Teoretická informatika TIN - 2025/2026

2. Test 13. 11. 2025

Čas na řešení: 120 minut

Jméno/přihlašovací jméno:

Hodnocení:

Poznámka: *Pokud při vypracování zkoušky použijete jinou notaci a konvence, než byly zavedeny na přednáškách, je nutné takovou notaci popsat. Písemnou zkoušku zpracujte čitelně a úhledně.*

Příklad 1
5 bodů

a) Formálně zapište *Pumping lemma pro bezkontextové jazyky.*

b) Nechť $\Sigma = \{a, b, c\}$. Sestrojte (rozšířený) zásobníkový automat pro jazyk

$$L = \{w \in \Sigma^* \mid \#_a(w) > \#_b(w) + \#_c(w)\},$$

kde $\#_x(w)$ značí počet znaků $x \in \{a, b, c\}$ v řetězci $w \in \Sigma^*$. Automat popište přechodovým diagramem.

c) Uvažujme gramatiku $G = (\{E\}, \{+, *, (,), i\}, P, E)$, kde P je množina pravidel

$$E \to E + E \mid E * E \mid (\ E\ ) \mid i$$

i) Rozhodněte a dokažte, zda G je *jednoznačná*.

ii) Pro gramatiku G sestrojte zásobníkový automat A, který modeluje *syntaktickou analýzu shora dolů*. Automat A popište ve shodě s definicí zásobníkového automatu (tj. nikoliv diagramem).

TIN — 2. Test 13. 11. 2025 **Čas: 120 minut**

Prostor pro řešení příkladu 1.

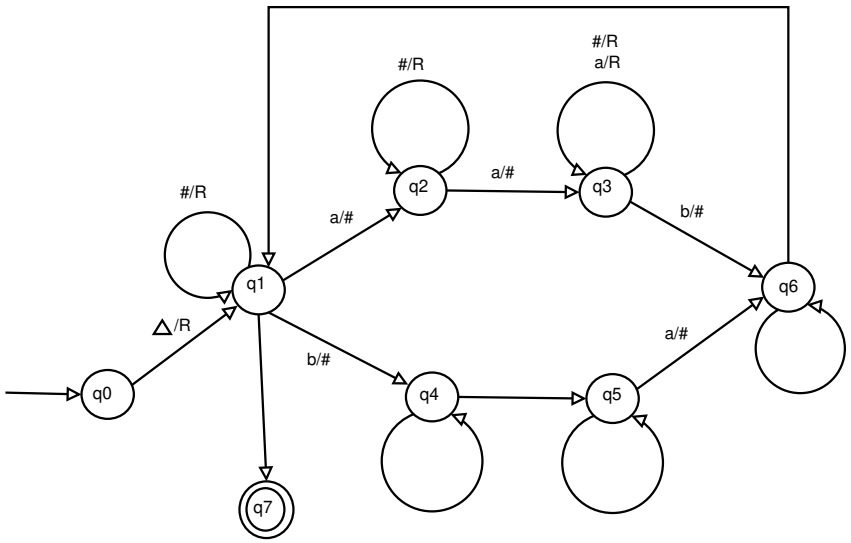
Příklad 2
5 bodů

a) Nechť $\Sigma = \{a, b\}$. Do níže uvedeného diagramu doplňte 1 nový přechod (hranu i její označení) a doplňte označení k neoznačeným přechodům tak, aby vznikl *deterministický* Turingův stroj T, který přijímá jazyk

$$L_1 = \{w \in \{a, b\}^* \mid \#_a(w) = 2\#_b(w)\}$$

kde $\#_x(w)$ značí počet symbolů $x \in \{a, b\}$ v řetězci $w \in \{a, b\}^*$.

Poznámka: V rámci řešení je dovoleno použít „vícenásobné“ přechody, jako již jsou uvedeny např. ze stavu q_3 do q_3.



b) Dokažte, že jazyk L_2 je *rekurzivně vyčíslitelný*.

$$L_2 = \{<M_1> \# <M_2> \# <n> \mid |L(M_1) \cup L(M_2)| > n\},$$

kde $<M>$ značí kód Turingova stroje M a $<n>$ značí kód čísla n.

Poznámka: V důkazu stačí uvést slovní popis fungování Turingova stroje (deterministického či nedeterministického, dle libosti s více páskami), který akceptuje jazyk L_2.

c) Nechť $\Sigma = \{a, b\}$. Rozhodněte a dokažte, zda existuje *lineárně ohraničený automat* M takový, že množina $\{w \in \Sigma^* \mid M \text{ na } w \text{ cyklí}\}$ *obsahuje právě jeden prvek* a zároveň $L(M)$ je *nekonečná množina*.

Prostor pro řešení příkladu 2.

Příklad 3
5 bodů

a) Formálně definujte i) *časovou složitost* deterministického Turingova stroje, který přijímá jazyk L a ii) *třídu jazyků* $DTIME[n^2]$.

b) Uvažme $\Sigma = \{0, 1\}$ a předpokládejte, že $\mathbf{P} \neq \mathbf{NP}$. Rozhodněte a stručně dokažte, zda platí následující tvrzení:

i) Existuje nekonečný jazyk L_1 takový, že $L_1 \in DTIME[n] \setminus DTIME[1]$.

ii) Existuje regulární jazyk L_2, který nepatří do třídy $DSPACE[1]$.

iii) Existuje jazyk $L_3 \in \mathbf{EXP} \setminus \mathbf{P}$.

Poznámka: U třídy DSPACE[1] *nepočítáme buňky pásky, na nichž je zapsán vstup za předpokladu, že obsah těchto buněk není modifikován.*

c) Uvažme orientovaný graf $G = (V, E)$, kde V je množina vrcholů a E je množina hran. Funkce $Adj(v)$ vrací množinu následníků vrcholu v.

Definujme predikát P: $P(G) = \texttt{true} \iff \exists v_s \in V$: existuje v G neprázdná cesta z v_s do v_s (tj. obsahuje aspoň jednu hranu).

Uvažme algoritmus $hasCycle(G)$, který vrací `true` právě tehdy když $P(G) = \texttt{true}$

i) Analyzujte a zdůvodněte asymptotickou časovou složitost algoritmu $hasCycle(G)$ v nejhorším případě.

ii) Analyzujte a zdůvodněte asymptotickou časovou složitost algoritmu $hasCycle(G)$ v nejlepším případě.

Poznámka 1: Předpokládejte uniformní cenové kritérium, kde složitost každého řádku programu je 1.

Poznámka 2: Časovou složitost analyzujte vzhledem k $|V|$ a $|E|$.

```
1  Function hasCycle(G)
2      foreach v_s ∈ V do
3          foreach v ∈ V do
4              v.visited := false
5          Q := emptyQueue()
6          Q.enqueue(v_s)
7          v_s.visited = true
8          while Q.nonempty() do
9              v := Q.dequeue()
10             foreach v' ∈ Adj(v) do
11                 if v' = v_s then return true
12                 if v'.visited = false then
13                     v'.visited = true
14                     Q.enqueue(v')
15     return false
```

Bonusový příklad: Navrhněte algoritmus, který řeší stejný problém a má optimální asymptotickou časovou složitost v nejhorším případě. Analyzujte tuto složitost.

Prostor pro řešení příkladu 3.

TIN — 2. Test 13. 11. 2025 **Čas: 120 minut**

Příklad 4
3 body

Uvažme modifikovanou datovou strukturu *vector* implementovanou níže:

Operace *vector*.pushBack(*value*) vloží na konec nový prvek s hodnotou *value*. Tato operace má složitost 1 (v tomto příkladě neřešíme možnou realokaci). Proměnná *size* obsahuje aktuální velikost struktury *vector* a proměnná *elements* obsahuje počet platných prvků ve struktuře *vector*.

Funkce `insert`(*value*) zavolá operaci *vector*.pushBack(*value*) a aktualizuje proměnné *elements* a *size*.

Funkce `remove`(*pos*) zneplatňuje prvek na pozici *pos*. Pokud dojde k situaci (řádek 15), kdy počet platných prvků je menší než polovina velikosti struktury *vector*, tak proběhne "setřesení"prvků na začátek. Operace firstEmpty() vrací první pozici (bráno od 0), na které se nachází zneplatněný prvek a operace lastNonEmpty() vrací poslední pozici, na které je platný prvek. Obě operace mají složitost 1. Po "setřesení"všech prvků se velikost struktury nastaví na počet platných prvků.

i) Analyzujte a zdůvodněte asymptotickou časovou složitost (vhledem k velikosti *size*) funkce *remove*() v nejhorším případě.

ii) Analyzujte a zdůvodněte amortizovanou časovou složitost libovolné posloupnosti m volání funkcí *insert*() a *remove*().

Poznámka 1: Předpokládejte uniformní cenové kritérium, kde složitost každého řádku programu je 1.

Poznámka 2: Pokud použijete metodu účtů, je nutné pro zisk plného počtu bodů rovněž argumentovat, že navržené kreditové schéma je korektní. Body lze ale získat i za částečné řešení či jakýkoliv smysluplný pokus o analýzu amortizované složitosti.

```
1  global: vector = emptyVector()
2  global: elements = 0
3  global: size = 0
4  function insert(value)
5  |   vector.pushBack(value)
6  |   elements = elements + 1
7  |   size = size + 1
8  function remove(pos)
9  |   if vector[pos] ≠ ⊥ then
10 |   |   vector[pos] := ⊥
11 |   |   elements = elements − 1
12 |   if elements = 0 then
13 |   |   size := 0
14 |   else
15 |   |   if elements < ⌈size/2⌉ then
16 |   |   |   while true do
17 |   |   |   |   x := firstEmpty()
18 |   |   |   |   y :=
   |   |   |   |    lastNonEmpty()
19 |   |   |   |   if x > y then
20 |   |   |   |   |   break
21 |   |   |   |   vector[x] = vector[y]
22 |   |   |   |   vector[y] = ⊥
23 |   |   |   size := elements
```

Prostor pro řešení příkladu 4.