11MAG – 2. cvičení
Výpočetní složitost

13. října 2014

1 Měření času [5 minut]

V Matlabu měříme dobu, strávenou vykonáváním nějaké funkce, buď pomocí profileru (příkaz `profile`, nebo příkazy `tic` a `toc`). V Pythonu slouží k podobným účelům modul `timeit`. V C/C++ se pohybujete na nízké úrovni systémových volání, nějaké informace naleznete například na stránce `http://stackoverflow.com/questions/307596/time-difference-in-c`.

2 Vykreslování grafů [5 minut]

Pro vykreslení použijte v Matlabu funkci `plot`, v Pythonu modul `pyPlot`, v ostatních jazycích dle úvahy (pro C++ by asi mohl jít použít *koolplot*, `http://koolplot.codecutter.org/`).

3 Výpočet časové náročnosti Fibonacciho posloupnosti [80 minut]

Podle přednášek postupně naprogramujte všechna schémata

1. Algoritmus 1 (přímý výpočet pomocí zlatého řezu) jako funkci `y=fibo_1(n)`,
2. Algoritmus 2 (rekurentní) jako funkci `y=fibo_2(n)`,
3. Algoritmus 3 (dynamický program shora dolů) jako funkci `y=fibo_3(n)`,
4. Algoritmus 4 (dynamický program zdola nahoru) jako funkci `y=fibo_4(n)`,
5. jako bonus i Algoritmus 5 (maticový) jako funkci `y=fibo_5(n)`.

U každé implementace měřte čas výpočtu pro různá n a výsledek vykreslete do grafu. Porovnejte výstup s odhadem $\mathcal{O}(n)$ z přednášek.

Jedna z možností, jak měřit čas výpočtu pro různá n v Matlabu, je tento:

```
function times = measure_execution_time(fn_handle, nvals)
% fn_handle ... function handle to the function fibo_#(n)
% nvals ..... vector of values to use for ‘n‘

% initialise output
times=zeros(sizeof(nvals));

% number of repetitions
reps = 5;

% pick ‘n‘ from the vector, call the function several times
% and compute the average execution time
for pos = 1:lenght(nvals)
    % we could have iterated directly over ‘nvals‘, but we
    % need the position of the element as well ...
    n = nvals(pos)
    % initialise the average execution time
    exe_time = 0;
    % the execution time will be averaged over ‘reps‘ cases
    for i = 1:reps
        % start timer
        tic;
        % execute the function
        y = fn_handle(n);
        % compute the average execution time
        exe_time = exe_time + toc/reps;
    end
    times(pos) = exe_time;
end
```

Při implementaci si dejte pozor na to, že rekurentní výpočet (Algoritmus 2) má velmi prudce rostoucí časovou složitost, není jej proto vhodné zkoušet pro příliš velká n – podle zkušeností stačí $n = 2, 4, 6, 10, 14, 18, 22, 26$. Pro ostatní algoritmy použijte $n = 2^k$, kde $k = 1, 2, 3, \ldots, 10$.