Digitální učební materiál

Číslo projektu:	CZ.1.07/1.5.00/34.0548
Název školy:	Gymnázium, Trutnov, Jiráskovo náměstí 325
Název materiálu:	VY_32_INOVACE_144_IVT
Autor:	Ing. Pavel Bezděk
Tematický okruh:	Algoritmy
Datum tvorby:	červenec 2013
Ročník:	4. ročník a oktáva
Anotace:	Algoritmus IV. – Výpočetní složitost algoritmu
Metodický pokyn:	Při výuce nutno postupovat individuálně. Části DUM – „ Pro hloubavé“ jsou určeny pro zájemce o studium na technických a matematicko-fyzikálních oborech vysokých škol.

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Autor	**Ing. Pavel Bezděk**		
Vytvořeno dne	**13. 7. 2013**		
Odpilotováno dne	**2. 12. 2013**	**ve třídě**	**8.Y**
Vzdělávací oblast	**Informatika a informační a komunikační technologie**		
Vzdělávací obor	**Informatika a výpočetní technika**		
Tematický okruh	**Algoritmus**		
Téma	**Algoritmus IV. - Výpočetní složitost algoritmů**		
Klíčová slova	**Algoritmus, časová složitost algoritmu**		

Výpočetní složitost algoritmů

Algoritmická analýza

Algoritmická analýza je ***rozložení*** určitého postupu ***na konečný počet přesně určených kroků*** (operací).

Výsledkem algoritmické analýzy je algoritmus.

Krok algoritmu je operace proveditelná v konstantním čase.
Např. **aritmetická operace** (+,-,….), **porovnání dvou hodnot** (čísel), **přiřazení** (pro jednoduché typy, ale ne pro pole),…….

Složitost algoritmu

Pokud řešíme nějakou programátorskou úlohu, často nás napadne více různých řešení a potřebujeme se rozhodnout, které z nich je „nejlepší“. Abychom to mohli posoudit, potřebujeme si zavést měřítka, podle kterých budeme různé algoritmy porovnávat.

Nás **u každého algoritmu budou zajímat dvě vlastnosti: čas, po který algoritmus běží, a paměť, kterou při tom spotřebuje.**

1. **Čas nebudeme měřit v sekundách (protože stejný program na různých počítačích běží rozdílnou dobu), ale v počtu provedených operací. Pro jednoduchost budeme předpokládat, že aritmetické operace, přiřazování, porovnávání apod. nás stojí jednotkový čas.** Ona to není úplná pravda, tyto operace se ve skutečnosti přeloží na procesorové instrukce, které se teprve zpracovávají. Ale nám postačí vědět, že těch instrukcí bude vždy konstantní počet.

2. **Množství použité paměti můžeme zjistit tak, že prostě spočítáme, kolik bajtů paměti náš program použil.** Nám obvykle bude stačit menší přesnost, takže všechna čísla budeme považovat za stejně velká a velikost jednoho prohlásíme za jednotku prostoru.

Budeme proto oba parametry algoritmu určovat v závislosti na velikosti vstupu a hledat funkci, která nám tuto závislost popíše.

Časová složitost algoritmu **„doba výpočtu“ – měří se počtem instrukcí.**

Je to funkce, která každé hodnotě N udávající velikost konkrétního řešeného problému**, přiřazuje počet operací vykonaných při výpočtu daného algoritmu. Tato funkce je zpravidla rostoucí (ale může být i neklesající).**

Časová složitost algoritmu – funkce, která vyjadřuje, kolik kroků (operací) max. udělá daný algoritmus pro vstup velikosti n.

Paměťová (prostorová) složitost algoritmu **paměťové nároky – měří se v bytech nebo počtem jednoduchých proměnných, které budou při výpočtu zapotřebí.**

Závislost paměťových nároků algoritmu na velikosti řešeného problému nebo vstupních dat.

Časová složitost problému – je **časová složitost „nejoptimálnějšího“ algoritmu, který řeší daný problém**, tedy

„*Složitost problému* je složitost *nejlepšího* algoritmu, který ho řeší.“

Podobné souvislosti platí i pro paměťovou složitost programu.

Časová složitost algoritmu x paměťová složitost algoritmu

Nejlepší algoritmus má nejmenší složitost.

Ale časová a paměťová složitost jdou často proti sobě. V řadě případů platí, že čím více času se snažíme ušetřit, tím více paměti nás to pak stojí, kvůli chytré reprezentaci dat v paměti a různým vyhledávacím strukturám.

Musíme si vybrat, zda-li chceme menší složitost časovou nebo paměťovou. Který z těchto faktorů je pro nás důležitější, se musíme rozhodnout vždy u konkrétního příkladu. Nás u valné většiny algoritmů bude nejdříve zajímat časová složitost a až pak složitost paměťová.

Budeme se tedy snažit, aby algoritmus měl co nejmenší časovou složitost, i když tím často zvedneme paměťovou složitost.

Paměti mají totiž dnešní počítače dost, a tak se málokdy stane, že vymyslíme algoritmus, který má dokonalý čas, ale nestačí nám na něj paměť.

Ale přesto si musíme dávat pozor na paměťová omezení, např. šachové algoritmy.

Program faktoriál v C++

```
/* FAKTORIAL */
#include <iostream>         /* hlavickovy soubor vstupy a vostupy */
#include <math.h>           /* hlavickovy soubor matematicky */
using namespace std;
int  f,fak;
long double faktorial;
void faktor();
int main()
{
faktor();
                  return 0;
}
void faktor()
{cout<<endl;
 cout<<("   Zadej hodnotu cisla ( maximalne cislo 1754 ),")<<endl;
 cout<<("   pro ktere chces vypocitat faktorial: ");
cin >>fak; cout<<endl;
if (fak>1754){cout<<("  Byla prekrocena hodnota 1754!")<<endl;
            cout<<("  Vysledna hodnota faktorialu je tak velka, ze ji PC uz nedokaze
spocitat!!")<<endl;
            cout<<("  Cislo ma vice nez 4930 cislic!!!")<<endl;
            cout<<("   ")<<fak<<"! = chyba"<<endl;   }
else {
faktorial=1;
for  (f=1;f<=fak;f++)
   {faktorial=faktorial*f;
   }cout<<endl;
    cout<<("       Faktorial cisla:")<<endl;
    cout<<"       "<<fak<<"! = "<<faktorial<<endl;}
}
```

Program Faktoriál Pascal

```
Program Faktorial;
uses Crt;
var n,i:integer;  f,faktorialN:extended;
              {extended má rozsah 10^-4932.. 10^4932}
begin
 writeln;  writeln; writeln('Program Faktorial');
 writeln; writeln('Maximalni hodnota: 1754!');
 writeln('Hodnoty vetsi nez 1754!, jsou moc velke i pro pocitac!');
 writeln('Po zadani hodnoty N stiskni enter!');
 writeln;  write('Zadej N: ');
 readln(n);
 if n<=1754 then begin
                  writeln('Spravne zadana hodnota z mnoziny {1..1754}'); writeln;
                  f:=1;
                  for i:=1 to n do f:=f*i;
                  faktorialN:=f;
                  writeln('Faktorial:  ',n,'! = ',faktorialN);
                      end
              else writeln('Hodnota N mimo mnozinu {1..1754}');
 writeln;   writeln('Po precteni vysledku stiskni enter!');
 repeat until keypressed;
 end.
```

Program faktoriál s funkcí

ale pouze do 170!

```
program faktorial;
uses Crt;
var x:integer;
 function fact(n:integer): double;    {double má rozsah  10^-324.. 10^308}
      var f:double; i:integer;
            begin f:=1; for i:=1 to n do f:=f*i; fact:=f; end;
begin
clrscr; {   vymaze obrazovku}
writeln; writeln(' Program Faktorial');
writeln; writeln(' Jaky faktorial chces zjistit?  ');
writeln(' Maximalne muzes zjistit hodnotu 170! '); write(' Zadej cislo:');
read(x); writeln('--------------');
write('  ',x,'!=');  write(fact(x));
repeat until keypressed;
end.
```

Časová a paměťová složitost faktoriálu

Časová

Jednotka času = jedna jednoduchá operace, např. přiřazení.

Odhadneme kolik operací přiřazení provedeme:

1. f:=1; 1 přiřazení
2. f:=f*i; (**for** i:=1 **to** n **do** f:=f*i; od 1 do n) 1 x n n přiřazení

Časová složitost = n+1 závisí lineárně na n lineární

Paměťová

Kolik program bude potřebovat paměťových buněk ? Nebudeme rozlišovat jejich velikost.

Uložení hodnot v proměnných x; f; i; n; fact; potřebujeme 5 paměťových buněk

Paměťová složitost = 5 nezávisí na n konstantní

Použité zdroje

BÖHM, Martin. *Programátorská kuchařka: Recepty z programátorské kuchařky* [online]. Praha: KSP MFF UK Praha, 2011/2012 [cit. 2013-07-13]. KSP, Korespondenční seminář z programování: Programátorské kuchařky, 24. ročník KSP. Dostupné z: http://ksp.mff.cuni.cz/tasks/24/cook1.html