Turingův stroj

jeho funkce a přínos světu (nejen) programování

Výukový materiál

Obsah

1. Práce s příručkou pro pedagogy
2. Co je Turingův stroj a proč je dobré ho znát
3. Jak funguje tento model
4. Jak spustit interaktivní mód
5. Představení základních příkazů, práce s interaktivním módem
6. Využití pro výuku základů algoritmizace
7. Představení dalších způsobů programování modelu
8. Představení pokročilých příkazů
9. Využití pro výuku pokročilejších témat z oblasti algoritmizace ilých příkazů
10. Řešené úlohy pro zamyšlení i další zpracování
11. Řešení zadaných úloh

1. Práce s příručkou pro pedagogy

Tato příručka popisuje interaktivní možnost, jak obeznámit studenty s funkcí a historií Turingova stroje, ale také se základy programování a algoritmizace. Materiál je určen zejména pro studenty sedmých až osmých ročníků základních škol a odpovídajících ročníků víceletých gymnázií. Je samozřejmě možné ho využít i pro starší studenty, avšak v takových případech doporučujeme projít látku o něco rychleji a například vynechat úlohy užívající stejných principů, aby studenty hodina stále bavila. Je také vhodné, aby hodiny s využitím této příručky vedli učitelé se základním vhledem do světa programování, ale není to nezbytně nutné.

A jak přesně s příručkou pracovat? Doporučeným postupem je nejprve celou příručku pročíst a tím získat přehled o možném průběhu hodiny. V příručce jsou obsaženy i čistě informativní kapitoly, například o tom co to vlastně Turingův stroj je. Tyto kapitoly je taktéž vhodné si přečíst, aby měl pedagog přehled a zároveň věděl, proč je tento princip v informatice důležitý a proč by ho studenti měli znát. Je nutností zde poznamenat, že není nutné, ba ani doporučené předávat studentům všechny informace a realizovat všechna cvičení v této příručce. Množství informací i času, kterým tímto tématem daná třída stráví, by mělo záviset zejména na schopnostech a zájmu dané skupiny.

2. Co je Turingův stroj a proč je dobré ho znát

Tato kapitola bude sloužit jako stručný úvod do toho, co to je Turingův stroj, jak funguje a co přinesl světu. Jedním z cílů této kapitoly je to, aby měl pedagog povědomí o tom, proč by studenti měli Turingův stroj znát. Dalším cílem je samozřejmě i to, aby se informace dostali i ke studentům a to zaprvé pro jejich vzdělání, ale také pro zvýšení jejich motivace a zájmu o toto téma.

Turingův stroj je ve skutečnosti čistě teoretický koncept, vymyšlený Alanem Turingem. Jedná se vlastně o model fungujícího počítače, usnadňující pohled na programování. Turingův stroj je zajímavý tím, že jeho funkce je v principu jednoduchá a k ovládání se využívá pouze několik základních příkazů, avšak i přesto je velice výpočetně silný. Dodnes se výkonnost některých strojů a programovacích jazyků hodnotí podle toho, jsou-li tzv. Turingovsky úplné. Turingovsky úplným nazýváme takový nástroj, který má stejnou výpočetní sílu, jako právě Turingův stroj.

Turingův stroj je čistě teoretický koncept, a proto existuje mnoho různých představ, možností a modifikací toho, jak přesně by měl tento stroj vypadat. Co je však všem těmto představám společné, je to, že se Turingův stroj skládá z čtecí hlavy a několika nekonečných pásek. Na pásky programátor zapisuje kód a následně je předává stoji, který podle jednotlivých příkazů něco vykonává. Čtecí hlava je pak ta část stoje, která po takovéto pásce přejíždí a stroj řídí. Vždy najede na nějaké políčko na pásce, vykoná příkaz a přesune se na další políčko. Nekonečná páska může být také využita jako paměť například pro uchovávání mezivýsledků.

3. Jak funguje tento model

Model, s nímž pracuje tato příručka, používá dvě libovolně dlouhé pásky. První je páska s pamětí. Na tu si stroj může ukládat přirozená čísla. Na druhou pásku může uživatel vpisovat příkazy, které si přeje provést. Navíc stroj využívá tzv. ukazatele, standardně se využívají dva ukazatele, avšak model jich umožňuje využít i více.

V nynější podobě lze model spustit ve třech možných módech. Nejdůležitější pro výuku je interaktivní mód, díky kterému se studenti zajímavým způsobem naučí, co dělají které příkazy a jak funguje práce s paměťovou i programovou páskou. Interaktivní mód je uzpůsoben co nejjednodušší obsluze, a proto je i jeho spuštění oproti jiným módům znatelně jednodušší. Další mód umožňuje vstup ze souboru, což mu dává značnou výhodu při tvorbě delších a složitějších kódů, které je třeba debugovat, nebo je archivovat. Poslední mód zpracovává vstup přímo z kódu, konkrétně z listu. Pro spuštění těchto dvou módů je třeba sepsat o něco větší část kódu. Přesný význam této části kódu, jakožto i celého programu je popsán v následujících několika odstavcích.

Celý program je postaven na principech objektově orientovaného programování[1]. Celý program obsluhuje třída Head, představující čtecí hlavu stroje. Další důležitou třídou je třída Memory reprezentující paměťovou pásku stroje. Poslední větší třída má na starosti zpracovávání příkazů ze vstupu a budeme jí prozatím zjednodušeně říkat Commands. Pro jasné označení bez dlouhého popisování budeme tuto funkci označovat také výrazem „dávkovač příkazů“.

Objekt třídy Head si pamatuje odkaz na paměť, kterou využívá, neboli na objekt třídy Memory. Dále si pak pamatuje odkaz na dávkovač příkazů typu Commands. Poslední proměnou třídy Head je pravdivostní hodnota udávající, zda si uživatel přeje, aby program vypisoval mezivýsledky, nebo nikoliv. Třída Head má pouze jedinou metodu a tou je DoNext, které si z programové pásky přečte aktuální příkaz a vykoná ho. K přečtení příkazu z pásky využije objektu třídy Commands, konkrétně jeho metody GetCommand[2].

V popisu výše je pro lepší přehlednost zjednodušen pohled na třídu Commands, a proto bude v tomto odstavci vše uvedeno na správnou míru. Vzhledem k různým možnostem zadávání vstupu při použití různých módů je nutno měnit způsob, jakým „Commands“ zpracovává vstup. Proto neexistuje pouze jedna taková třída, nýbrž existuje samostatná třída pro každý druh zadávání vstupu. Konkrétně se jedná o třídy InteractiveCommands, FileCommands a ListCommands. Všechny tyto třídy však implementují rozhraní (častěji známé pod názvem interface) ICommands, což zajišťuje, že je lze libovolně zaměňovat a využívat tu, která se zrovna hodí nejvíce.

1 Co to je objektově orientované programování si můžete přečíst například na stránce http://pehapko.cz/oop/uvod

2 Nejedná-li se o nějakou specifickou situaci, ve které je nutno použít jinou z metod.

4. Jak spustit interaktivní mód

Model doporučuji otevírat v programu Visual Studio, ve kterém byl vytvořen. Pro spuštění interaktivního módu určeného pro výuku, je potřeba pouze mít v souboru Program.cs ve funkci Main napsaný příkaz:
“InteractiveCommands actualCommands = new InteractiveCommands();”

Text v tomto souboru by tedy měl vypadat třeba takto:

```
using System;
using System.Collections.Generic;

namespace Turing
{
    0 references
    class Program
    {
        0 references
        static void Main()
        {
            InteractiveCommands podavacPrikazu = new InteractiveCommands();
        }
    }
```

Po spuštění programu (ve Visual Studiu zelenou šipkou na horní liště) se objeví nové okno, ve kterém budete s modelem interagovat. Ovládání by mělo být z velké části intuitivní.

5. Představení základních příkazů, práce s interaktivním módem

Dříve, než se pustíme do vysvětlování konkrétních příkazů, ujasníme ještě jeden důležitý fakt. Pro funkci některých příkazů je potřeba zadat kromě samotného příkazu i nějaký parametr. Například, když uživatel potřebuje přesunout ukazatel, je nutné stroji sdělit, kam přesně je ho potřeba přesunout. Parametr se vždy píše do dalšího pole programové pásky za příkaz. V interaktivním módu našeho modelu je tedy potřeba nejprve zadat příkaz a až následně budete vyzváni k zadání parametru.

Nyní už přejdeme k vysvětlování konkrétních příkazů.

ADD

Add je příkaz pro sčítání. Funguje tak, že sečte hodnoty pod všemi ukazateli na pásce s pamětí a výsledek následně zapíše do paměti pod první ukazatel. Můžeme to tedy chápat i tak, že hodnoty pod všemi ukazateli kromě prvního, přičte právě k hodnotě pod prvním ukazatelem.
SUB – Podobně jako příkaz Add funguje i příkaz Sub, který ovšem hodnoty nepřičítá, nýbrž odečítá. Odečte tedy hodnoty pod všemi ukazateli krom prvního od hodnoty pod prvním ukazatelem.

MOVE

Dalším zásadním příkazem je příkaz Move. Ten se využívá pro změnu pozice ukazatelů. Move je jeden z příkazů, které je třeba doplnit parametrem. Parametr specifikuje, na jakou pozici má stroj přesunout první ukazatel. Druhý ukazatel se pak přesune na místo, kde byl do té doby první ukazatel, třetí ukazatel se přesune tam, kde byl druhý ukazatel a tak dále, v závislosti na tom, kolik zrovna používáme ukazatelů.

PRINT

Se používá pro vracení hodnoty uživateli. Po přečtení tohoto příkazu vytiskne stroj hodnotu uloženou v paměti pod prvním ukazatelem. Zároveň je možné ho využít jako parametr po zadání jiného příkazu. V tom případě bude jako parametr použité číslo uložené v paměti pod prvním ukazatelem.

EXIT

Ukončí program. Tento příkaz by měl být na konci každé programové pásky.

HELP

Je speciální příkaz, který můžete využít při používání interaktivního módu. Nejedná se o příkaz, který by uměl Turingův stroj. Je určen pouze pro pomoc uživatelům. Nevíte-li si v průběhu programování rady s tím, co znamená který příkaz, stačí zadat slovo ‚help'. Program se vás pak zeptá, s jakým příkazem potřebujete pomoci, a připomene vám jeho význam.

6. Využití pro výuku základů algoritmizace

Tato kapitola bude pojatá spíše jako seznam informací, které je možné studentům sdělit během toho, co si budou zkoušet práci s interaktivním módem. Tyto informace budou mít povětšinou podobu obeznámení s tím, jak daná věc funguje obecně všude v programování a zároveň poukázání na to, kde si toho můžeme všimnout při práci s tímto modelem.

I v běžném programovacím jazyce máme omezený počet klíčových slov, která můžeme využít.
- Navíc si můžeme také nadefinovat svá vlastní.

Program se vždy (nepracujeme-li například v nějakém ezoterickém programovacím jazyce[1]) vykonává odshora dolů, tak jak je napsán ve zdrojovém kódu.

I v běžných programovacích jazycích existují příkazy (neboli funkce), které vyžadují nějaké další parametry. Stejně jako v modelu tedy existují funkce, které parametr nepotřebují, funkce, které nějaký parametr potřebují, ale i funkce, které potřebují hned několik parametrů.
- Pro ujasnění je možné uvést jednoduchý příklad funkce ‚Sečti', která požaduje dva parametry a ty následně sečte a vrátí výsledek.
- Dále je možné zmínit, že funkce může podporovat i tzv. dobrovolné nebo nepovinné parametry.
  Jako příklad pro lepší pochopení je možno uvést například situaci, kdy by výše zmíněná funkce ‚Sečti' podporovala i sčítání tří čísel. V tu chvíli by byl třetí parametr nepovinný, protože funkce bude bez problémů fungovat i když ho dostane, i když ne.

Reálná paměť, kterou využívají dnešní počítače, není vlastně moc odlišná od té, kterou používáme my při práci s tímto modelem. Rozdíl je samozřejmě v počtu potřebných a používaných buněk. Další rozdíl pak je v reprezentaci dat, do reálné paměti si počítač umí ukládat pouze hodnoty 1 a 0 a až počítač si je pak přepočítá na hodnoty, se kterými dál pracuje.
- Bude-li studenty toto téma zajímat, je to pravděpodobně vhodná chvíle pro krátké představení principů binární soustavy.
  - Je vhodné popsat i jiné číselné soustavy a fakt, že princip je u všech soustav stejný.
  - Dále je zde možnost vysvětlit, jak funguje sčítání v binární soustavě.
    - Projeví-li žáci o toto téma extrémní zájem, doporučujeme jim představit i logické operace. Je na vás, zda budete používat pojmy konjunkce a disjunkce, nebo and a or. Je však žádoucí být v pojmenovávání konzistentní a případně i vysvětlit žákům v jakých oborech se které pojmenování používá.
- I v reálném programování se prakticky všechno (listy, pole, atp.) indexuje (=čísluje) od nuly, stejně jako je v modelu indexovaná paměť, co můžeme vidět na pozicích a pohybech ukazatelů.

1 Ezoterické programovací jazyky jsou často absurdní programovací jazyky navržená tak, aby se co nejvíce odlišily od běžných standardů programování. Mezi nejznámější můžeme zařadit například jazyky Brainfuck, OstraJAVA, Whitespace a LOLCODE.

7. Představení dalších způsobů programování modelu

Asi už vás napadlo, že pro nějaké větší, složitější programy je tato interaktivní forma poněkud nešťastná. Programátor nesmí nic opomenout, protože oproti psaní zdrojového kódu nemůže zpětně něco dopsat na začátek, aniž by vrátil čas nebo začal znovu. Musí tedy už od začátku naprosto přesně vědět, co a jakým způsobem chce programovat. Další možností je, že si program předem promyslí a napíše někam na papír. V tu chvíli se však můžeme tázat, není-li hloupé, aby programátor musel svůj kód psát vícekrát. Odpověď zní ano a právě proto existují i jiné možnosti, jak model programovat.

Ke zprovoznění jiného, než interaktivního módu je doporučeno chápat, jak program pracuje, toto je v základu popsáno v kapitole 5. Jak funguje tento model. K běhu programu je třeba až do ukončení příkazem exit na programové pásce provádět metodu DoNext, kterou má na starosti čtecí hlava. Pro jiný, než interaktivní mód je tedy nejprve potřeba tuto čtecí hlavu, neboli objekt třídy Head, vytvořit. K jejímu vytvoření jsou však potřeba objekty třídy Memory a Commands a k jejich vytvoření zase objekty typu List a Pointer.

V pořadí je tedy zapotřebí udělat (některé celky lze samozřejmě provést i v jiném pořadí, ale pro jasnost zde udávám jeden jednoznačný správný postup):

1. Vytvořit všechny potřebné ukazatele a dát je do jednoho listu
2. Vytvořit list celých čísel (typu int)
3. Vytvořit objekt třídy Memory, kterému dva výše zmíněné listy předáme
4. Vytvořit objekt třídy FileCommands, nebo ListCommands podle popisu výše
5. Vytvořit objekt třídy Head, kterému předáme podavač příkazů, paměť a pravdivostní hodnotu podle toho, chceme-li tisknout mezivýsledky

Příklad korektního programu pracujícího se vstupem ze souboru:

```
static void Main()
{
    Pointer prvniUkazatel = new Pointer(0);
    Pointer druhyUkazatel = new Pointer(0);
    List<Pointer> listUkazatelu = new List<Pointer> { prvniUkazatel, druhyUkazatel };
    List<int> obsahPameti = new List<int> { 1, 1, 1 };
    Memory pamet = new Memory(obsahPameti, listUkazatelu);
    FileCommands podavacPrikazu = new FileCommands("cesta/k/souboru/s/příkazy/na/programovou/pásku.txt");
    Head head = new Head(podavacPrikazu,pamet,true);

    while (true)
    {
        head.DoNext();
    }
```

Příklad korektního programu pracujícího se vstupem z listu:

```
static void Main()
{
    Pointer prvniUkazatel = new Pointer(0);
    Pointer druhyUkazatel = new Pointer(0);
    List<Pointer> listUkazatelu = new List<Pointer> { prvniUkazatel, druhyUkazatel };
    List<int> obsahPameti = new List<int> { 1, 1, 1 };
    Memory pamet = new Memory(obsahPameti, listUkazatelu);
    List<string> obsahProgramovePasky = new List<string> { "add", "add", "move", "2", "add", "exit" };
    ListCommands podavacPrikazu = new ListCommands(obsahProgramovePasky);
    Head head = new Head(podavacPrikazu,pamet,true);

    while (true)
    {
        head.DoNext();
    }
}
```

8. Představení pokročilých příkazů

Možná jste se už v interaktivním módu setkali s hlášením o tom, že nějaký příkaz nelze v tomto módu použít. Tato kapitola bude právě o příkazech, které při práci v interaktivním módu nemá smysl používat.

Prvním z těchto příkazů je příkaz Go. Díky němu můžeme například tvořit cykly. Jedná se o jeden z příkazů, které vyžadují zadání parametru. Příkaz Go přesune čtecí hlavu nad jiné políčko programové pásky. Právě to, nad které políčko se hlava přesune, specifikuje parametr a to konkrétně indexem daného políčka. (Políčka se číslují od prvního příkazu a číslování je od nuly.)

Druhým z pokročilejších příkazů je příkaz If. Tímto příkazem vytváříme podmínky. Příkaz dělá to, že když se v paměti pod prvním ukazatelem nachází nula, přeskočí další příkaz a ve vykonávání pokračuje až od následujícího. Nejlepší využití nabízí příkaz If v kombinaci s příkazem Go. Pokud příkaz Go v této kombinaci odkazuje na index paměťové pásky za aktuálním políčkem, podmínka určuje, vykonají-li se všechny příkazy až do tohoto políčka. Odkazuje-li příkaz Go na některé z předcházejících políček, rozhoduje podmínka, jestli se předcházející část kódu vykoná znovu, nebo nikoliv.

9. Využití pro výuku pokročilejších témat z oblasti algoritmizace ilých příkazů

Tato kapitola je strukturována obdobně jako kapitola číslo 5, aneb Využití pro výuku základů algoritmizace. Jedná se tedy o seznam informací, které je doporučené studentům sdělit během práce v některém z pokročilých módů.

To co se děje, když použijeme Go s odkazem na některé z předchozích políček, se nazývá cyklus. Cykly jsou spolu s podmínkami nejdůležitější postupy při algoritmizaci.

- Cykly mohou být podmíněné. Obvykle nechceme, aby nějaký cyklus v programu běžel do nekonečna. Proto obvykle mluvíme o podmíněných cyklech, které se při splnění určité podmínky zastaví.
  - Podmínkou může být například to, že cyklus proběhl už n-krát. Takovým cyklům se říká for-cykly.
  - Pokud je podmínka jiná, nazýváme takový cyklus while-cyklem.

Příkazem If tvoříme podmínky. Jak je uvedeno výše, podmínky patří spolu s cykly mezi nejdůležitější postupy při algoritmizaci.

- Ve většině programovacích jazyků je klíčové slovo pro vytvoření podmínky stejné, jako v tomto modelu, tedy If.

10. Řešené úlohy pro zamyšlení i další zpracování

Obsahem této kapitoly je soubor několika úloh pro procvičení nabitých znalostí. Úlohy se vzájemně liší v různých ohledech. Cílem většiny úloh je dosáhnout určité podoby paměti, ale jsou zde i úlohy, jejichž cílem je vypsat nějaké číslo, popřípadě čísla na výstup stroje. Je důležité dát u každé úlohy pozor, je-li specifikováno, jak má vypadat paměť při spuštění programu. Není-li to totiž jasně dáno, je možné si vstupní paměť libovolně přizpůsobit. Jaký počet a vstupní umístění ukazatelů řešitel zvolí, závisí vždy pouze na jeho uvážení. O správném řešení jednotlivých úloh pojednává následující kapitola.

1. Vytvořte program, který z paměti s pěti buňkami o hodnotě jedna vytvoří paměť, kde každá buňka bude mít hodnotu o jedna vyšší, než je její index.
2. Vytvořte program, který z paměti s pěti buňkami o hodnotě jedna vytvoří paměť, kde každá buňka bude mít hodnotu stejnou, jako její index.
3. Vytvořte program, který bude postupně vypisovat všechny mocniny čísla 2 menší, než 100. Vstupní paměť si můžete libovolně přizpůsobit. Uvažujte pouze mocniny s mocnitelem ≥ 1.
4. Vytvořte program řešící úlohu 3 z maximálně 12 příkazů.
5. Vytvořte program, který bude postupně vypisovat všechny mocniny čísla 3 menší, než 100. Vstupní paměť si můžete libovolně přizpůsobit. Uvažujte pouze mocniny s mocnitelem ≥ 1.
6. Vytvořte program, který vypíše číslo 100. Vstupní paměť má deset buněk, všechny s hodnotou 1.
7. Vytvořte program, který bude odpovídat zadání úlohy 6 a který zároveň bude mít co nejméně příkazů. Zkuste napsat kratší program, než spolužáci.
8. Popište obecný postup, jak napsat program, který v paměti z jedniček vytvoří konkrétní číslo.

11. Řešení zadaných úloh

Tato kapitola obsahuje doplňující poznámky k řešení jednotlivých úloh zadaných v předchozí kapitole. Správná řešení ve smyslu funkčních programů je k dispozici v modelu ve složce Solutions. Řešení jsou ve složce řazena tak, jako jsou zadána v předchozí kapitole. Má-li tedy pedagog zájem o řešení úlohy 4, stačí nahlédnout do souboru 4.txt. Na počátku každého vzorového programu je za znakem ‚#' zapsáno, jakou vstupní paměť program očekává. U některých programů je také specifikováno, jaké ukazatele jsou pro běh potřeba. Není-li počet a pozice ukazatelů specifikována, předpokládá se použití dvou ukazatelů, s pozicí na indexu 0.

1. Můžeme se na problém podívat tak, že každá z buněk má mít o jedna vyšší hodnotu, než buňka předchozí. Vzhledem k tomu, že z počátku mají všechny buňky hodnotu 1, stačí vždy přičíst k původní hodnotě buňky správnou hodnotu buňky předchozí. Buňka na indexu 0 má mít hodnotu 1, což už na počátku běhu programu má. Je tedy zapotřebí přesunout první ukazatel na pozici za ní a hodnotu přičíst. Tím vzniklo číslo 2. Opět se posuneme o buňku dále a znovu zavoláme příkaz Add. Ten tentokrát k 1 přičte číslo 2. Tento postup opakujeme, dokud nevytvoříme požadovaných 5 správných čísel.

2. Tento problém je v zásadě stejný, jako problém 1. Co je potřeba si uvědomit je, že číslování bude začínat až o buňku později. V minulém případě byla 1 na indexu 0 a první sečtení se ukládalo na index 1. Nyní budeme první sečtení ukládat na index 2. Pak už je jen zapotřebí odečíst 1 od 1 na prvním indexu.

3. Mocniny čísla dva vytvoříme tak, že budeme mít dva ukazatele, oba na stejném indexu a sečteme je. Tím vznikne dvojnásobek původního čísla. Byla-li tedy v buňce některá z mocnin čísla 2, vznikne vyšší mocnina tohoto čísla. Stačí tedy 1 buňka v paměti, ve které bude nejprve 1, jakožto nejnižší požadovaná mocnina. Tu nejprve vypíšeme, poté dvojnásobíme a opět vypíšeme a takto stále dokola, než narazíme na čísla větší, než 100.

4. Kdyby se zadání z ničeho nic změnilo a studenti museli vypisovat mocniny až do 1000, asi by je to poněkud vyvedlo z míry. Aby se toto v reálném světě nestávalo, používají programátoři cykly namísto psaní stejného bloku kódu několikrát pod sebe. V takovém případě stačí změnit jedno číslo a část programu se 1000krát bez problémů vykoná. Tato úloha tedy navádí k tomu, aby studenti pro tento program využili právě cyklů. Co se týká konkrétní implementace, je potřeba někde uchovávat, kolikrát se už program spustil, aby bylo možné ho včas zastavit. Avšak vzhledem k tomu, že máme k dispozici If, který kontroluje, je-li číslo 0, je praktičtější jiný postup. V jedné z buněk bude na vstupní paměti číslo, kolikrát je potřeba cyklus opakovat. Toto číslo se následně v rámci každého opakování cyklu, neboli v rámci každé iterace, sníží o 1. Když bude číslo 0, cyklus se ukončí. Toho lze osáhnout jednoduše tak, že ze příkazem If bude příkaz Go odkazující na počátek cyklu. Bude-li při vyhodnocování If toto odečtené číslo 0, příkaz Go se přeskočí a program se pomocí příkazu Exit může ukončit.

5. V této úloze je potřeba si uvědomit, že ukazatele můžeme používat libovolně. Z předešlých úloh studenti vědí, že jsou-li dva ukazatele na stejné buňce a je použit příkaz Add, číslo v dané buňce se zdvojnásobí. Stačí tedy použít tři ukazatele, aby se číslo ztrojnásobilo a bylo tedy možné vytvářet mocniny čísla 3. Když už je známa tato úvaha, stačí postupovat obdobně, jako u úlohy 3.

6. Nejjednodušší možností je využít dvě buňky, jednu nemněnou s číslem 1 a jednu, ve které budeme tvořit výsledek. Pak už stačí správně nastavit ukazatele a 99krát přičíst číslo 1.

8. Je potřeba mít k dispozici dvě buňky v paměti. V jedné buňce číslo 1 do druhé budeme ukládat mezivýsledky. Nastavíme první ukazatel na druhou zmíněnou buňku, druhý na buňku s hodnotou 1. Nyní stačí jen použít příkaz Add o jedna méněkrát, než jaké číslo je potřeba vytvořit.