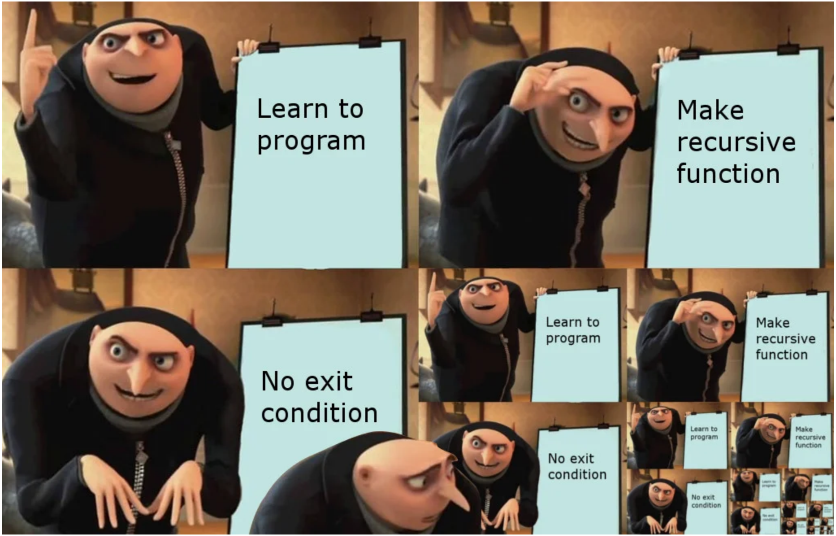
DELTA TopGun

(06) Rekurze

Luboš Zápotočný, Michal Havelka, Tomáš Faltejsek

2022

Obsah


Learn to program
Make recursive function
No exit condition
Learn to program
Make recursive function
No exit condition
Learn to program
Make recursive function
No exit condition

Rekurze

Funkci či procedura, která volá sama sebe.

Rekurze

Funkci či procedura, která volá sama sebe.

Existuje i nepřímá rekurze, ve které se rekurzivní krok může provést až po několika zavolání jiných funkcí.

Rekurze

Funkci či procedura, která volá sama sebe.

Existuje i nepřímá rekurze, ve které se rekurzivní krok může provést až po několika zavolání jiných funkcí.

Většina rekurzivních algoritmů lze zapsat lineárním způsobem.

Rekurze

Funkci či procedura, která volá sama sebe.

Existuje i nepřímá rekurze, ve které se rekurzivní krok může provést až po několika zavolání jiných funkcí.

Většina rekurzivních algoritmů lze zapsat lineárním způsobem.

Rekurze je jádro funkcionálního programování.

Rekurze a stack

Rekurentní rovnice

$$F(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 1, & \text{pro n} = 1 \\ F(n-1) + F(n-2), & \text{pro n} \geq 2 \end{cases}$$

Rekurentní rovnice

$$F(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 1, & \text{pro n} = 1 \\ F(n-1) + F(n-2), & \text{pro n} \geq 2 \end{cases}$$

Fibonacciho číslo

Rekurentní rovnice

$$F(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 1, & \text{pro n} = 1 \\ F(n-1) + F(n-2), & \text{pro n} \geq 2 \end{cases}$$

Fibonacciho číslo

$$F(n) = \begin{cases} 1, & \text{pro n} = 0 \\ n * F(n-1) & \text{pro n} > 0 \end{cases}$$

Faktoriál

Rekurentní rovnice

$$F(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 1, & \text{pro n} = 1 \\ F(n-1) + F(n-2), & \text{pro n} \geq 2 \end{cases}$$

Fibonacciho číslo

$$F(n) = \begin{cases} 1, & \text{pro n} = 0 \\ n * F(n-1) & \text{pro n} > 0 \end{cases}$$

Faktoriál

$$T(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 2 * T(n-1) + 1, & \text{pro n} > 0 \end{cases}$$

Rekurentní rovnice

$$F(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 1, & \text{pro n} = 1 \\ F(n-1) + F(n-2), & \text{pro n} \geq 2 \end{cases}$$

Fibonacciho číslo

$$F(n) = \begin{cases} 1, & \text{pro n} = 0 \\ n * F(n-1) & \text{pro n} > 0 \end{cases}$$

Faktoriál

$$T(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 2 * T(n-1) + 1, & \text{pro n} > 0 \end{cases}$$

Počet nutných přemístění při přesunu Hanojské věze

Výpis spojového seznamu od konce

Koncová rekurze

Koncová rekurze je taková rekurze, ve které je jako poslední instrukce rekurzivní volání sama sebe.

Koncová rekurze

Koncová rekurze je taková rekurze, ve které je jako poslední instrukce rekurzivní volání sama sebe.

Poslední instrukce musí být doopravdy pouze jedno zavolání funkce.

Koncová rekurze

Koncová rekurze je taková rekurze, ve které je jako poslední instrukce rekurzivní volání sama sebe.

Poslední instrukce musí být doopravdy pouze jedno zavolání funkce.

Typická implementace Fibonacciho čísla není koncová rekurze, protože poslední instrukcí je součet dvou čísel.

Fibonacciho číslo

```c
#include <stdio.h>

unsigned long long fib(int n) {
    if (n < 2) return n;
    return fib(n-1) + fib(n-2);
}

int main () {
    printf("%llu\n", fib(45));
}
```

Fibonacciho číslo

```c
#include <stdio.h>

unsigned long long fib(int n) {
    if (n < 2) return n;
    return fib(n-1) + fib(n-2);
}

int main () {
    printf("%llu\n", fib(45));
}
```

Časová náročnost

time ./fib

real 0m3.019s

Fibonacciho číslo - koncová rekurze

```
#include <stdio.h>

unsigned long long fib_tail(int a, int b, int n) {
    if (n == 0) return a;
    return fib_tail(b, a+b, n-1);
}

unsigned long long fib(int n) {
    return fib_tail(0, 1, n);
}

int main () {
    printf("%llu\n", fib(45));
}
```

Fibonacciho číslo - koncová rekurze

```
#include <stdio.h>

unsigned long long fib_tail(int a, int b, int n) {
    if (n == 0) return a;
    return fib_tail(b, a+b, n-1);
}

unsigned long long fib(int n) {
    return fib_tail(0, 1, n);
}

int main () {
    printf("%llu\n", fib(45));
}
```

Časová náročnost

time ./fib-tail

real 0m0.022s

Koncová rekurze a stack

Při provádění koncové rekurze může být původní stack frame přepoužit

Koncová rekurze a stack

Při provádění koncové rekurze může být původní stack frame přepoužit

Pouze se změní registry na hodnoty nově předaných parametrů a skočí se na začátek původního rámce.

Rozděl a panuj

Programovací paradigma, které rozděluje problém na několik menších podproblémů, které už je jednodušší vyřešit.

Rozděl a panuj

Programovací paradigma, které rozděluje problém na několik menších podproblémů, které už je jednodušší vyřešit.

Povahově to bývají rekurzivní algoritmy.

Rozděl a panuj

Programovací paradigma, které rozděluje problém na několik menších podproblémů, které už je jednodušší vyřešit.

Povahově to bývají rekurzivní algoritmy.

Zároveň je nutné do časové složitosti zakomponovat režii pro spojení výsledků podproblemů.

Odvození asymptotické složitosti

Merge sort

Odvození asymptotické složitosti

Merge sort

- Rozdělí neseřazenou množinu dat na dvě podmnožiny o přibližně stejné velikosti.

Odvození asymptotické složitosti

Merge sort

- Rozdělí neseřazenou množinu dat na dvě podmnožiny o přibližně stejné velikosti.
- Seřadí obě podmnožiny.

Odvození asymptotické složitosti

Merge sort

- Rozdělí neseřazenou množinu dat na dvě podmnožiny o přibližně stejné velikosti.
- Seřadí obě podmnožiny.
- Spojí seřazené podmnožiny do jedné seřazené množiny.

Odvození asymptotické složitosti

Merge sort

- Rozdělí neseřazenou množinu dat na dvě podmnožiny o přibližně stejné velikosti.
- Seřadí obě podmnožiny.
- Spojí seřazené podmnožiny do jedné seřazené množiny.

[Wikipedia]

Odvození asymptotické složitosti

Master theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Odvození asymptotické složitosti

Master theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Případ 1:

$$f(n) = \mathcal{O}\left(n^{log_b(a)-\epsilon}\right) \implies T(n) = \Theta\left(n^{log_b(a)}\right)$$

Odvození asymptotické složitosti

Master theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Případ 1:

$$f(n) = \mathcal{O}\left(n^{log_b(a)-\epsilon}\right) \implies T(n) = \Theta\left(n^{log_b(a)}\right)$$

Případ 2:

$$f(n) = \Theta\left(n^{log_b(a)}\right) \implies T(n) = \Theta\left(n^{log_b(a)} log(n)\right)$$

Odvození asymptotické složitosti

Master theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Případ 1:

$$f(n) = \mathcal{O}\left(n^{log_b(a)-\epsilon}\right) \implies T(n) = \Theta\left(n^{log_b(a)}\right)$$

Případ 2:

$$f(n) = \Theta\left(n^{log_b(a)}\right) \implies T(n) = \Theta\left(n^{log_b(a)} log(n)\right)$$

Případ 3:

$$f(n) = \Omega\left(n^{log_b(a)+\epsilon}\right) \implies T(n) = \Theta\left(f(n)\right)$$

[Wikipedia]

Odvození asymptotické složitosti

Jakou asymptotickou složitost má tedy merge sort?

Násobení dvou čísel

Mějme dvě n-ciferná čísla x a y, která chceme vynásobit

n je mocnina dvou

Obě čísla rozdělíme na horních $\frac{n}{2}$ a dolních $\frac{n}{2}$ cifer

Násobení dvou čísel

Mějme dvě n-ciferná čísla x a y, která chceme vynásobit
n je mocnina dvou
Obě čísla rozdělíme na horních $\frac{n}{2}$ a dolních $\frac{n}{2}$ cifer

$$
\begin{aligned}
x &= x_U \cdot 10^{\frac{n}{2}} + x_L, \\
y &= y_U \cdot 10^{\frac{n}{2}} + y_L,
\end{aligned}
$$

Násobení dvou čísel

Mějme dvě n-ciferná čísla x a y, která chceme vynásobit

n je mocnina dvou

Obě čísla rozdělíme na horních $\frac{n}{2}$ a dolních $\frac{n}{2}$ cifer

$$x = x_U \cdot 10^{\frac{n}{2}} + x_L,$$

$$y = y_U \cdot 10^{\frac{n}{2}} + y_L,$$

Výsledný součin $x \cdot y$ dvou n-ciferných čísel x a y pak můžeme poskládat ze 4 součinů $\frac{n}{2}$ ciferných čísel:

Násobení dvou čísel

Mějme dvě n-ciferná čísla x a y, která chceme vynásobit

n je mocnina dvou

Obě čísla rozdělíme na horních $\frac{n}{2}$ a dolních $\frac{n}{2}$ cifer

$$x = x_U \cdot 10^{\frac{n}{2}} + x_L,$$

$$y = y_U \cdot 10^{\frac{n}{2}} + y_L,$$

Výsledný součin $x \cdot y$ dvou n-ciferných čísel x a y pak můžeme poskládat ze 4 součinů $\frac{n}{2}$ ciferných čísel:

$$x \cdot y = x_U \cdot y_U \cdot 10^n + (x_U \cdot y_L + x_L \cdot y_U) \cdot 10^{\frac{n}{2}} + x_L \cdot y_L$$

Násobení dvou čísel

Mějme dvě n-ciferná čísla x a y, která chceme vynásobit
n je mocnina dvou
Obě čísla rozdělíme na horních $\frac{n}{2}$ a dolních $\frac{n}{2}$ cifer

$$x = x_U \cdot 10^{\frac{n}{2}} + x_L,$$
$$y = y_U \cdot 10^{\frac{n}{2}} + y_L,$$

Výsledný součin $x \cdot y$ dvou n-ciferných čísel x a y pak můžeme poskládat ze 4 součinů $\frac{n}{2}$ ciferných čísel:

$$x \cdot y = x_U \cdot y_U \cdot 10^n + (x_U \cdot y_L + x_L \cdot y_U) \cdot 10^{\frac{n}{2}} + x_L \cdot y_L$$

Jakou asymptotickou složitost má tedy tento styl násobení?

Karacubovo násobení

Vylepšení předchozího snímku.

Karacubovo násobení

Vylepšení předchozího snímku.

$$x_U \cdot y_L + x_L \cdot y_U = (x_U + x_L) \cdot (y_U + y_L) - x_U \cdot y_U - x_L \cdot y_L$$