Cvičení 7

Příklad 1: Odsimulujte činnost následujícího programu pro stroj RAM, pokud jako vstup dostane posloupnost (délky 1) tvořenou jediným číslem 4, tj. vypište posloupnost jednotlivých konfigurací, kterými stroj RAM projde během tohoto výpočtu:

```
0: R2 := READ()
1: R1 := 2
2: goto 5
3: R1 := R1 * R1
4: R2 := R2 − 1
5: if (R2 > 0) goto 3
6: WRITE(R1)
7: halt
```

Určete, co tento program počítá, tj. co vydá jako výstup, když na vstupu dostane číslo n.

Řešení: Pro hodnoty n, kde $n \geq 0$, vypíše na výstup hodnotu 2^{2^n}. Pro hodnoty n, kde $n < 0$, vypíše na výstup hodnotu 2.

Příklad 2: Určete, co dělá následující program pro stroj RAM, tj. popište detailně jeho činnost pro libovolný vstup a popište, co bude výstupem.

Poznámka: Pro přehlednost zde nejsou na rozdíl od předchozího příkladu explicitně uvedeny adresy instrukcí, ale jsou místo nich použita symbolická návěští.

```
    R4 := 4
    R3 := READ()
    R1 := R4 + R3
    R0 := 0
L1: if (R1 = R4) goto L2
    [R1] := R0
    R1 := R1 − 1
    goto L1
L2: R2 := READ()
    if (R2 ≤ 0) goto L3
    if (R2 > R3) goto L3
    R1 := R4 + R2
    R0 := [R1]
    R0 := R0 + 1
    [R1] := R0
    goto L2
L3: R2 := 1
L4: if (R2 > R3) goto L5
    R1 := R4 + R2
    R0 := [R1]
    WRITE(R0)
    R2 := R2 + 1
    goto L4
L5: halt
```

Příklad 3: Pro každý z následujících problémů navrhněte program pro stroj RAM, který ho řeší.

Poznámka: Při konstrukci stroje nemusíte řešit chybná data na vstupu, která neodpovídají zadání.

a) VSTUP: celá čísla x, y (tj. $x, y \in \mathbb{Z}$)

VÝSTUP: hodnota $x + y$

Řešení:

$R_0 :=$ READ ()
$R_1 :=$ READ ()
$R_0 := R_0 + R_1$
WRITE (R_0)
halt

b) VSTUP: celá čísla x, y (tj. $x, y \in \mathbb{Z}$)
VÝSTUP: $\max\{x, y\}$

Řešení:

$R_0 :=$ READ ()
$R_1 :=$ READ ()
if $(R_0 \geq R_1)$ **goto** L_1
$R_0 := R_1$
L_1: WRITE (R_0)
halt

c) VSTUP: přirozené číslo n (tj. $n \in \mathbb{N}$)
VÝSTUP: sekvence čísel $1, 2, \ldots, n$

Poznámka: Pro $n = 0$ bude sekvence na výstupu prázdná.

Řešení:

$R_1 :=$ READ ()
$R_0 := 1$
L_1: **if** $(R_0 > R_1)$ **goto** L_2
WRITE (R_0)
$R_0 := R_0 + 1$
goto L_1
L_2: **halt**

d) VSTUP: sekvence čísel $a_1, a_2, \ldots, a_n, 0$, kde $n \geq 0$ a $a_i \in \mathbb{Z} - \{0\}$ pro $1 \leq i \leq n$
VÝSTUP: $\prod_{i=1}^{n} a_i$

Poznámka: Zápis $\prod_{i=1}^{n} a_i$ označuje součin $a_1 \cdot a_2 \cdot \ldots \cdot a_n$. Pro $n = 0$ bude výstupem hodnota 1.

e) VSTUP: sekvence čísel $a_1, a_2, \ldots, a_n, 0$, kde $n \geq 0$ a $a_i \in \mathbb{Z} - \{0\}$ pro $1 \leq i \leq n$
VÝSTUP: sekvence čísel $a_n, a_{n-1}, \ldots, a_1$

Příklad 4: Sestavte program pro stroj RAM, který přečte ze vstupu číslo n a vypíše na výstup n-té Fibonacciho číslo F_n. Můžete předpokládat, že číslo n na vstupu je nezáporné

(tj. nemusíte řešit situaci, kdy $n < 0$). Připomeňme, že Fibonacciho čísla $F_0, F_1, F_2, \ldots$ jsou definována následujícím rekurentním vztahem:

$$F_n = \begin{cases} 0 & \text{pro } n = 0 \\ 1 & \text{pro } n = 1 \\ F_{n-1} + F_{n-2} & \text{pro } n > 1 \end{cases}$$

Řešení:

$R_1 :=$ READ ()
$R_0 := 0$
$R_2 := 0$
$R_3 := 1$
L_1: **if** $(R_0 \geq R_1)$ **goto** L_2
$R_4 := R_2 + R_3$
$R_2 := R_3$
$R_3 := R_4$
$R_0 := R_0 + 1$
goto L_1
L_2: WRITE (R_2)
halt

Příklad 5: Vezměme si následující Algoritmus 1. Vstupem tohoto algoritmu může být libovolné přirozené číslo n (tj. hodnoty proměnné n mohou být libovolná neomezeně velká přirozená čísla).

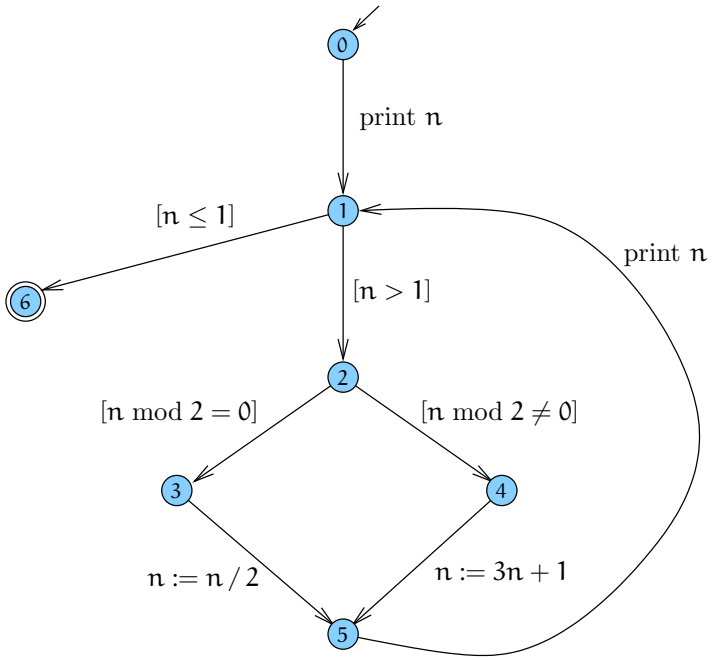
Algoritmus 1:

```
PrintSeq(n):
    print n
    while n > 1 do
        if n mod 2 = 0 then
            n := n / 2
        else
            n := 3 * n + 1
        print n
```

a) Nakreslete graf řídícího toku tohoto algoritmu.

b) Popište výpočet, který tento algoritmus provede, pokud jako vstup dostane číslo 5. Vypište posloupnost jednotlivých konfigurací při tomto výpočtu.

c) Kolik kroků provede tento algoritmus, když jako vstup dostane číslo 7 ? Co bude výstupem?



Obrázek 1: Graf řídícího toku

d) Vytvořte program pro stroj RAM realizující činnost tohoto algoritmu.

Poznámka: Hodnotu proměnné n načte tento stroj RAM ze vstupu.

Řešení:

a) Graf řídícího toku je uveden na Obrázku 1.

b) Výpočet vypadá následovně

Krok	Stav	n
0	0	5
1	1	5
2	2	5
3	4	5
4	5	16
5	1	16
6	2	16
7	3	16
8	5	8
9	1	8
10	2	8
11	3	8
12	5	4
13	1	4
14	2	4
15	3	4
16	5	2
17	1	2
18	2	2
19	3	2
20	5	1
21	1	1
22	6	1

c) Výstupem bude posloupnost $7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$.

Celkový počet kroků se určí následovně: Cyklem se projde celkem 16 krát. Při každém průchodu cyklem se provedou 4 kroky. Na začátku se provede mimo cyklus jeden krok a konci také jeden krok. Celkem se tedy provede $1 + 16 \cdot 4 + 1 = 66$ kroků.

d)

```
    R2 := READ()
    WRITE(R2)
L1: if (R2 ≤ 1) goto L4
    R1 := ⌊R2 / 2⌋
    R0 := R1 * 2
    R0 := R2 − R0
    if (R0 ≠ 0) goto L2
    R2 := R1
    goto L3
L2: R2 := R2 * 3
    R2 := R2 + 1
L3: WRITE(R2)
    goto L1
L4: halt
```

nebo

```
    R2 := READ()
L1: WRITE(R2)
    if (R2 ≤ 1) goto L3
    R1 := ⌊R2 / 2⌋
    R0 := R1 * 2
    R0 := R2 − R0
    if (R0 ≠ 0) goto L2
    R2 := R1
    goto L1
L2: R2 := R2 * 3
    R2 := R2 + 1
    goto L1
L3: halt
```

Příklad 6: Uvažujme následující algoritmus 2 popsaný pseudokódem.

Algoritmus 2: Třídění přímým vkládáním

```
INSERTION-SORT ():
    n := READ ()
    for i := 0 to n − 1 do
        A[i] := READ ()
    for j := 1 to n − 1 do
        x := A[j]
        i := j − 1
        while i ≥ 0 and A[i] > x do
            A[i + 1] := A[i]
            i := i − 1
        A[i + 1] := x
    for i := 0 to n − 1 do
        WRITE (A[i])
```

a) Nakreslete graf řídícího toku reprezentující tento pseudokód.

b) Implementujte tento algoritmus jako program pro stroj RAM.

c) Popište, jak různé části vámi vytvořeného kódu programu pro stroj RAM odpovídají jednotlivým hranám v grafu řídícího toku.

Řešení:

```
        R5 := 7
        R4 := READ ()
        R2 := 0
L1:     if (R2 ≥ R4) goto L2
        R0 := READ ()
        R1 := R5 + R2
        [R1] := R0
        R2 := R2 + 1
        goto L1
L2:     R3 := 1
L3:     if (R3 ≥ R4) goto L6
        R1 := R5 + R3
        R6 := [R1]
        R2 := R3 − 1
L4:     if (R2 < 0) goto L5
        R1 := R5 + R2
        R0 := [R1]
        if (R0 ≤ R6) goto L5
        R1 := R1 + 1
        [R1] := R0
        R2 := R2 − 1
        goto L4
L5:     R1 := R5 + R2
        R1 := R1 + 1
        [R1] := R6
        R3 := R3 + 1
        goto L3
L6:     R2 := 1
L7:     if (R2 ≥ R4) goto L8
        R1 := R5 + R2
        R0 := [R1]
        WRITE (R0)
        R2 := R2 + 1
        goto L7
L8:     halt
```

Příklad 7: Popište, jak k libovolnému Turingovu stroji $\mathcal{M}$ vytvořit program pro stroj RAM realizující stejný algoritmus jako stroj $\mathcal{M}$. Uvažujte následující varianty Turingových strojů:

a) Turingův stroj s jednou jednostranně nekonečnou páskou

b) Turingův stroj s jednou oboustranně nekonečnou páskou

c) Turingův stroj s více oboustranně nekonečnými páskami

Poznámka: Není třeba, abyste dané programy pro stroj RAM explicitně vytvářeli. Stačí, pokud slovně, ale dostatečně podrobně, popíšete, jak budou dané stroje RAM konkrétně fungovat.

Příklad 8: Sestavte program pro stroj RAM, který přečte ze vstupu dvě čísla x a k a na výstup vypíše hodnotu k-tého bitu čísla x (tj. 0 nebo 1), přičemž bity jsou číslovány od 0 a 0-tý bit je nejméně významný bit. Můžete předpokládat, že $x \geq 0$ a $k \geq 0$ (tj. nemusíte řešit situace, kdy $x < 0$ nebo $k < 0$).

Řešení:

$$
\begin{array}{ll}
 & R_1 := \textsc{read}() \\
 & R_0 := \textsc{read}() \\
L_1: & \textbf{if } (R_0 \leq 0) \textbf{ goto } L_2 \\
 & R_1 := \lfloor R_1 / 2 \rfloor \\
 & R_0 := R_0 - 1 \\
 & \textbf{goto } L_1 \\
L_2: & R_0 := \lfloor R_1 / 2 \rfloor \\
 & R_0 := R_0 * 2 \\
 & R_1 := R_1 - R0 \\
 & \textsc{write}(R_1) \\
 & \textbf{halt}
\end{array}
$$

Příklad 9: Navrhněte program pro stroj RAM, který načte ze vstupu dvě čísla x a y (můžete předpokládat, že $x \geq 0$ a $y \geq 0$) a na výstup vypíše jejich součin $x \cdot y$. Aby to nebylo tak jednoduché, musíte navíc dodržet následující omezení:

- Ve vašem programu ***nesmíte*** použít aritmetické instrukce s operacemi násobení a dělení. Můžete ovšem použít aritmetickou instrukci pro bitový posun doprava o jeden bit:

  $$R_i := \mathit{rshift}(R_j)$$

  která provede to samé, co by provedla instrukce $R_i := \lfloor R_j / 2 \rfloor$.

- Celkový počet instrukcí, který váš program provede, musí být polynomiální vzhledem k počtům bitů nutných pro zápis čísel x a y.

- Napadá vás nějaký způsob, jak vyřešit tento problém bez použití instrukce typu $R_i := \mathit{rshift}(R_j)$, tj. jak spočítat hodnotu $x \cdot y$ na stroji RAM, který má z aritmetickým operací k dispozici pouze sčítání a odčítání, tak, aby celkový počet kroků byl polynomiální vzhledem počtu bitů čísel x a y?

Řešení:

$$
\begin{array}{ll}
 & \mathsf{R}_0 := \textsc{read}\,() \\
 & \mathsf{R}_1 := \textsc{read}\,() \\
 & \mathsf{R}_2 := 0 \\
\mathsf{L}_1\colon & \textbf{if } (\mathsf{R}_1 \leq 0) \textbf{ goto } \mathsf{L}_3 \\
 & \mathsf{R}_3 := \mathit{rshift}\,(\mathsf{R}_1) \\
 & \mathsf{R}_4 := \mathsf{R}_3 + \mathsf{R}_3 \\
 & \mathsf{R}_4 := \mathsf{R}_1 - \mathsf{R}_4 \\
 & \textbf{if } (\mathsf{R}_4 = 0) \textbf{ goto } \mathsf{L}_2 \\
 & \mathsf{R}_2 := \mathsf{R}_2 + \mathsf{R}_0 \\
\mathsf{L}_2\colon & \mathsf{R}_1 := \mathsf{R}_3 \\
 & \mathsf{R}_0 := \mathsf{R}_0 + \mathsf{R}_0 \\
 & \textbf{goto } \mathsf{L}_1 \\
\mathsf{L}_3\colon & \textsc{write}\,(\mathsf{R}_2) \\
 & \textbf{halt}
\end{array}
$$