Rekurze

Přednáška 2

Rekurze

- Rekurze je jeden ze způsobů řešení algoritmů
- Rekurze je popis, kdy objekt je součástí sám sebe (například: namíříme-li kameru na monitor, který zobrazuje obraz z kamery či soustava protilehlých zrcadel)
- Rekurze musí obsahovat podmínku ukončení (konečnost algoritmu)
- Rekurze je situace, kdy se jako součást řešení problému objeví stejný problém/řešení
- Většinu případů rekurzí lze převést na nerekurzivní algoritmus (a naopak)

Rekurze

- Rekurze je spojená s vysokými nároky na využití paměti zásobníku
- I vlastní kód rekurzivního volání je rozsáhlejší o režii s voláním funkce a předáváním parametrů. Tato část zabere také více času oproti realizaci cyklem
- Rekurentní algoritmy mohou vést k přehlednějšímu řešení (pro některé typy úloh)
- V programování se objevuje rekurze u funkcí a u datových typů

Rekurze

- U datových typů se jedná o struktury s uzly, které se odkazují/propojují s jinými uzly stejného typu. Proměnná reprezentující uzel, musí tedy obsahovat stejnou proměnnou, nebo několik, které na ni navazují. (Binární strom se skládá z uzlu a dvou stromů …).
- V případě, že by proměnná obsahovala přímo stejnou proměnnou, došlo by k rekurzi přímo při jejím vytváření a tím by jedna proměnná zaplnila celou paměť. Proto se pro návazné struktury musí používat odkaz (ukazatel).
- Druhým důvodem proč nemůže proměnná obsahovat sama sebe je to, že v daném místě ještě není dokončena její definice a proto se nezná její velikost (a překladač pro ni tedy nemůže vyhradit místo, které nezná a zahlásí chybu).

Rekurze

- Příklady rekurze
  - strom (z větve jdou větve, co je ještě větev? …)
  - většina her - pravidla v každém tahu jsou stejná, princip řešení je tedy stejný i když se data mění.
  - matrjošky (https://www.educative.io/collection/page/10370001/760001/1000001)
  - fraktály (https://www.root.cz/clanky/fraktaly-kolem-nas/)
  - Google.com -> zadat heslo „Recursion“

Matrjoška

Matrjoška

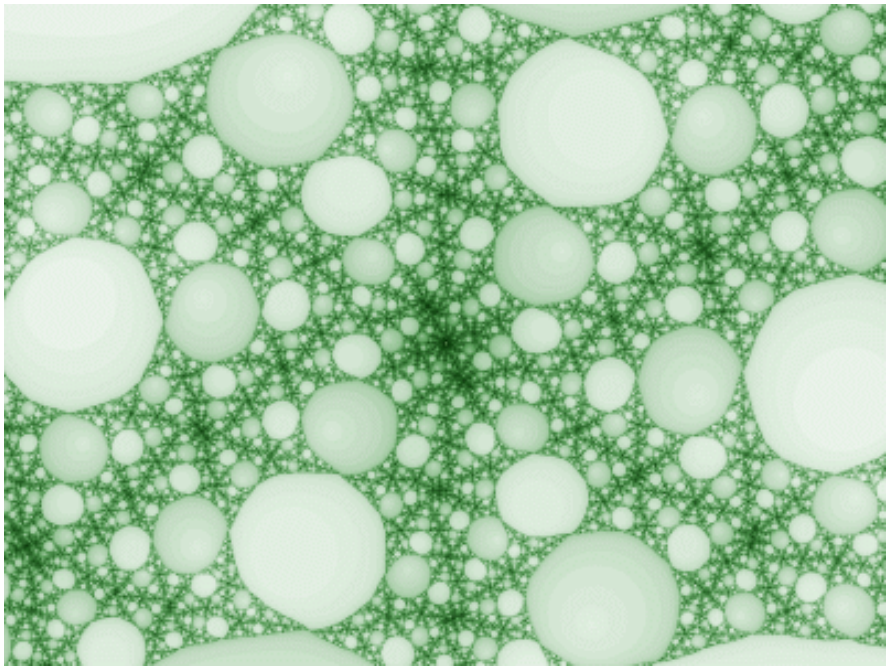
Matrjoška

Matrjoška

Fraktály

- Sobě-podobný obrazec
- Opticky složitý tvar
- Generování jednoduchými pravidly

- Mraky, sněhové vločky, řeky, cévy



Rekurze u funkcí

- Znamená to, že pro výpočet funkce je využita tatáž funkce.
- Funkce `f1` je volána zatímco jedno z předchozích volání `f1` ještě nebylo ukončeno návratem.
- Druhy:
  - **přímá rekurze** - volá se přímo
  - **nepřímá rekurze** - volá se zprostředkovaně přes jinou funkci

  - **lineární rekurze** – funkce volá sama sebe jen jednou.
    - Faktoriál
    - Hledání pozice dané hodnoty v setříděném poli.
  - **stromová rekurze** – funkce volá sama sebe několikrát
    - třídicí algoritmy:

Druhy rekurzí

- **Tail recursion** – rekurzní volání je posledním příkazem
  -> `return f()`
- **Strukturální rekurze** - volání funkce přes jinou funkci, nebo v jiném místě, než samostatně na konci
- **Backtracking** - zpětné vyčíslování rekurze (obecně funkce)
- **Mutual (indirect) recursion** - dva objekty, závisí na sobě
  - funkce volající se navzájem
  - objekty ukazující na sebe navzájem
  - např. binární strom se skládá z uzlu a dvou stromů
- **Corecursion** - duální k rekurzi (iterace), od jednoduchého ke složitému.

Obecný zápis rekurze

vstup x
test konečné podmínky,
    pokud je splněna, ukonči funkci a vrať se (do předchozí úrovně)
výpočet před rekurentním voláním
    (pro jednotlivá volání)
rekurentní volání s upraveným x
    (jedno, nebo v několika větvích)
test a zpracování návratové hodnoty
výpočet po rekurentním volání
    (jednom či více)
návrat do předchozí úrovně

Faktoriál

- použití například ve statistice (pravděpodobnost, rozložení, …)
- platí
  - n! = n.(n-1)! n > 0
  - n! = 1 n = 0
- algoritmus nerekurzivní:

```
vstup x
pokud (x == 1)
    výsledek = 1
jinak
    výsledek = 1
    pro n = 1 až x; krok 1
        výsledek = n * výsledek
```

Faktoriál

- n! = n. (n-1)! - princip rekurze, v popisu (definici) řešení se objevuje původní prvek
- podmínka ukončení: pro n = 0; 0! = 1
- algoritmus rekurzivní:

```
vstup x
pokud (x == 0)
    výsledek = 1
jinak
    výsledek = x * rekurze(x-1)
```

Faktoriál - Ukázka hierarchie volání a parametrů na zásobníku

- hodnota x -> levé tři sloupce (stav zásobníku)
- hodnota nejvíce vpravo -> hodnota x v aktuální funkci
- počáteční hodnota 2

			volající funkce	volání s parametrem 2
2			první zanoření	volání s parametrem 1
2	1		druhé zanoření	volání s parametrem 0
2	1	0	třetí zanoření	return 1
2	1		druhé zanoření	return 1 . 1 = 1
2			první zanoření	return 1 . 2 = 2
			volající funkce	obdržený výsledek = 2

Faktoriál

- Problém při výpočtu – přetečení
  - Jak zjistit přetečení?
  - Jak vypočítat maximální číslo, jehož faktoriál lze vypočítat pro daný datový typ?

Tisk řetězce - aneb cesta tam a zase zpátky

- Vytiskněte ho pomocí rekurze tak jak je
- Vytiskněte ho pomocí rekurze pozpátku
- Neznáme délku řetězce

Tisk řetězce - po řadě

```
vstupy: řetězec, aktuální pozice=0
Pokud jsi na konci řetězce, vrať se
     na předchozí úroveň
Vytiskni aktuální znak
Zavolej tento postup se stejným řetězcem,
     pozici posuň na další hodnotu v poli
```

Tisk řetězce - pozpátku

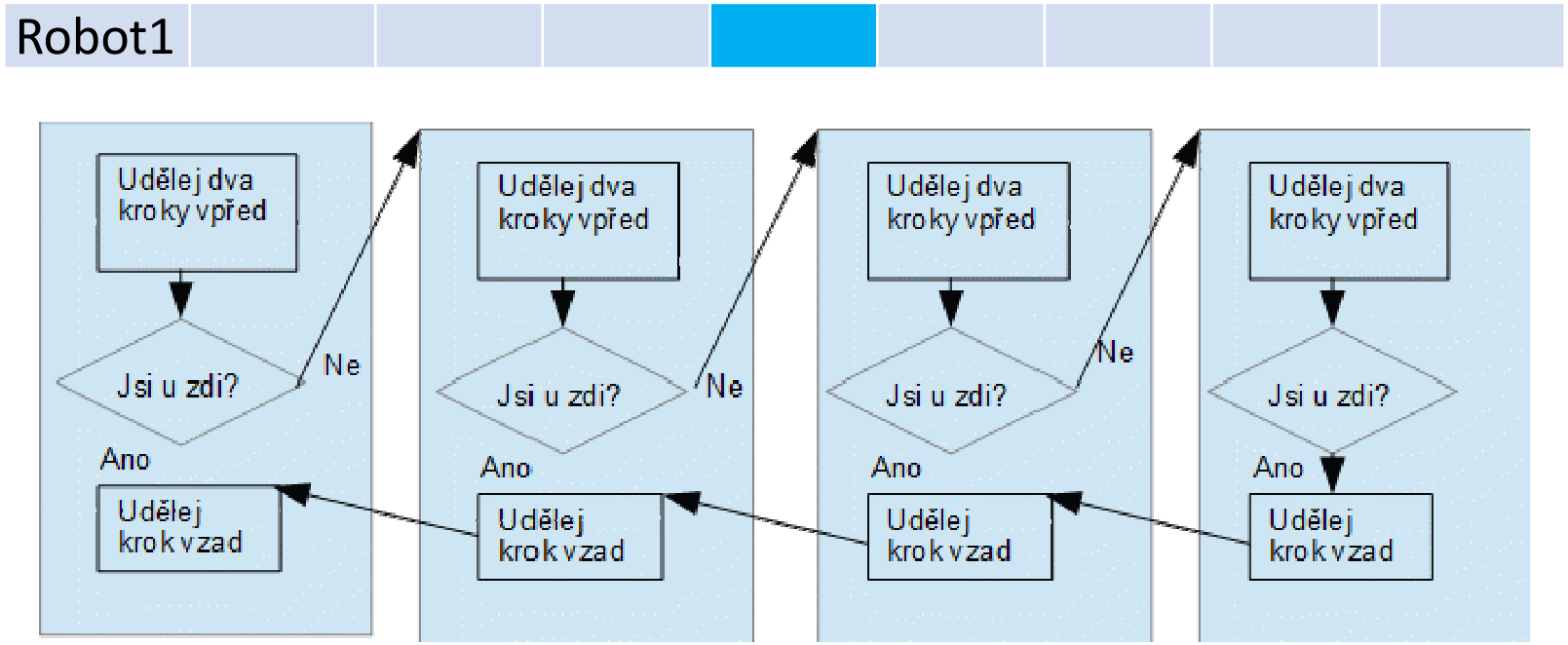
```
vstupy: řetězec, aktuální pozice=0
Pokud jsi na konci řetězce, vrať se
     na předchozí úroveň
Zavolej tento postup se stejným řetězcem,
     pozici posuň na další hodnotu v poli
Vytiskni aktuální znak
```

Robot Karel

- Definice úlohy:
  - Dojděte s figurkou na šachovnici do poloviny vzdálenosti od okraje pole/zdi
  - Robot jde doprava
  - Úplně vpravo je zeď
- Algoritmus:

```
Pokud je před tebou zeď, vrať se do předchozí
      úrovně (podmínka konečnosti)
Udělej dva kroky, pokud nenarazíš na zeď
Zavolej tento algoritmus (další úroveň)
Vrať se o jeden krok
```

Robot Karel


Robot1
Udělej dva kroky vpřed
Jsi u zdi?
Ne
Ano
Udělej krok vzad
Udělej dva kroky vpřed
Jsi u zdi?
Ne
Ano
Udělej krok vzad
Udělej dva kroky vpřed
Jsi u zdi?
Ne
Ano
Udělej krok vzad
Udělej dva kroky vpřed
Jsi u zdi?
Ano
Udělej krok vzad

Robot Karel

- Nutno dodefinovat/rozhodnout co se stane, narazí-li na okraj po prvním kroku. Nevrátí se v tom případě přesně do poloviny.

Je slovo palindrom?

- Palindrom - text, který je stejný při čtení zepředu i zezadu
- Obecně tedy test symetrie, může být i s tolerancí
- Předávání tří parametrů - rychlé plnění zásobníku, dá se zmenšit počet parametrů ?

```
Palindrom (text, začátek, delka)
Pokud je delka 1 nebo 0 znaků vrať true
Pokud nejsou první a poslední stejné vrať false
Zavolej fci Palindrom (text, začátek+1, delka-2)
```

Půlení intervalů

- je možné ji považovat za grafickou metodu
- hledání kořenů rovnic (průsečík s nulou), nebo pro extrémy
- průsečík s nulou – máme-li kladnou a zápornou hodnotu (spojité rovnice), musí mezi nimi být průsečík s nulou (kořen)
- vypočteme hodnotu uprostřed stávajících a nahradíme jí tu se stejnou polaritou hodnoty
- ukončení při dostatečně přesném přiblížení k průsečíku s nulou
  - našli jsme přímo nulu
  - ukončení pro hodnotu y v dané toleranci od nuly („strmé“ průsečíky)
  - ukončení pro danou vzdálenost krajních bodů od sebe („ploché“ průsečíky)
  - ukončení z důvodu velkého množství kroků

Půlení intervalů - algoritmus

vstup – dvě souřadnice x a funkce y = f(x)

vypočteme hodnoty v krajních bodech x

tolerance hodnota = velká

dokud tolerance hodnoty = velká
- -vypočti souřadnici x uprostřed mezi krajními body
- -vypočti hodnotu y v tomto bodě
- -nahraď krajní bod, který má hodnotu se stejným znaménkem, vypočteným středem
- -spočítej novou tolerance hodnoty

výsledek je x s menší hodnotou

Hanojské věže

- Demonstrace úlohy:
  - máme tři pozice a na první je pyramida ze 4 (například) disků
  - úkolem je přesunout disky na poslední pozici
  - podmínky řešení
    - disky přenášíme po jednom
    - nesmíme umístit větší disk na menší
- Zkuste tuto úlohu vyřešit
- Jak postupovat?

Hanojské věže

- Možné řešení:
  - přenes všechny disky až na spodní disk na pomocnou pozici
  - přesuň zbývající spodní disk na cílovou pozici
  - přenes disky (přenášené v prvním kroku) z pomocné pozice na cílovou
- Jak přenést všechny disky až na spodní ?
  - výška pyramidy = výška pyramidy - 1
  - zavolám stejný algoritmus
- Nepřipomíná vám to něco?

Hanojské věže - algoritmus

```
Vstup:
   počet disků
   označení pozicí: Zdroj, Pomocná, Cíl
Algoritmus:
   Pokud je více jak jeden disk
      Přesun(počet disků-1; pozice Zdroj, Cíl, Pomocný)
   Zbývající samostatný disk ze Zdroj do Pomocný
   Pokud je více jak jeden disk
      Přesun(počet disků-1; pozice Cíl, Pomocný, Zdroj)
```

Hanojské věže - příklad

```
  XIX      I       I
 XXIXX     I       I
XXXIXXX    I       I
```

```
          0->1
   I       I       I
   I       I       I
XXXIXXX  XXIXX    XIX
```

```
          0->2
   I       I       I
   I      XIX      I
   I     XXIXX  XXXIXXX
```

```
          1->2
   I       I       I
   I       I     XXIXX
  XIX      I    XXXIXXX
```

```
          0->2
   I       I       I
 XXIXX     I       I
XXXIXXX    I      XIX
```

```
          2->1
   I       I       I
   I      XIX      I
XXXIXXX  XXIXX     I
```

```
          1->0
   I       I       I
   I       I       I
 XIX    XXIXX  XXXIXXX
```

```
          0->2
   I       I      XIX
   I       I     XXIXX
   I       I    XXXIXXX
```

Hornerovo schéma

- použití
  - zjištění hodnoty polynomiální funkce v bodě
  - při převodu mezi číselnými soustavami
- hodnota polynomu

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- v tomto vyčíslení je (n-1) násobení a (n-1) sčítání. Dále je nutné získat mocniny x (při iterativním počítání dalších n operací; při každé mocnině n(n-1)/2).
- při úpravě Hornerovým schématem:

$$f(x) = (..((a_n x + a_{n-1})x + \cdots + a_1)x + a_0$$

- zůstane počet násobení a sčítání, není ale nutné počítat mocniny. Tím se také ušetří pomocná proměnná.
- Převeďte tento algoritmus na rekurentní

Hornerovo schéma

- pa – pole koeficientů
- degree – řád polynomu
- x – hodnota bodu

```
double compute_by_horner(size_t degree, double pa[], double x)
        {
        if (degree == 0)
                return pa[0];
        return compute_by_horner(degree-1, pa, x) * x + pa[degree];
        }
```

- Jaké musí být pořadí koeficientů v poli?

Shrnutí

- Rekurentní je taková funkce, která volá sama sebe
- Obvyklý průběh
  - vstup x
  - test konečné podmínky (pokud ano, tak)
    - ukonči aktuální průběh
    - návrat o úroveň výše
  - výpočet
  - rekurentní volání s upraveným x
  - výpočet
  - návrat do předchozí úrovně

Děkuji za pozornost

Bonus

```
Něco_se_nepovedlo (chyba, osoba):

Pokud není definován (osoba.nadřízený)
      Přijmi_následky(osoba, chyba)
Jinak
      Přiznej(osoba, chyba)
      Vyšiluj(osoba.nadřízený, chyba)
      Nabídni_výpověď(osoba)
      Něco_se_nepovedlo (chyba, osoba.nadřízený)
```

Video: https://www.youtube.com/watch?v=BlCI8iXmm_o