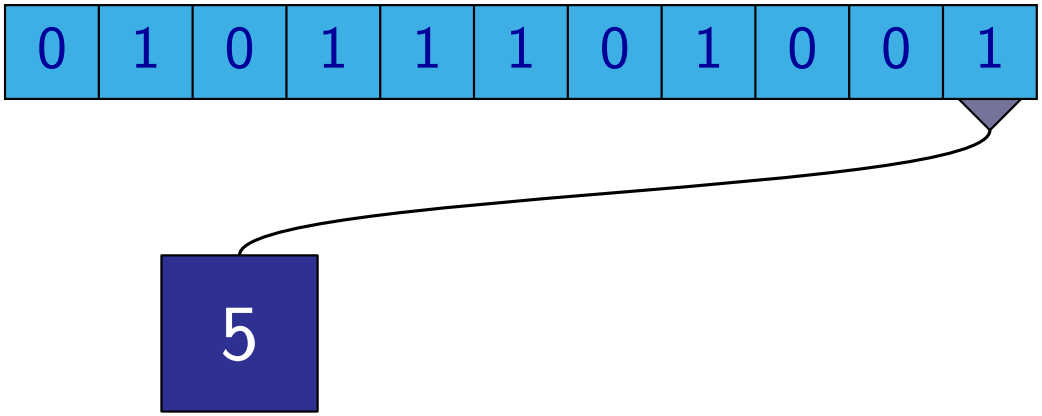
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



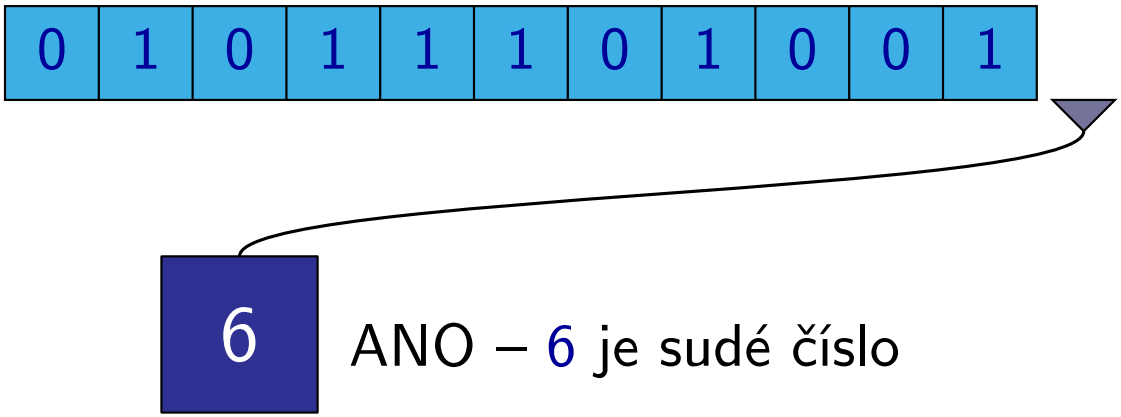
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



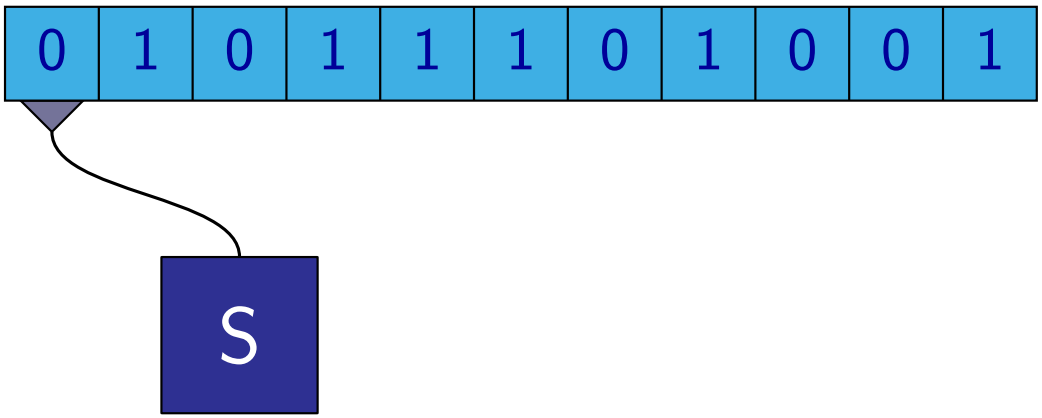
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



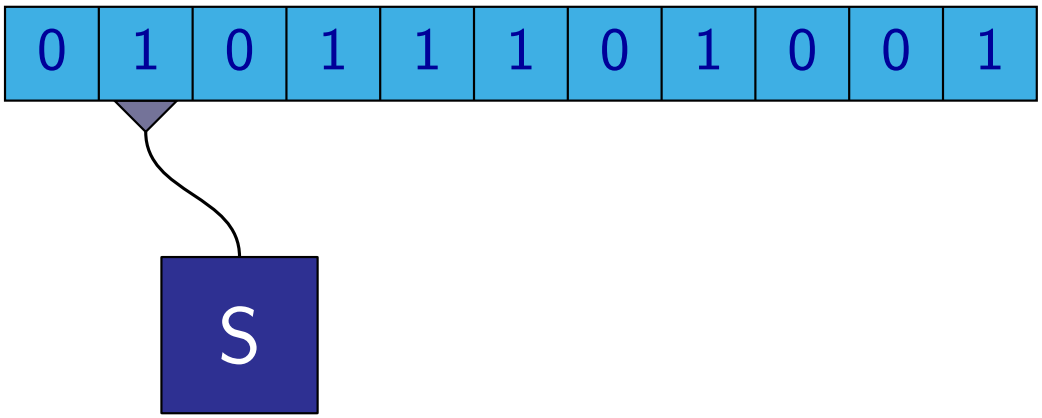
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



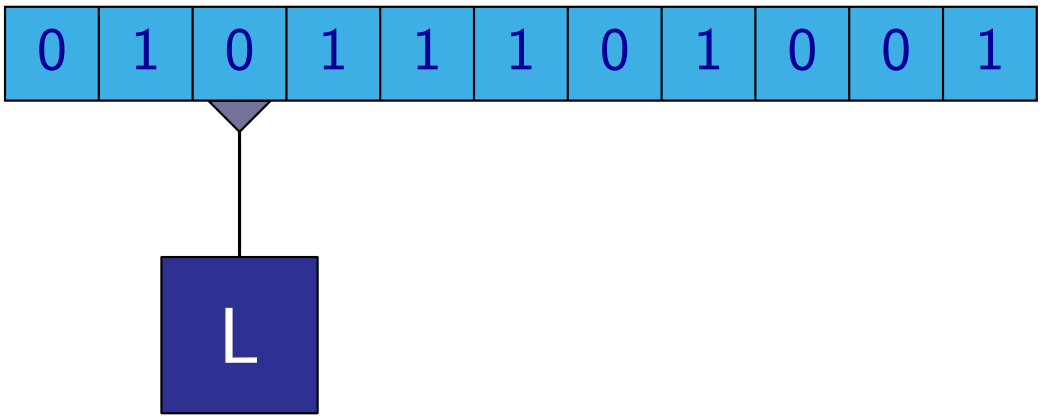
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



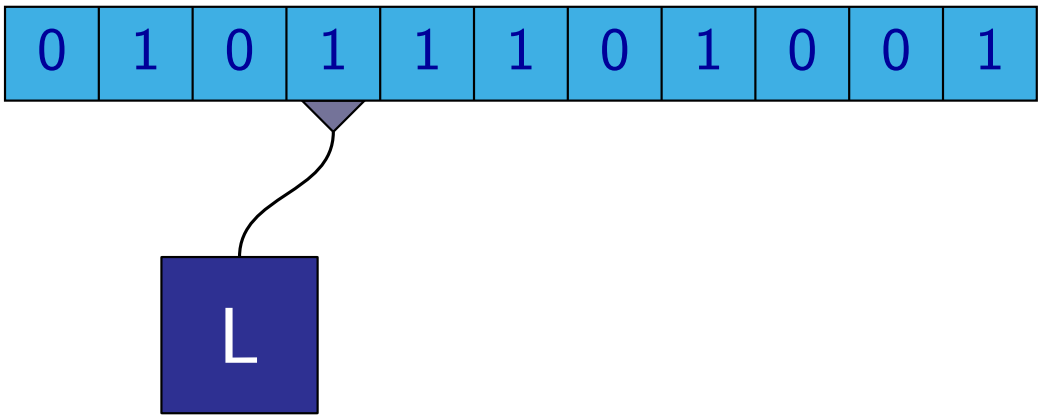
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



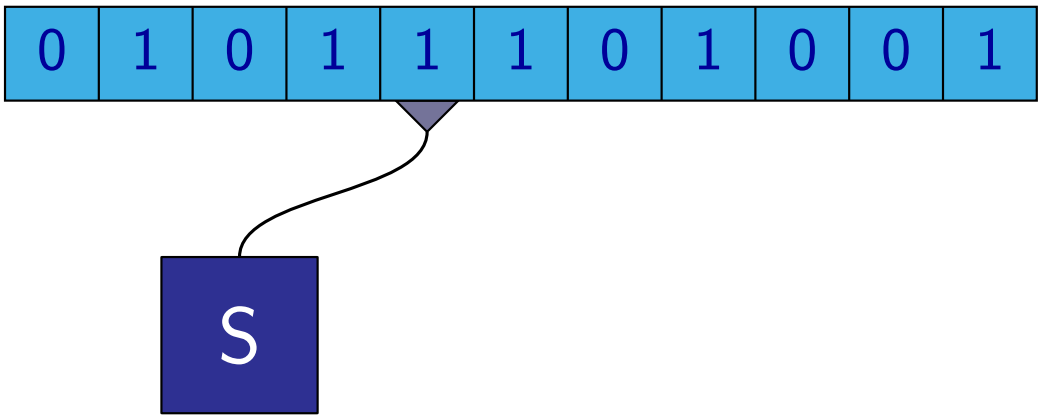
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



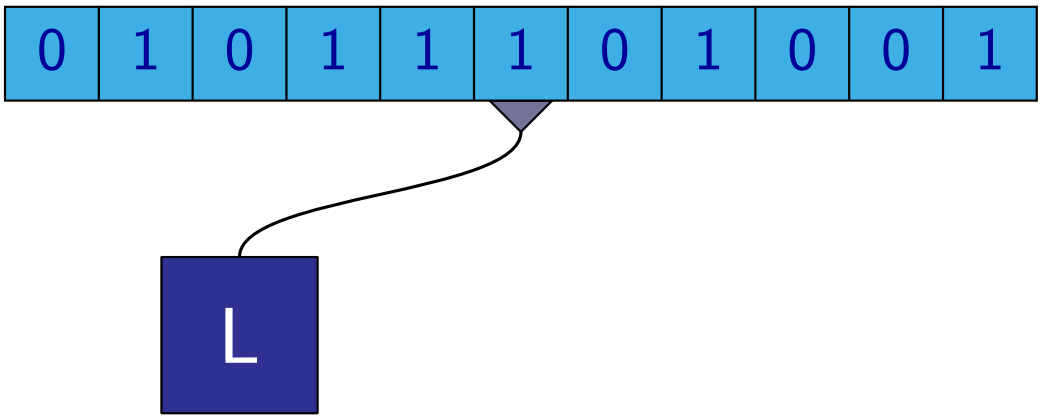
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



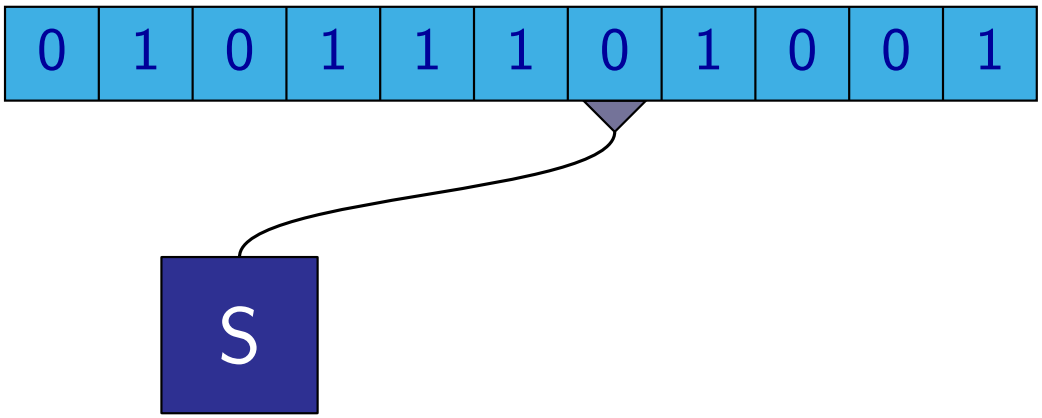
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



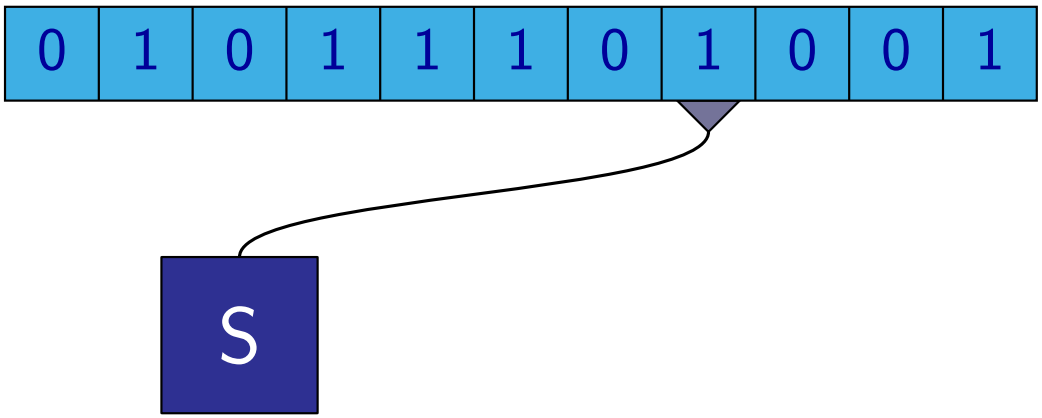
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



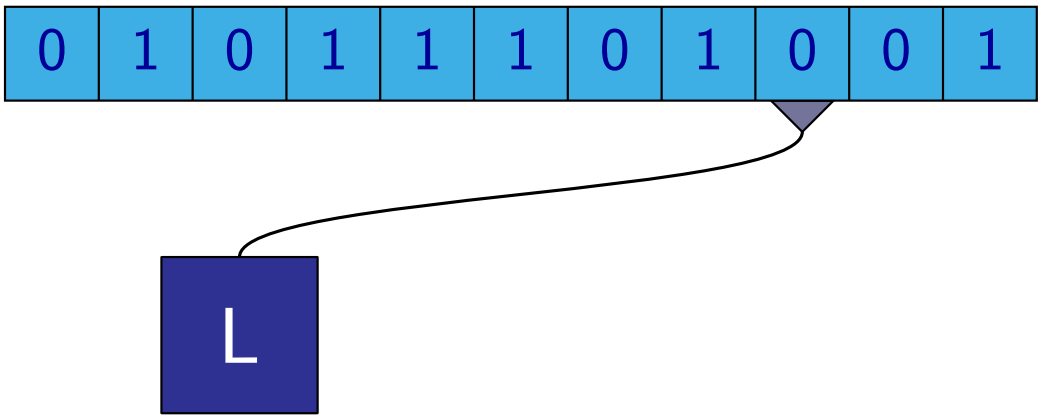
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



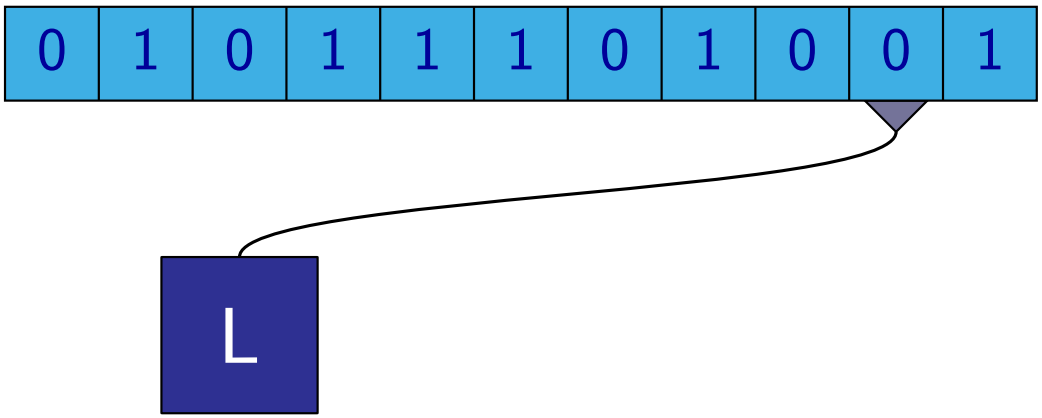
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



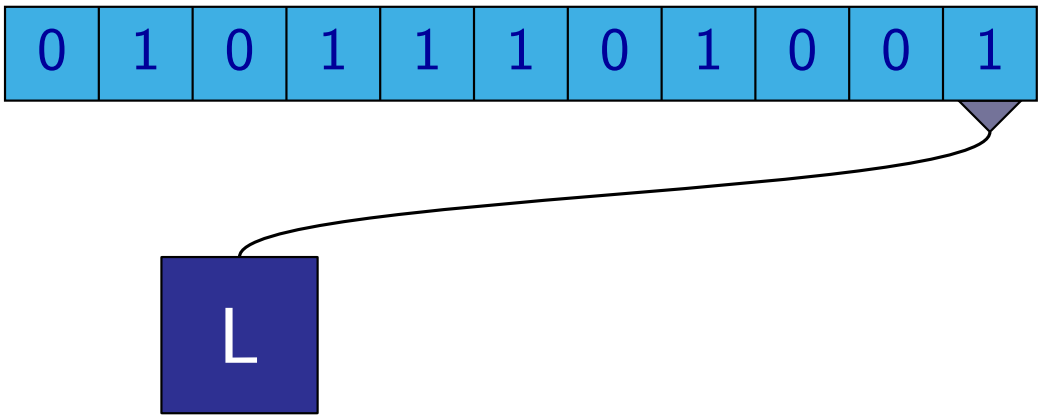
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



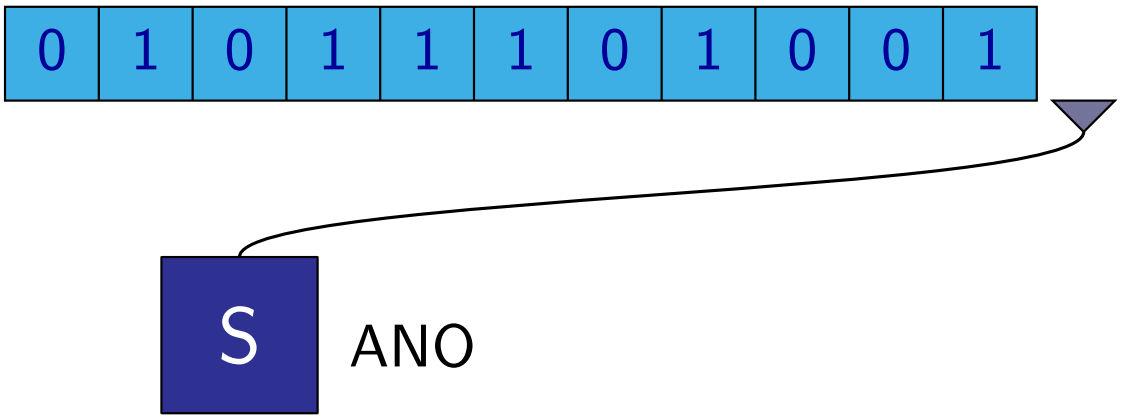
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



Rozpoznávání jazyka

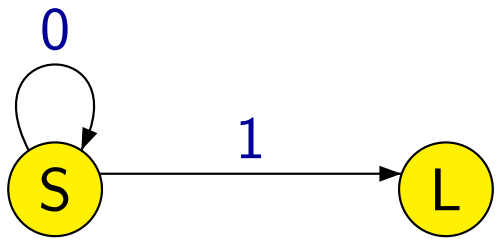
Chovaní tohoto zařízení můžeme popsat grafem:

Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:

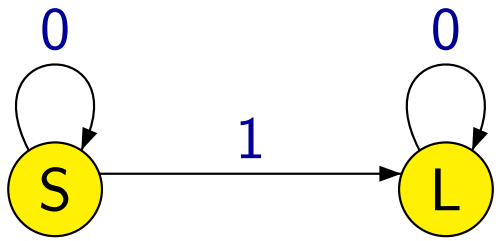
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



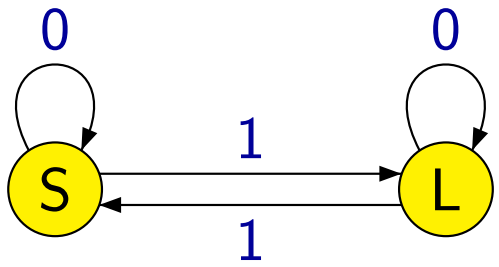
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



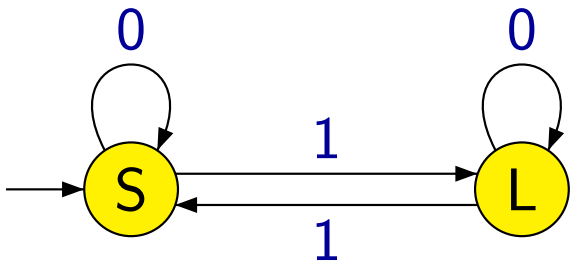
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



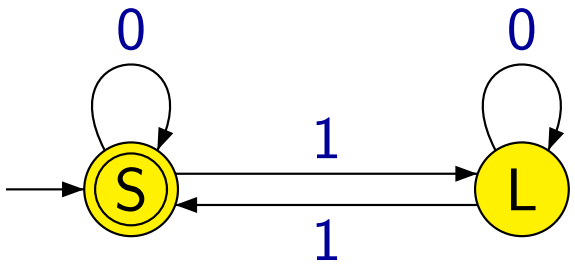
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



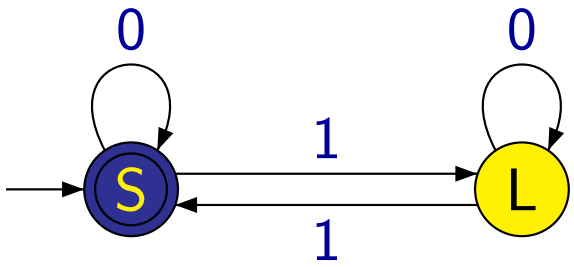
Rozpoznávání jazyka

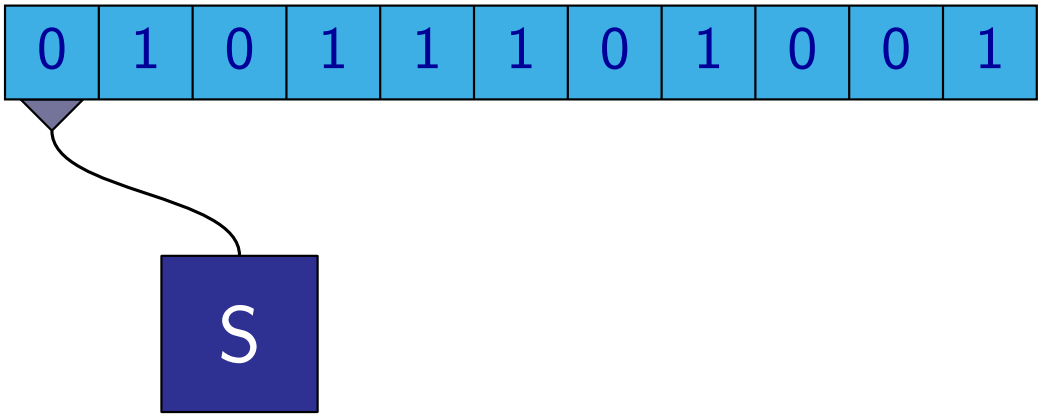
Chovaní tohoto zařízení můžeme popsat grafem:



Rozpoznávání jazyka

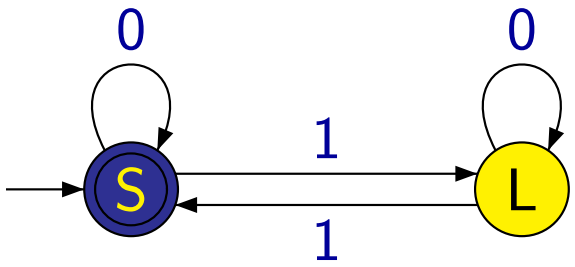
Chovaní tohoto zařízení můžeme popsat grafem:

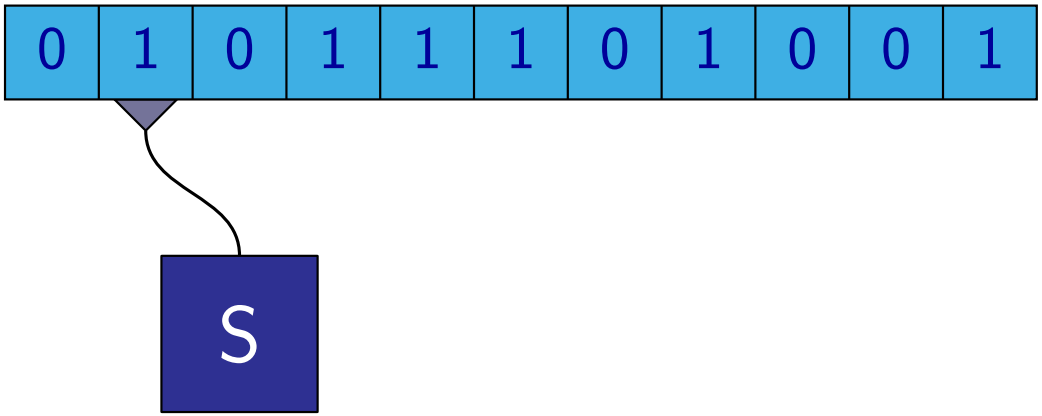




Rozpoznávání jazyka

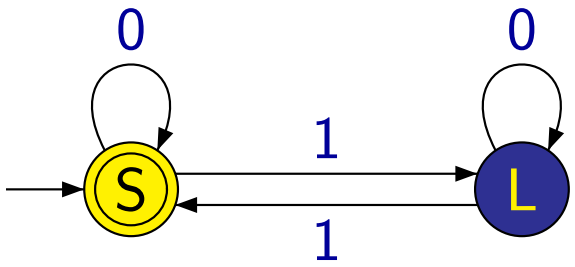
Chovaní tohoto zařízení můžeme popsat grafem:

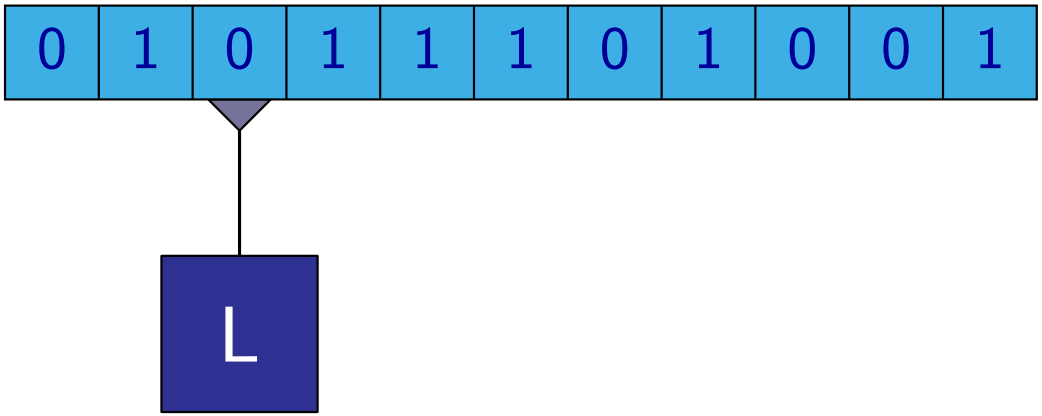




Rozpoznávání jazyka

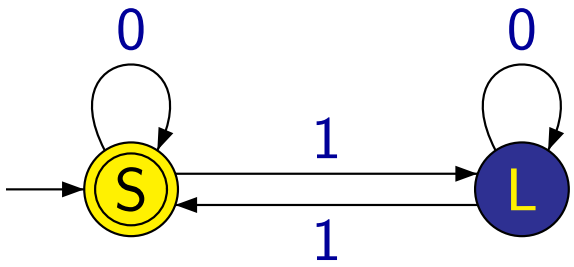
Chovaní tohoto zařízení můžeme popsat grafem:

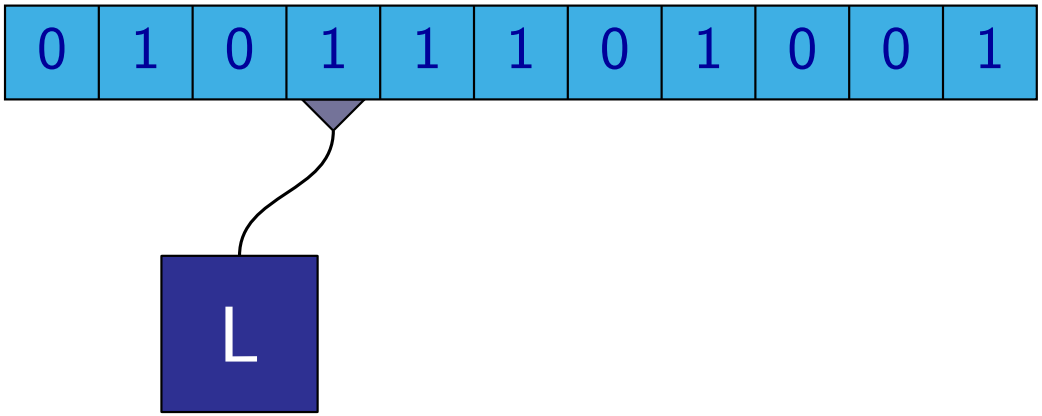




Rozpoznávání jazyka

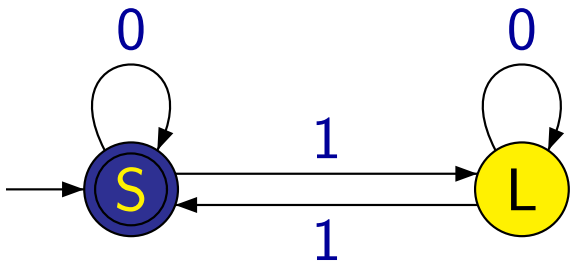
Chovaní tohoto zařízení můžeme popsat grafem:

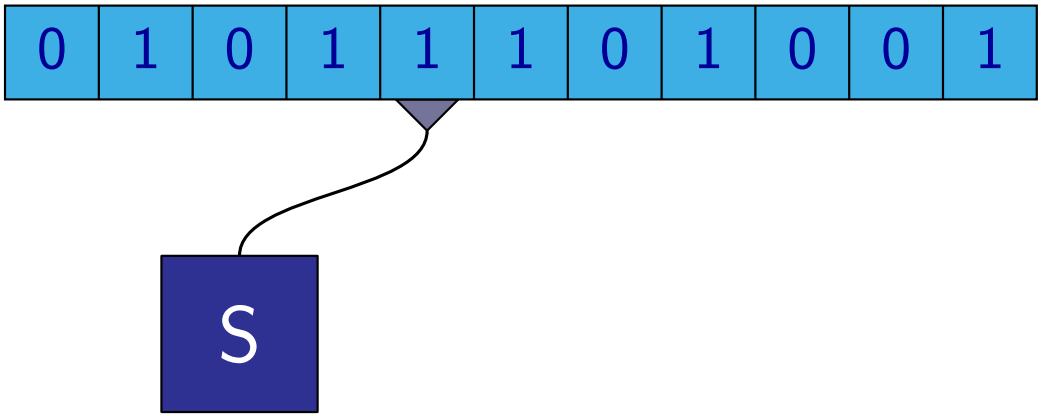




Rozpoznávání jazyka

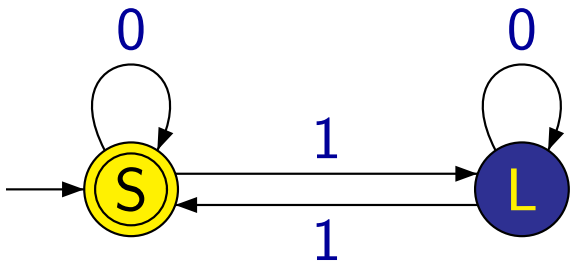
Chovaní tohoto zařízení můžeme popsat grafem:

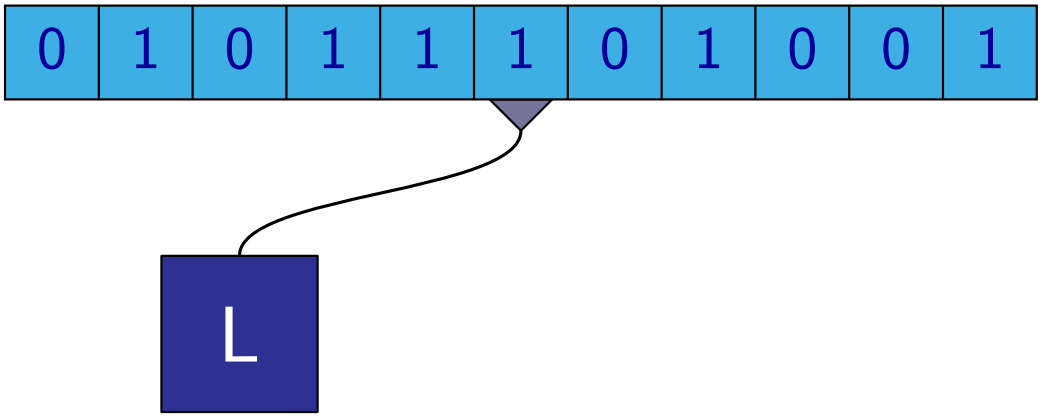




Rozpoznávání jazyka

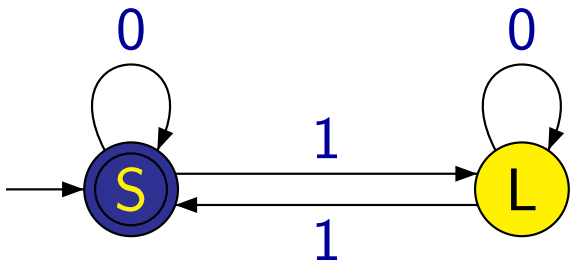
Chovaní tohoto zařízení můžeme popsat grafem:

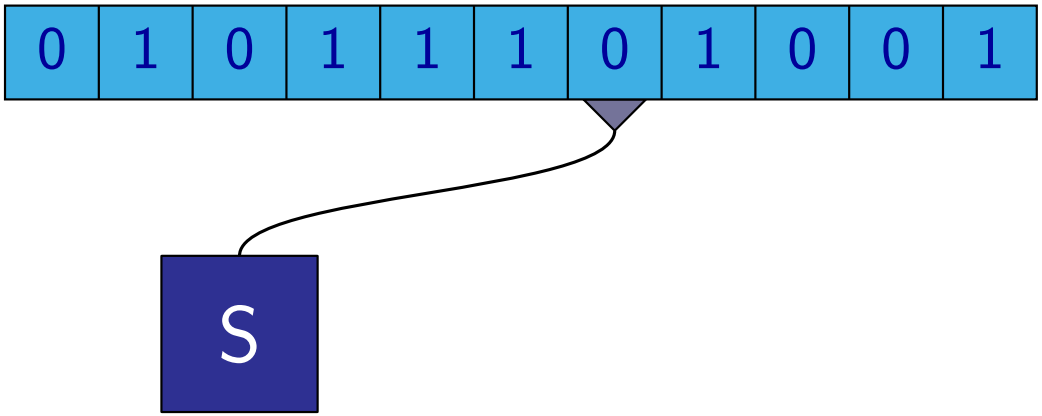




Rozpoznávání jazyka

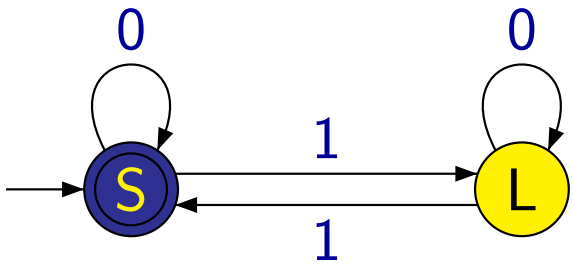
Chovaní tohoto zařízení můžeme popsat grafem:

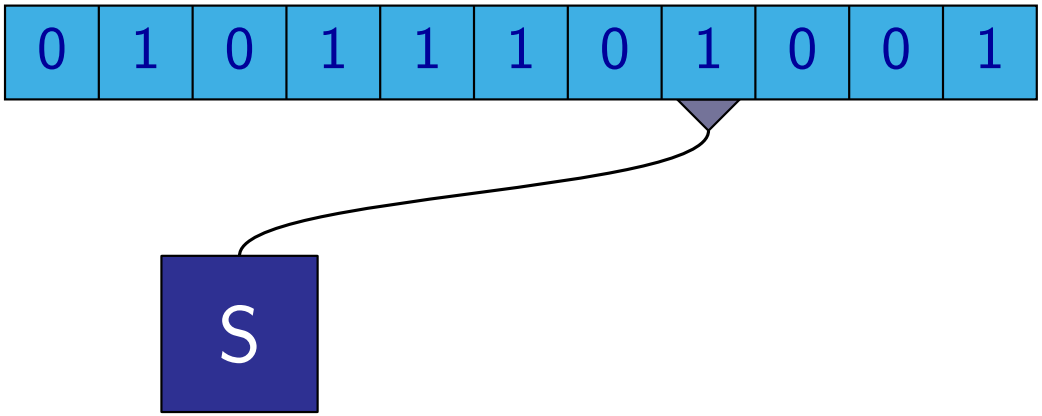




Rozpoznávání jazyka

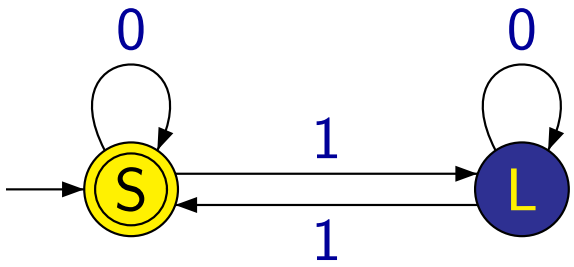
Chovaní tohoto zařízení můžeme popsat grafem:

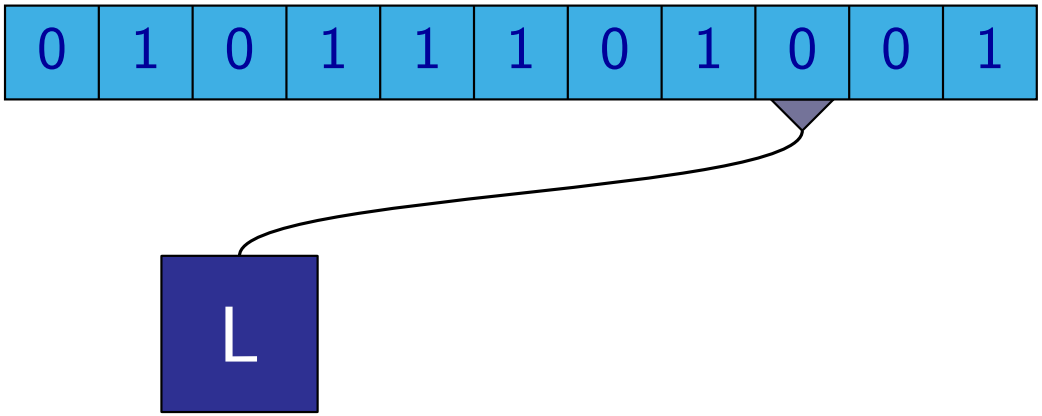




Rozpoznávání jazyka

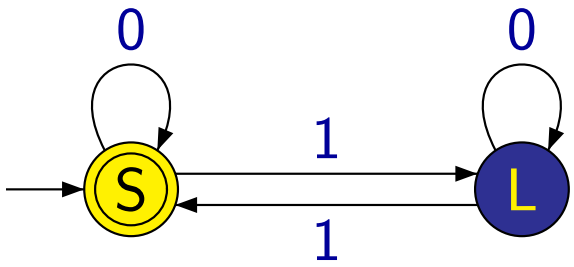
Chovaní tohoto zařízení můžeme popsat grafem:

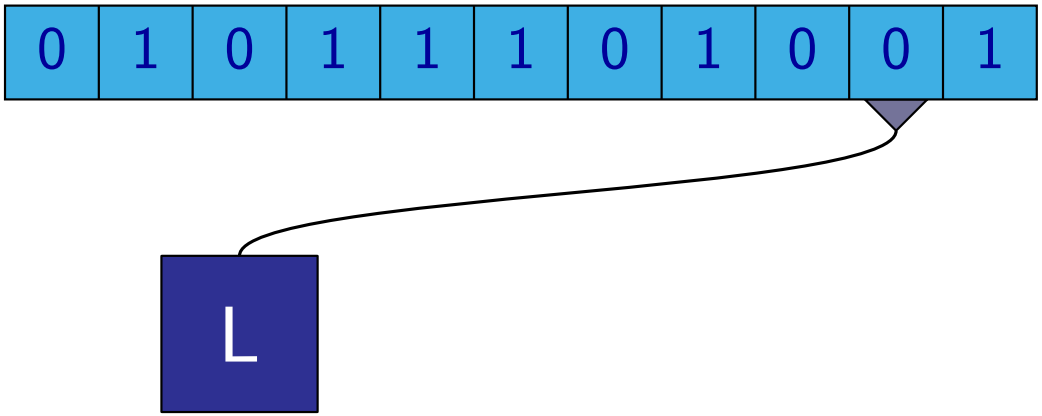




Rozpoznávání jazyka

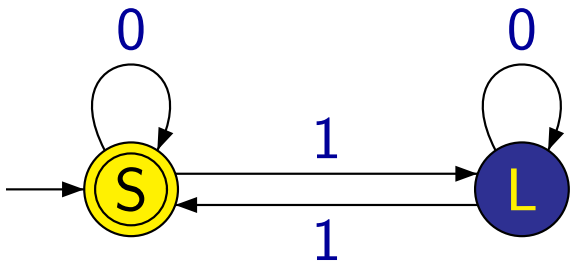
Chovaní tohoto zařízení můžeme popsat grafem:

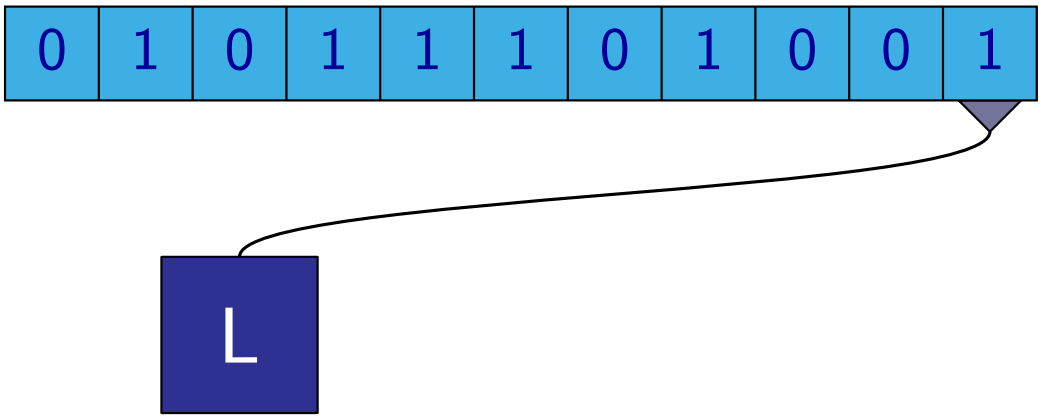




Rozpoznávání jazyka

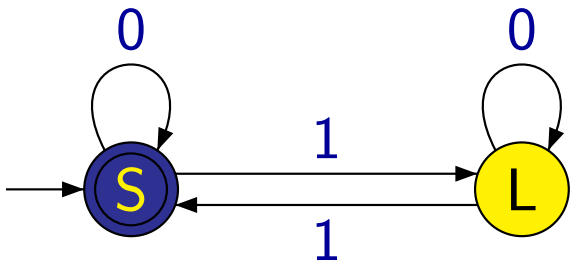
Chovaní tohoto zařízení můžeme popsat grafem:

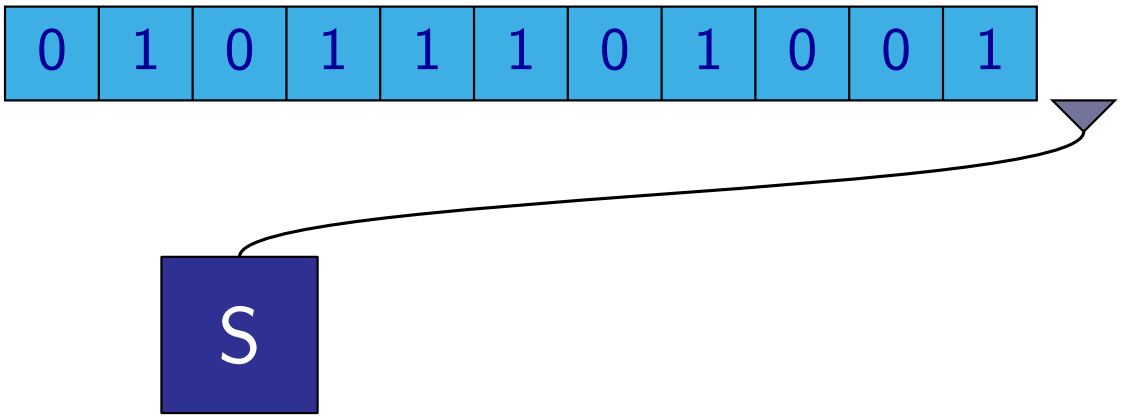




Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:





Vyhledávání v textu

Problém

Vstup: t – dlouhý text, s – hledaný řetězec

Výstup: ANO – pokud se řetězec s nachází v textu t,
NE – pokud se tam nenachází

Například chceme zjistit, zda se v textu `aaababaabab` nachází řetězec `abaa`.

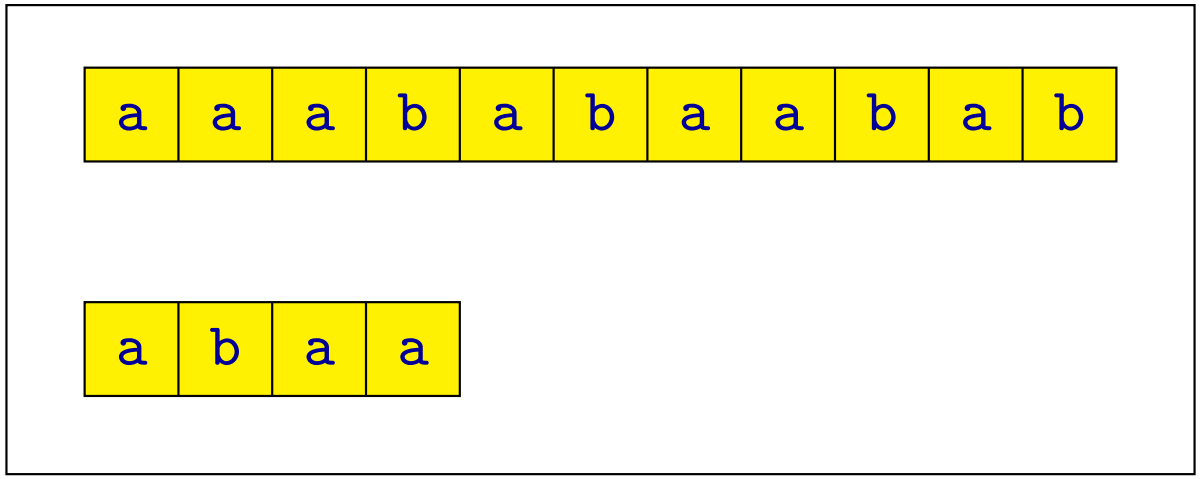
Vyhledávání v textu

Jednoduchý algoritmus, který nás asi napadne v první chvíli:

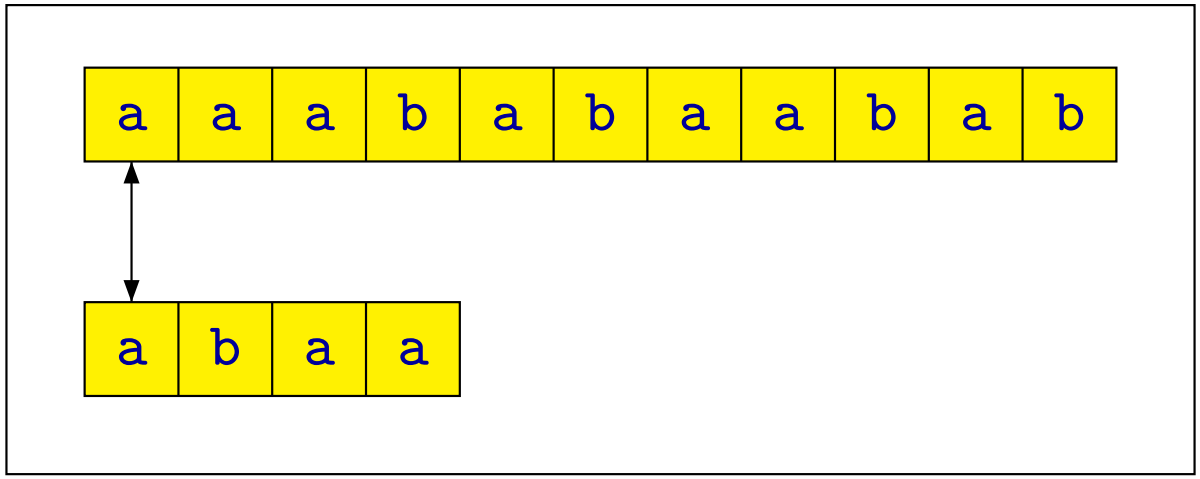
NAIVNÍ-VYHLEDÁVÁNÍ(t, s)

```
1  n ← length(t)
2  m ← length(s)
3  for i ← 0 to n − m
4        do nalezen ← TRUE
5           for j ← 0 to m − 1
6                 do if s[j] ≠ t[i + j]
7                       then nalezen ← FALSE; break
8           if nalezen
9              then return TRUE
10 return FALSE
```

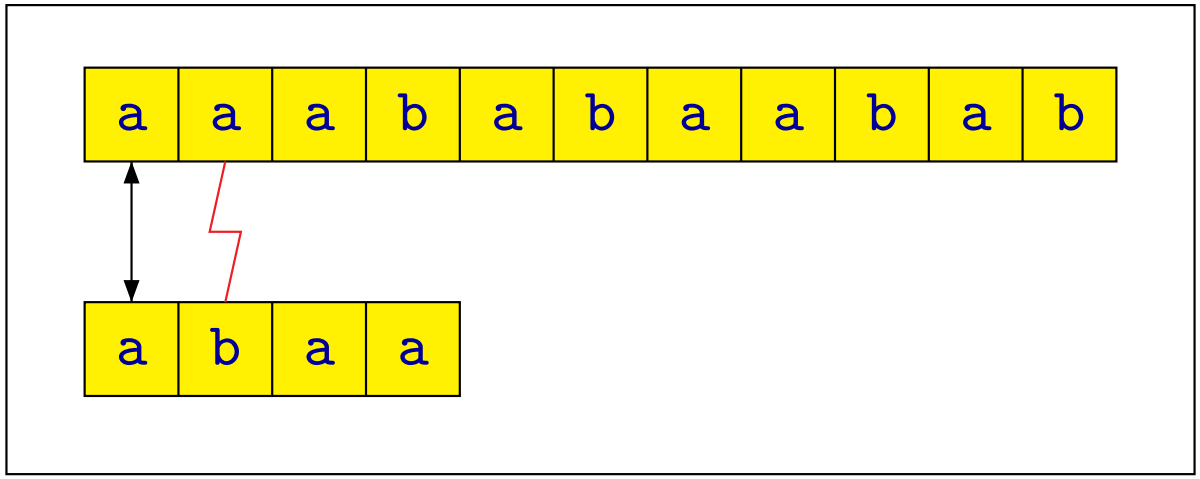
Vyhledávání v textu



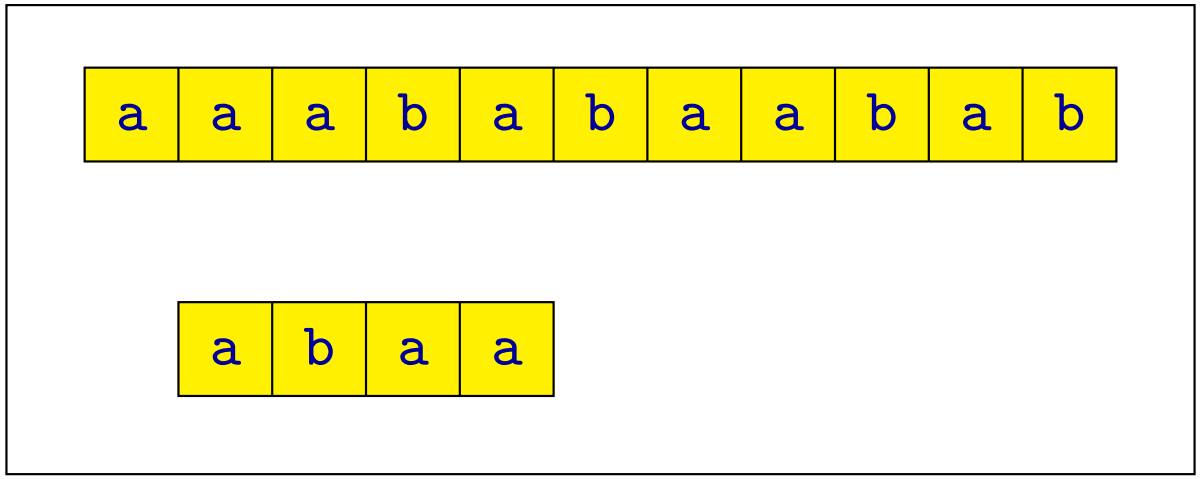
Vyhledávání v textu



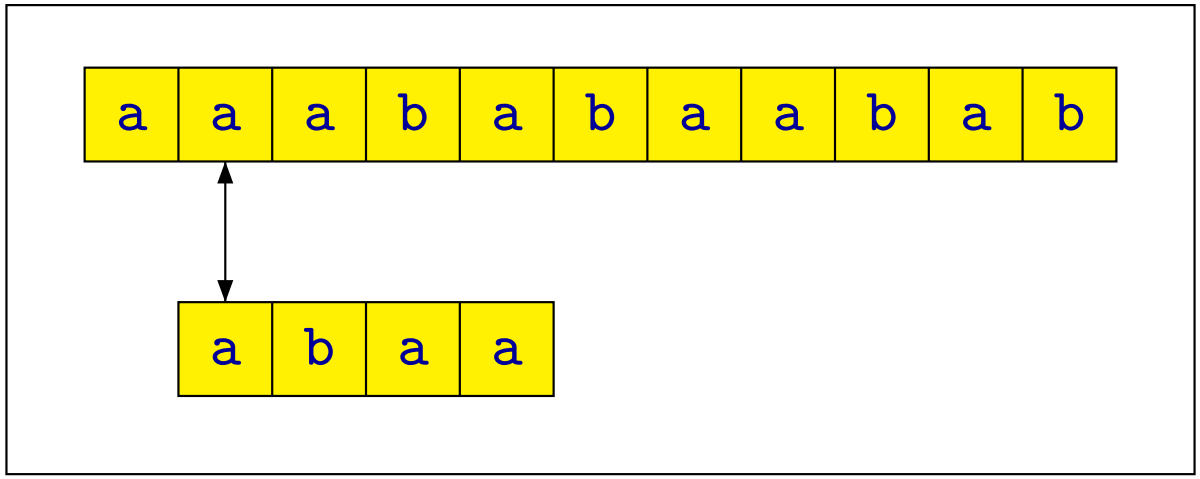
Vyhledávání v textu



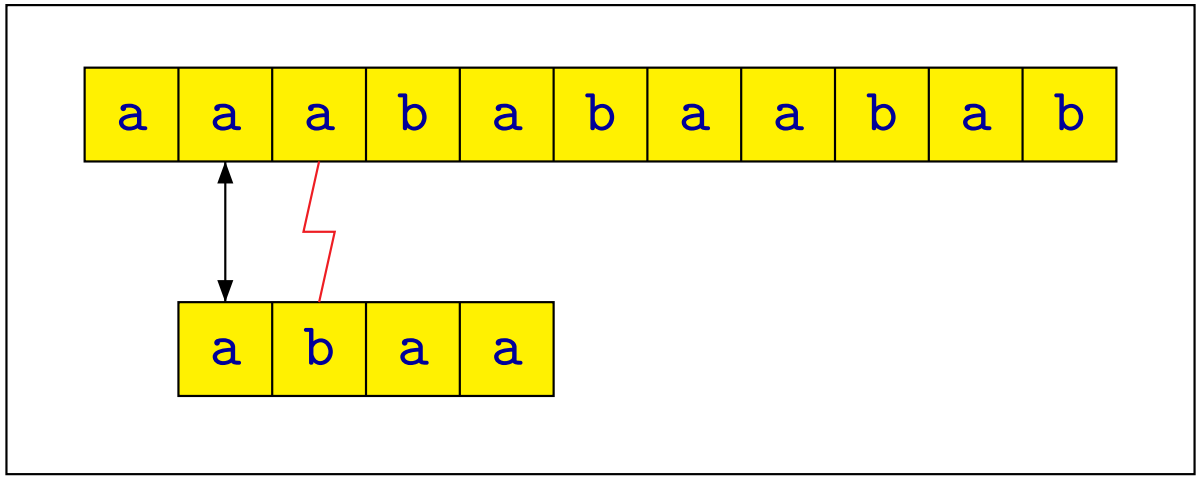
Vyhledávání v textu



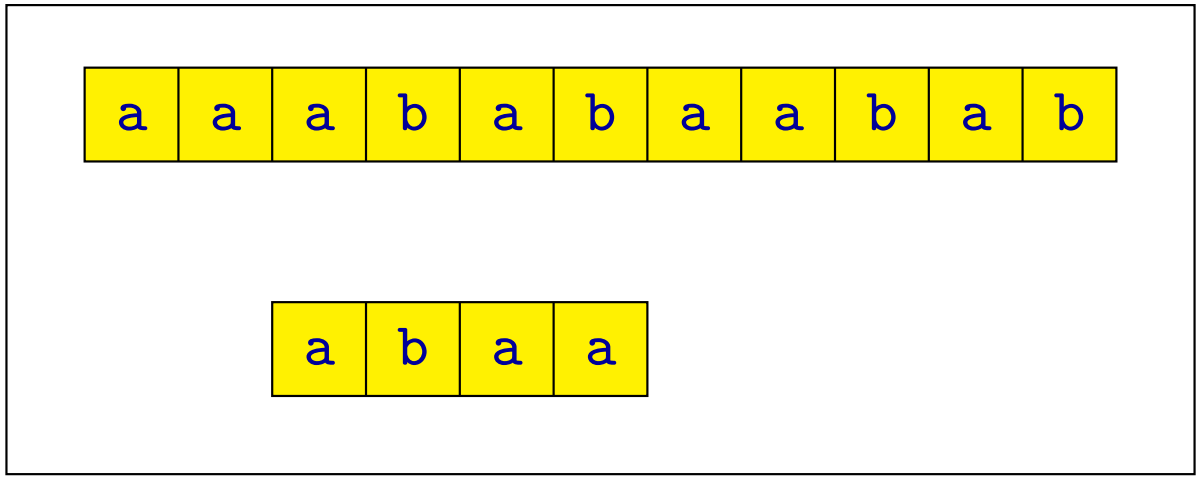
Vyhledávání v textu



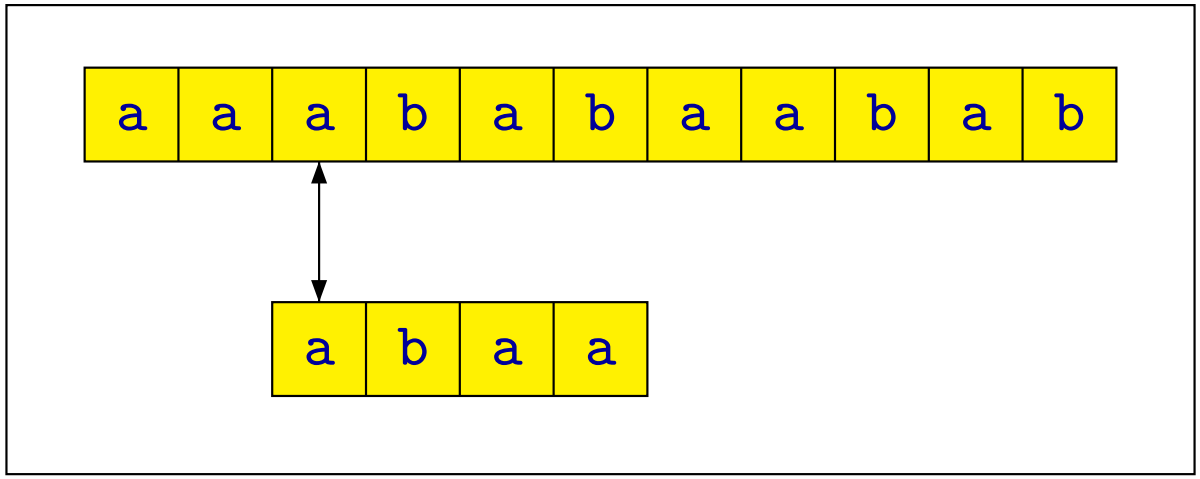
Vyhledávání v textu



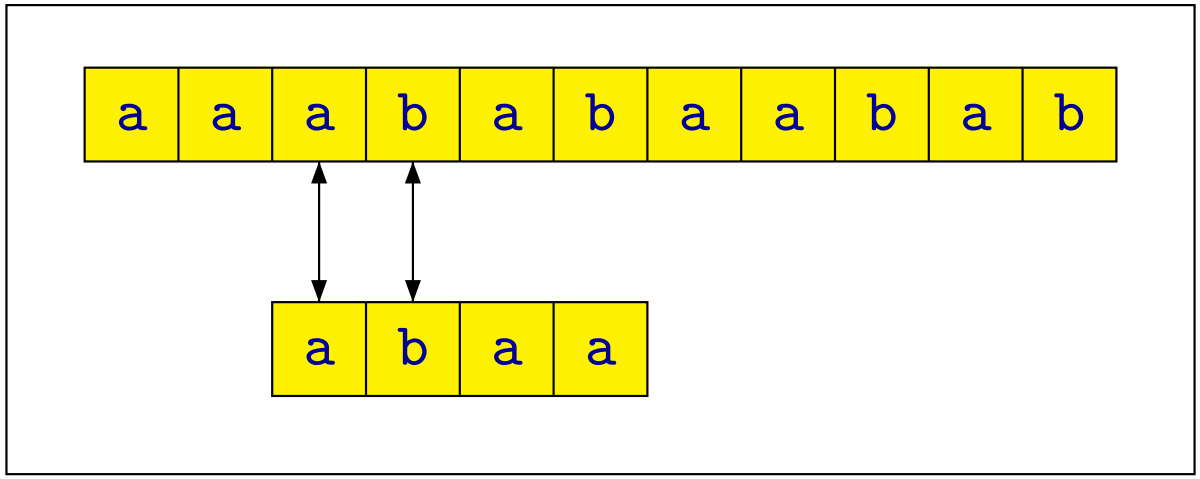
Vyhledávání v textu



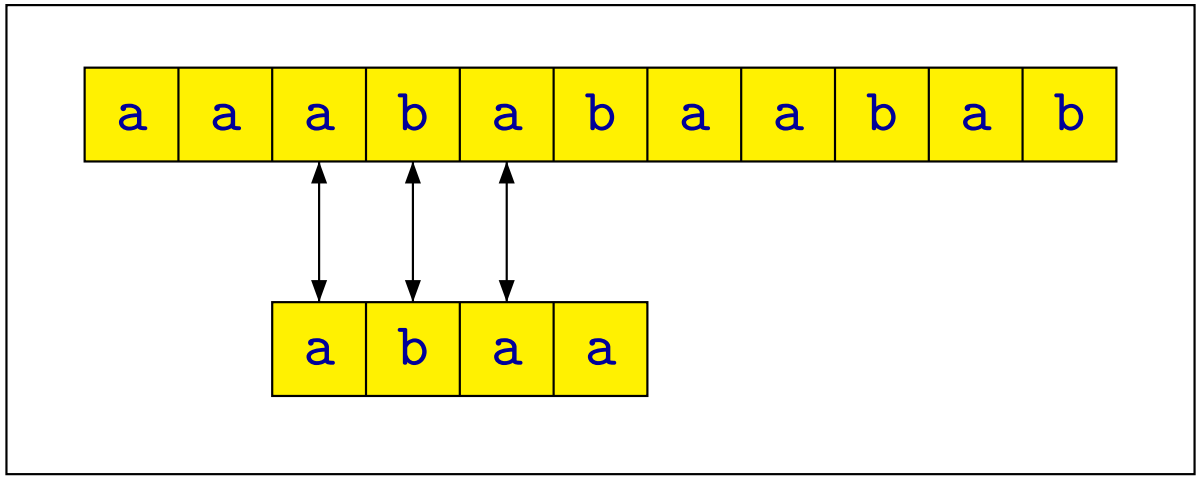
Vyhledávání v textu



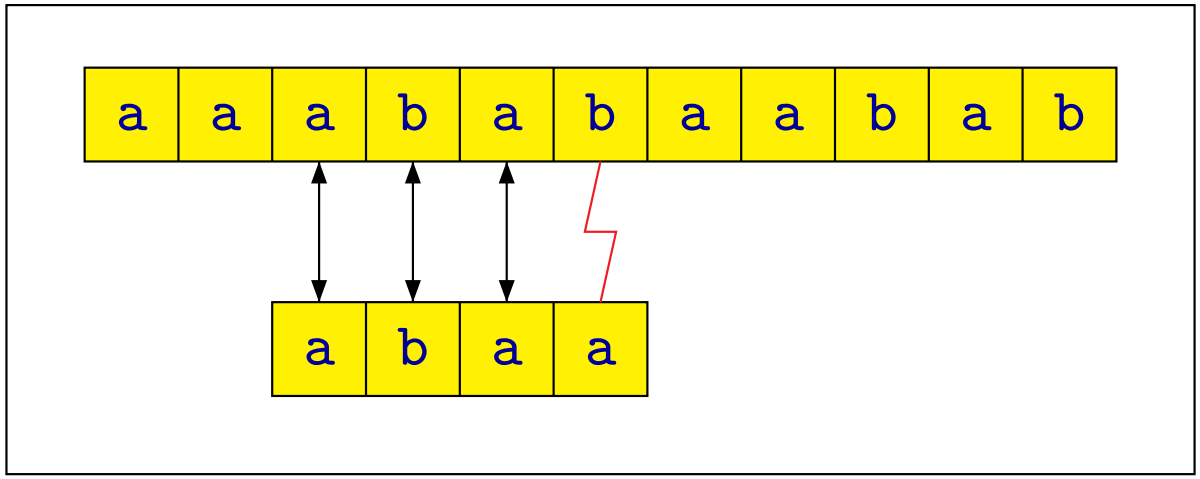
Vyhledávání v textu



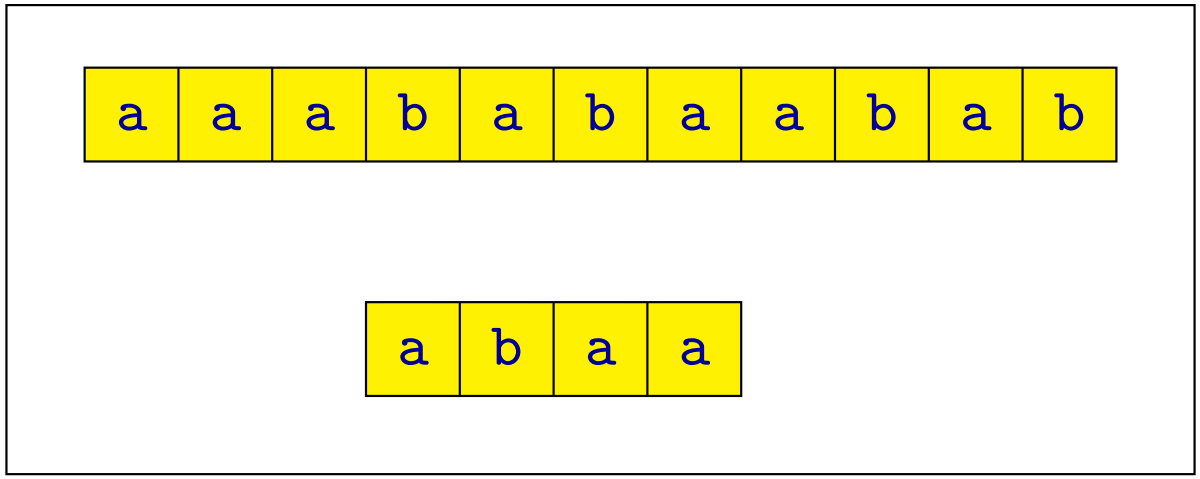
Vyhledávání v textu



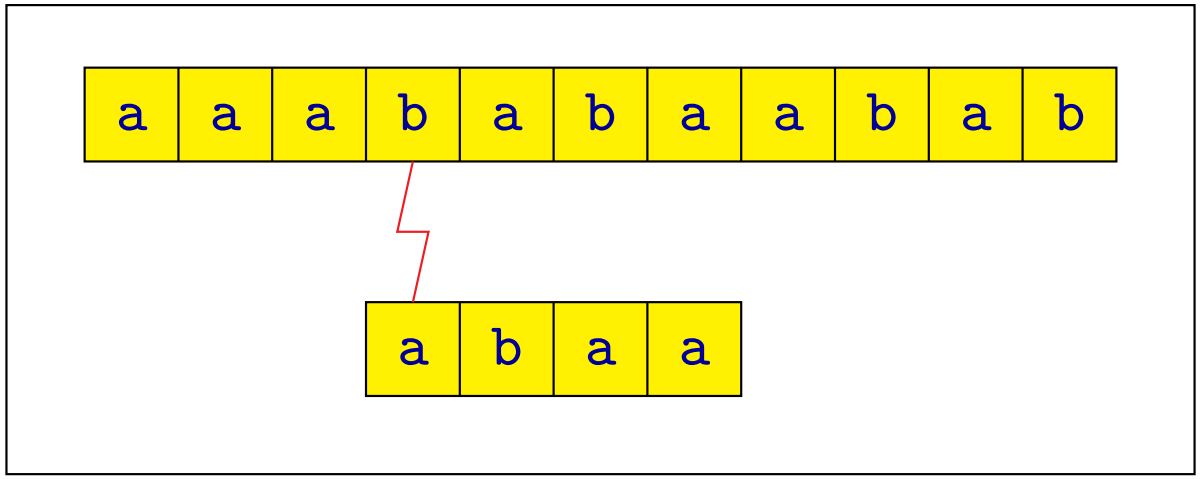
Vyhledávání v textu



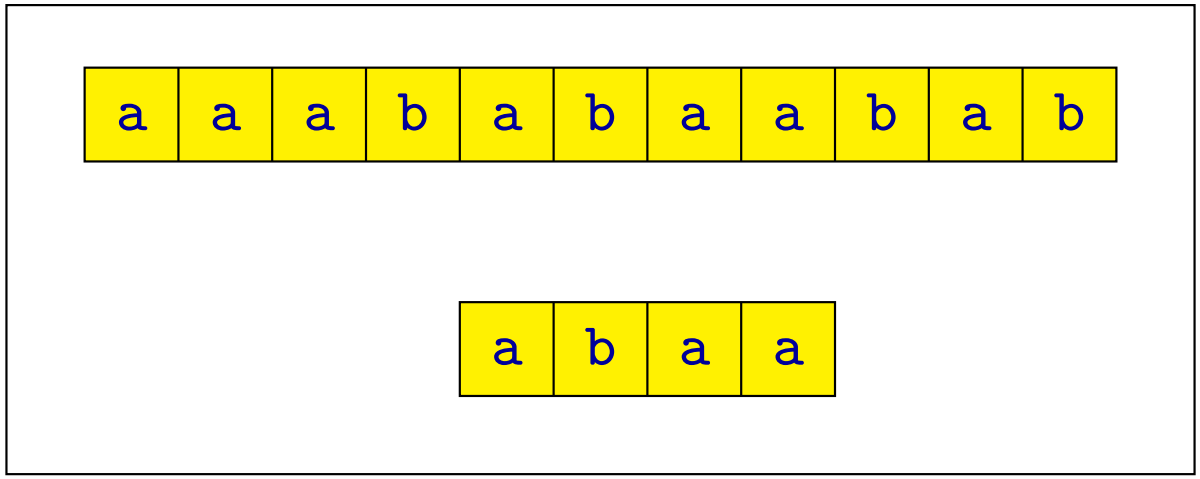
Vyhledávání v textu



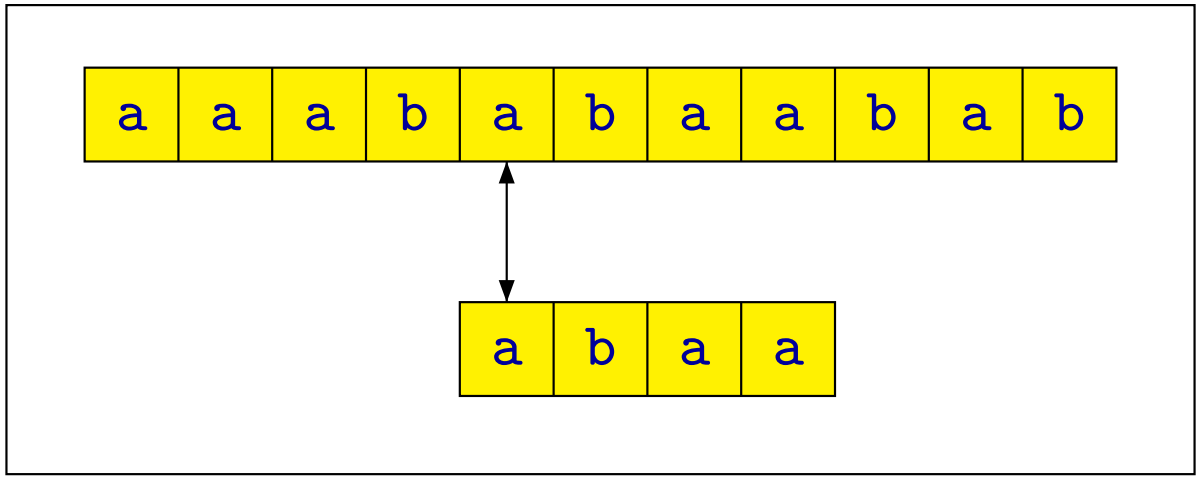
Vyhledávání v textu



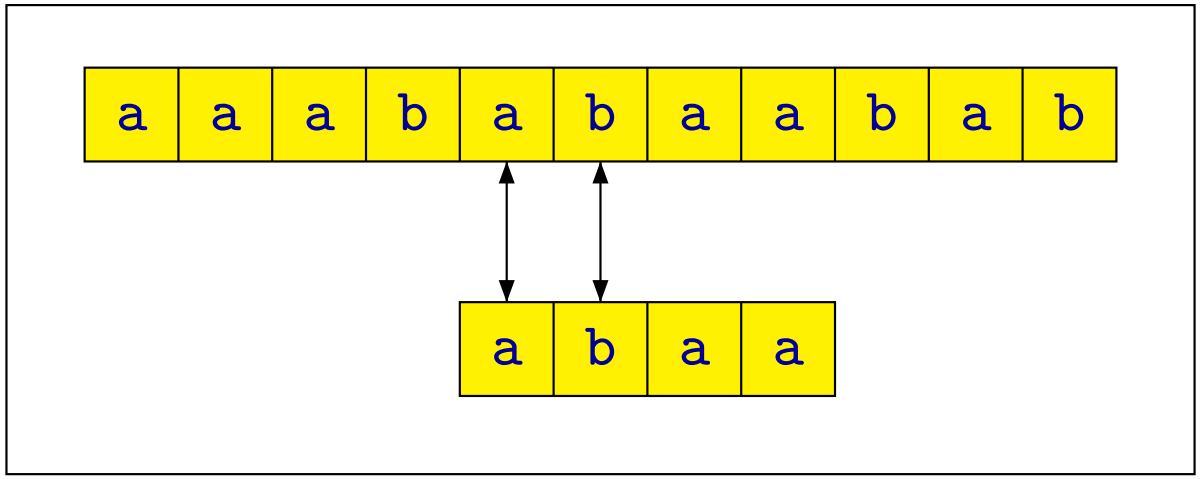
Vyhledávání v textu



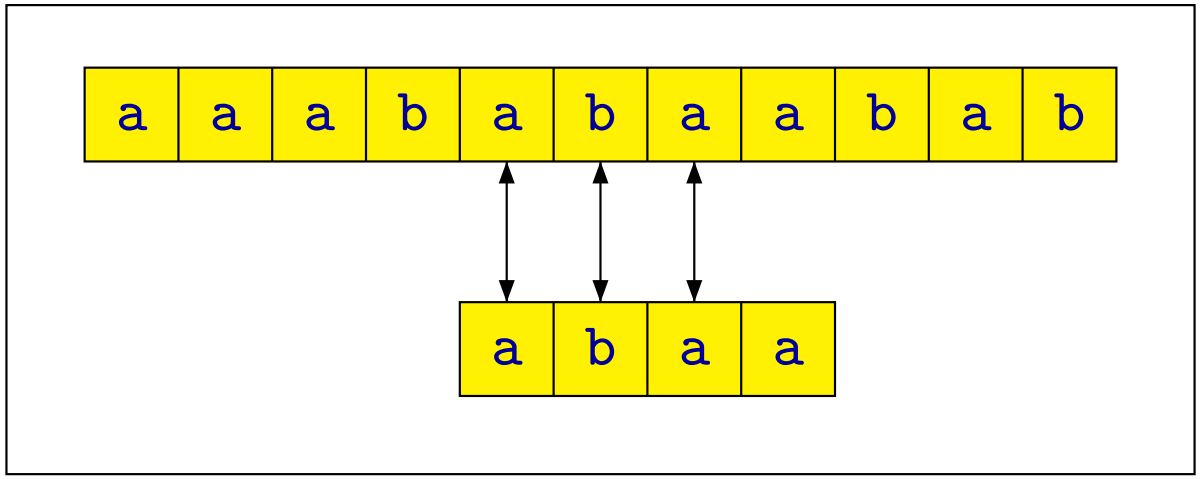
Vyhledávání v textu



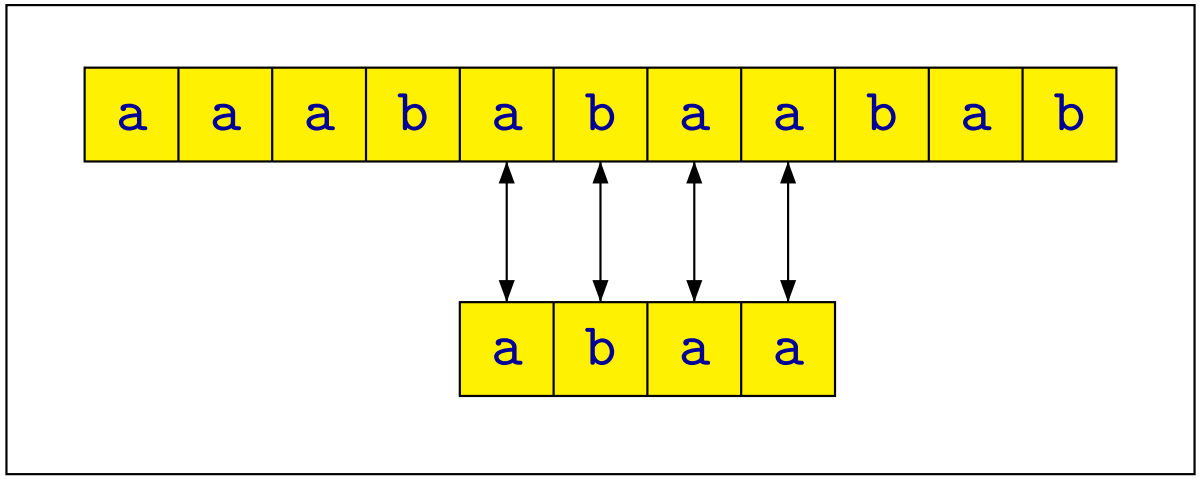
Vyhledávání v textu



Vyhledávání v textu



Vyhledávání v textu



Vyhledávání v textu

Pozorování: Nijak nevyužíváme informaci o části slova s, která souhlasí, a v dalším kroku začínáme zase od začátku slova s.

Nápad: Slovo t číst znak po znaku a pamatovat si jaká část slova s je shodná s koncem dosud načteného textu. Upravovat tento údaj vždy jen na základě dalšího jednoho načteného znaku:

LEPŠÍ-VYHLEDÁVÁNÍ(t, s)

```
1  n ← length(t)
2  m ← length(s)
3  q ← 0
4  for i ← 0 to n − 1
5        do q ← δ(q, t[i])
6           if q = m
7              then return TRUE
8  return FALSE
```

Prefixy, sufixy a podslova

V této souvislosti se nám hodí tři následující pojmy:

Definice

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

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

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

Příklad:

- Prefixy slova `abaab` jsou ε, `a`, `ab`, `aba`, `abaa`, `abaab`.
- Sufixy slova `abaab` jsou ε, `b`, `ab`, `aab`, `baab`, `abaab`.
- Podslova slova `abaab` jsou ε, `a`, `b`, `ab`, `ba`, `ab`, `aba`, `baa`, `abb`, `abaa`, `baab`, `abaab`.

Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

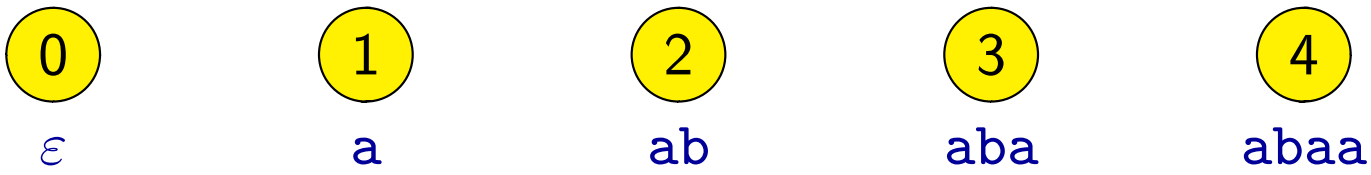
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:



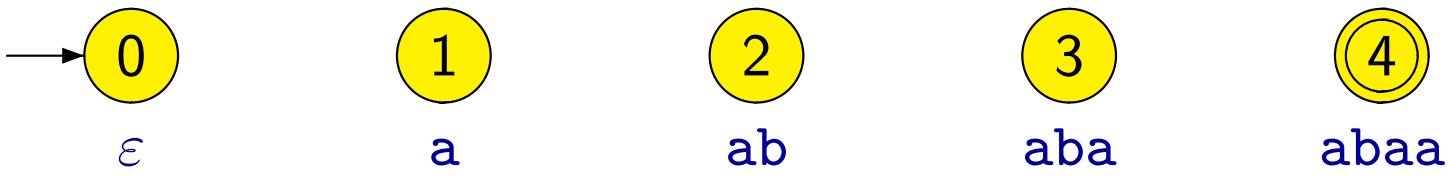
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:



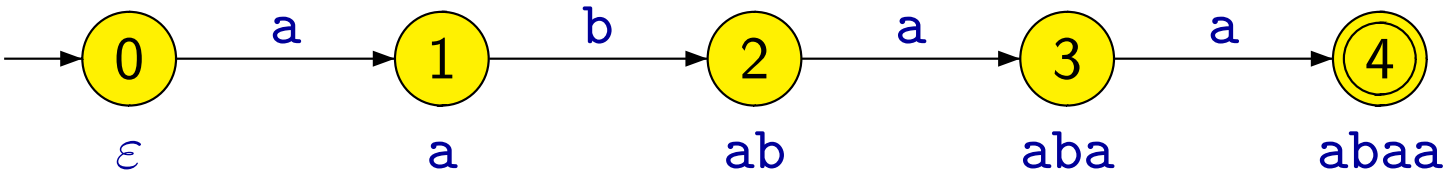
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:



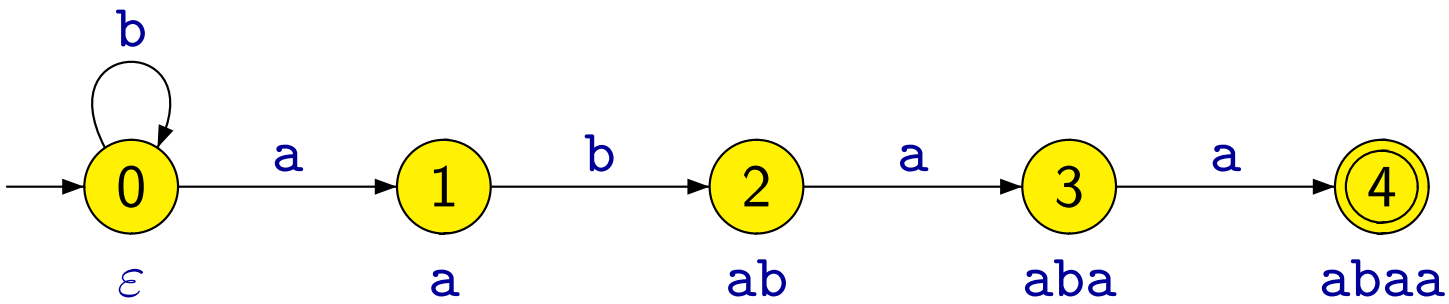
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:



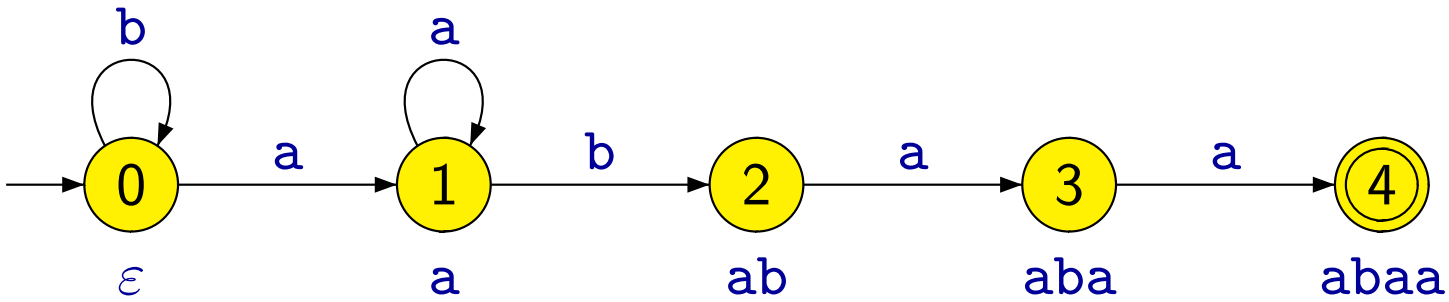
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:



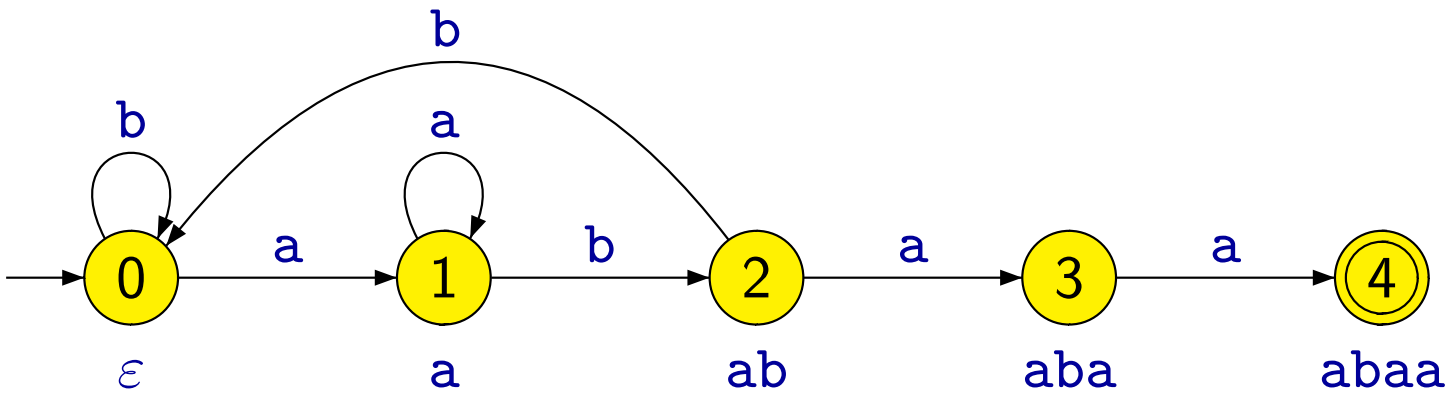
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:



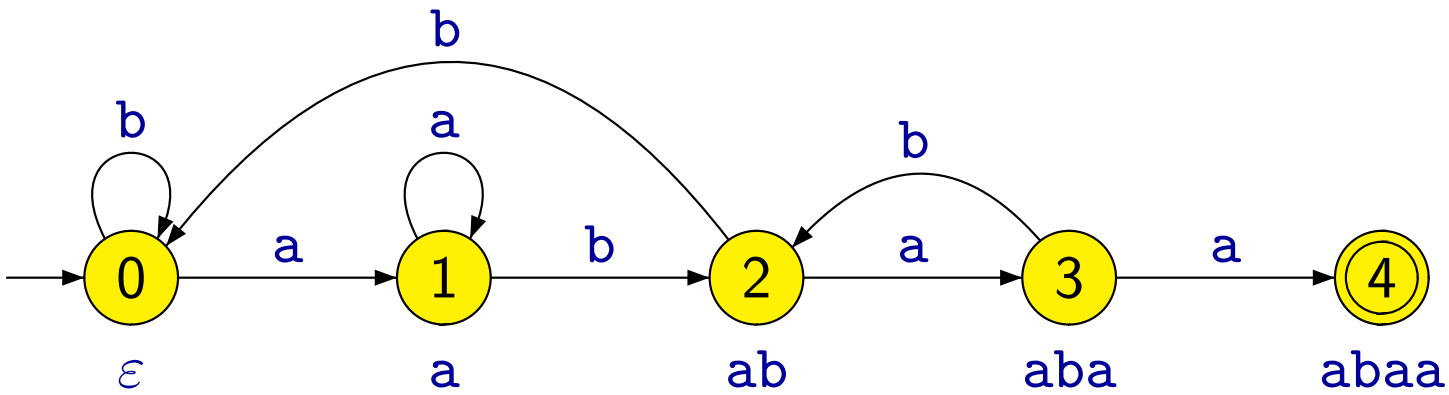
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:



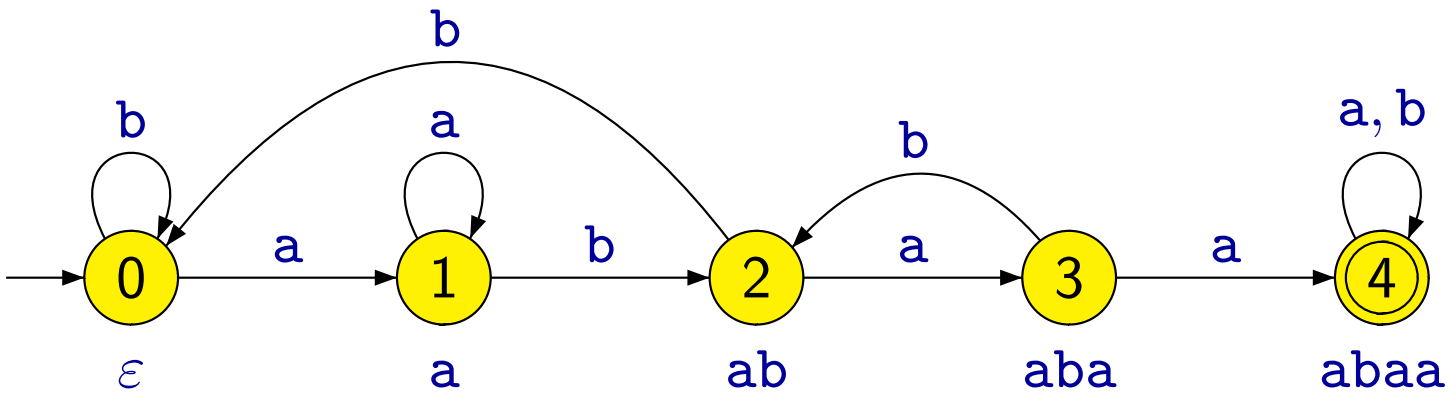
Vyhledávání v textu

Hodnota q, kterou si během výpočtu pamatujeme je tedy délka prefixu slova s, který současně sufixem dosud přečtené části slova t.

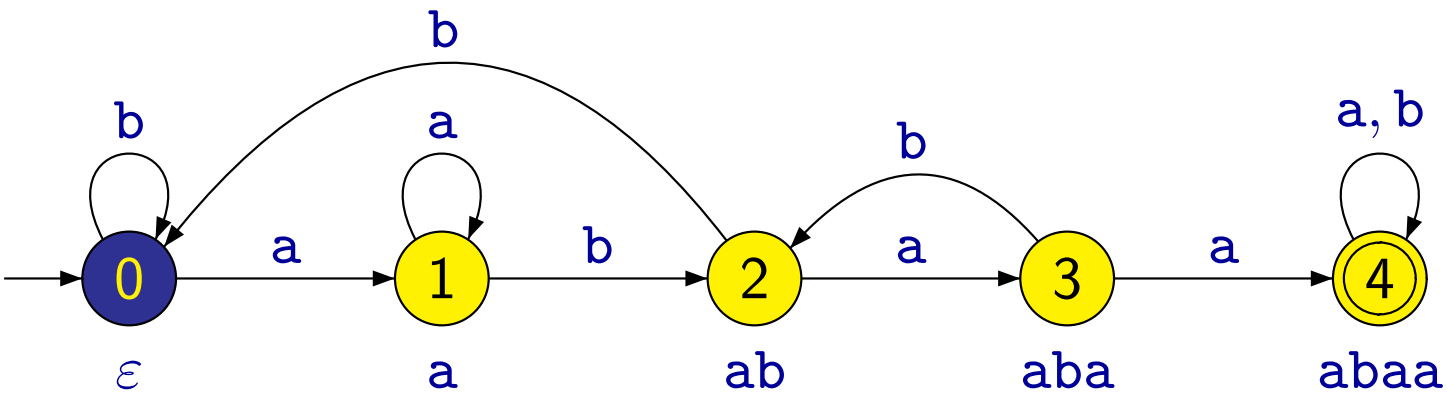
Poznámka: Pokud je takových prefixů víc, pamatujeme si délku nejdelšího z nich.

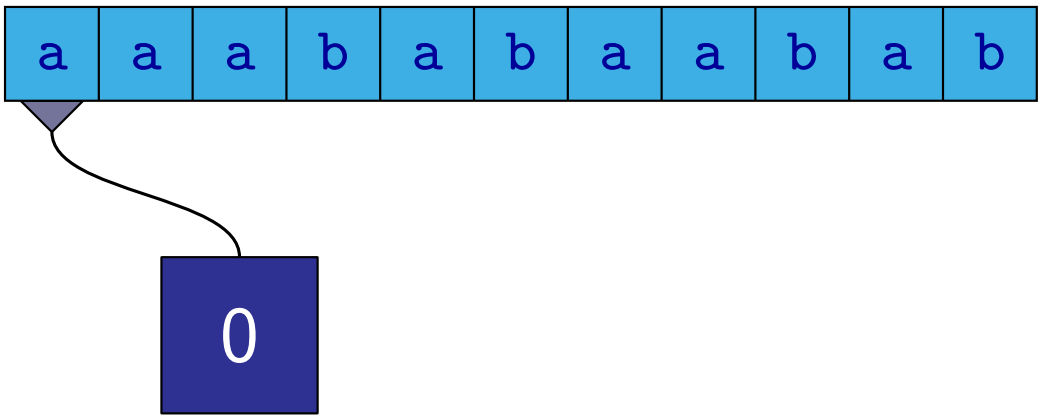
Proměnná zjevně může nabývat jen hodnot $\{0, 1, \ldots, m\}$. Hodnoty m nabývá jen v případě, že bylo nalezeno slovo s.

Chování tohoto systému si opět můžeme znázornit jako graf:

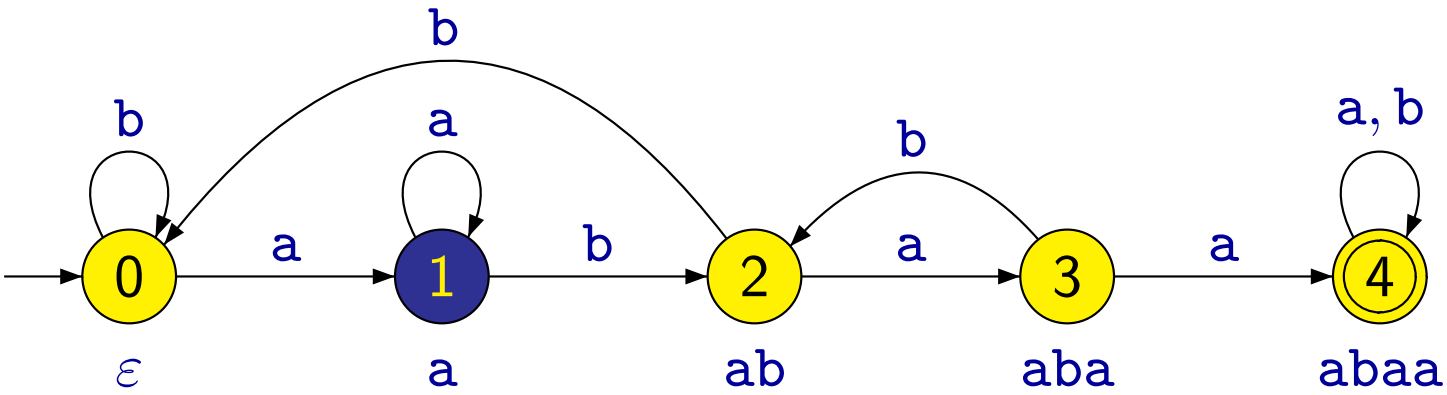


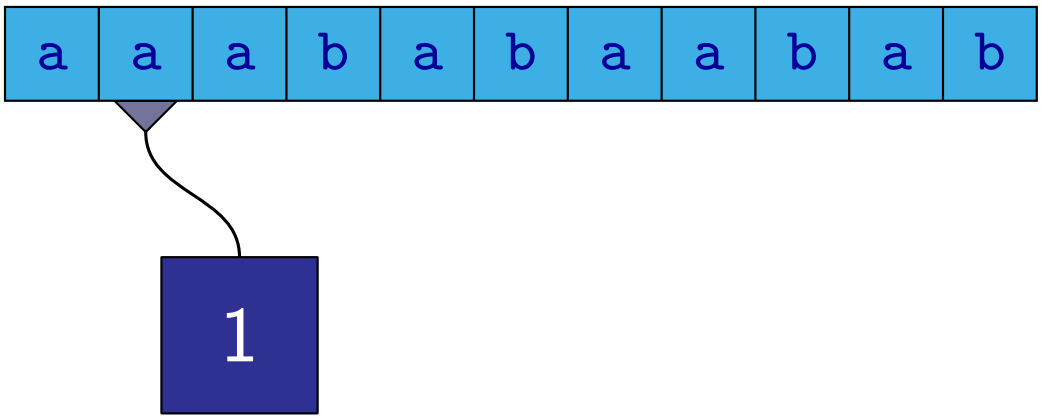
Vyhledávání v textu



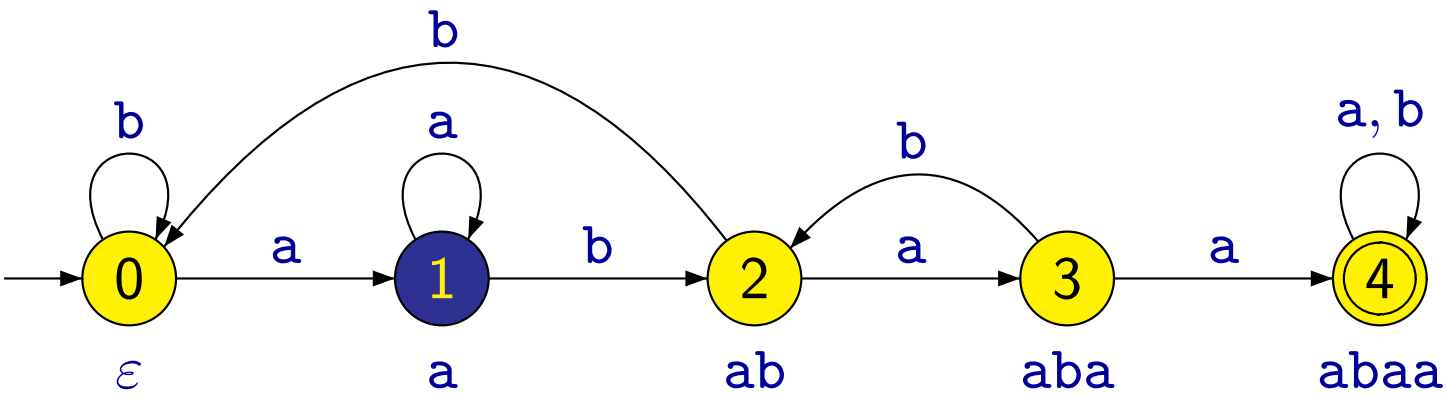


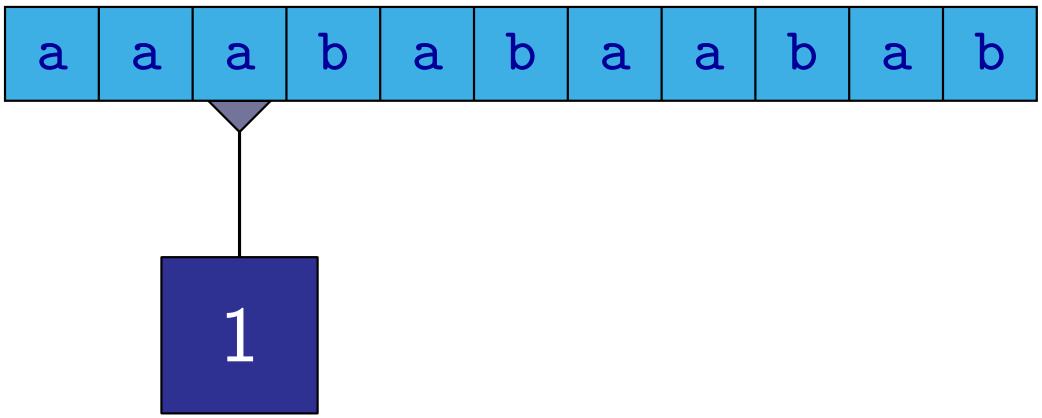
Vyhledávání v textu



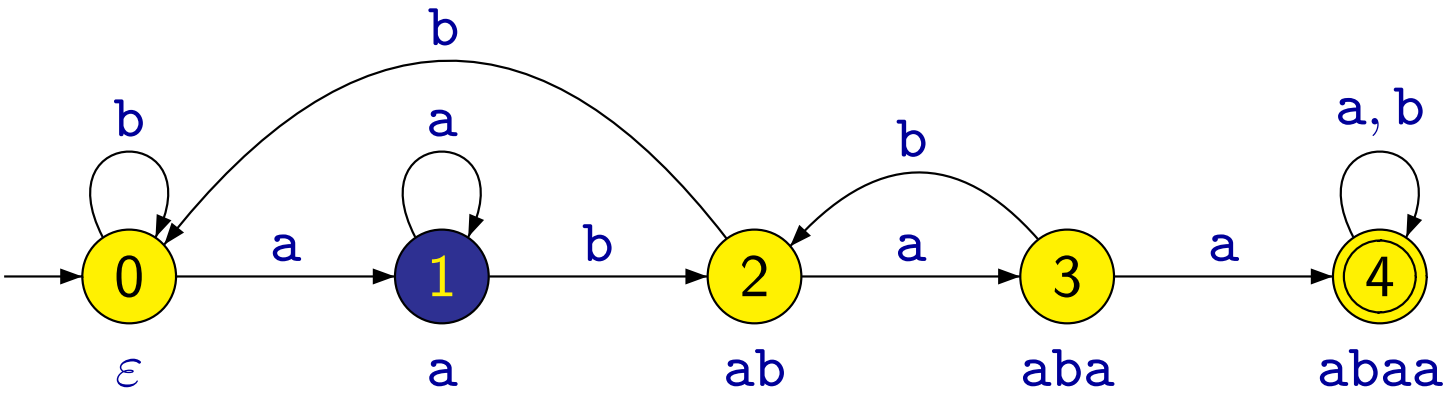


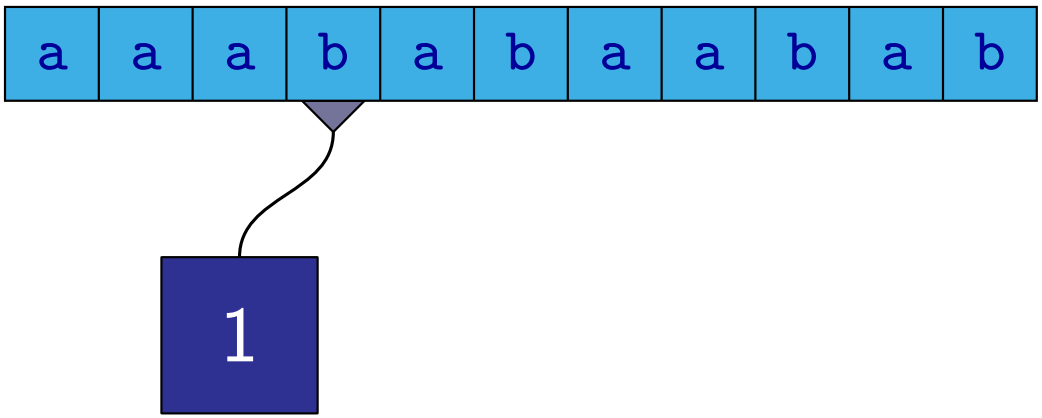
Vyhledávání v textu



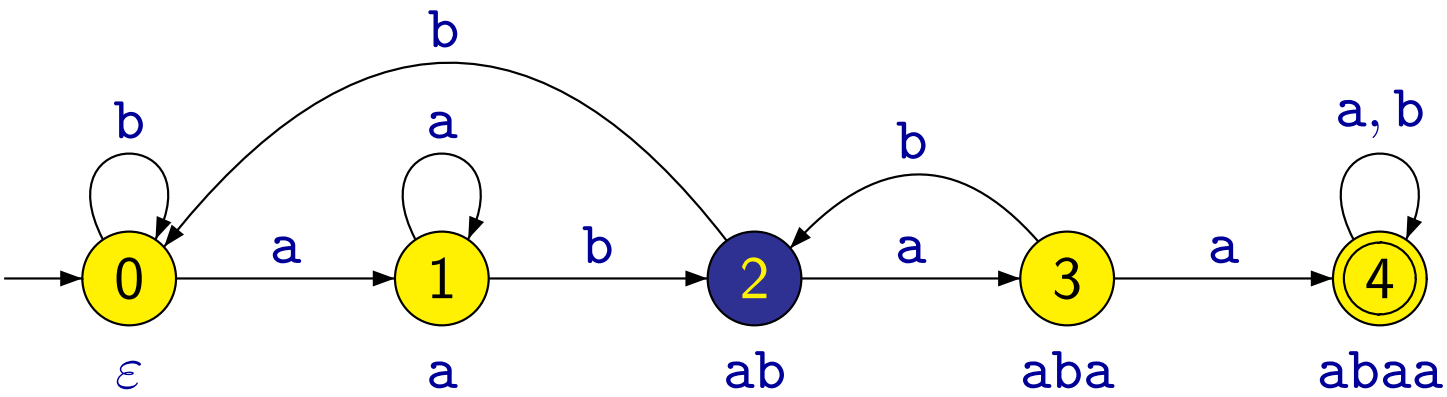


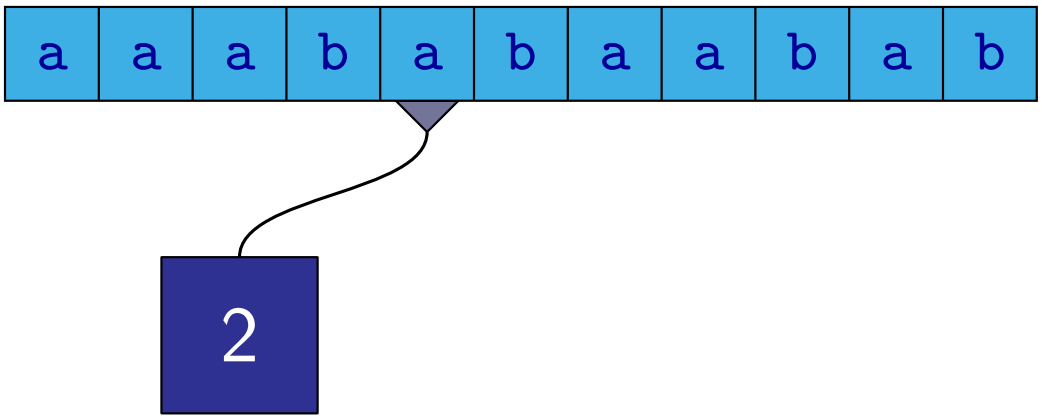
Vyhledávání v textu



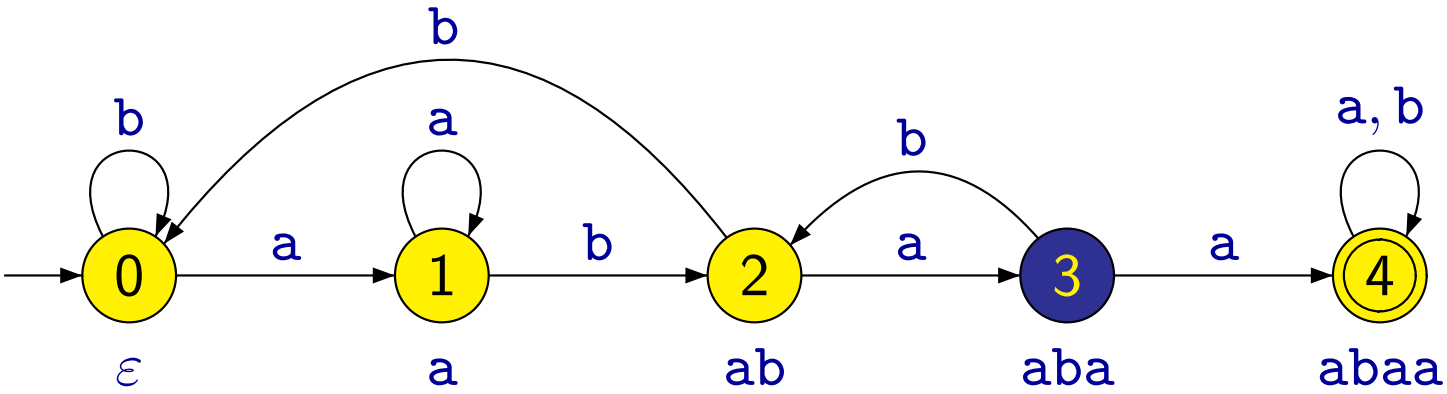


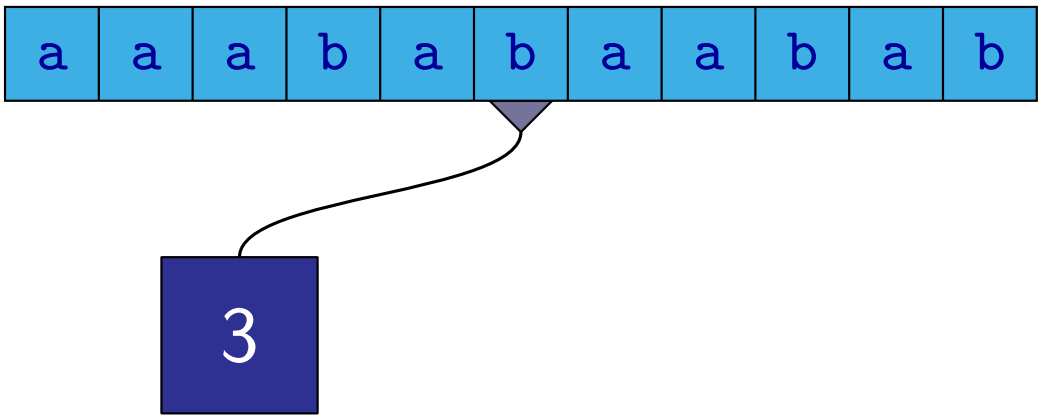
Vyhledávání v textu



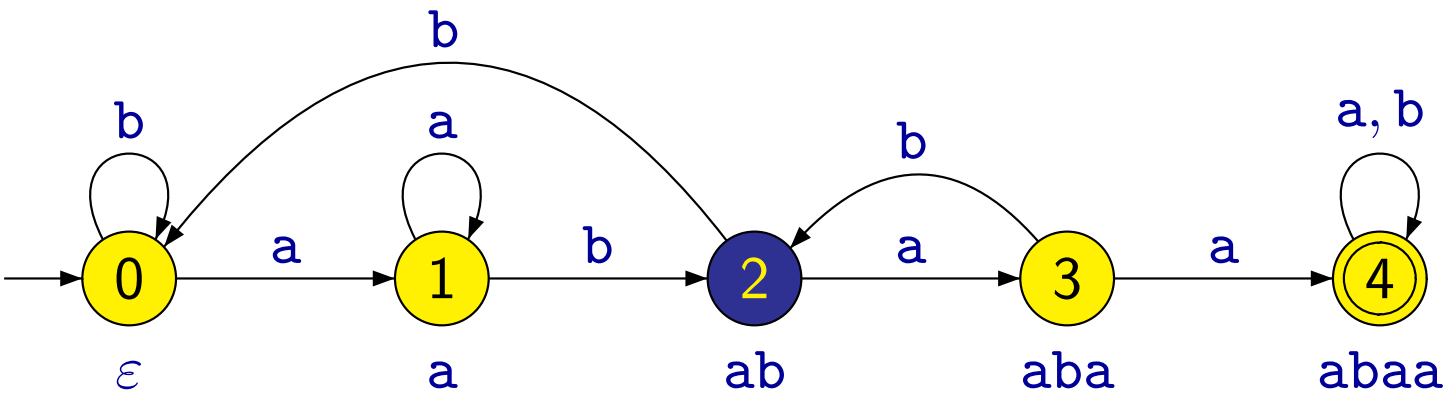


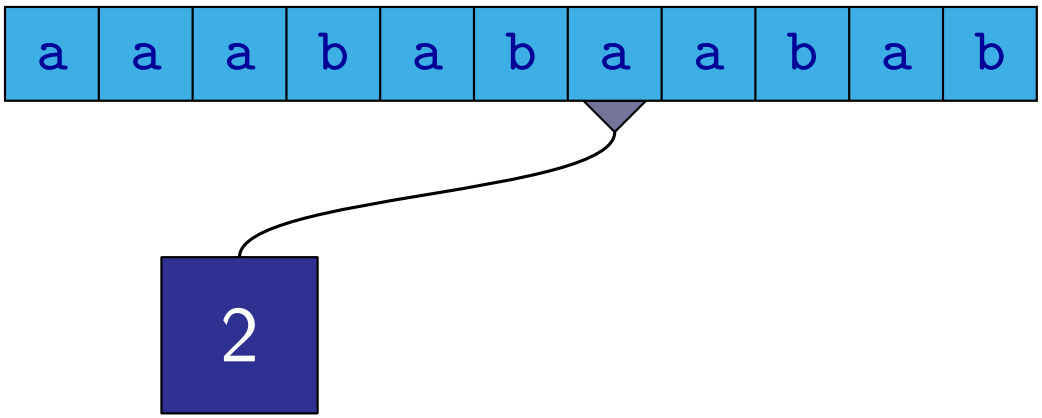
Vyhledávání v textu



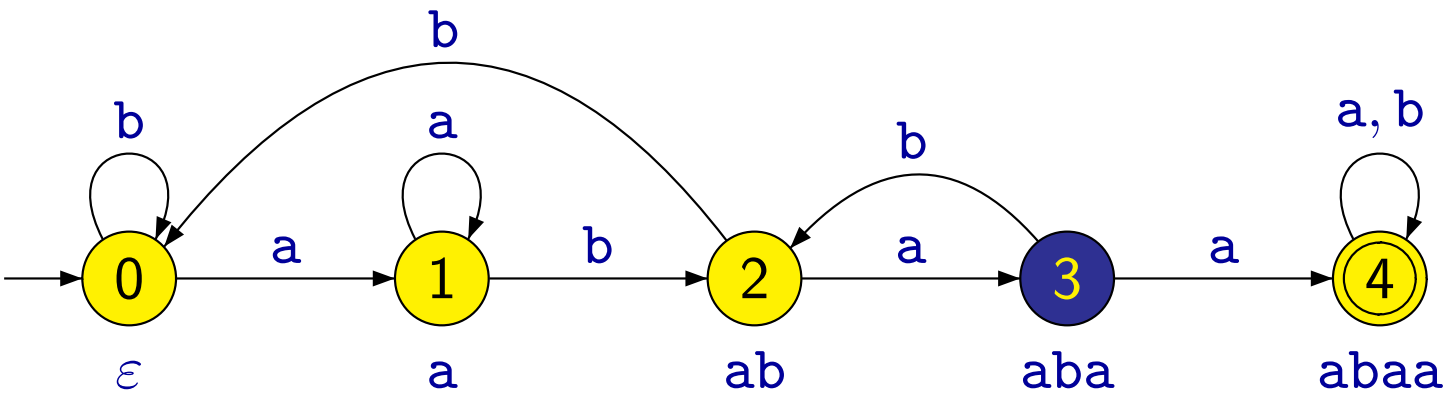


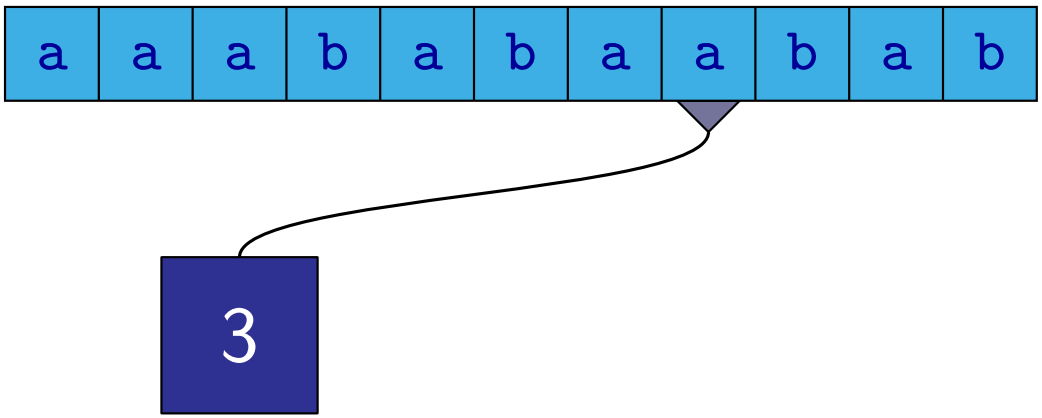
Vyhledávání v textu



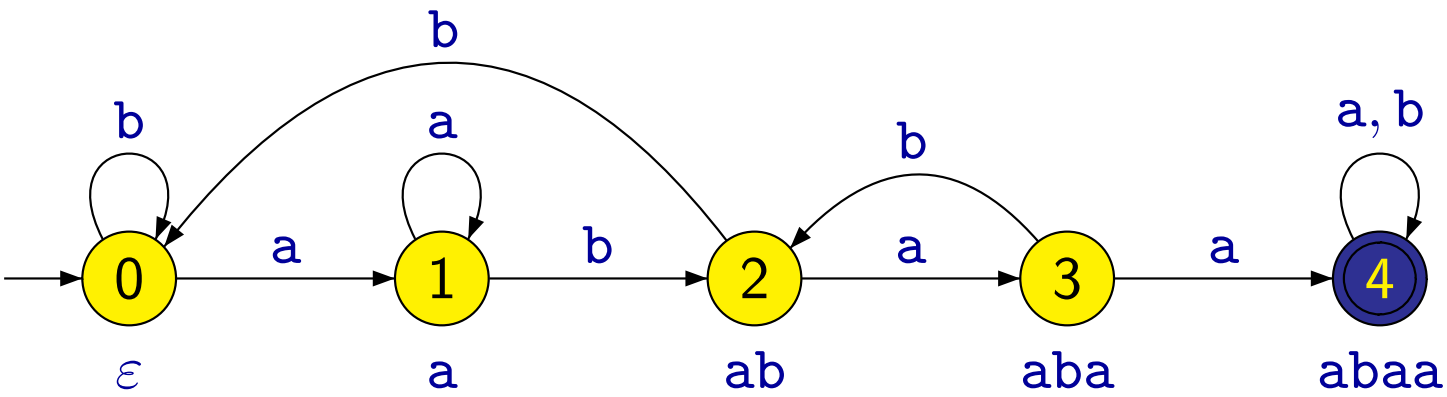


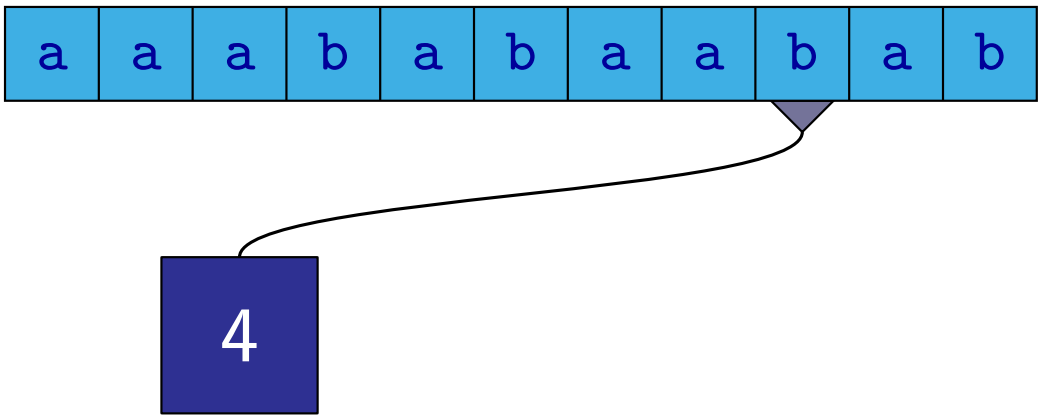
Vyhledávání v textu



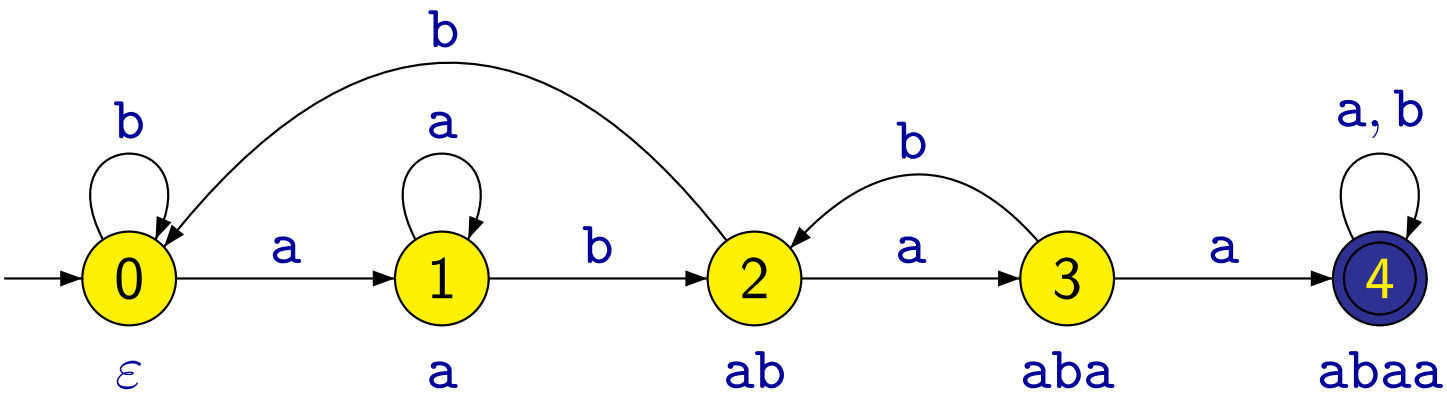


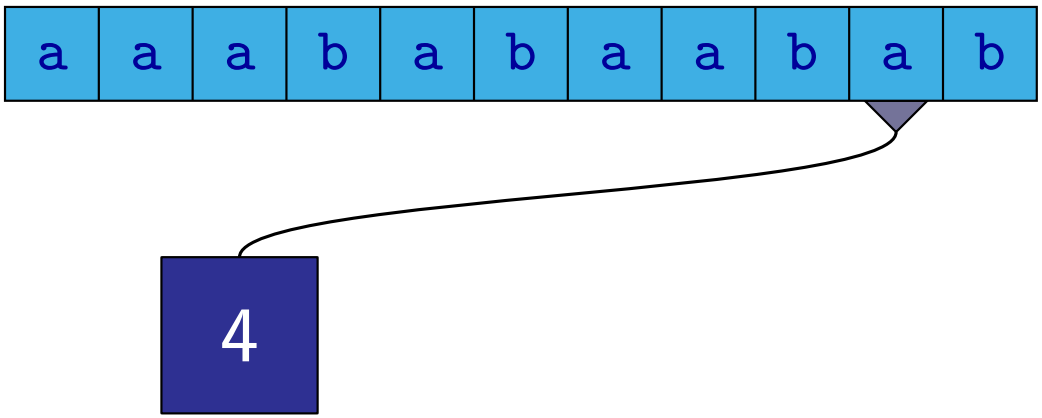
Vyhledávání v textu



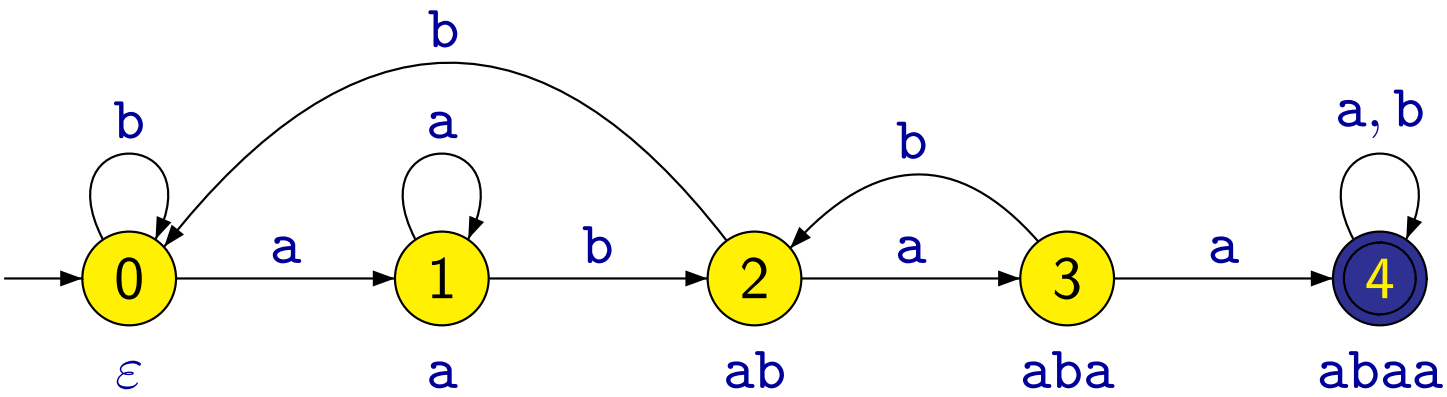


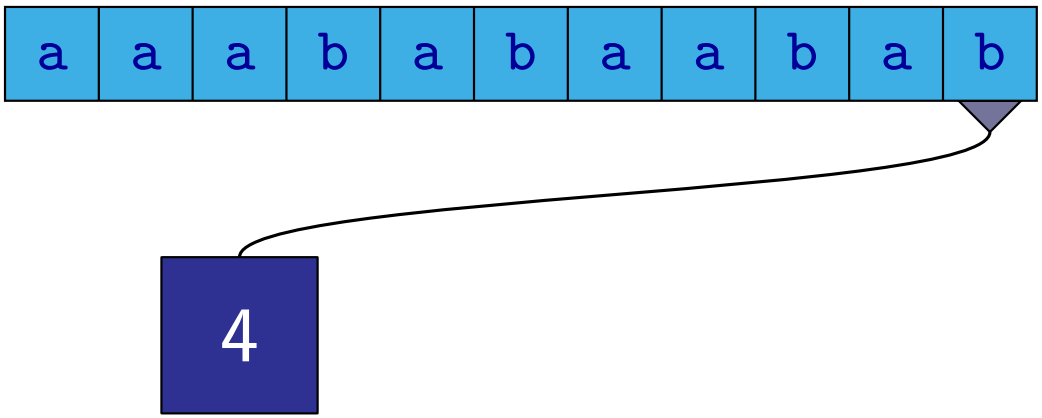
Vyhledávání v textu



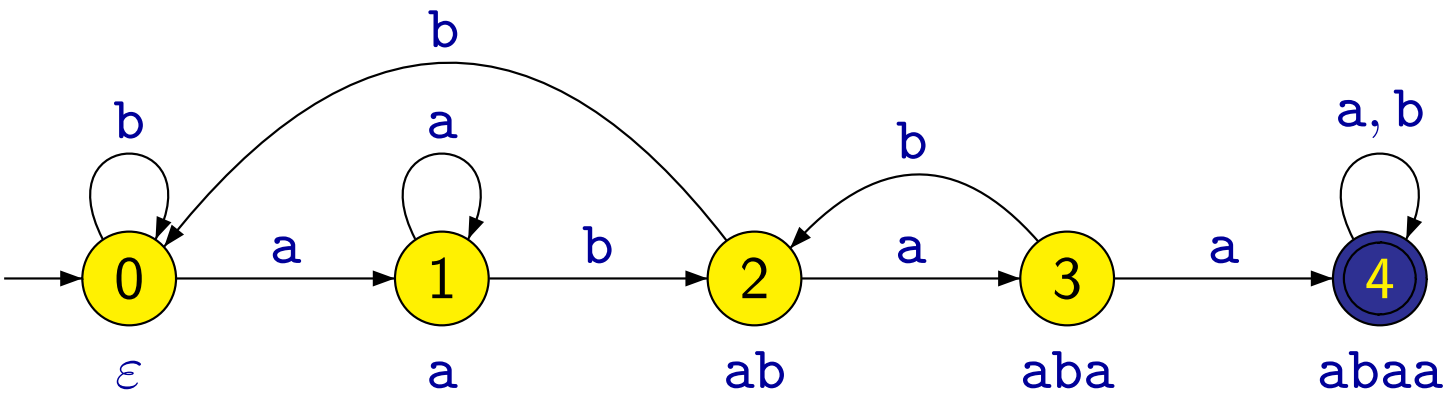


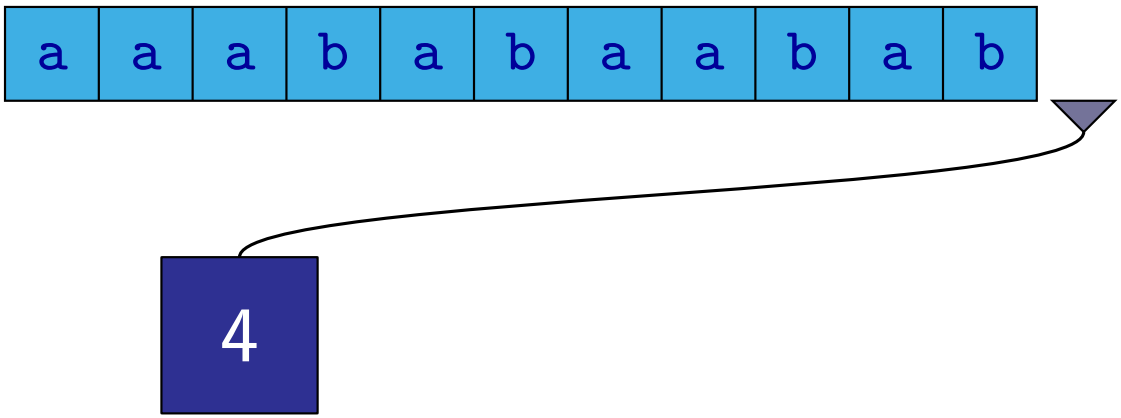
Vyhledávání v textu



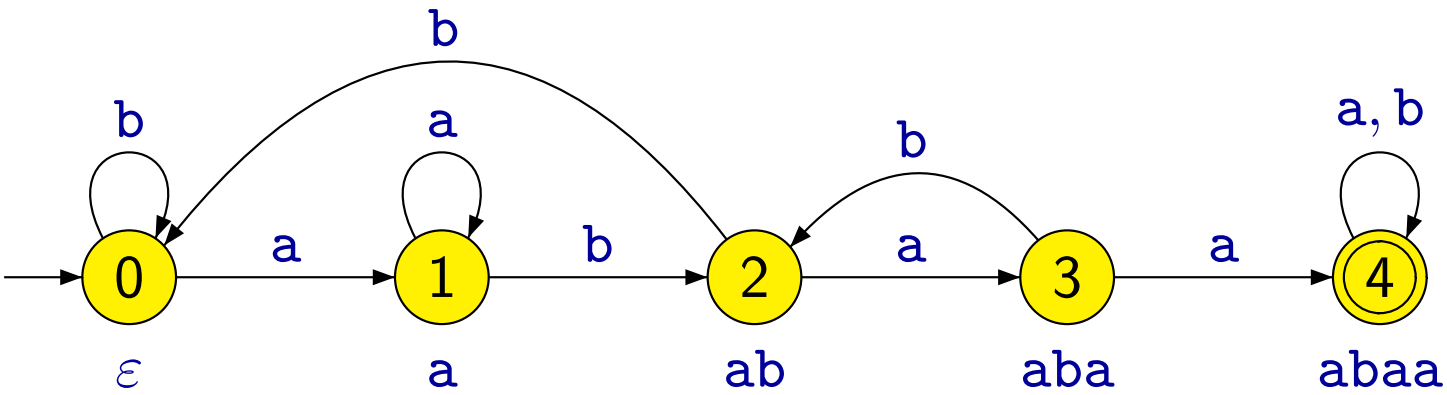


Vyhledávání v textu





Vyhledávání v textu



Místo grafu můžeme stejnou informaci reprezentovat tabulkou:

	a	b
→ 0	1	0
1	1	2
2	3	0
3	4	2
← 4	4	4

Vyhledávání v textu

Jak zvolit pro danou dvojici q a t_i novou hodnotu q, tj. hodnotu $\delta(q, t_i)$?

Vyhledávání v textu

Jak zvolit pro danou dvojici q a t_i novou hodnotu q, tj. hodnotu $\delta(q, t_i)$?

Zvolíme $\delta(q, t_i) = q'$ takové, že slovo $s_0 s_1 \cdots s_{q'-1}$ je nejdelším prefixem slova s takovým, že je současně suffixem slova $s_0 s_1 \cdots s_{q-1} t_i$.

Poznámka: Předpokládáme, že $s = s_0 s_1 \cdots s_{m-1}$.

Vyhledávání v textu

Jak zvolit pro danou dvojici q a t_i novou hodnotu q, tj. hodnotu $\delta(q, t_i)$?

Zvolíme $\delta(q, t_i) = q'$ takové, že slovo $s_0 s_1 \cdots s_{q'-1}$ je nejdelším prefixem slova s takovým, že je současně suffixem slova $s_0 s_1 \cdots s_{q-1} t_i$.

Poznámka: Předpokládáme, že $s = s_0 s_1 \cdots s_{m-1}$.

Poznámka: Výše uvedené úvahy vedou k algoritmu nazývanému podle jeho autorů Knuth-Morris-Pratt.

V tomto algoritmu se vyhneme tomu, že bychom výše uvedenou tabulku skutečně sestrojili. Místo ní se sestrojí určitá její stručnější reprezentace, která ale umožňuje hodnoty z tabulky rychle vypočítat.

Zde se ale nebudeme tímto algoritmem blíže zabývat.