

# VYČÍSLITELNOST A SLOŽITOST



**Kurz:**      **Datové struktury a algoritmy**

---

**Lektor:**    Doc. Ing. Radim Burget, Ph.D.

**Autor:**     Doc. Ing. Radim Burget, Ph.D.



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Vytvoření této videopřednášky bylo podpořeno projektem č. CZ.1.07/2.2.00/28.0098  
Evropského sociálního fondu a státním rozpočtem České republiky.

# Motivace

- Jak vedení obhájit, že řešení stále nedosahuje parametrů, kterých by si představovali?



## Možné odpovědi:

- **A:** Nevím, možná nahrazení mne někým zkušenějším pomůže. ☒
- **B:** Tento problém nelze vyřešit. (je sice mnohdy pravda, velmi obtížné prokázat) ☒
- **C:** Spadá to do skupiny problémů, které jsou srovnatelné s jinými a které ani vědci za posledních X let nedokázali vyřešit. ☒

# Cíl přednášky

## 1. Teorie vyčísitelnosti

- Jak vypadá výpočetní model současných výpočetních systémů
- Deterministický konečný automat, Nedeterministický konečný automat, Zásobníkový automat, Turingův stroj + jeho varianty
- Existují i nevyčíslitelné problémy

## 2. Teorie složitosti – jak posuzovat algoritmy?

- Absolutní
- Asymptotická složitost
- Třídy složitosti P, NP, NP-těžké
- Problém ekvivalence P vs. NP. – jeden z top desítky největších problémů matematiky současnosti

# Úvod

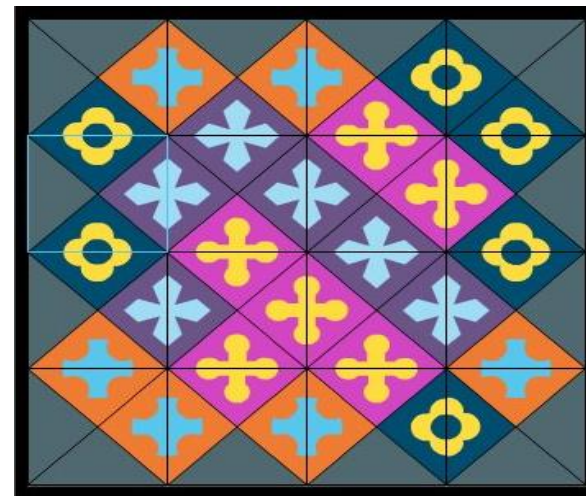
- Teoretická informatika
  - Informace – reprezentovat & zpřístupnit (vyhledat, dopočítat)
  - Říká, co nemá smysl algoritmicky řešit
  - Kde jsou hranice spočítatelnosti
  - Jak se dostat na hranici spočítatelnosti.

# Úvod – Motivace

- Pole 16 x 16 s 256 políčky, Eternity II (hra v prodeji v knihkupectví)
- Úkolem uspořádat políčka tak, aby na sebe navazovala, prodejce slibuje za vyřešení zajímavou finanční odměnu (\$2 mil.)

S pomocí znalostí z tohoto kurzu:

- 12x8 vyřešíte za cca 2 minut (a získáte nápovědu políčka)
- 12x12 vyřešíte za cca 1,5 hod (a získali nápovědu dalšího políčka)
- 16x16 za pár dnů máme vyhráno?  
viz konec přednášky...



Zdroj: [www.eternityii.com](http://www.eternityii.com)

# Úvod

- Proč je důležité znát teorii vyčísitelnosti
  - Existují problémy, kterými netřeba ztrácet čas – nelze je vyřešit
- Proč je důležité znát teorii složitosti
  - Volba optimálních datových struktur
  - Optimální výkon
  - Možnost výpočtu v přijatelném čase
  - Volba jazyka (C, C++, JAVA, C#, ...) má téměř vždy výrazně nižší dopad na výkon aplikace nežli volba datových struktur

# Kde hraje složitost významnou roli

- Směrovací tabulky L3 přepínačů (až 10x rychlejší)
- Návrh analogových obvodů
- Obrazové zpracování
- Data-mining
- Zpracování hlasu
- Zpracování zvuku
- Dolování znalostí z textu
- Systémy řízení báze dat
- Návrh procesorů
- Analýzy finančních trhů
- Předpověď počasí
- Simulace biologických systémů

# Volba datových struktur

- **Datové struktury** a **algoritmy** jsou základem pro veškeré programovací jazyky
  - Volba datových typů: `T[ ]`, `LinkedList<T>`, `List<T>`, `Stack<T>`, `Queue<T>`, `Map<K, T>`, `HashSet<T>`, `TreeMap<K, T>` a `TreeSet<T>` má výrazný dopad na výkonnost aplikace
    - o řády vyšší než volba programovacího prostředí



# Vyčísлитelnost



# Teorie vyčísitelnosti

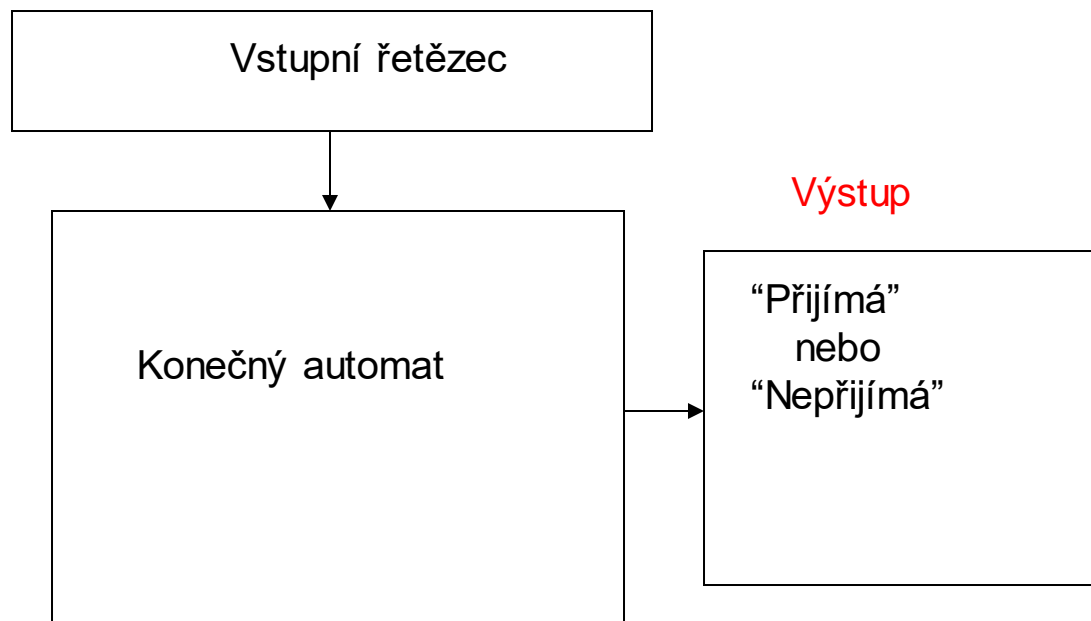
- je vědní obor na pomezí matematiky a informatiky
- zkoumá otázky algoritmické řešitelnosti problémů (ne z pohledu času, ale zdali vůbec)
- hranice využití algoritmicky pracujících postupů
- Vyčísitelnost je zkoumána pomocí teoretických výpočetních modelů, např.:
  - Deterministický konečný automat
  - Nedeterministický konečný automat
  - Zásobníkový automat
  - Turingův stroj
  - ... mnoho dalších

# Vyčísitelnost

- Mají všechny stroje stejnou vyjadřovací sílu? **NE!**
  - Základní členění:
    - Deterministický konečný automat
    - Nedeterministický konečný automat
    - Deterministický zásobníkový automat
    - Nedeterministický zásobníkový automat
    - Turingův stroj + varianty
- a mnoho dalších ...

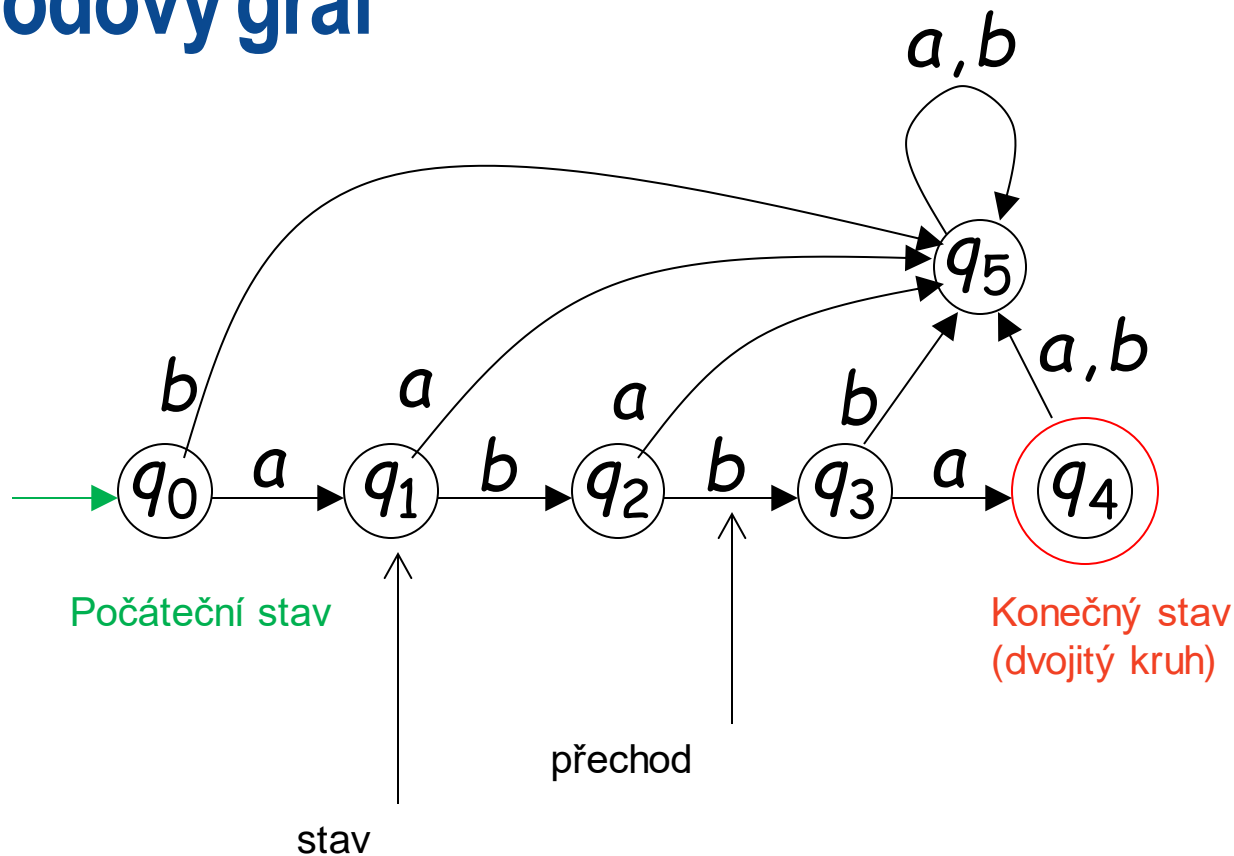
# Deterministický konečný automat (DFA)

Vstupní páska



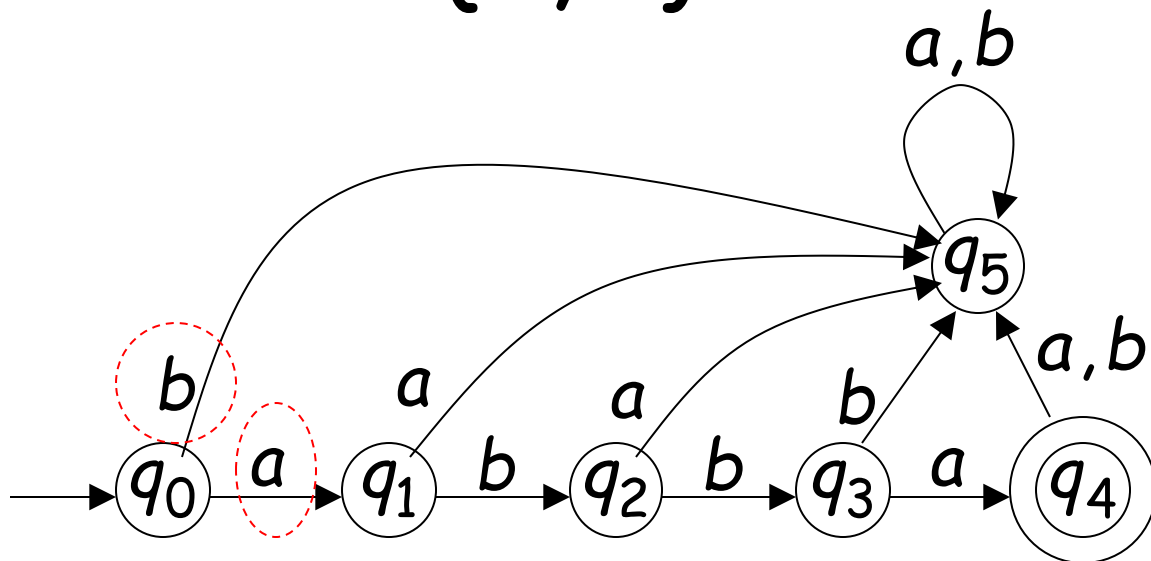
Deterministic Finite Automaton (DFA)

# Přechodový graf



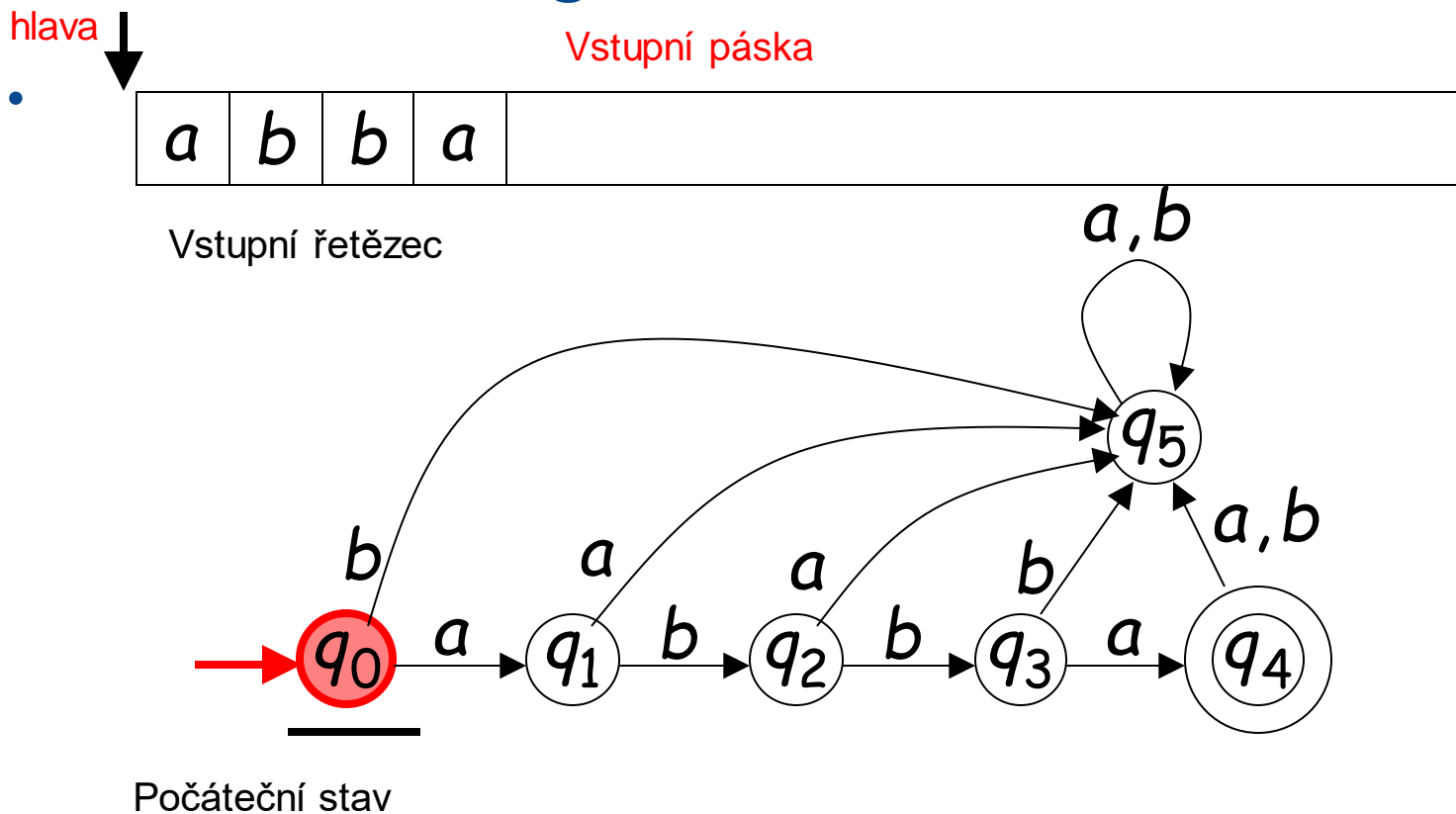
Abeceda

$$\Sigma = \{a, b\}$$



Pro každý stav je zde přechod pro každý symbol v abecedě

# Počáteční konfigurace

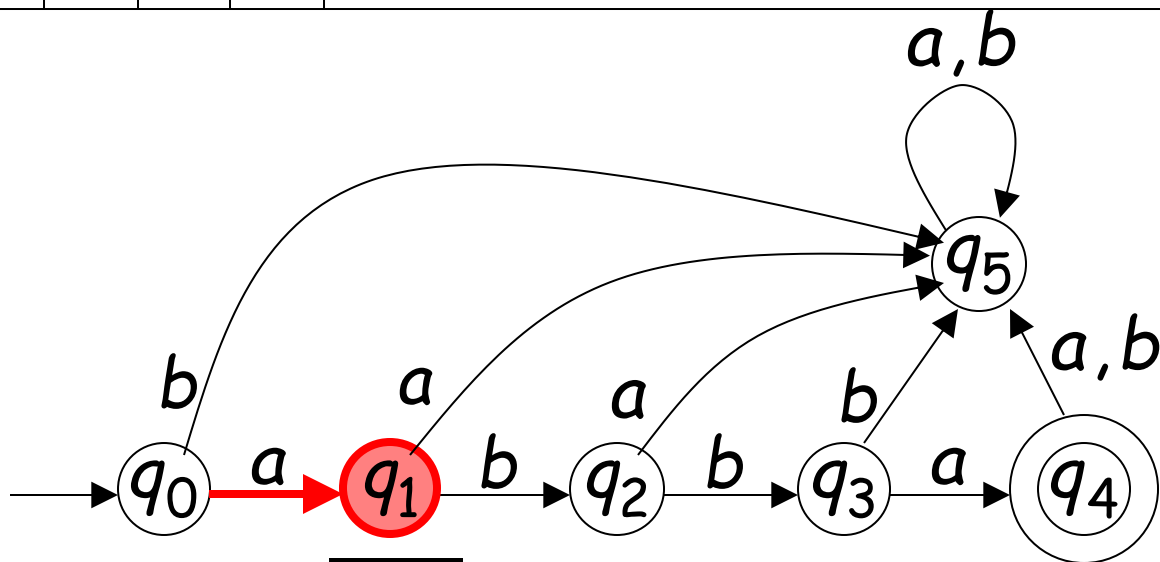


# Čtení vstupního řetězce vstupní pásky

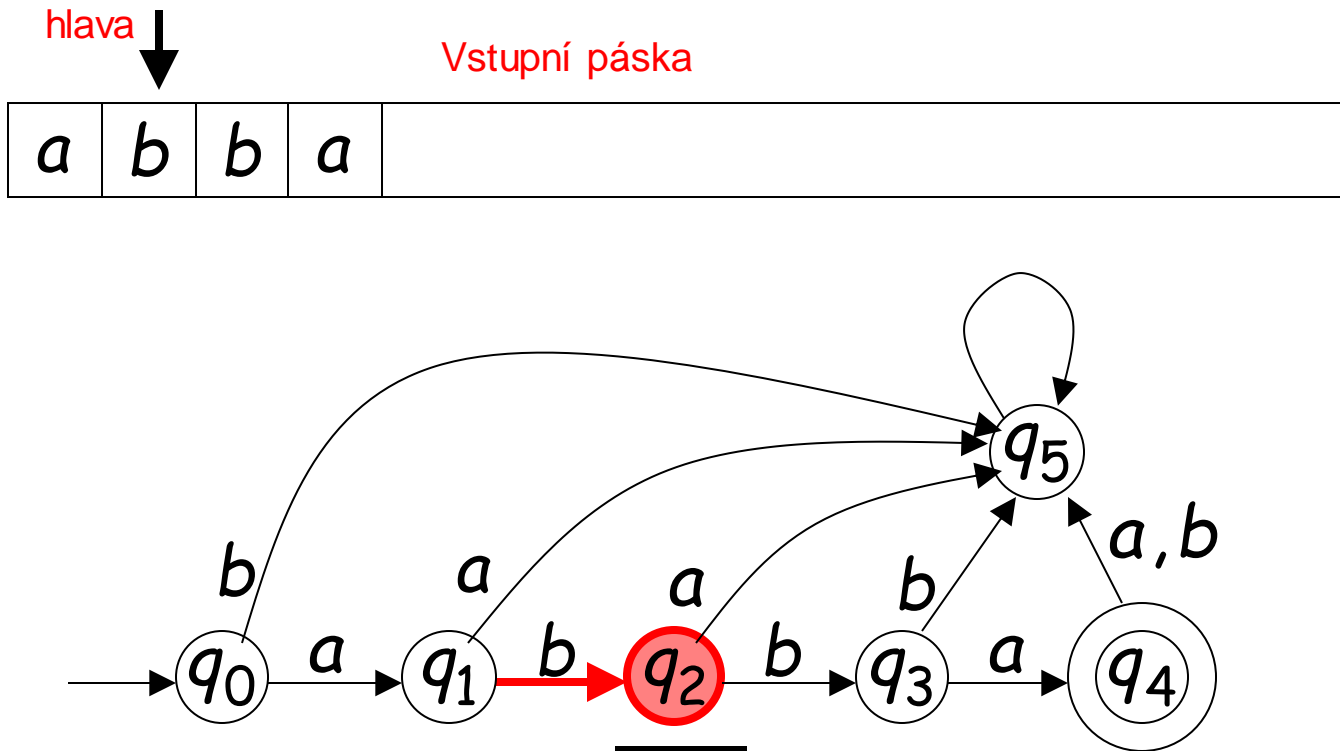
hlava

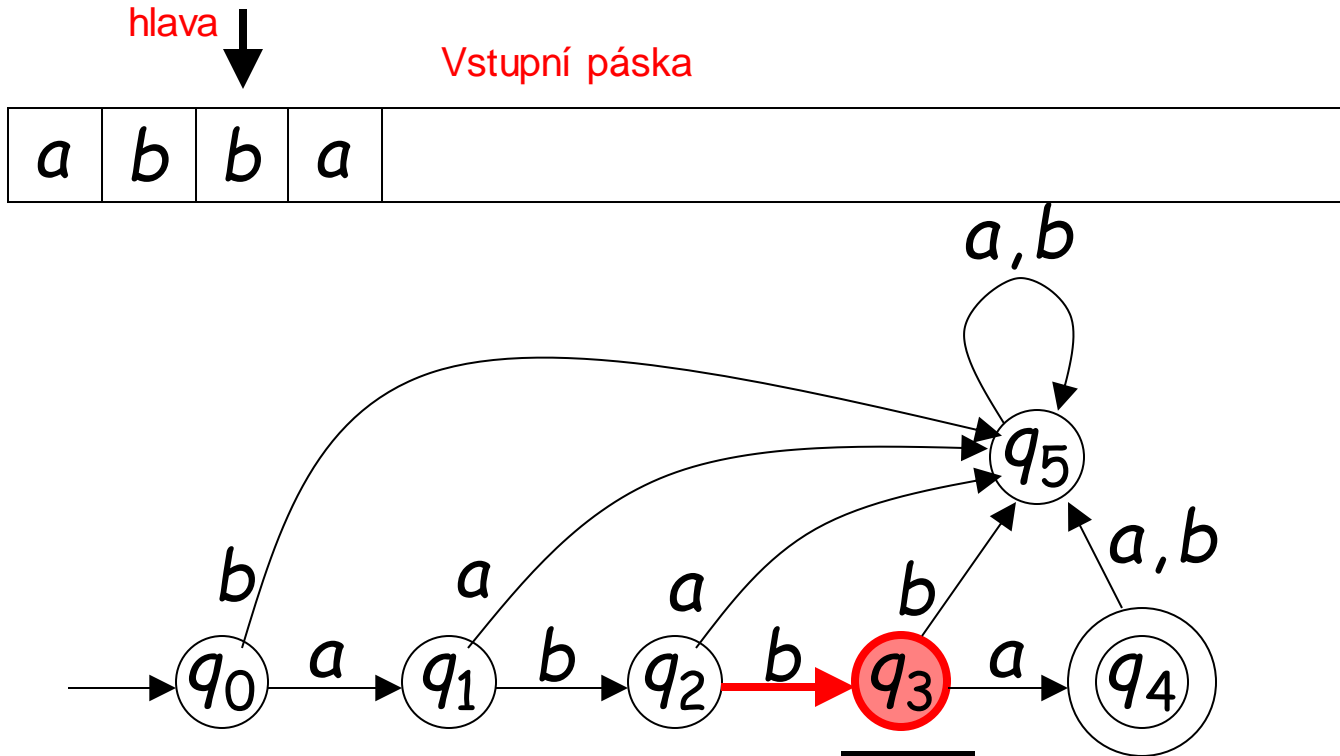


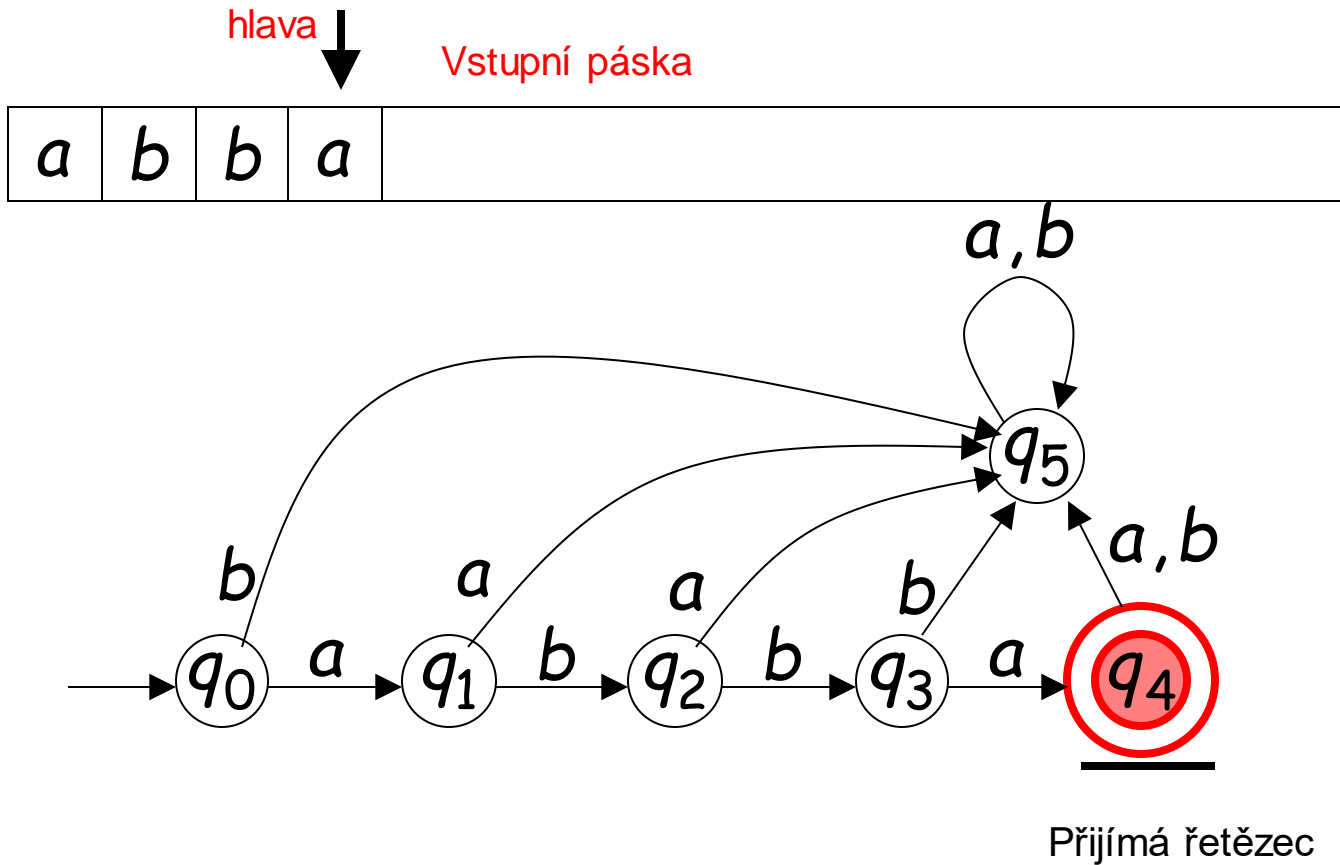
Vstupní páska











Případ zamítnutí

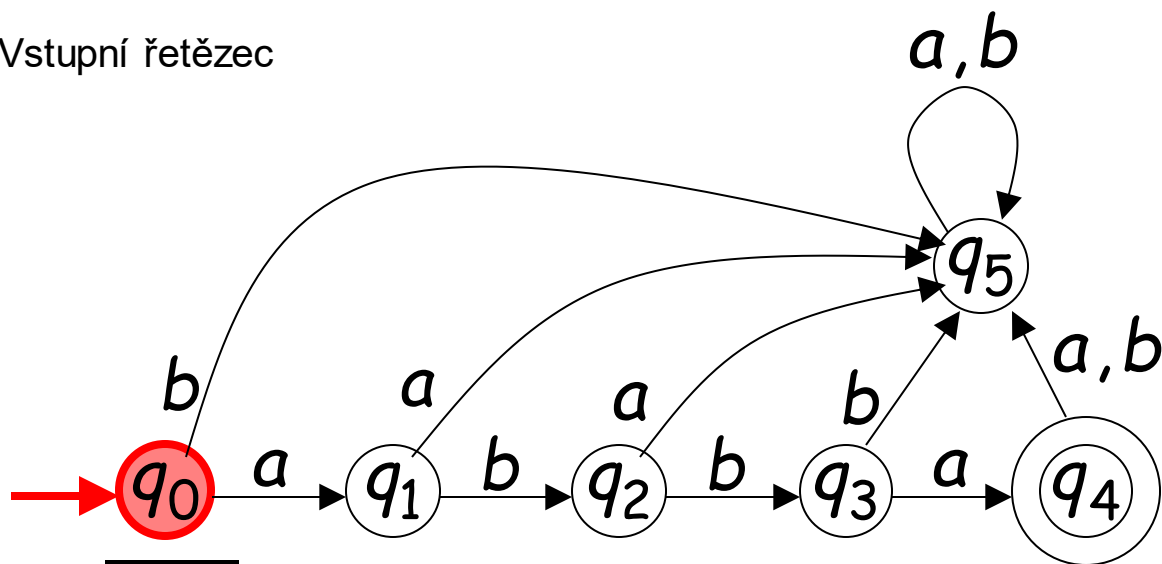
# Příklad – nepřijímá řetězec

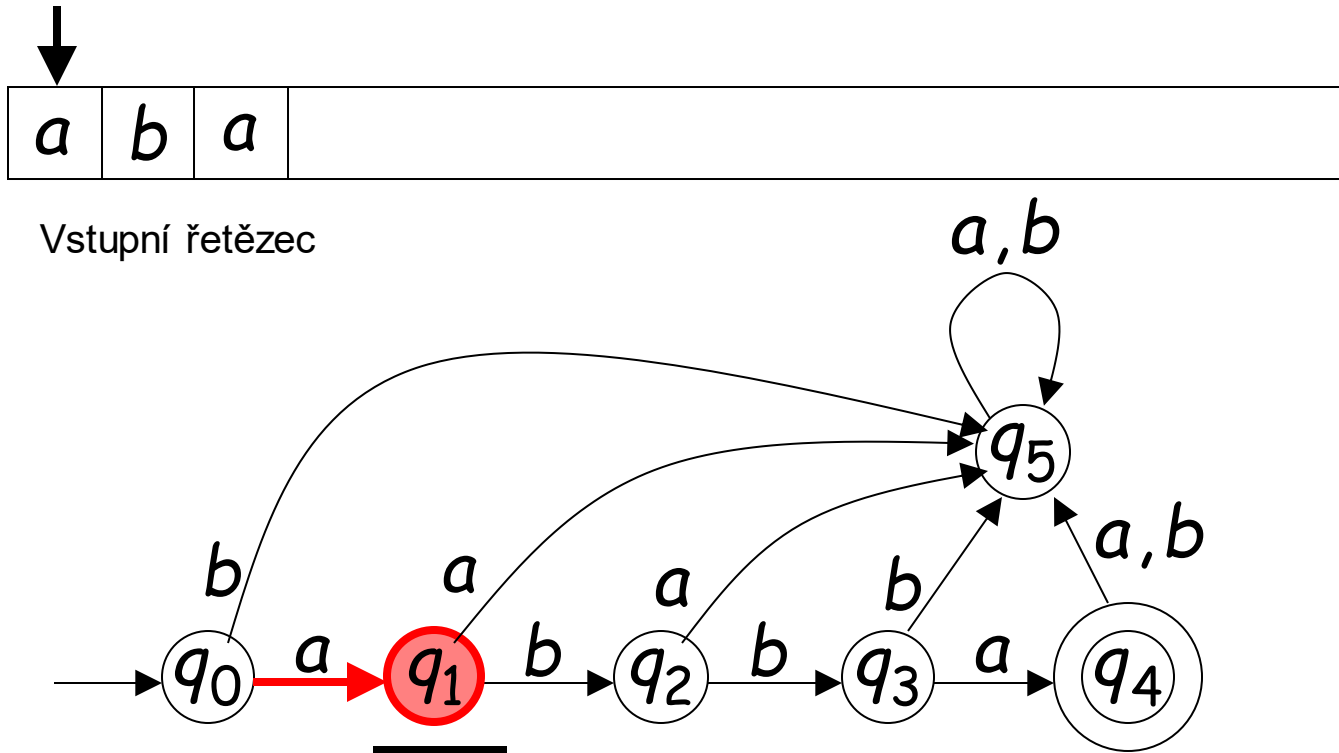


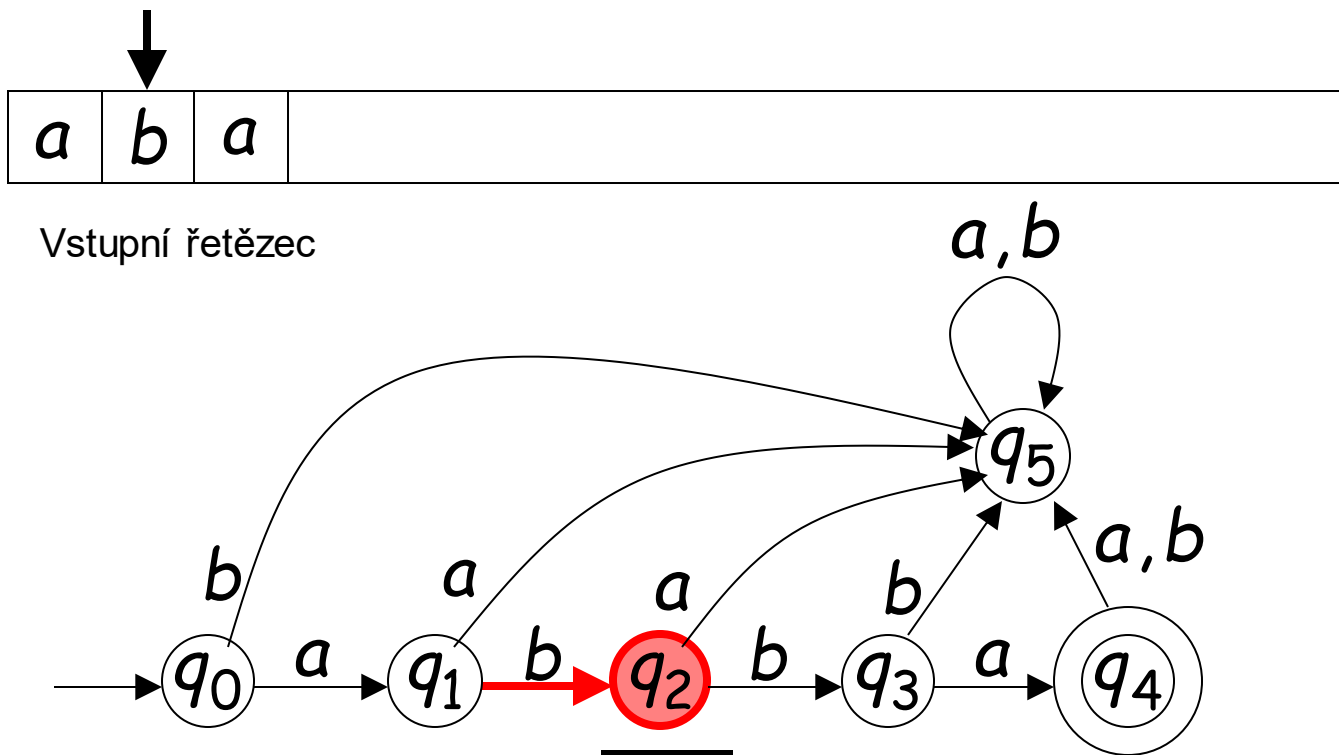
•



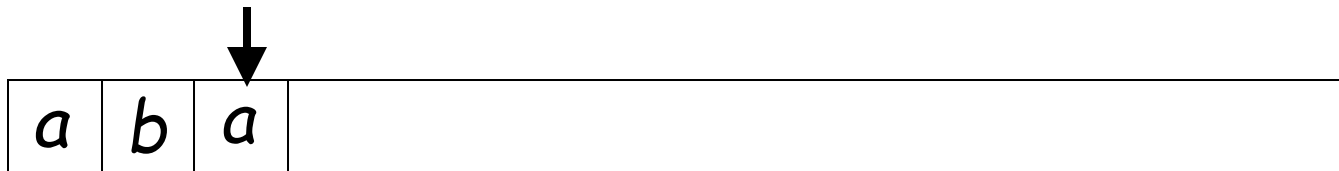
Vstupní řetězec



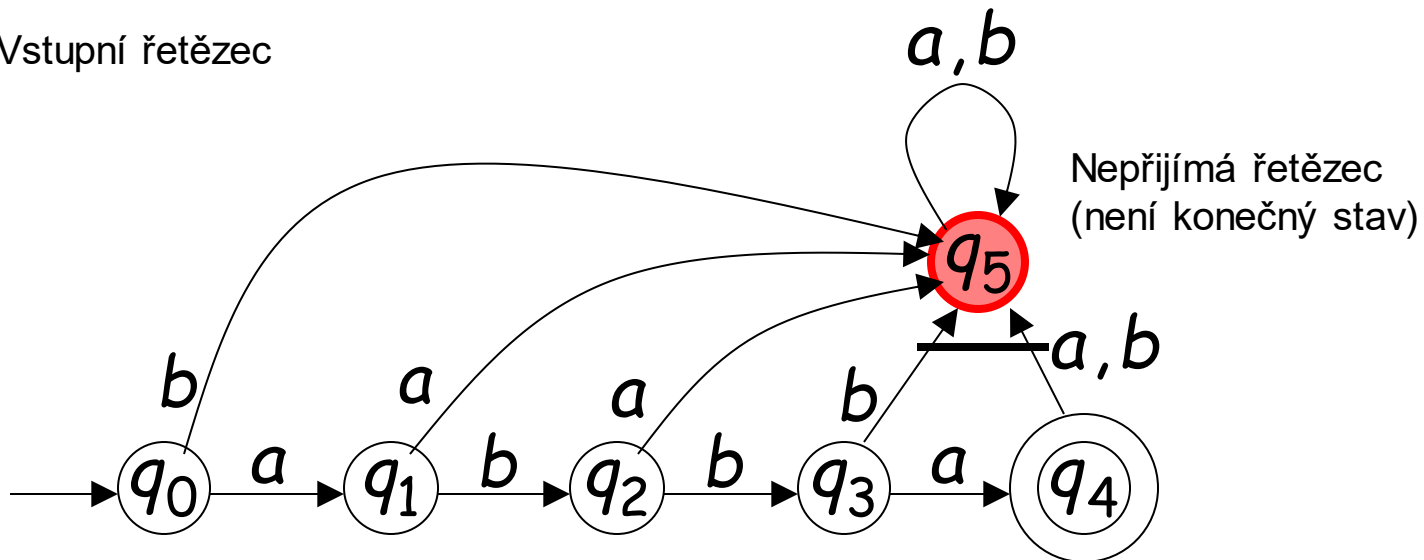




Vstup dokončen



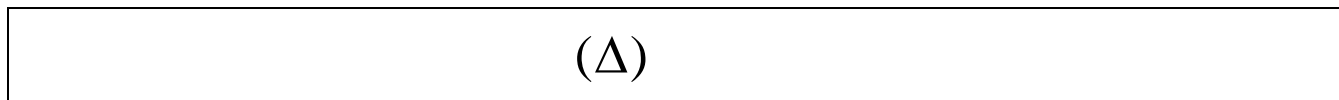
Vstupní řetězec



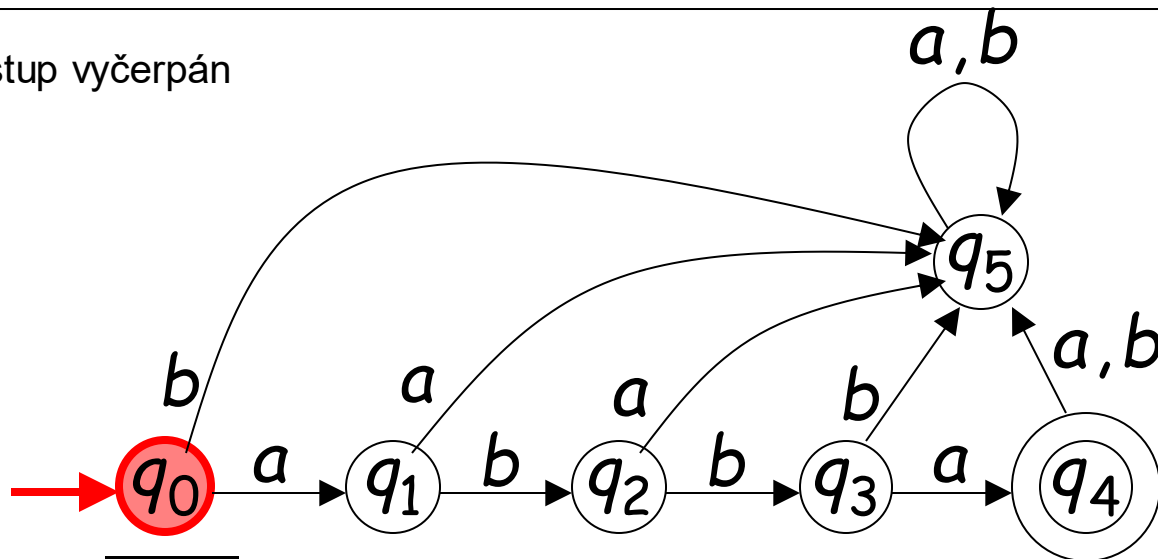
# Další případ zamítnutí



Páska je prázdná



Vstup vyčerpán

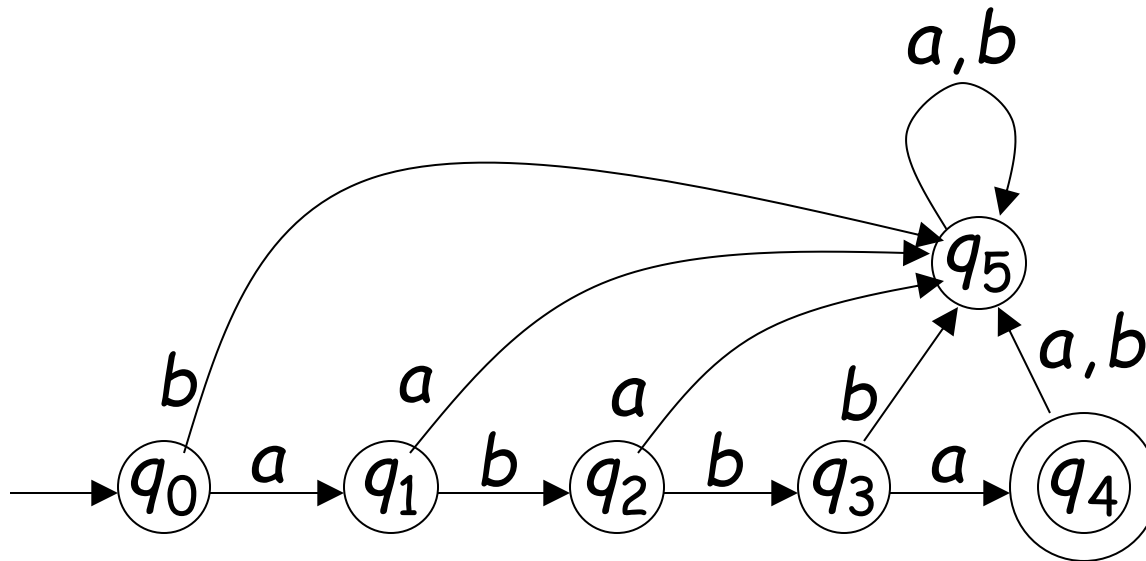


Nepřijímá



Přijatý řetězec:

$$L = \{abba\}$$



### Pro přijetí řetězce:

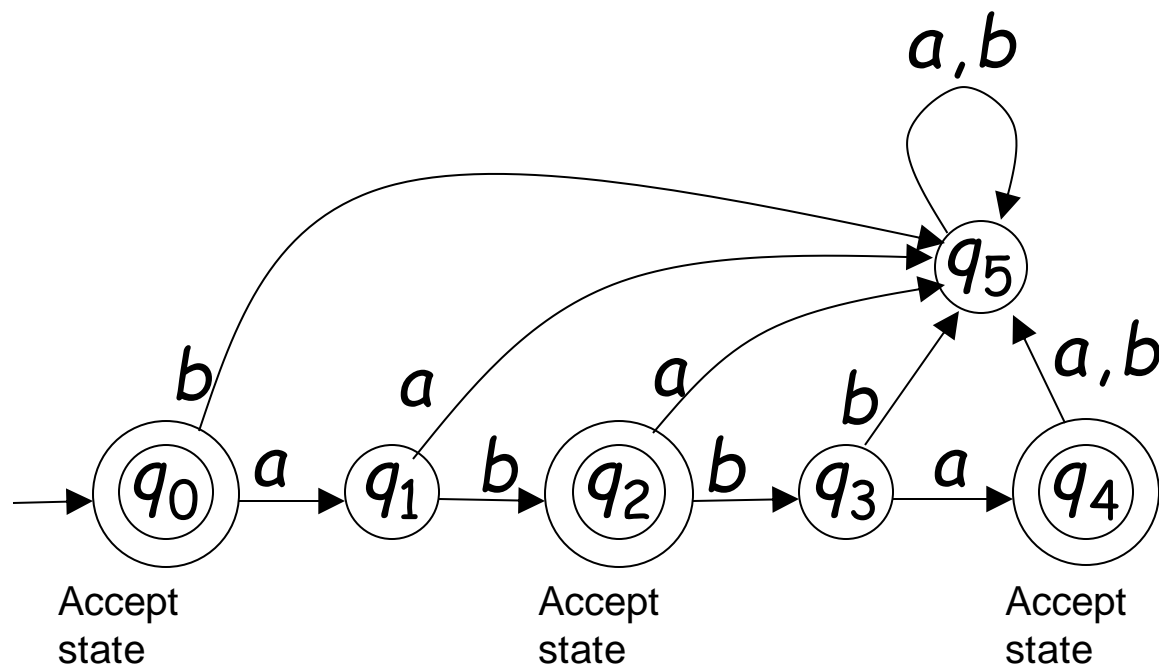
Veškeré znaky vstupní pásky byly přečteny  
Automat skončil ve stavu, který je konečným stavem

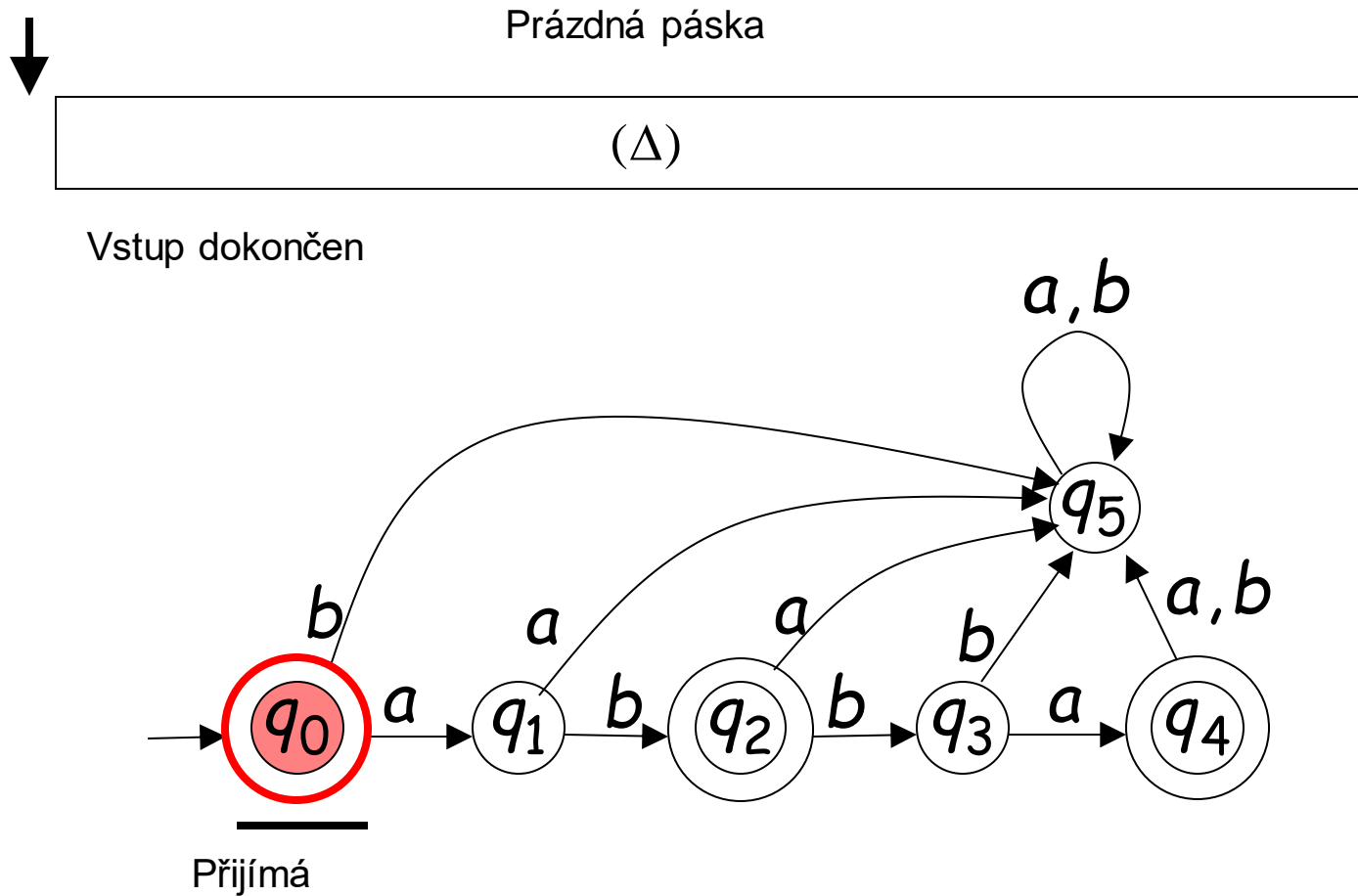
### Pro nepřijetí řetězce:

Veškeré znaky vstupní pásky byly přečteny  
Automat skončil ve stavu, který **není** konečným stavem

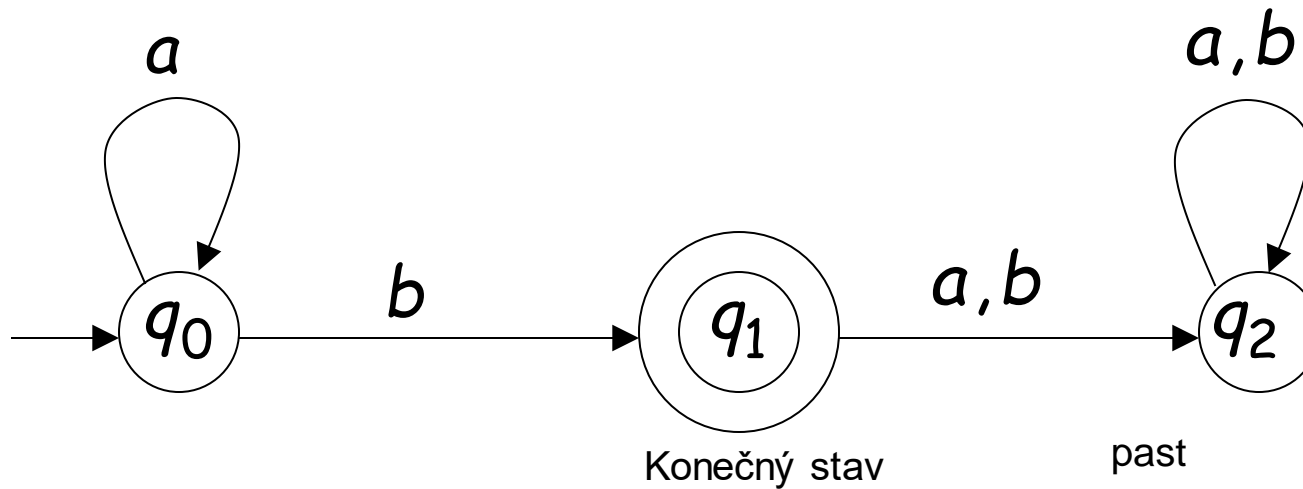
## Další příklad

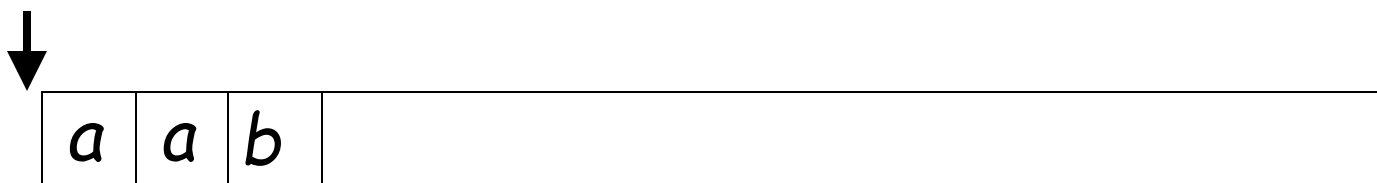
- $L = \{\Delta, ab, abba\}$



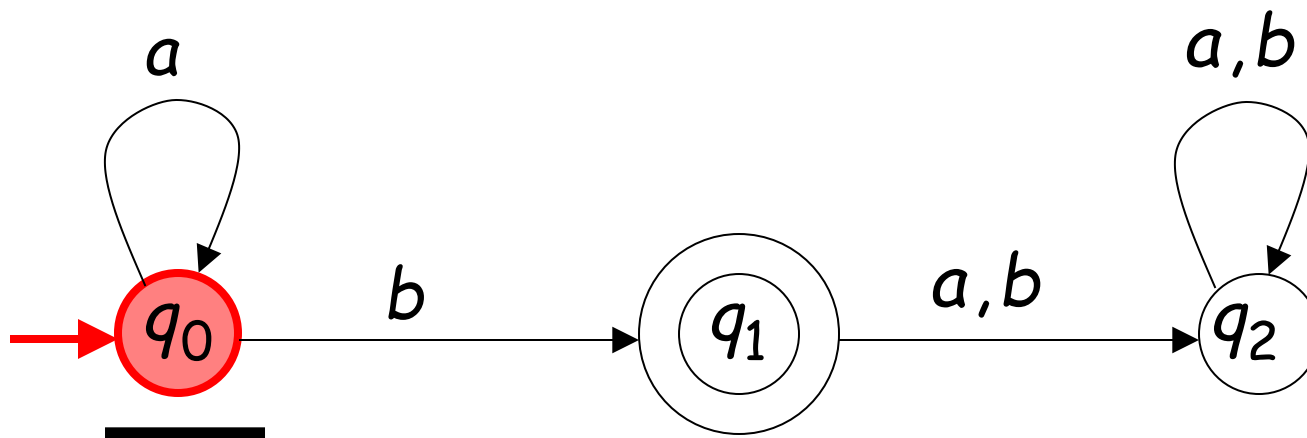


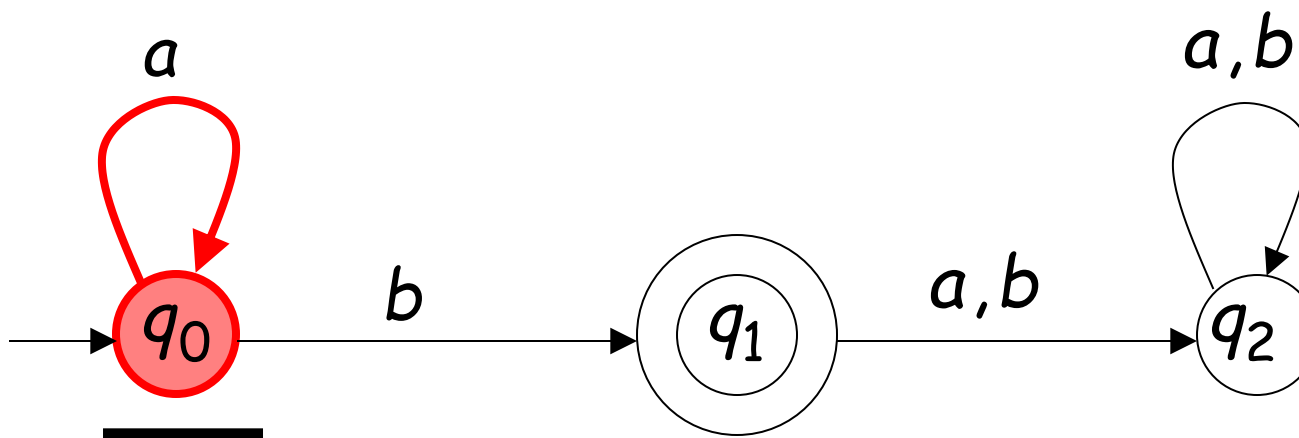
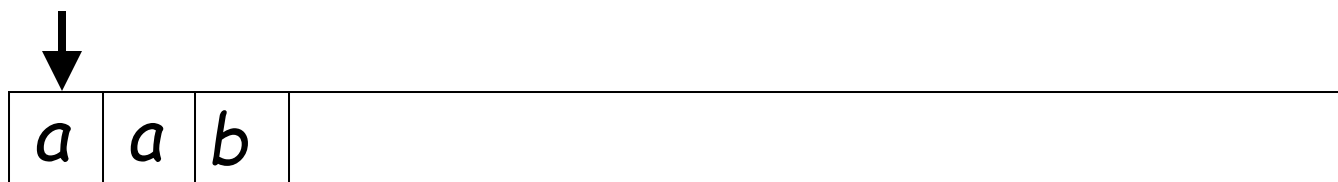
# Další případ

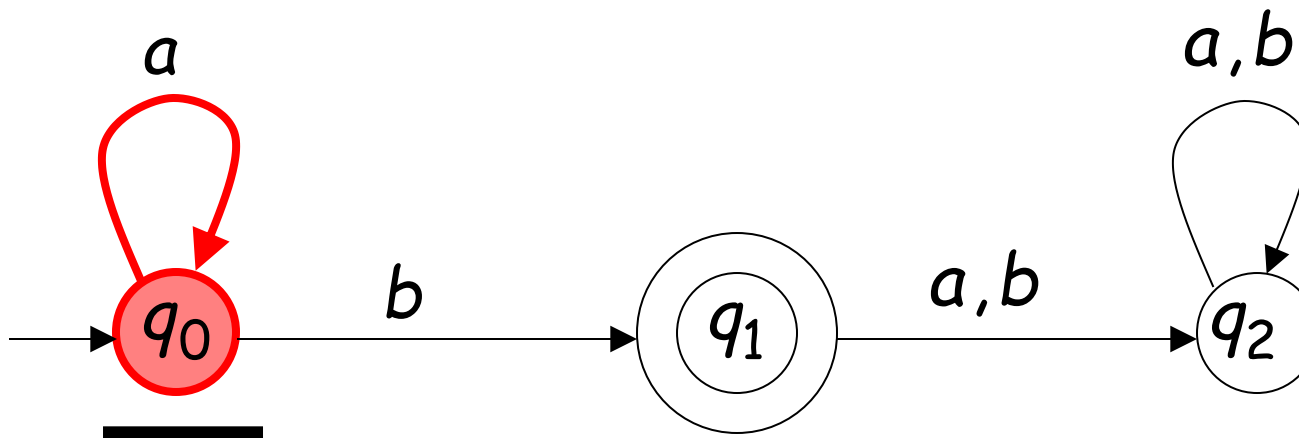
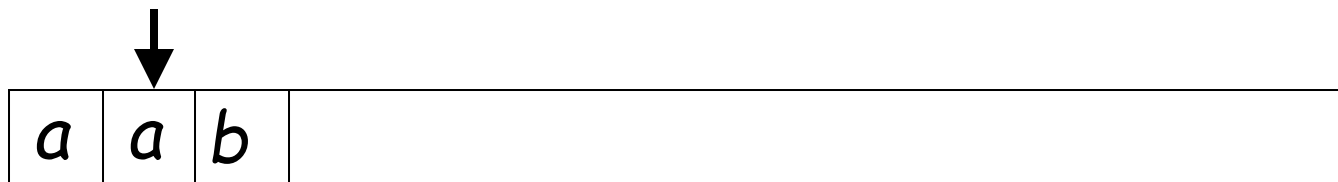




Vstupní řetězec

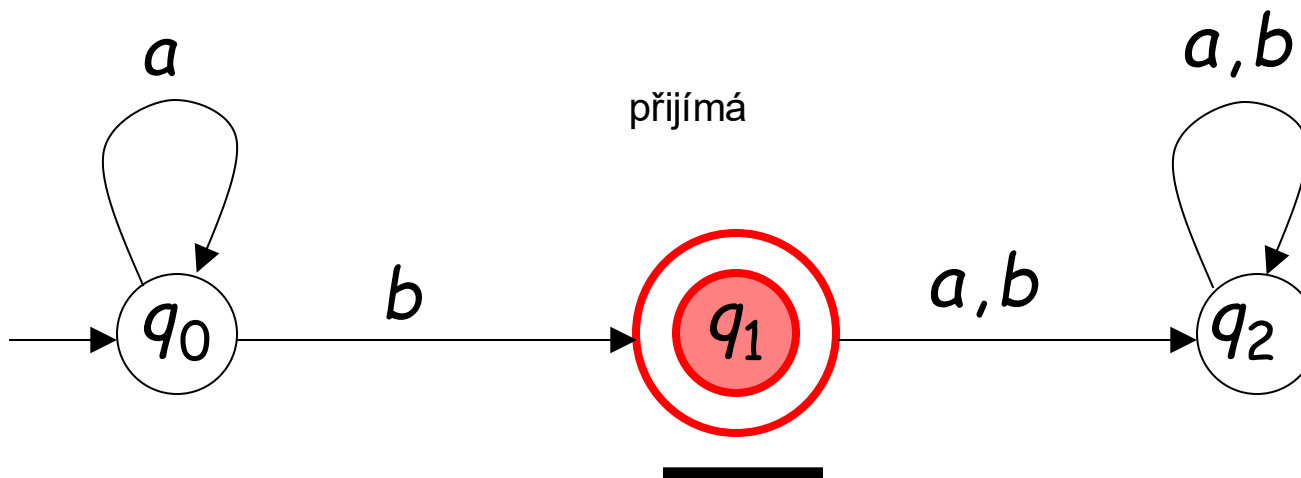
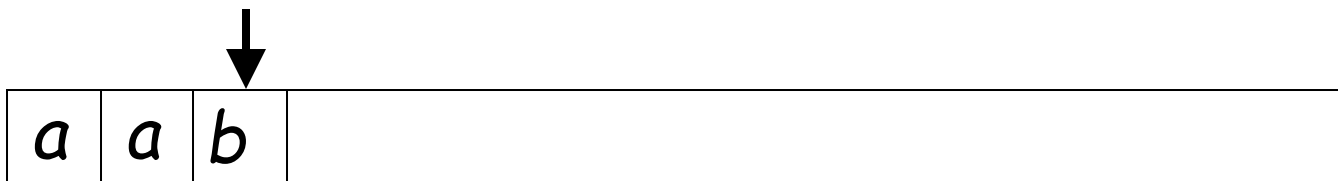




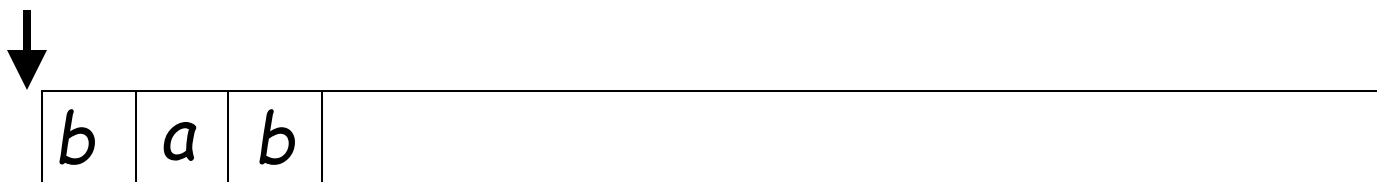




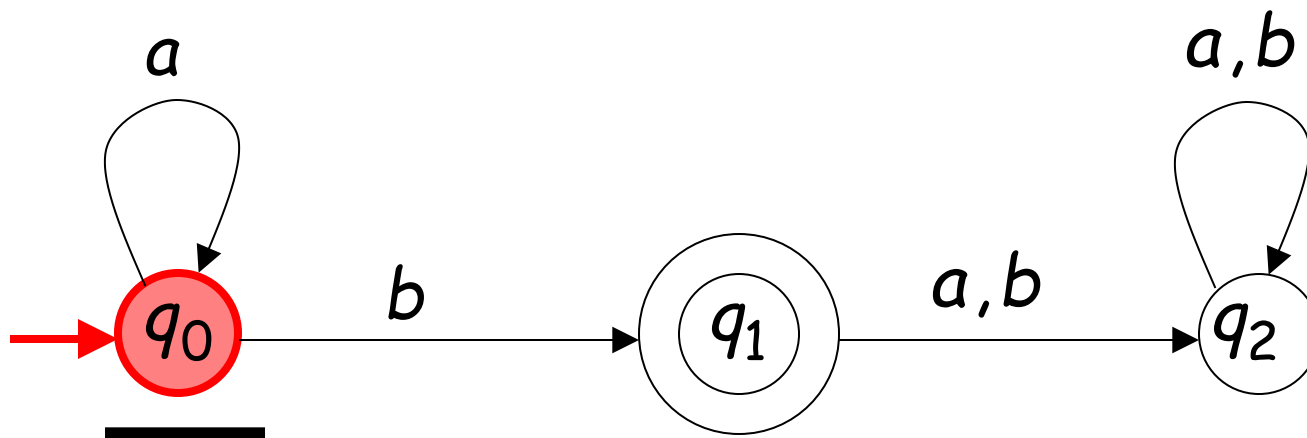
vstup dokončen

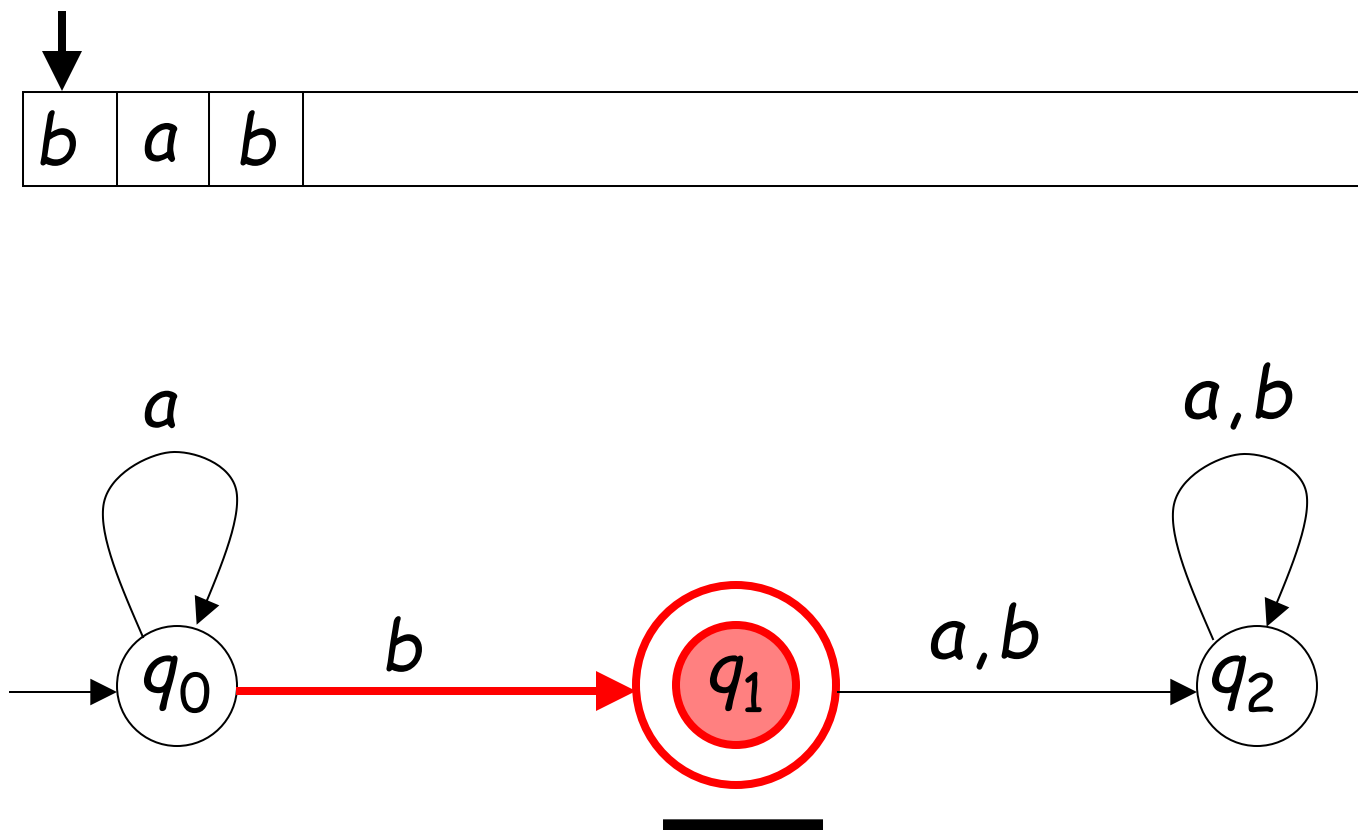


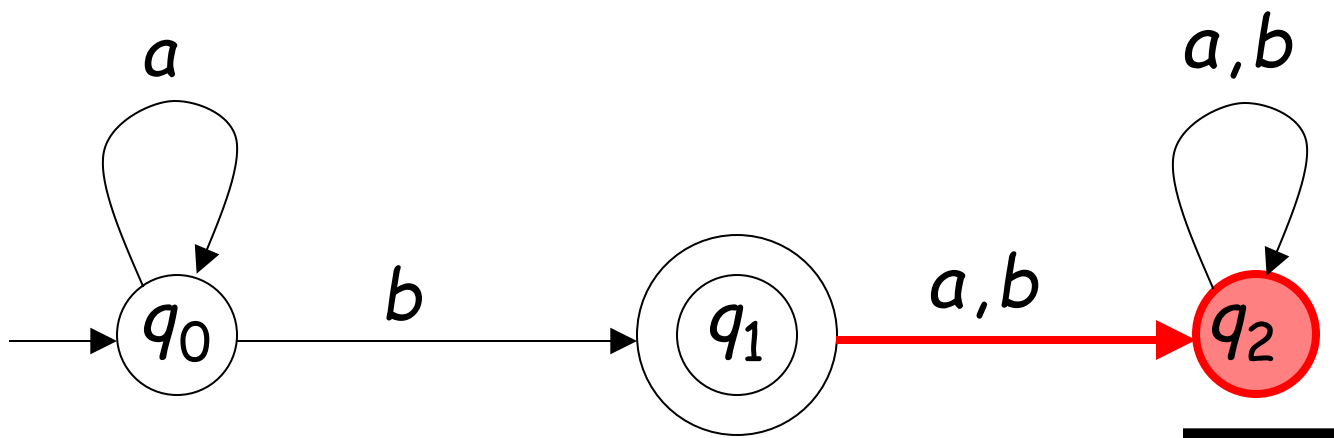
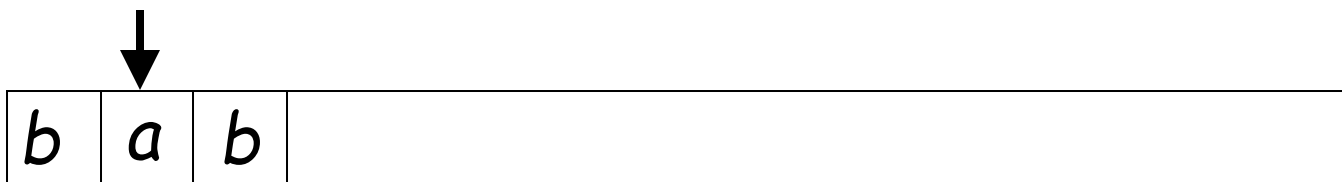
## A rejection case



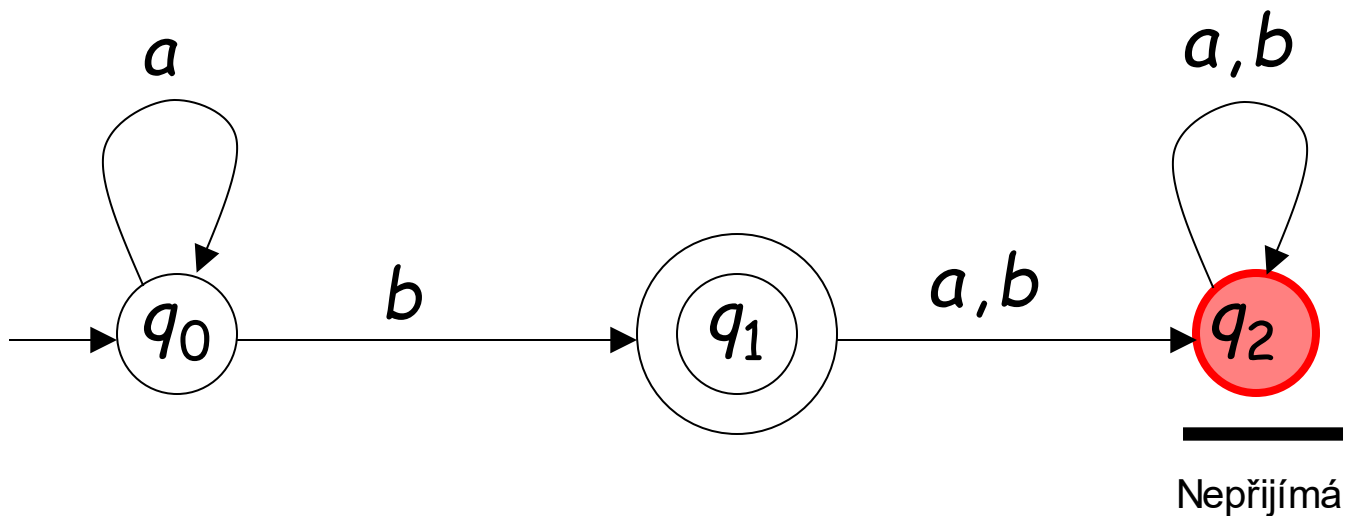
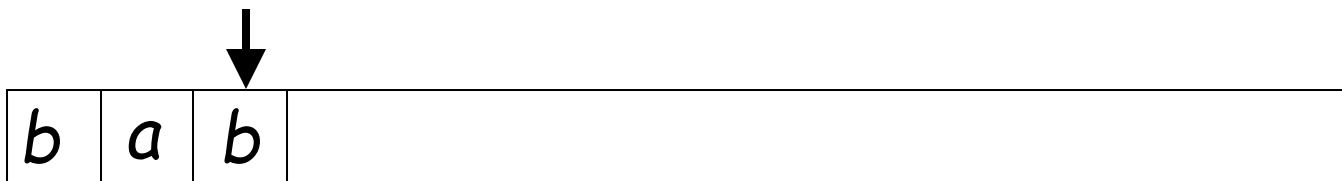
Vstupní řetězec





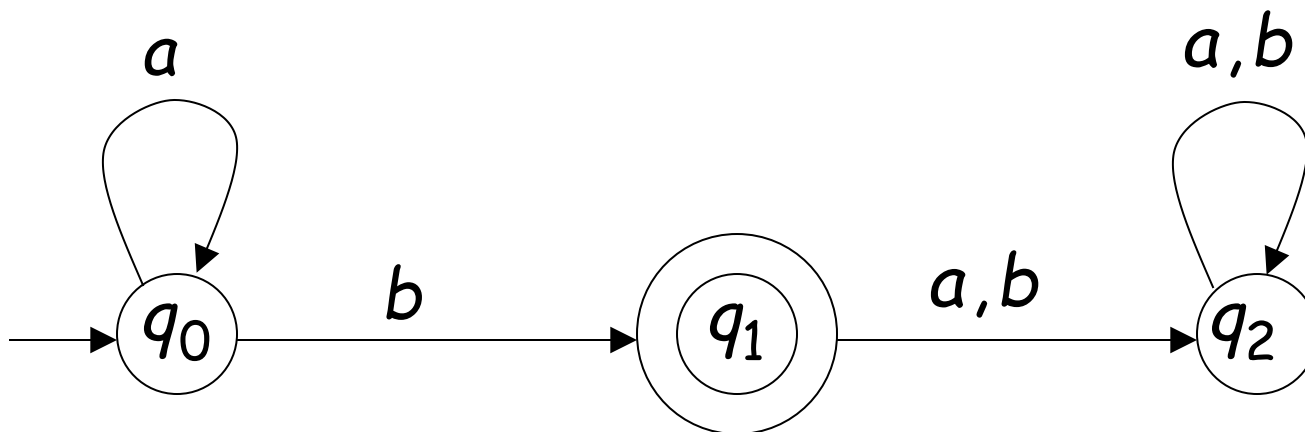


Vstup dokončen



Jazyk přijat:

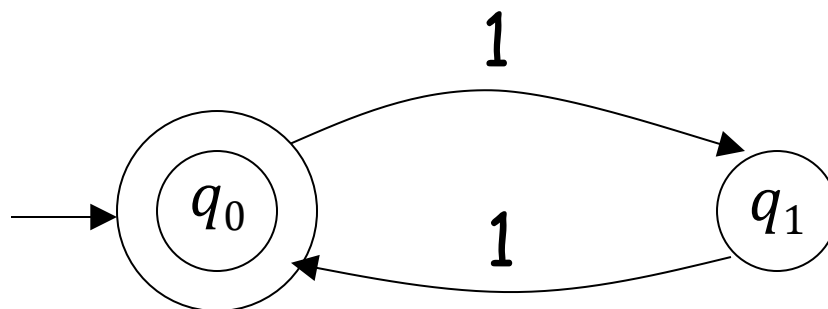
$$L = \{a^n b : n \geq 0\}$$



# Další příklad

Abeceda:

$$\Sigma = \{1\}$$



Přijímaný jazyk:

$$\text{SUDÝ POČET JDNIČEK} = \{x: x \in \Sigma^* \text{ a délka je sudá}\}$$

$$= \{\Delta, 11, 1111, 111111, \dots\}$$

# Formální definice

- Deterministický konečný automat (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

$Q$  : množina vnitřních stavů

$\Sigma$  : vstupní abeceda  $\Delta \notin \Sigma$

$\delta$  : přechodová funkce

$q_0$  : počáteční stav

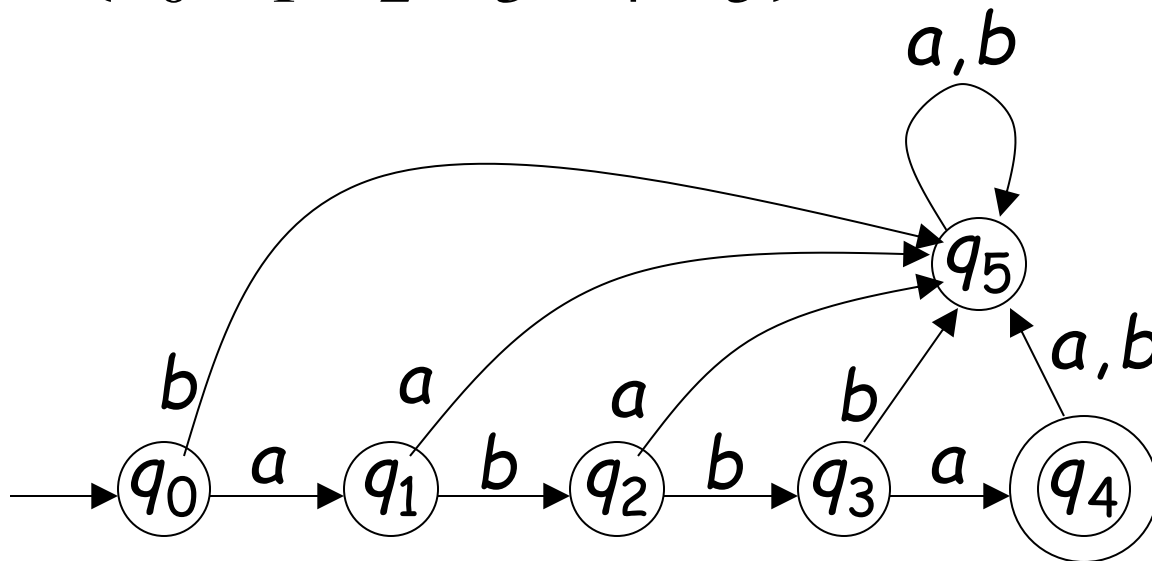
$F$  : množina konečných stavů



# Množina stavů $Q$

- Příklad:

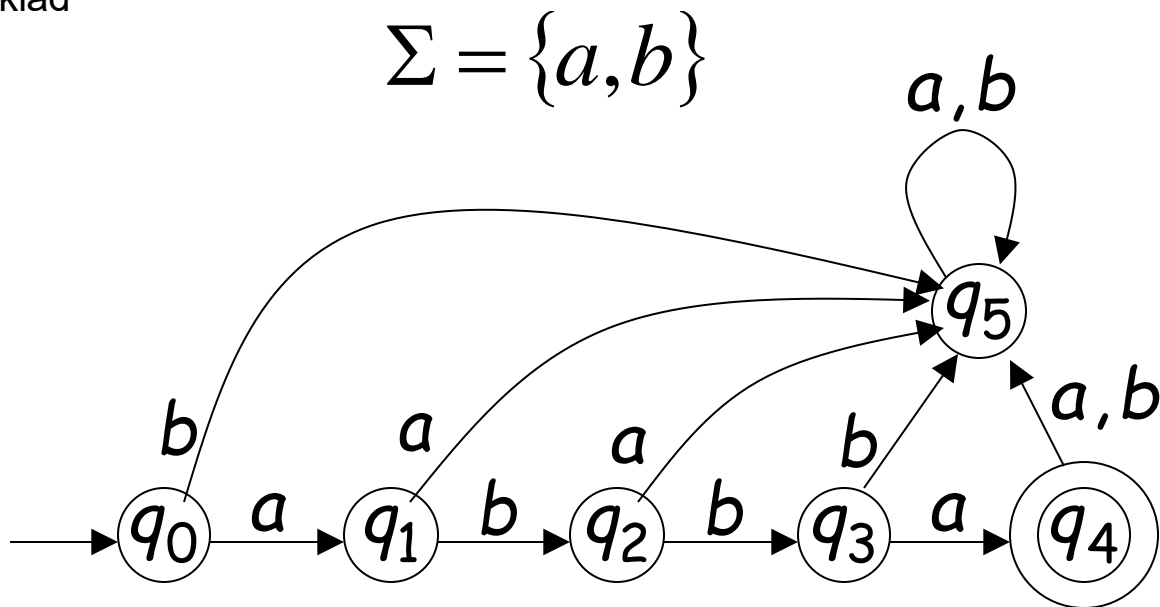
$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$



# Vstupní abeceda $\Sigma$

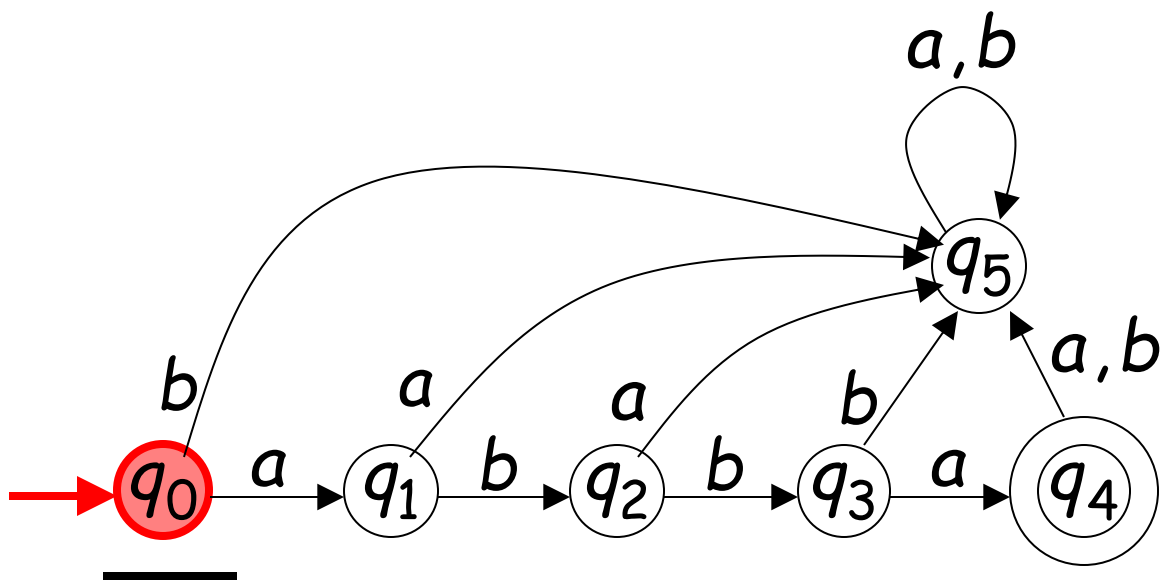
- $\Delta \notin \Sigma$  : vstupní abeceda nikdy neobsahuje prázdný znak  $\Delta$

Příklad



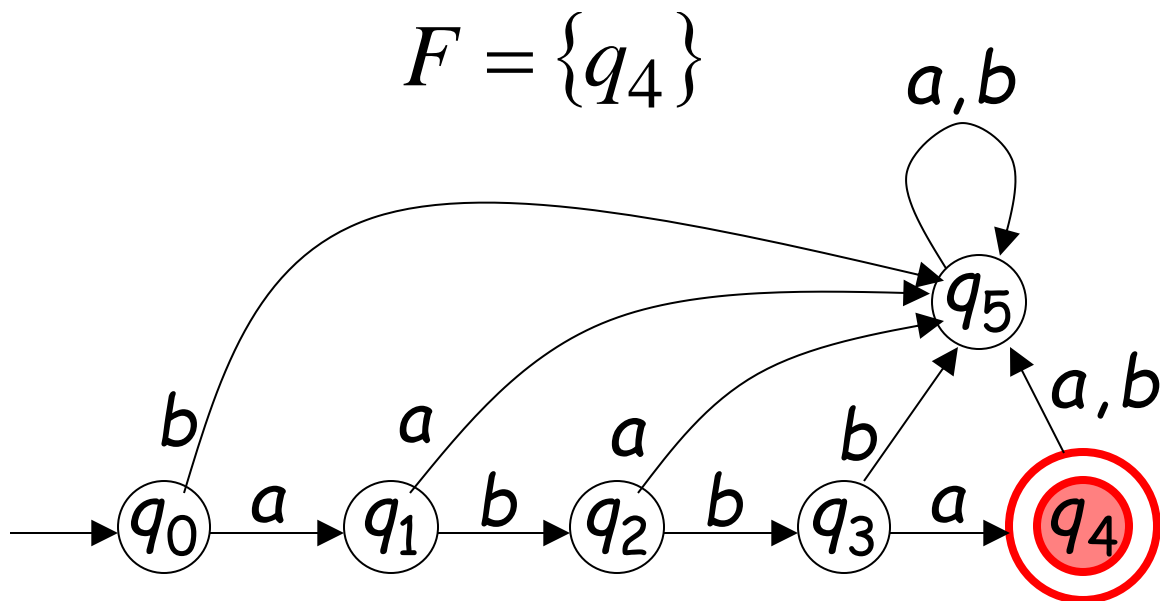
# Počáteční stav $q_0$

Příklad



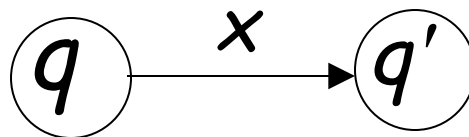
# Množina konečných stavů $F \subseteq Q$

Příklad



**Přechodová funkce**  $\delta : Q \times \Sigma \rightarrow Q$

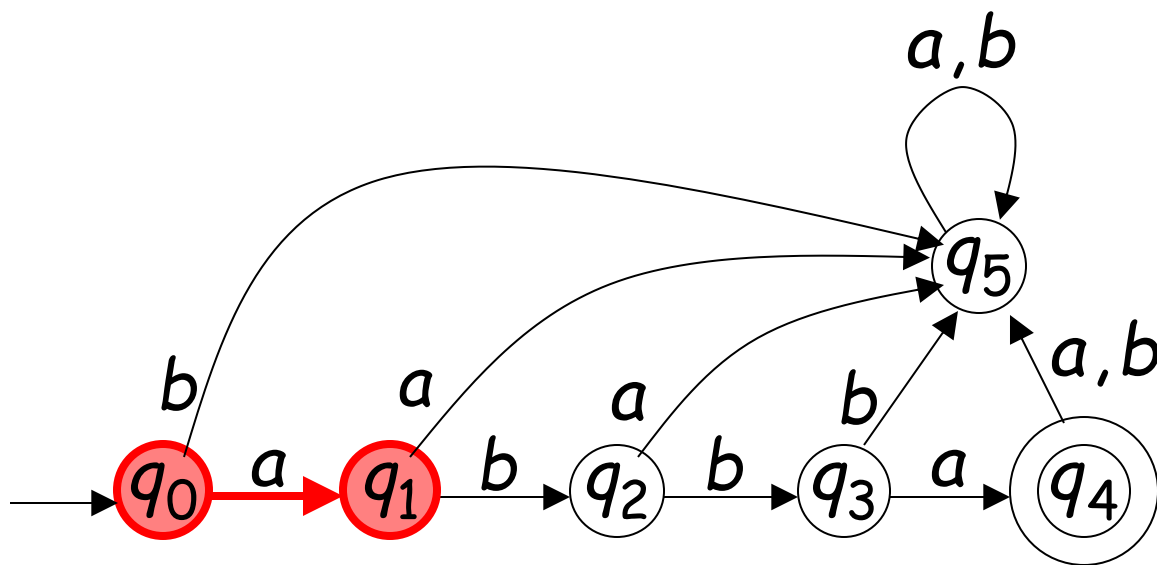
$$\delta(q, x) = q'$$



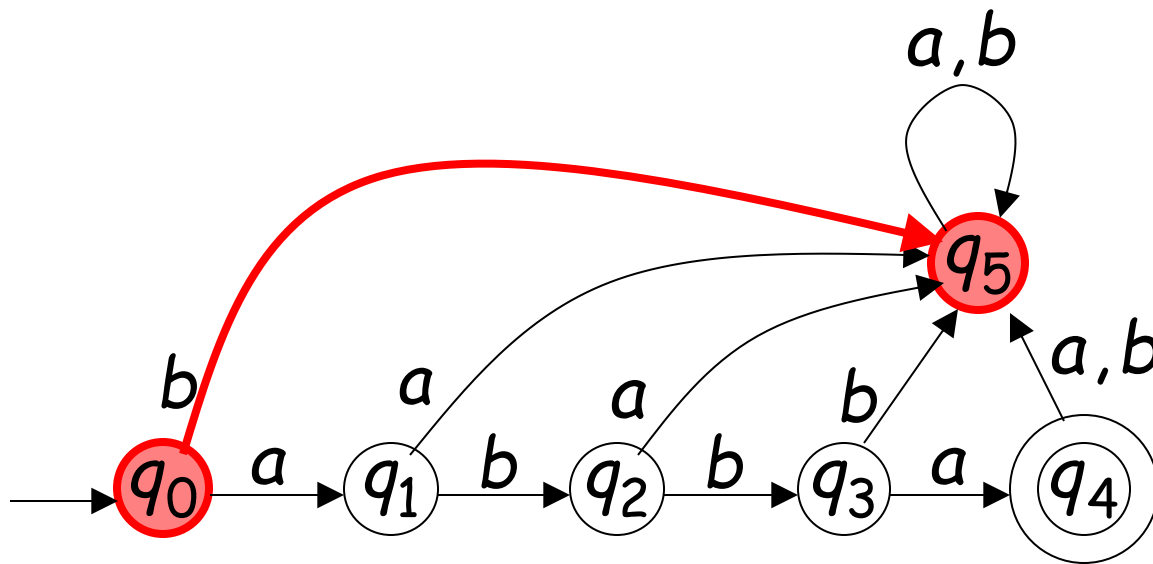
Vyjadřuje výsledek přechodu ze stavu  $q$ , kde na pásce je symbol  $x$

# Příklad

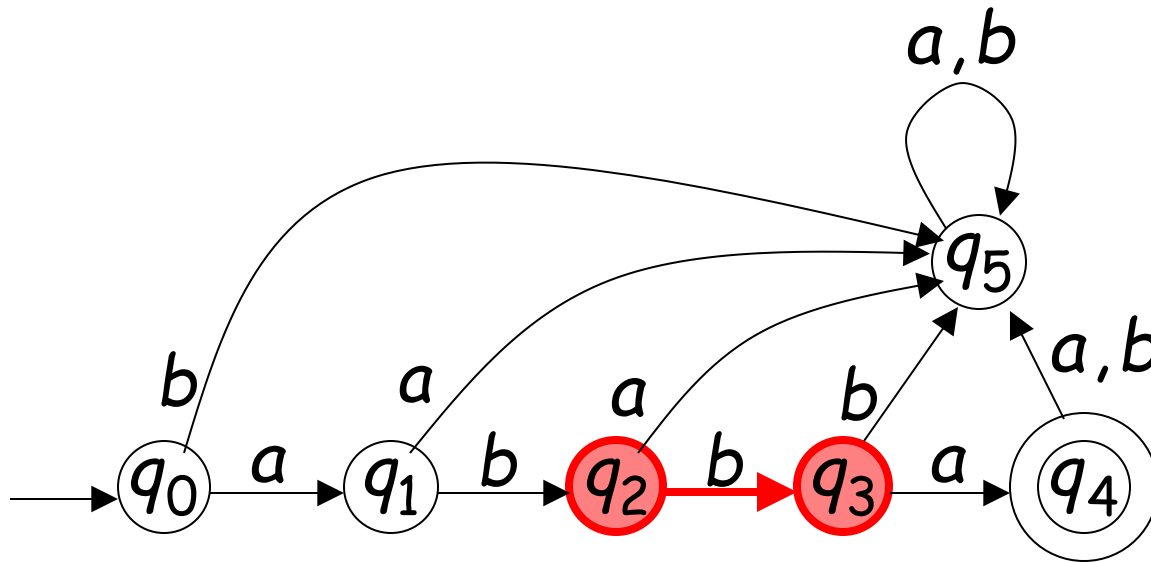
$$\delta(q_0, a) = q_1$$



$$\delta(q_0, b) = q_5$$



$$\delta(q_2, b) = q_3$$

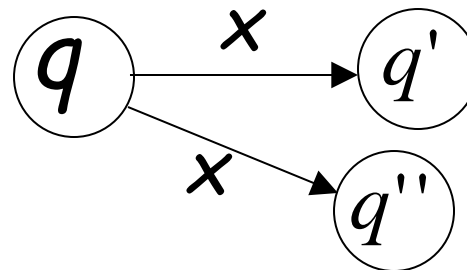




# Nedeterministický konečný automat

- Vše jako deterministický, jen přechodová funkce je definována jako:  $\delta : Q \times \Sigma \rightarrow 2^Q$

$$\delta(q, x) = q'$$

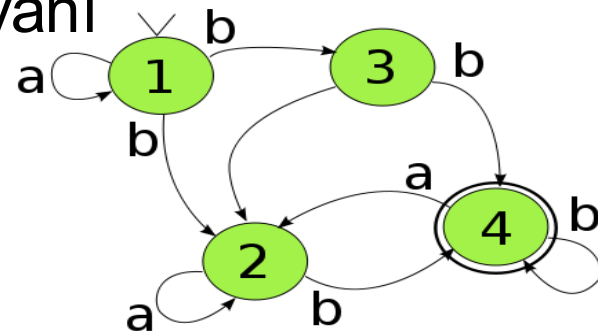


Vyjadřuje výsledek přechodu ze stavu  $q$ , kde na pásce je symbol  $x$

Zatímco v předchozím případě existovala vždy nejvýše jedna možnost, Nedeterministický automat umožňuje více variant

# Příklad

- Inicializace ve stavu 1
- Konečný stav: 4 (tučný okraj)
- Rozhoduje, zdali přijímá či nepřijímá vstupní řetězec :
  - Př. 1: aabaaaba
  - Př. 2: aaaaaaab
- Ne-deterministický = na základě konkrétního vstupu může nastat více než jedna možnost pokračování



# Deterministický vs. nedeterministický

- Jak rozumět pojmu "nedeterministický":
  - je teoretický výpočetní model, který má schopnost ve všech variantách nalézt jedno správné řešení
  - Schopnost vytvářet nekonečné množství paralelních procesů

# Deterministický zásobníkový automat (PDA)

$Q$ : konečná množina stavů

$\Sigma$ : konečná množina vstupních symbolů

$\Gamma$ : konečná zásobníková abeceda

$\delta$ : přechodová funkce, kde:

$q_i$  je stav z množiny  $Q$ .

$a$  je symbol z množiny  $\Sigma$  nebo  $a = \varepsilon$  (the empty string).

$\tau_m$  je zásobníkový symbol,  $\tau_m \in \Gamma$ .  $P = (Q, \Sigma, \Gamma, \delta(q_i, a, \tau_m) \rightarrow \{(q_k, \tau_n), \dots\}, q_0, \tau_0, F)$

a výstup je konečná množina dvojic:

$q_k$  nový stav.

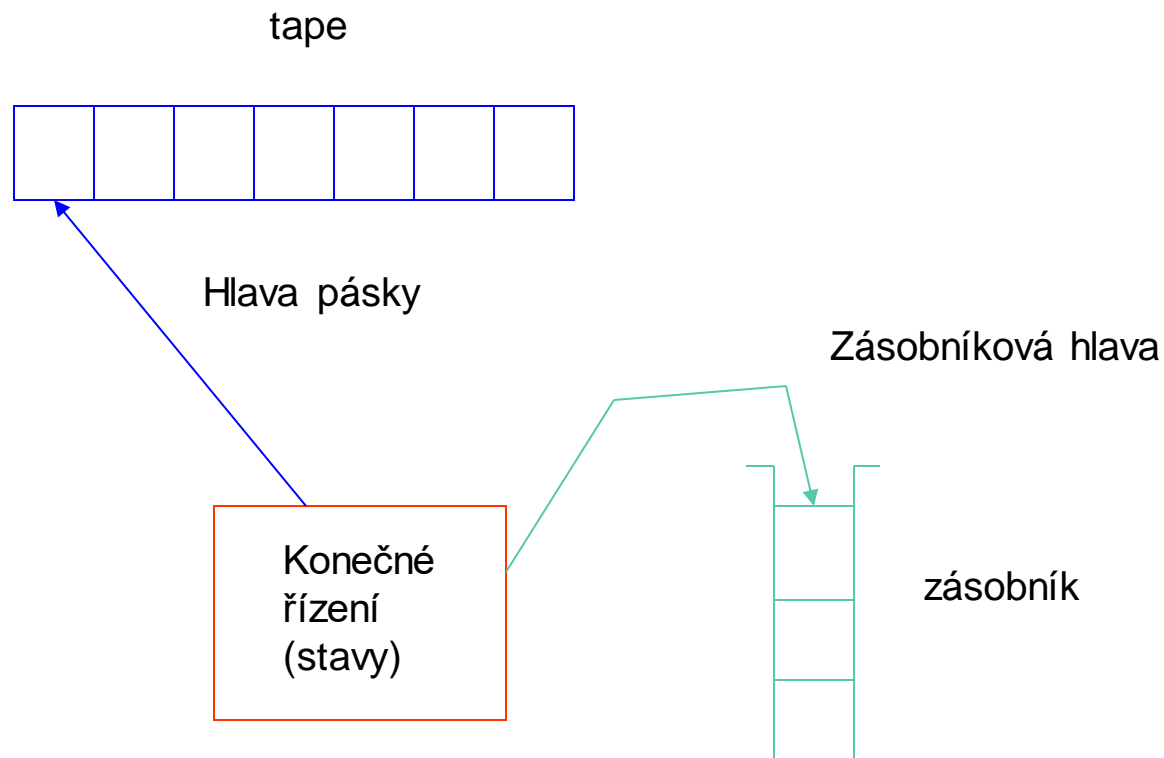
$\tau_n$  symbol, který nahrazuje symbol  $\tau_m$  na vrcholu zásobníku.

Pokud  $\tau_n = \Delta$ , potom poslední symbol je ze zásobníku odstraněn

$q_0$ : počáteční stav.

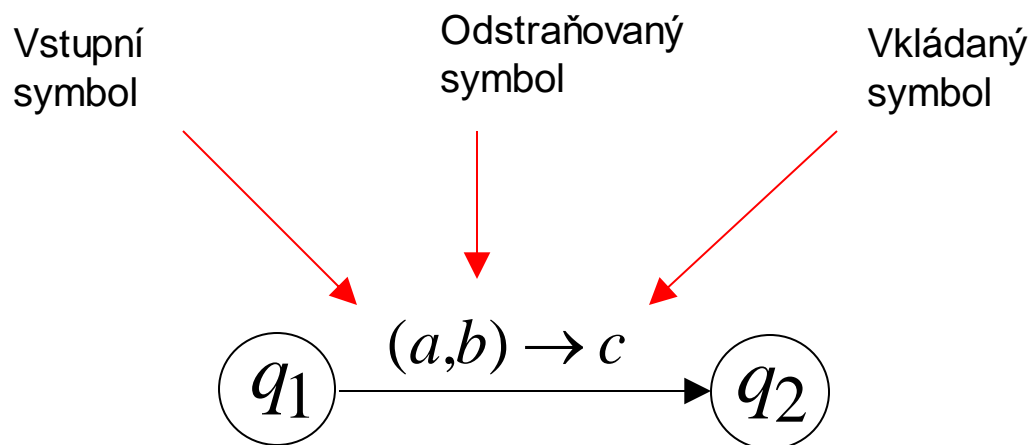
$\tau_0$ : Původní stav zásobníku, PDA obsahuje na počátku pouze tento symbol (\$).

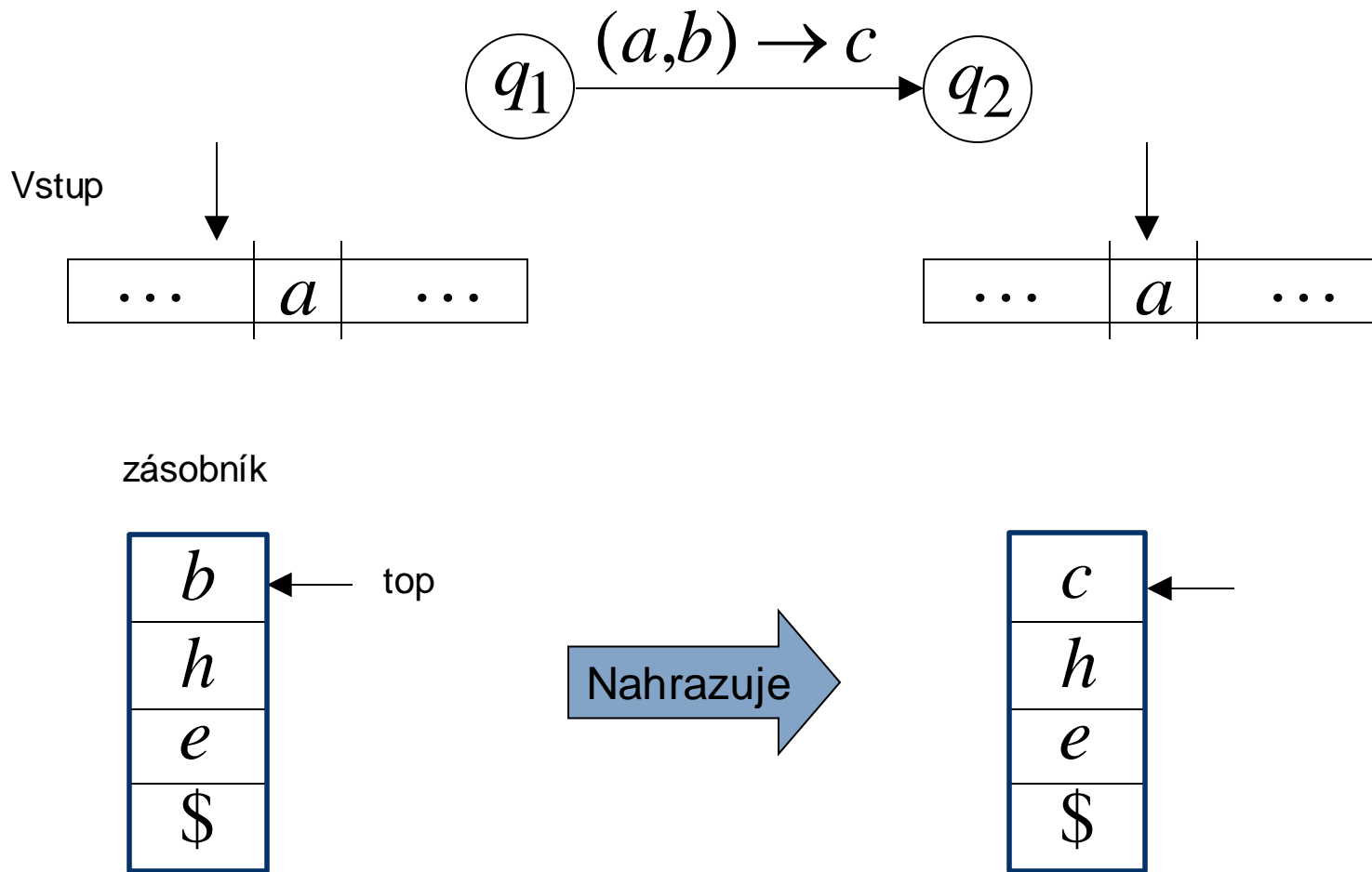
$F$ : Množina konečných stavů.

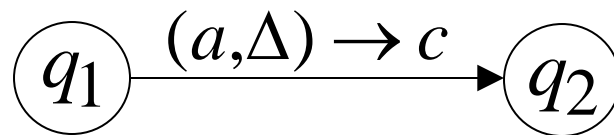


# Přechodová funkce

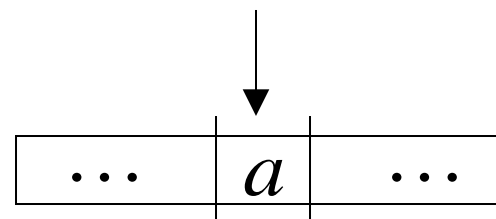
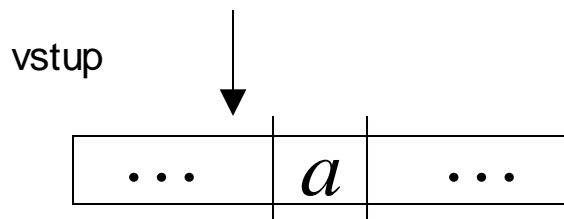
- Vše velmi podobné deterministickému kon. automatu
- Navíc ještě zásobník a pozměněná přechodová funkce



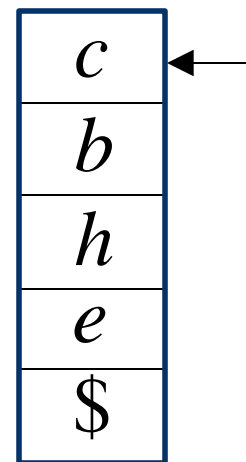
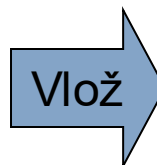
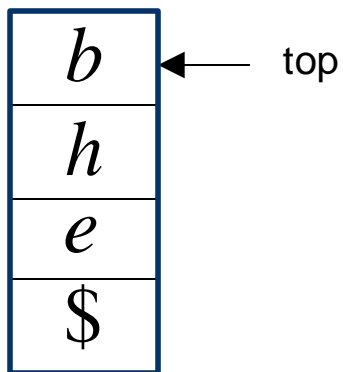




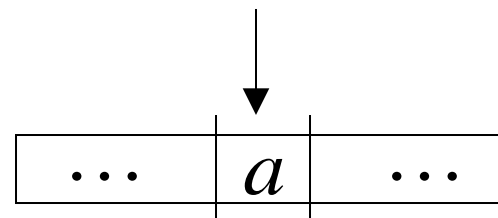
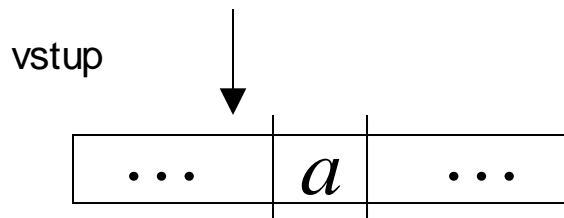
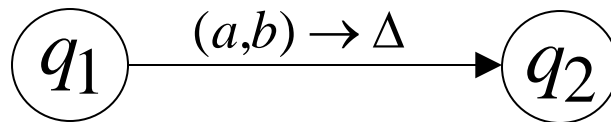
$\Delta$  ... prázdný symbol



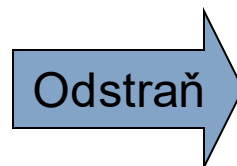
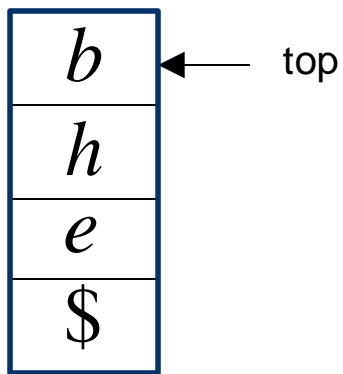
zásobník



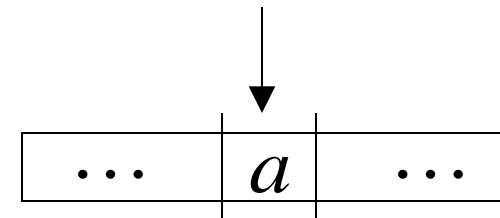
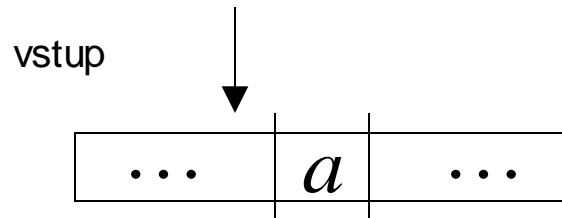
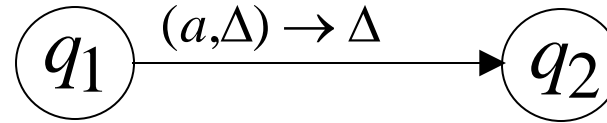




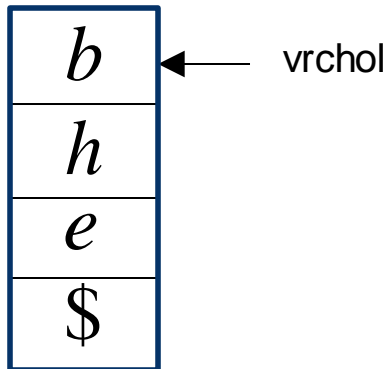
zásobník



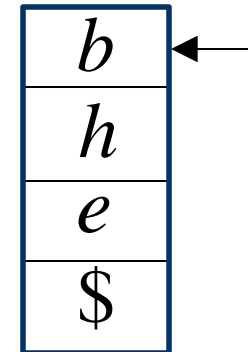
Odstraň



zásobník

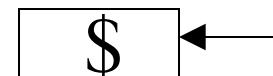
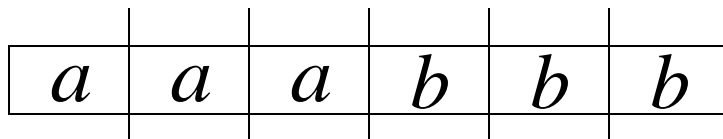
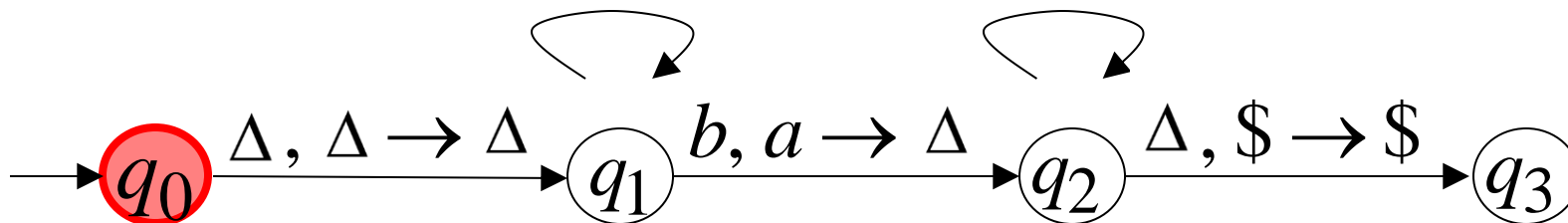


Beze změny



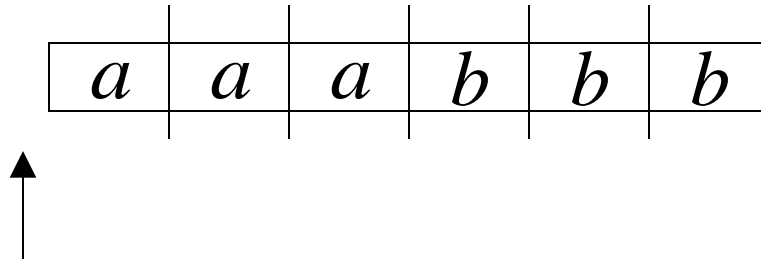
Čas 0

vstup

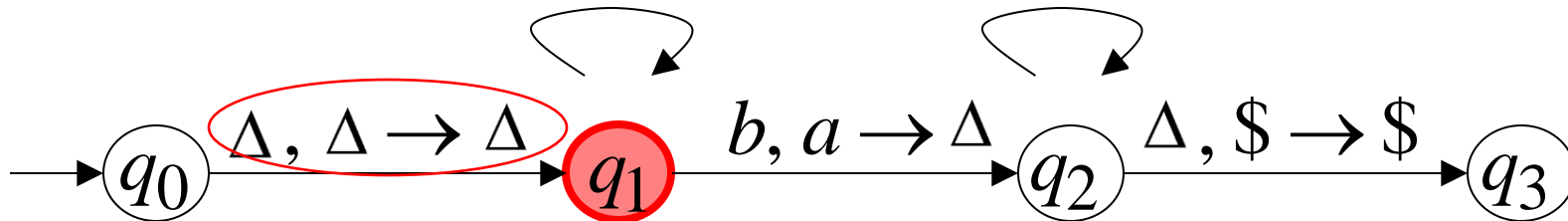
Zásobník  
(prázdný)Aktuální  
stav $a, \Delta \rightarrow a$  $b, a \rightarrow \Delta$ 

vstup

Čas 1

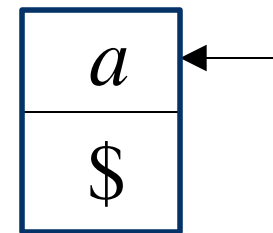
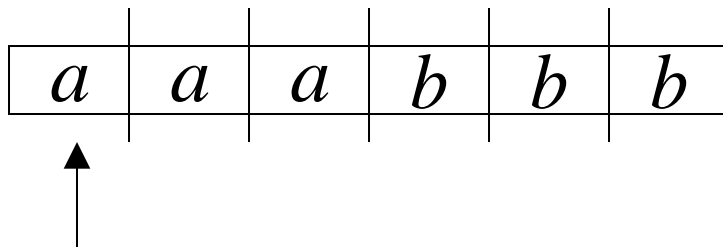


Zásobník

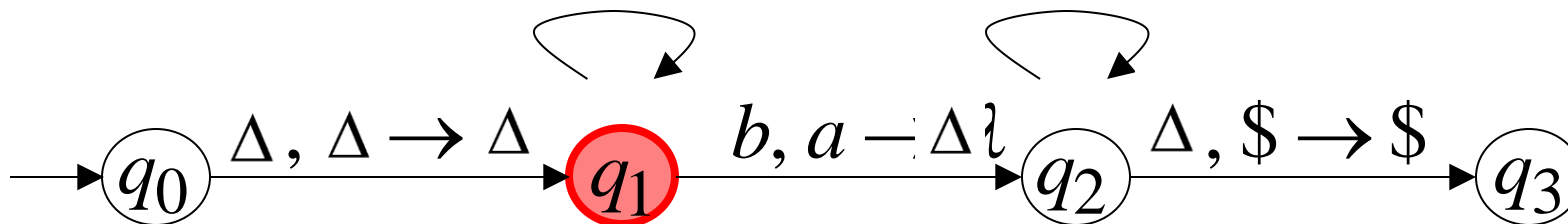
 $a, \Delta \rightarrow a$  $b, a \rightarrow \Delta$ 

Čas 2

vstup

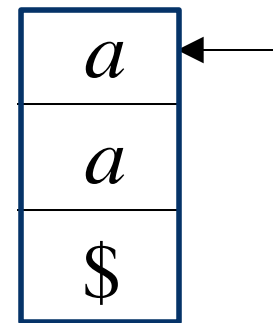
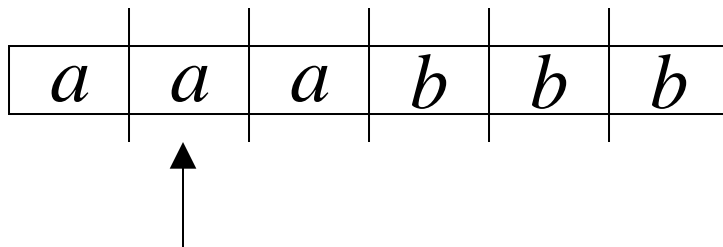


Zásobník

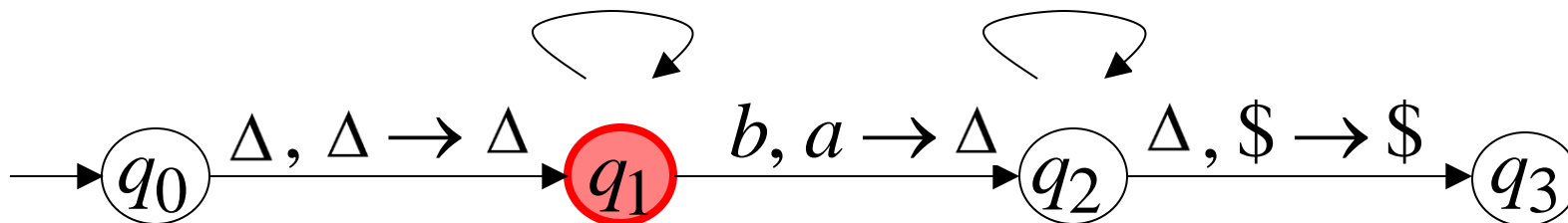
 $a, \Delta \rightarrow a$ 
 $b, a \rightarrow \Delta$ 


vstup

Čas 3

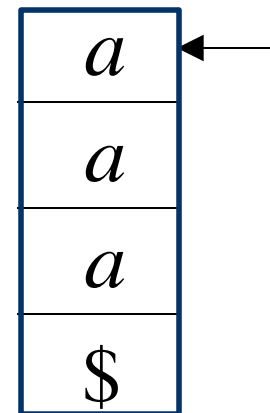
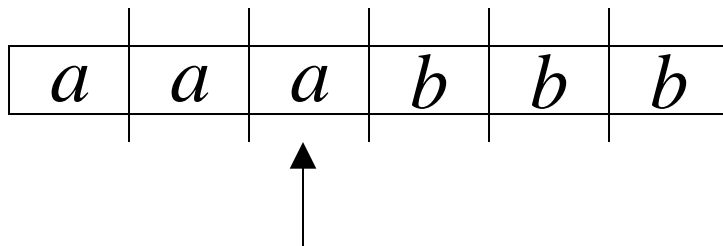


Zásobník

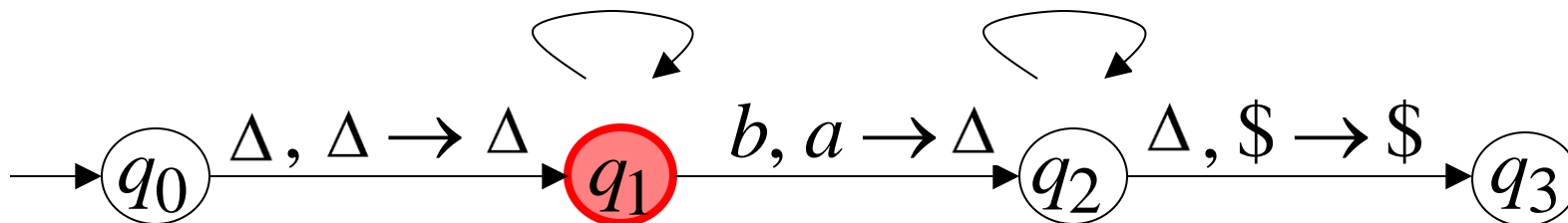
 $a, \Delta \rightarrow a$ 
 $b, a \rightarrow \Delta$ 


vstup

Čas 4

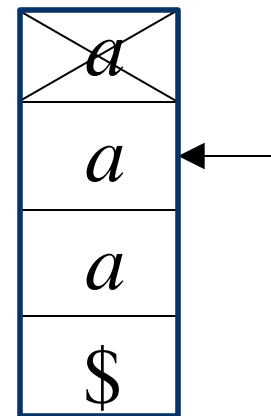
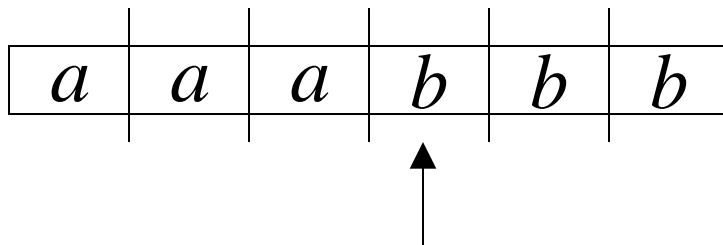


Zásobník

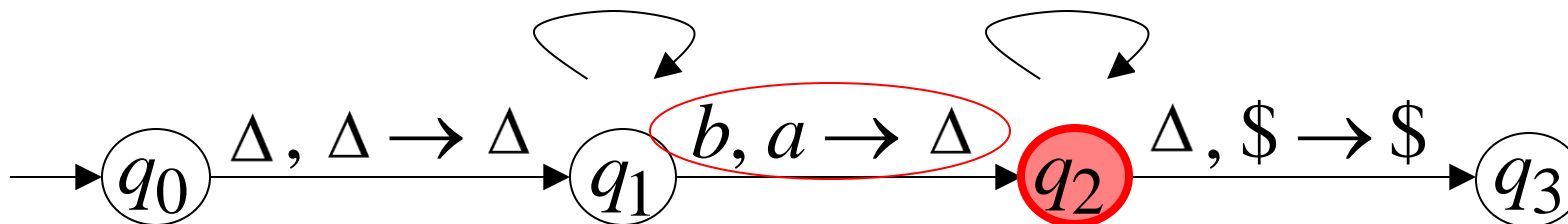
 $a, \Delta \rightarrow a$ 
 $b, a \rightarrow \Delta$ 


vstup

Čas 5



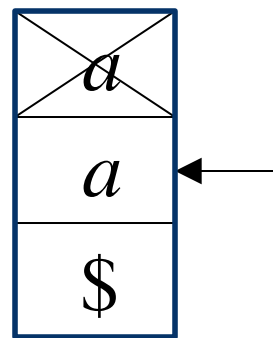
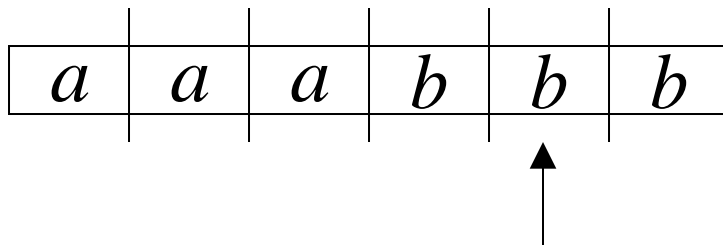
Zásobník

 $a, \Delta \rightarrow a$  $b, a \rightarrow \Delta$ 

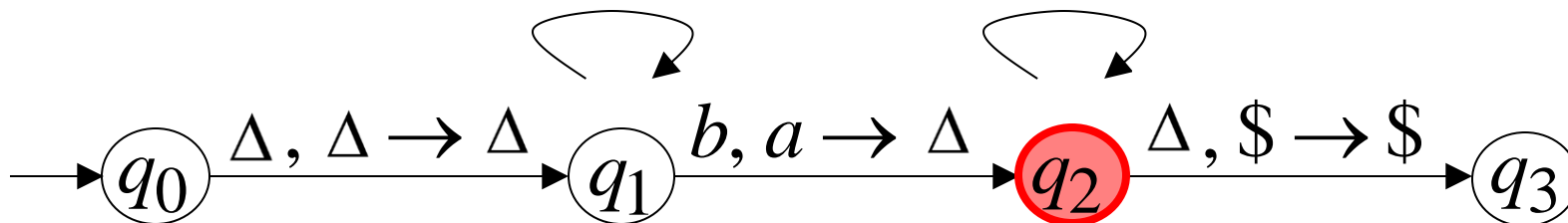


vstup

Čas 6

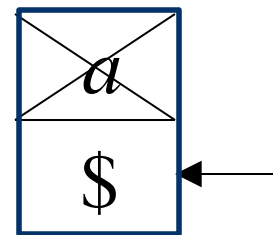
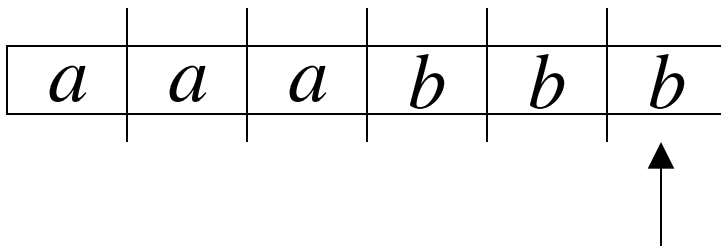


Zásobník

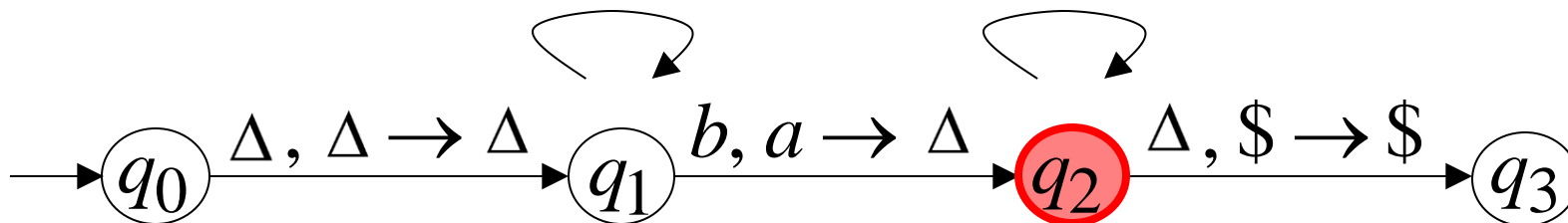
 $a, \Delta \rightarrow a$  $b, a \rightarrow \Delta$ 

Čas 7

vstup

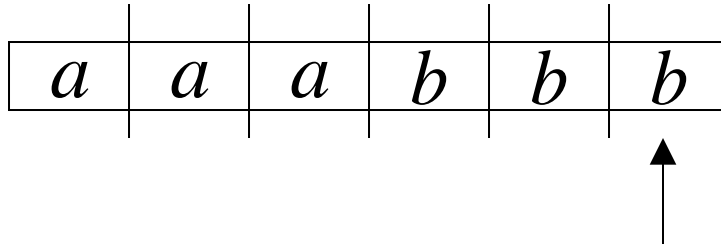


Zásobník

 $a, \Delta \rightarrow a$  $b, a \rightarrow \Delta$ 

Čas 8

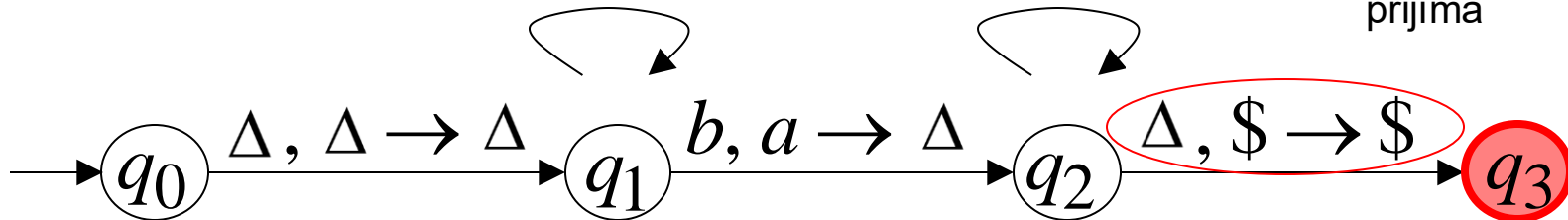
vstup



Zásobník

 $a, \Delta \rightarrow a$  $b, a \rightarrow \Delta$ 

přijímá



# Odmítnutí řetězce

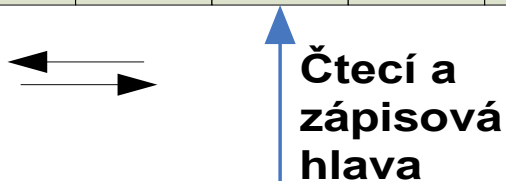
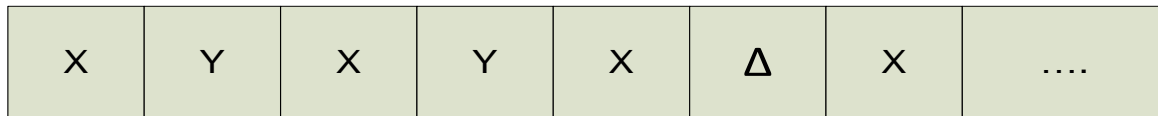
- Pokud automat odstraní poslední symbol  $\tau_0$  ze zásobníku, potom se zastaví
- Zásobníkový automat přijímá vstupní řetězec pokud:
  - Vyprázdní zásobník, přečte veškeré vstupy vstupní pásky a skončí v konečném stavu (\*existuje více variant)

# Turingův stroj

- Vědní obor - Vyčíslitelnost
  - Zkoumá zdali lze spočítat (vyčíslit)
  - Nezkoumá jak dlouho (tím se zabývá vědní obor spočítatelnosti)
- Turingův stroj - matematický model výpočetního stroje
  - Vznikl za účelem zkoumání vyčíslitelnosti – co lze a co nelze (nemá cenu se pokoušet) řešit
  - Jednoduchý princip – nehledat v tom nic složitého
  - Několik základních operací
  - Vychází z teorie automatů (deterministický konečný automat, nedeterministický konečný automat, zásobníkový automat)
  - Pojem „Turingovsky úplný“ – vyjadřovací síla je ekvivalentní Turingově stroji

# Turingův stroj

**Páska**



**Čtecí a  
zápisová  
hlava**

**Stavové řízení**



Operace čtení hlavy, zápis na pozici hlavy, provádění příkazů

Pro posun hlavy navíc dva speciální symboly: R (doprava), L (doleva)

# Turingův stroj – definice

- Turingův stroj  $T$  je definován jako uspořádaná 6-ice  $T=(Q, \Sigma, \Gamma, \delta, q_0, q_F)$ , kde

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_F) \quad (9)$$

- $Q$  reprezentuje konečnou množinu vnitřních stavů,
- $\Sigma$  je konečná množina páskové abecedy, nazývaná **vstupní abeceda**.  $\Delta$  (prázdný symbol) není součástí  $\Sigma$  (je ale možné  $\Delta$  do množiny explicitně vložit)
- $\Gamma$  je konečná množina vnitřních symbolů. Je to tzv. **pásková abeceda** a platí pro ni  $\Gamma \subset \Sigma$ ,  $\Delta \in \Gamma$
- $\delta$  je parciální funkce a popisuje chování Turingova stroje, jinak řečeno program. V tomto případě nelze měnit program za běhu počítače, tak jak jsme zvyklí z dnešních výpočetních strojů. Je nazývána **přechodová funkce** a definována je jako

$$\delta: (Q \setminus \{q_F\}) \times \Gamma \rightarrow Q \times (\Gamma \cup \{L, R\}) \quad (10)$$

Kde  $\{L, R\} \notin \Gamma$ , a  $L$  je symbol pro přesun hlavy o jednu pozici doleva a  $R$  je symbol pro přesun hlavy o jednu pozici vpravo.

- $q_0$  je počáteční stav  $q_0 \in Q$  a
- $q_F$  je konečný stav  $q_F \in Q$ .

# Turingův stroj – příklad



$$\begin{aligned}Q &= \{q_1, q_2\} \\ \Sigma &= \{X, Y, \Delta\} \\ \Gamma &= \{F, X, Y, \Delta\} \\ \delta: \{q_1 Y \rightarrow q_1 R, q_1 X \rightarrow q_1 F\} \\ q_0 &= \{q_1\} \\ q_F &= \{q_2\}\end{aligned}$$

- 1) Přijímá Y
- 2) Dle programu  $\delta$  se ze stavu  $q_1$  a aktuálním symbolem Y na pásce dostává opět do stavu  $q_1$  a posouvá hlavu vpravo (R)
- 3) Dle programu  $\delta$  se ze stavu  $q_1$  a aktuálním symbolem X na hlavě dostává do stavu  $q_2$  a na hlavu zapisuje symbol F



# Turingův stroj – varianty

- **Turingový stroj** – nelze měnit program
- **Univerzální Turingový stroj** (lze jeho program přepsat = klasické PC, smartphone)
- **Nedeterministický**

$$\delta: (Q \setminus \{q_F\}) \times \Gamma \rightarrow Q \times 2^{(\Gamma \cup \{L,R\})} \quad (11)$$

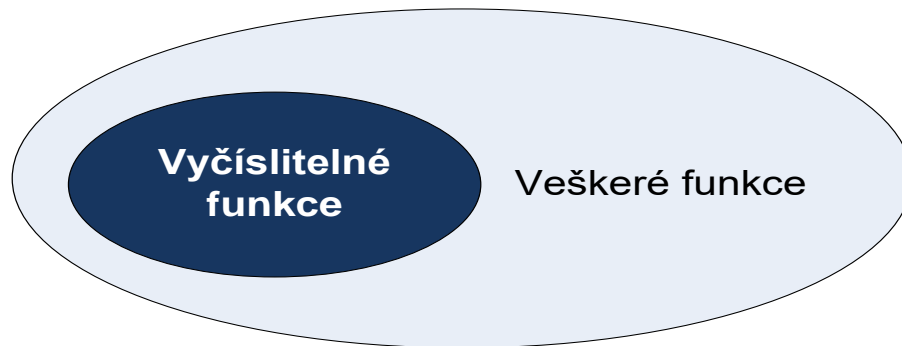
- Nedeterministický nebyl doposud sestrojen
- Případné sestrojení by mělo za následek konec veškeré dnes používané asymetrické kryptografie
- Neví se, zdali to vůbec je možné (tzv. problém P vs. NP)

# Turingův stroj – varianty

- **Paralelní** – z pohledu vyčísitelnosti je ekvivalentní s běžným (přináší jen vyšší výkon)
  - Lze jej simulovat pomocí klasického TS
- **Kvantový Turingův stroj**
  - Založený na superpozici stavů
  - Dokáže řešit „exponenciální explozi“ a převést ji na problém se složitostí v polynomiálním čase
  - Ne všechny problémy lze takto řešit

# Church-Turingova teze

- Hypotéza o povaze a výpočetní síle mechanických strojů
- **Church-Turingova teze říká, že každý algoritmus [1] může být vykonán Turingovým strojem.**
- Existuje-li tedy rozhodovací problém, pro jehož řešení neexistuje program například v jazyce JAVA, potom je tento problém algoritmicky neřešitelný a naopak.



# Nevyčísitelné problémy

- Zavedení pojmu:
- **Rozhodovací problém**
  - **Vstup programu** – jeho kódování můžeme interpretovat jako celé číslo (možná velmi velké)
  - **Řešení** – výstup programu, můžeme kódovat 1=ano, 0=ne
  - Rozhodovací problém je potom funkce  $f: \mathbb{N} \rightarrow \{0, 1\}$ , na základě vstupního čísla odpovídá ANO, či NE na základě konkrétního vstupu.
- Př.: Je vstupní přirozené číslo liché?

| Vstup: $n \in \mathbb{N}$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---------------------------|---|---|---|---|---|---|---|---|---|
| Výstup: $f(n)$            | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

# Nevyčíslitelné problémy – pokračování

- Příklady algoritmů řešící předchozí příklad:

```
int jeLiche1(int n) {  
    return n%2;  
}
```

```
int jeLiche2(int n) {  
    for(; n > 1; i-=2):  
        return n;  
}
```

- Důkaz nevyčíslitelného problému - diagonalizace

# Nevyčíslitelné problémy – pokračování

- Označme  $P$  jako množinu všech rozhodovacích problémů
  - Tzn. patří sem i **jeLiche1()** a **jeLiche2()**

| Algoritmy \ Vstup | 1        | 2        | 3        | ... | k           | ... |
|-------------------|----------|----------|----------|-----|-------------|-----|
| $P_1$             | $P_1(1)$ | $P_1(2)$ | $P_1(3)$ | ... | $P_1(k)$    |     |
| $P_2$             | $P_2(1)$ | $P_2(2)$ | $P_3(3)$ | ... | $P_1(k)$    |     |
| ...               | ...      | ...      | ...      | ... | ...         | ... |
| <b>jeLiche1()</b> | 1        | 0        | 1        |     | jeLiche1(k) |     |
| ...               | ...      | ...      | ...      | ... | ...         | ... |
| <b>jeLiche2()</b> | 1        | 0        | 1        |     | jeLiche2(k) |     |
| ...               | ...      | ...      | ...      | ... | ...         | ... |
| $P_n$             | $P_n(1)$ | $P_n(2)$ | $P_n(3)$ | ... | $P_n(k)$    | ... |
| ...               | ...      | ...      | ...      | ... | ...         | ... |

# Nevyčísitelné problémy – důkaz diagonalizací

- Dejme tomu, že bychom nyní chtěli program, který má na výstupu opačnou hodnotu daného algoritmu, tedy:

$$\begin{array}{ll} D(i) &= 0 & \text{je-li } P_i(i) == 1; \\ &= 1 & \text{je-li } P_i(i) == 0; \end{array}$$

(Tj. postupovalo by diagonálně v předchozí tabulce)

- Jestliže množina  $P$  obsahuje všechny algoritmy, s  $D$  se liší minimálně v jednom řádku
- Což je spor (**Důkaz pomocí diagonalizace**)

# Nevyčísitelné problémy - pokračování

- **Existují problémy algoritmicky neřešitelné (nerozhodnutelné).**

- **Problém zastavení Turingova stroje:**
  - Sestrojte program, který rozhodne, zdali libovolný program (vstup programu) poběží navždy bez zastavení či nikoli.

**NELZE SESTROJIT**



# Složitost algoritmů

- Lze vše algoritmizovatelné spočítat? (kryptografie)



# Analýza algoritmů

- Proč analyzovat algoritmy?
- Umožňuje odhadovat rychlost daného kódu před jeho spuštěním (& vzájemně srovnávat)
- Předpovídání zdrojů, které algoritmus potřebuje:
  - Výpočetní čas (vytížení CPU)
  - Paměťový prostor (spotřeba RAM)
  - Spotřeba šířky pásma pro komunikaci (sít')
- **Čas běhu** algoritmu je:
  - Celkový počet provedených elementárních operací (kroky stroje)
  - Ekvivalentní pojmu **složitost algoritmu**

# Algoritmická složitost

- Co měřit?
  - **Prostor (Paměť)**
  - **Čas**
  - Počet kroků
  - Počet konkrétních operací
    - Počet operací disku
    - Počet paketů sítě
  - Asymptotická složitost



# Asymptotická časová složitost

- Nejhorší případ – notace  $O$  (Omikron, big-O)
  - Horní hranice složitosti pro vstupní data dané délky
  - Nejčastěji používaný
- Průměrný případ – notace  $\Theta$  (Theta)
  - Odhaduje nejpravděpodobnější dobu trvání algoritmu
- Nejlepší případ –  $\Omega$  (Omega)
  - Spodní hranice běhu času

# Asymptotická časová složitost – příklad

- Sekvenční hledání prvku v seznamu velikosti  $n$

- Nejhorší případ:

- $n$  srovnání

- Nejlepší případ:

- 1 porovnání

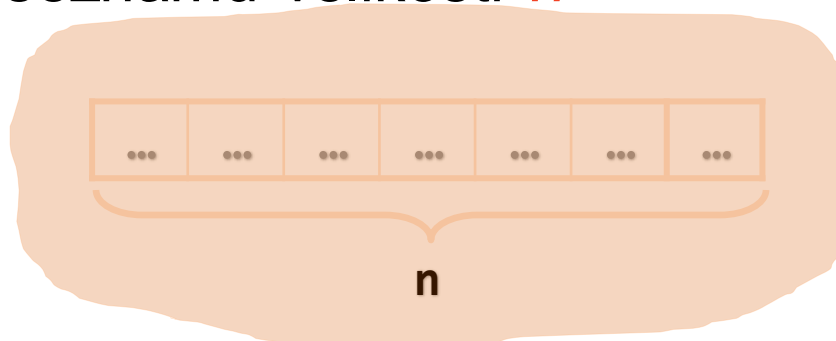
- Průměrný případ:

- $n/2$  porovnání

- Algoritmus běží v **lineárním čase**

- Lineární počet operací

Jaká je paměťová složitost?



# Časová složitost

- **Složitost algoritmu** je hrubý odhad počtu kroků, které daný algoritmus musí provést na základě délky vstupních dat
  - Měřeno skrze asymptotickou notaci
    - $O(g)$  kde  $g$  je funkce závislá na délce vstupních dat
  - Příklady:
    - Lineární složitost  $O(n)$  – všechny prvky jsou zpracované jednou (či  $k \cdot n$ , kde  $k$  je konstanta)
    - Kvadratická složitost  $O(n^2)$  – každý z prvků je zpracovaný maximálně  $n^2$  krát

# Asymptotická notace: Definice

- Asymptotická horní hranice
  - O-notace (Big O notace)
- Pro danou funkci  $g(n)$ , značíme  $O(g(n))$  množinou funkcí, které jsou odlišné od funkce  $g(n)$  od konstanty

**$O(g(n)) = \{f(n): \text{zde existuje kladná konstanta } c \text{ a } n_0 \text{ pro které platí } f(n) \leq c * g(n) \text{ pro všechny } n \geq n_0\}$**

- Příklady:
  - $3 * n^2 + n/2 + 12 \in O(n^2)$
  - $4 * n * \log_2(3 * n + 1) + 2 * n - 1 \in O(n * \log n)$

# Asymptotická složitost

| Složitost    | Notace      | Popis   |
|--------------|-------------|---|
| Konstantní   | $O(1)$      | Konstantní počet operací, nezávislý na velikosti vstupních dat, např.<br>$n = 1\,000\,000 \rightarrow 1\text{-}2$ operací   |
| Logaritmická | $O(\log n)$ | Počet operací odpovídá $\log_2(n)$ , kde $n$ je velikost vstupních dat, např. $n = 1\,000\,000\,000 \rightarrow 30$ operací |
| Lineární     | $O(n)$      | Počet operací je závislé lineárně na vstupních datech, např. $n = 10\,000 \rightarrow 5\,000$ operací                       |

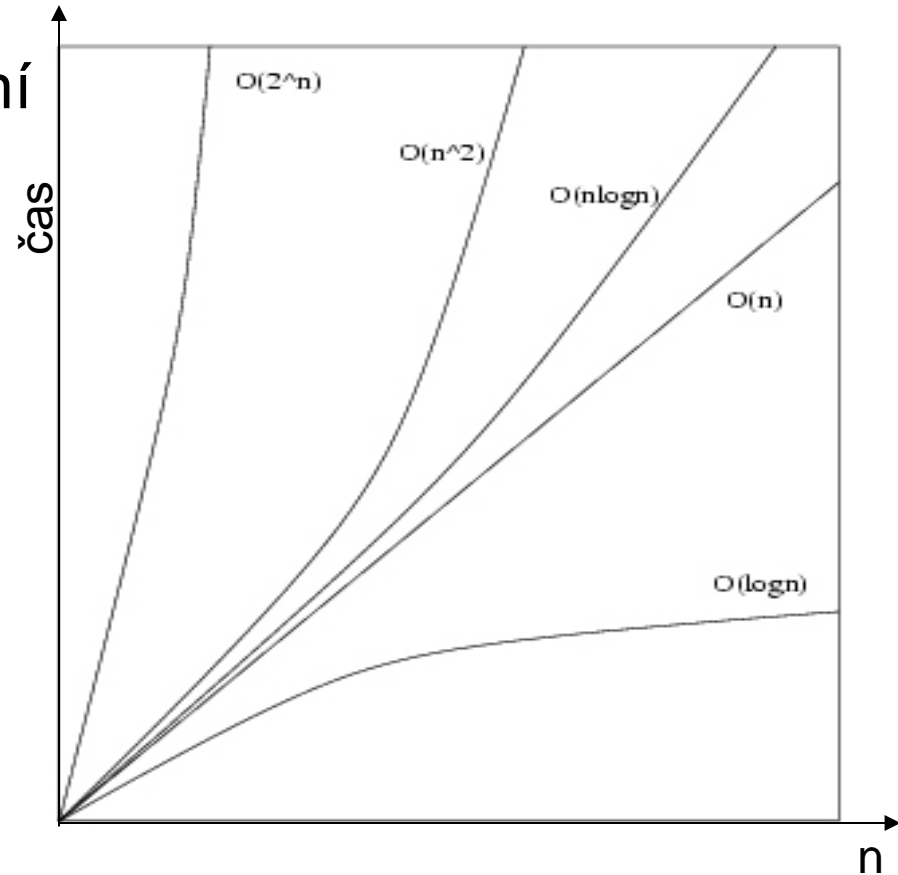


# Asymptotická složitost (2)

| Složitost     | Notace                              | Popis  |
|---------------|-------------------------------------|--|
| Kvadratická   | $O(n^2)$                            | Počet operací odpovídá kvadrátu velikosti vstupních dat, např. $n = 500 \rightarrow 250\,000$ operací        |
| Kubická       | $O(n^3)$                            | Počet operací je závislý kubicky na velikosti vstupních dat, např. $n = 200 \rightarrow 8\,000\,000$ operací |
| Exponenciální | $O(2^n)$ ,<br>$O(k^n)$ ,<br>$O(n!)$ | Exponenciální počet operací, rychle roste, např. $n = 20 \rightarrow 1\,048\,576$ operací                    |

# Asymptotická složitost

- $n \cdot \log(n)$  ... kvazi lineární



# Časová složitost a rychlost

| Složitost            | 10      | 20     | 50      | 100    | 1 000  | 10 000 | 100 000 |
|----------------------|---------|--------|---------|--------|--------|--------|---------|
| $O(1)$               | < 1 s   | < 1 s  | < 1 s   | < 1 s  | < 1 s  | < 1 s  | < 1 s   |
| $O(\log(n))$         | < 1 s   | < 1 s  | < 1 s   | < 1 s  | < 1 s  | < 1 s  | < 1 s   |
| $O(n)$               | < 1 s   | < 1 s  | < 1 s   | < 1 s  | < 1 s  | < 1 s  | < 1 s   |
| $O(n \cdot \log(n))$ | < 1 s   | < 1 s  | < 1 s   | < 1 s  | < 1 s  | < 1 s  | < 1 s   |
| $O(n^2)$             | < 1 s   | < 1 s  | < 1 s   | < 1 s  | < 1 s  | 2 s    | 3-4 min |
| $O(n^3)$             | < 1 s   | < 1 s  | < 1 s   | < 1 s  | 20 s   | 5 hod  | 231 dnů |
| $O(2^n)$             | < 1 s   | < 1 s  | 260 dnů | Dlouho | Dlouho | Dlouho | Dlouho  |
| $O(n!)$              | < 1 s   | Dlouho | Dlouho  | Dlouho | Dlouho | Dlouho | Dlouho  |
| $O(n^n)$             | 3-4 min | Dlouho | Dlouho  | Dlouho | Dlouho | Dlouho | Dlouho  |

# Časová a paměťová složitost

- Složitost může být vyjádřena jako rovnice o několika proměnných, např.:
  - Algoritmus vyplňující matici o rozměrech  $n * m$  přirozenými čísly 1, 2, ... poběží v  $O(n*m)$
  - DFS průchod grafem s  $n$  vrcholy a  $m$  hranami poběží v  $O(n + m)$
- Paměťové nároky by měly být také brány v potaz a mnohdy jsou velice důležité:
  - Čas běhu  $O(n)$ , paměťové nároky  $O(n^2)$
  - $n = 50\,000 \rightarrow \text{OutOfMemoryException}$

# Polynomiální algoritmy

- **Polynomiální-čas algoritmu** je takový alg., jehož nejhorší časovou složitost lze shora ohraničit polynomiální funkcí:

$$W(n) \in O(p(n))$$

- Příklad nejhoršího případu časové složitosti
  - Polynomiální-čas:  $\log n$ ,  $2n$ ,  $3n^3 + 4n$ ,  $2 * n \log n$ ,  $n^k$
  - Ne-polynomiální-čas:  $2^n$ ,  $3^n$ ,  $n^n$ ,  $n!$
- Ne-polynomiální algoritmy pro vyšší hodnoty  $n$  jsou nepoužitelné (kryptografie)

# Analýza složitosti algoritmů

- Příklady



# Příklady složitosti

- Běží v lineárním čase  $O(n)$  kde  $n$  je délka pole
- Počet kroků je  $\sim n$

```
int FindMaxElement(int[] array)
{
    int max = array[0];
    for (int i=0; i<array.length; i++)
    {
        if (array[i] > max)
        {
            max = array[i];
        }
    }
    return max;
}
```

# Příklady složitosti

- Běží v lineárním čase  $O(1)$
- Počet kroků je  $\sim 1\ 000\ 0000\ 000$ , který tentokrát nijak nesouvisí se vstupem array

```
int FindMaxElement(int[] array)
{
    int max = array[0];
    for (int i=0; i< 1 000 000 000; i++)
    {
        if (array[i] > max)
        {
            max = array[i];
        }
    }
    return max;
}
```



## Příklady složitosti (2)

- Běží v kubickém čase  $O(n^3)$
- Počet kroků je  $\sim n^3$

```
decimal Sum3(int n)
{
    decimal sum = 0;
    for (int a=0; a<n; a++)
        for (int b=0; b<n; b++)
            for (int c=0; c<n; c++)
                sum += a*b*c;
    return sum;
}
```

## Příklady složitosti (2)

- Běží v kvadratickém čase  $O(n^2)$
- Počet kroků je  $\sim n^2$

```
decimal Sum3(int n)
{
    decimal sum = 0;
    for (int a=0; a<n; a++)
        for (int b=0; b<n; b++)
            for (int c=0; c<1 000 000 000 000; c++)
                sum += a*b*c;
    return sum;
}
```

## Příklady složitosti (2)

- Běží v lineárním čase  $O(2n)$ , cyklus není vnořený
- Počet kroků je  $\sim 2n$

```
decimal Sum3(int n, int m)
{
    decimal sum = 0;
    for (int a=0; a<n; a++)
        sum += a;
    }
    for (int b=0; b<m; b++)
        sum += b;
    return sum;
}
```

## Příklady složitosti (3)

- Běží v kvadratickém čase  $O(n*m)$
- Počet kroků je  $\sim n*m$

```
long SumMN(int n, int m)
{
    long sum = 0;
    for (int x=0; x<n; x++)
        for (int y=0; y<m; y++)
            sum += x*y;
    return sum;
}
```

# Příklady složitosti (4)

- Běží v kvadratickém čase  $O(n*m)$
- Počet kroků je:  $\sim n*m + \min(m,n)*n$

```
long SumMN(int n, int m)
{
    long sum = 0;
    for (int x=0; x<n; x++)
        for (int y=0; y<m; y++)
            if (x==y)
                for (int i=0; i<n; i++)
                    sum += i*x*y;
    return sum;
}
```

# Příklady složitosti (5)

- Běží v exponenciálním čase  $O(2^n)$
- Počet kroků je:  $\sim 2^n$

```
decimal Calculation(int n)
{
    decimal result = 0;
    for (int i = 0; i < (1<<n); i++)
        result += i;
    return result;
}
```

# Příklady složitosti (6)

- Běží v lineárním čase  $O(n)$
- Počet kroků je:  $\sim n$

```
decimal Factorial(int n)
{
    if (n==0)
        return 1;
    else
        return n * Factorial(n-1);
}
```

# Příklady složitosti (7)

- Běží v exponenciálním čase  $O(2^n)$
- Počet kroků je  $\sim \text{Fib}(n+1)$ , kde  $\text{Fib}(k)$  je  $k$ -té Fibonačiho číslo

```
decimal Fibonacci(int n)
{
    if (n == 0)
        return 1;
    else if (n == 1)
        return 1;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```



# Srovnávání datových struktur

- Příklady



# Efektivita datových struktur

| Datová struktura   | Přidat | Vyhledat | Smazat | Vybrat dle indexu |
|--|--------|----------|--------|-------------------|
| <b>Pole (<math>T[]</math>)</b>   | $O(n)$ | $O(n)$   | $O(n)$ | $O(1)$            |
| <b>Seznam (<math>\text{LinkedList}&lt;T&gt;</math>)</b>  | $O(1)$ | $O(n)$   | $O(n)$ | $O(n)$            |
| <b>Pole proměnlivé délky<br/>(<math>\text{ArrayList}&lt;T&gt;</math>, <math>\text{Vector}&lt;T&gt;</math>)</b> | $O(1)$ | $O(n)$   | $O(n)$ | $O(1)$            |
| <b>Zásobník (<math>\text{Stack}&lt;T&gt;</math>)</b>   | $O(1)$ | -        | $O(1)$ | -                 |
| <b>Fronta (<math>\text{Queue}&lt;T&gt;</math>)</b>   | $O(1)$ | -        | $O(1)$ | -                 |

Pozn.: Souvislosti vám budou jasnější při probírání jednotlivých ADT.

# Výběr datové struktury

- Pole (**T**[])
  - Použít, jestliže je potřeba konstantní počet prvků a ty jsou zpracovávány dle indexu
- Pole s proměnlivou délkou (**List**<**T**>)
  - Použít, jsou-li prvky zpracovávány dle konkrétního indexu
- Lineární seznamy (**LinkedList**<**T**>)
  - Použít, když mají být prvky přidávány z obou stran lineárního seznamu (konec i začátek)
  - Jinak použijte array list (**List**<**T**>)

# Výběr datové struktury (2)

- Zásobník (**Stack**<T>)
  - Použít pro implementaci LIFO (last-in-first-out) chování
  - **List**<T> může být také použito (ekvivalentní)
- Fronta (**Queue**<T>)
  - Použít pro implementaci FIFO (first-in-first-out) chování
  - **LinkedList**<T> může být také použito (ekvivalentní)
- Slovník založený na Hash tabulkách (**HashMap**<K, T>)
  - Použít, když je zapotřebí rychle přidávat a vyhledávat dvojici klíč-hodnota
  - Prvky jsou v hash tabulce v libovolném pořadí („náhodném“)

# Výběr datové struktury (3)

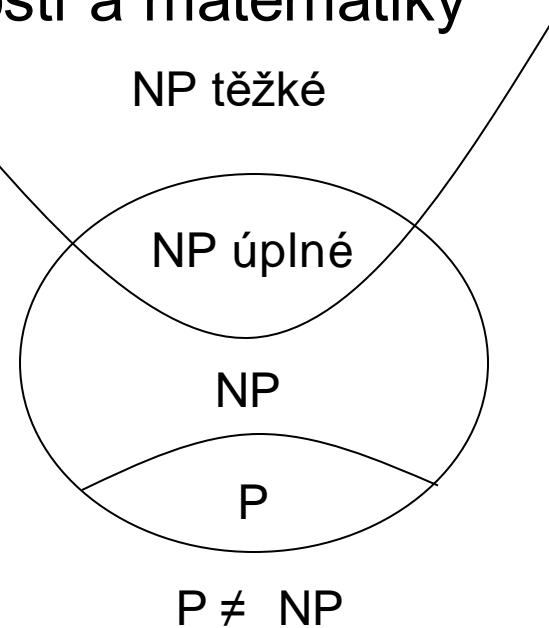
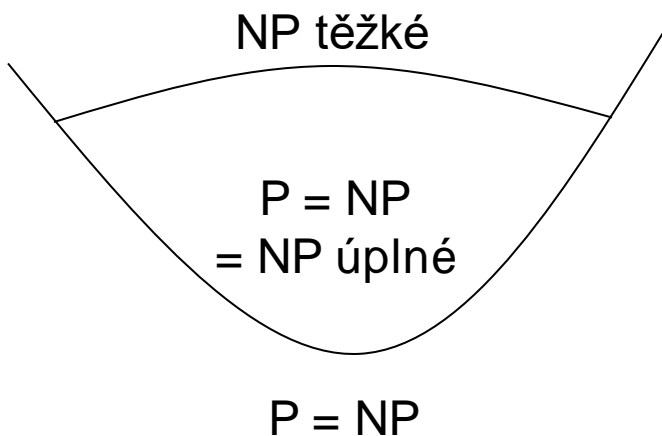
- Slovník založený na vyhledávacích stromech (**SortedDictionary<K,T>**)
  - Použít, když je zapotřebí rychle přidávat a vyhledávat dvojici klíč-hodnota a **seřazený dle klíče**
- Množina založená na hash tabulce (**HashSet<T>**)
  - Použít pro reprezentaci množiny (unikátní hodnota se v množině může vyskytnout max 1x), to add and check belonging to the set fast
  - Prvky jsou v libovolném pořadí
- Množina založená na vyhledávacích stromech (**SortedSet<T>**)
  - Pro reprezentaci množiny unikátních hodnot **seřazených prvků dle klíče**
  - Mírně pomalejší, nežli **HashSet<T>**

# Třídy složitosti P, NP, NP-úplné, NP-těžké

- vyjadřují jak náročný výpočet je nezbytný, abychom problém vyřešili z pohledu výpočetního modelu TS
- Rozdělení:
  - Třída P
    - je možné je provést v polynomiálním čase na deterministickém TS
    - (schůdné algoritmy)
  - Třída NP
    - je možné je provést v polynomiálním čase na **nedeterministickém** TS
    - (neschůdné algoritmy)
    - Jejich součástí jsou i všechny algoritmy z P
  - Třída NP-úplné
    - ty nejtěžší úlohy z NP
    - podmnožina NP
  - Třída NP-těžké
    - přinejmenším tak těžké, jako nejtěžší z NP (nemusí být vykonatelné pomocí TS)

# Souvislosti – P vs. NP

- NP – úplné
  - Současně NP a současně NP-těžké
- Problém ekvivalence P vs. NP – jeden z největších problémů současnosti v oblasti složitosti a matematiky vůbec
- **Není známa odpověď !!**



# Ekvivalence P vs. NP

- Jak dokázat? Stačí sestrojít, převést jediný algoritmus z třídy NP-úplný do třídy P
- Velké důsledky na kryptografii
- Pravděpodobně  $P \neq NP$
- Clayův matematický ústav zařadil tuto otázku vztahu P vs. NP mezi 7 největších matematických problémů současnosti - každá z těchto úloh je přitom oceněna milionem dolarů. „Tržní“ cena rozhodnutí problému P/NP je zřejmě mnohem vyšší.



# Složitost – příklad z předchozí přednášky




- Před nedávnem byla na trh uvedena hra, kde autoři za její vyřešení slibují 2 mil dolarů\* (viz obrázek níže).
- Jaká je složitost tohoto problému? Má cenu se tím zabývat?



**Zdroj:** [physicsandcake.wordpress.com](http://physicsandcake.wordpress.com)

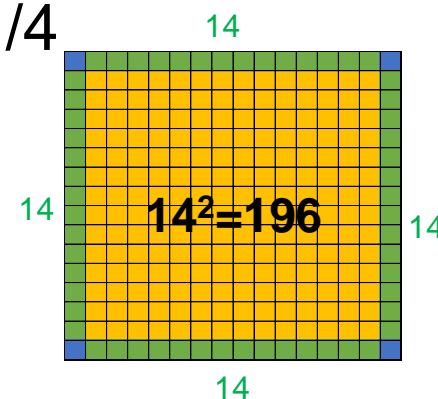
\* nekupovat, jsou to vyhozené peníze 😊

# Složitost – příklad z předchozí přednášky

- 4x rohové políčko (nelze rotovat, jen přeskládat) 
- 56x krajní políčko (nelze rotovat, jen přeskládat) 
- 196x vnitřní políčko (každé může být natočeno do 4 směrů) 

- Je mi jedno, jak je celé hrací pole natočeno /4

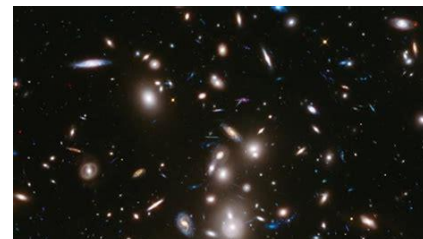
$$C = \frac{4!}{4} \cdot 56! \cdot 196! \cdot 4^{196} = 2,186 \cdot 10^{559}$$



# Složitost

$$2,186 \cdot 10^{59}$$

- Zrnko písku ...  $7.8 \times 10^{19}$  atomů
- Planeta Země ...  $10^{50}$
- Slunce  $1.19 \times 10^{57}$
- Sluneční soustava  $1.19 \times 10^{57}$
- Vesmír  $1.2 \times 10^{79}$  (naše galaxie cca 100 mld. hvězd, existuje cca 2 biliony galaxií)



I kdyby každý atom obsahoval další vesmír, jsme „pouze“ na cca  $1.44 \times 10^{158}$

Je vyloučeno, že je tento problém neřešitelný? Ne! ...ale „hrubou silou“ a s použitím stávajících automatů to nepůjde. Otázkou také je, kolik % řešení je správných.

# Shrnutí

- Vyčíslitelnost (= je to řešitelné?)
- Turingův stroj = matematický model počítače
  - Nejvyšší známá vyjadřovací síla z pohledu vyčíslitelnosti (ne spočítatelnosti – čas z pohledu vyčíslit. nehraje roli)
  - Všechny počítače jak je známe dnes
  - Běžná věc – univerzální varianta odpovídá každému PC, smartphone atp. (klasický TS nemůže měnit program)
  - Není jisté, že pracuje na stejných základech jako lidský mozek
- Existují i problémy nevyčíslitelné
  - Důkaz diagonalizace
  - Problém zastavení TS, ...

# Shrnutí

- Turingův stroj
  - Deterministický
  - Univerzální
  - Paralelní
  - Kvantový
  - Nedeterministický (nebyl doposud sestrojen)
- Pojem „Turingovsky úplný“ – jeho vyjadřovací síla je ekvivalentní Turingově stroji

# Shrnutí

- Složitost (=spočitatelnost = za jak dlouho)
- Absolutní (přesný vzorec)
- Asymptotická
  - Průměrná, nejhorší (big-O), nejlepší
  - logaritmická, lineární,  $n \log n$ , kvadratická, kubická, exponenciální, faktoriální, atp.
  - Nejčastěji používaná je nejhorší: např.:  $O(n^2)$
  - Různé datové struktury mají různou efektivitu na různé operace

# Shrnutí

- Třídy složitosti

- P – běh na Turingově stroji v polynomiálním čase
- NP – běh na Turingově stroji v nepolynomiálním čase
- NP-těžké - nejtěžší třída problémů ze skupiny NP ( $n!$ )

- Třídy složitosti

- Kryptografie založena na složitosti (NP-těžké) nespočítatelnosti (s výjimkou Vernamovy šifry – vojenství)

# Děkuji za pozornost