18 Rekurze, základní schéma, základní vlastnosti, souvislost rekurze a iterace, vlastnosti implementace, efektivita. (A4B36ALG)

7.června 2012, Václav Burda

Co říci na začátek:
Rekurze nám dáva možnost definice něčeho nekonečného konečným algoritmem. Rekurzivní volání funkce je tedy takové volání, ke kterému dojde ve chvíli, kdy funkce samotná ještě nedokončila svou činnost.

18.1 Základní schéma

1. inicializační algoritmus - rekurzivní funkce většinou potřebuje nějakou inicializační hodnotu, se kterou zahájí výpočet. To většinou zajišťuje jiná funkce, která není rekurzivní a která danou rekurzivní funkci volá a předává hodnotu, se kterou rekurzivní výpočet začíná
2. kontrola, zda vstupní parametr odpovídá stanoveným podmínkám. Pokud hodnota parametru odpovídá, rekurzivní funkce provede výpočet a vrátí hodnotu
3. rekurzivní funkce redefinuje řešení problému tak, že jej nějakým způsobem zmenší, zjednoduší nebo rozloží na dílčí podproblémy
4. volání funkcí, které řeší daný podproblém - tady nastává přímé nebo nepřímé volání sebe sama
5. sestavení výsledku
6. vrácení vypočtené hodnoty

18.2 Základní vlastnosti

Dělení rekurze 1:

- Lineární - v každé iteraci dojde k pouze jednomu zavolání sebe sama.

- Stromová - v každé iteraci dochází k více dělení na podprogramy.

Dělení rekuzre 2:

- Přímá - v dáné rekurzivní funkci dojde k volání téže funkce.
- Nepřímá - rekurzivní volání, ke kterému dochází přes další funkci (např. A volá B a B volá A).

Volání může probíhat přímo nebo nepřímo. Každá rekurze se nechá přepsat na nerekurzivní tvar například pomocí zásobníku (udržujeme si datovou strukturu, která nahrazuje implicitní hardwarový zásobník použitý při rekurzivním volání).

18.3 Vlastnosti implementace

Rekurzí lze dosáhnout zjednodušení řešené úlohy, ale je nutné brát v úvahu následující úskalí:

- Každé další volání rekurzívní funkce prohlubuje zapouzdření a tedy zabírá paměťový prostor a procesorový čas.
- V případě chybně ošetřeného ukončení rekurze dochází k vučerpání veškérých volných prostředků a k havárii programu.
- Použití může vést ke zvýšení složitosti výpočtu.

Rekurze musí obsahovat:

- Ukončovací podmínku.
- V každém kroku rekurze musí dojít ke zjednodušení problému.
- V algoritmu se nejprve musí ověřit, zda nenastala koncová situace. Když ne, provede se rekurzivní krok.

18.4 Souvislost rekurze a iterace

Příklad výpočtu Faktoriálu pomocí rekurze:

```
function Faktorial ( integer X )
   if X < 0 then return "Chybny argument" end if
   if X = 0 then return 1 end if
   return Faktorial(X-1) * X
end function
```

a pomocí iterace:

```
function Faktorial (integer X)
     integer nfact
     if X < 0 then return "Chybny argument" end if
     nfact = 1
     for i = 1 to X do
           nfact = nfact * i
     end for
     return nfact
end function
```

18.5 Efektivita

Hlavní nevýhodou rekurze je u stromové struktury několikanásobné počítání stejných výpočtů (v případě Fibonacciho posloupnosti jde o exponenciální složitost). Tento nedostatek lze vyřešit zavedením tabulky (pole), do které ukládáme již spočítané výsledky (ale potřebujeme další paměť pro tuto tabulku).

18.6 Zdroje

- http://cs.wikipedia.org/wiki/Rekurze#Efektivita_rekurzivn.C3.ADch_algoritm.C5.AF
- http://cs.wikipedia.org/wiki/Rekurzivní_funkce_(programování)
- http://service.felk.cvut.cz/courses/Y36ALG/balik/Alg5Balik2008.pdf
- http://www.algoritmy.net/article/23275/Rekurze-11