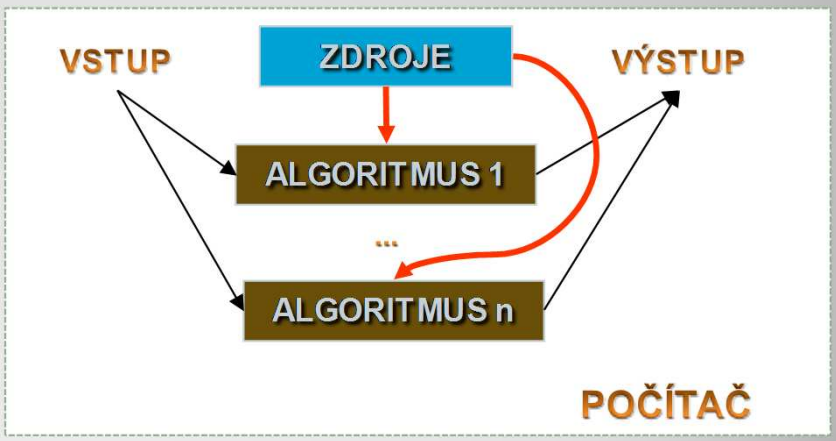
KIV/ZEP - 2011

Posuzování algoritmů, časová a paměťová složitost, základní tipy a triky pro urychlení

Definice

- Úkol: nalézt pro vstupní data taková výstupní data odpovídající řešení daného problému
  - Např. nalézt průnik dvou polygonů, atd.
- Více různých algoritmů k dispozici
  - různě kvalitní výstup
- Potřebné zdroje
  - specifický hardware
  - čas, peníze



Složitost algoritmů

- Časová vs. paměťová složitost
  - obvykle měníme paměť za čas a obráceně
- Složitost v nejhorším vs. očekávaném (průměrném) případě
  - *Příklady: nalezení minima, nalezení extrémů*
- Skutečná vs. asymptotická složitost
  - Pozor! na konstanty n_0 a c: algoritmus s horší asymptotickou složitostí může být výhodnější

Složitost algoritmů

- Algoritmická složitost nemusí nutně znamenat v praxi ještě vítězství, konstanty, využití cache a optimalizace kódu překladačem jsou důležité
  - nalezení extrémů párováním je pomalé!
  - sekvenční hledání může být rychlejší než binární (PPA1), lepší je van Emde Boas (vEB) tree

	Search time (ns)		
8	50	90	40
64	180	150	70
512	1,200	230	100
4,096	17,000	320	160
	Linear	Binary	vEB

- Rekurentní formule
  - *Příklady: Tower of Hanoi*
- Rozděl a panuj (D&C)
  - rozděl velký problém na menší
  - vyřeš menší a sluč výsledky

Složitost algoritmů

- Master Theorem
  - Nechť $T(n)=a\cdot T(n/b)+f(n)$ a $T(1)=\Theta(1)$, pak:
    - $T(n)=\Theta(n^{\log_b(a)})$, je-li $f(n)=O(n^{\log_b(a)-\varepsilon})$
    - $T(n)=\Theta(n^{\log_b(a)}\cdot\log(n))$, je-li $f(n)=\Theta(n^{\log_b(a)})$
    - $T(n)=\Theta(f(n))$, je-li $f(n)=\Omega(n^{\log_b(a+\varepsilon)})$ a zároveň $a\cdot f(n/b)\leq c\cdot f(n)$ pro $c>0$ a $c<1$ a $n>n_0$
    - nejsme schopni rozhodnout, je-li $f(n)=\Omega(n^{\log_b(a+\varepsilon)})$, ale zároveň neplatí, že $a\cdot f(n/b)\leq c\cdot f(n)$ pro $c>0$ a $c<1$ a $n>n_0$
  - jednoduše: složitost je odvislá od toho, co rychleji roste, zda $a\cdot T(n/b)$ nebo $f(n)$

Složitost algoritmů

- Master Theorem
  - Příklad 1: $T(n) = 3T\left(\frac{n}{2}\right) + 1 \rightarrow a = 3, b = 2, f(n) = 1$
    - $\log_b a \cong 1.585$ zvolíme-li $\varepsilon = 1$, pak $\log_b(a) - \varepsilon = 0.585$
    - $f(n) = 1 \leq c \cdot n^{0.585} \leq c \cdot n = O(n)$, pro libovolné c
    - tudíž: $T(n) = \Theta(n^{1.585})$
  - Příklad 2: $T(n) = 2T\left(\frac{n}{2}\right) + 2n \rightarrow a = 2, b = 2, f(n) = 2n$
    - $\log_b a = 1$: $c_1 \cdot n^{\log_b(a)} \leq 2n \leq c_2 \cdot n^{\log_b(a)}$, platí pro $c_1 = 1, c_2 = 2$
    - tudíž: $f(n) = 2n = \Theta(n) \rightarrow T(n) = \Theta(n \log n)$
  - Příklad 3: $T(n) = 3T\left(\frac{n}{2}\right) + n^2 \rightarrow a = 3, b = 2, f(n) = n^2$
    - $\log_b a \cong 1.585$ zvolíme-li $\varepsilon = 1$, pak $\log_b(a) + \varepsilon = 2$
    - ověříme podmínku: $3\left(\frac{n}{2}\right)^2 = \frac{3}{8}n^2 \leq c \cdot n^2, c < 1$
      - platí pro např. $c = \frac{4}{8}$
    - $f(n) = n^2 \geq c \cdot n^{\log_b(a+\varepsilon)} = c \cdot n^2 = \Omega(n^2)$, pro $c = 1$
    - tudíž: $T(n) = \Theta(n^2)$

Složitost algoritmů

- Zobecněný Master Theorem
  - Nechť $T(n) = \sum T(a_i \cdot n) + f(n)$, $0 < a_i < 1$, x je výsledkem rovnice $\sum a_i^x = 1$ a $f(n) = \Theta(n^d)$
    - $T(n) = \Theta(n^d)$, jestliže $x < d$, tj. $\sum a_i^d < 1$
    - $T(n) = \Theta(n^x)$, jestliže $x > d$, tj. $\sum a_i^d > 1$
    - $T(n) = \Theta(n^d \cdot \log n)$, jestliže $x = d$, tj. $\sum a_i^d = 1$

Složitost algoritmů

- Zatřízení dvou polí
  - Seřazená pole vs. neseřazená
- Dosahování efektivity řazením
  - Nalezení duplicit
  - Dělení intervalu
- Amortizace
  - Nalezení čísla v neseřazeném poli čísel

Brute-Force

Zbytečný kód

- Test s dopředu známým výsledkem
- Výpočet v cyklu s konstantním výsledkem

```
if (a == 0) {
        //do something
}
if (a == 1) {
        //do something else
}
```

```
if (!((x >= x_min && x <= x_max) &&
        (y >= y_min && y <= y_max))){
        //x nelezi v obdelniku
}
```

```
for (int i = 0; i < obj.get_Count() - 1; i++) {
        //delej neco, pocet prvku v obj se nebude menit
}
```

Zbytečný kód

- Podmínka namísto aritmeticko-logického výpočtu:

```
if (d == 0)
  return a*a;
else
  return a*a + b*d;
```

```
    if (d == 0)
00DB1723  cmp         dword ptr [ebp-144h],0
00DB172A  jne         wmain+36Dh (0DB173Dh)
        return a*a;
00DB172C  mov         eax,dword ptr [ebp-12Ch]
00DB1732  imul        eax,dword ptr [ebp-12Ch]
00DB1739  jmp         wmain+38Dh (0DB175Dh)
    else
00DB173B  jmp         wmain+38Bh (0DB175Bh)
        return a*a + b*d;
00DB173D  mov         eax,dword ptr [ebp-12Ch]
00DB1743  imul        eax,dword ptr [ebp-12Ch]
00DB174A  mov         ecx,dword ptr [ebp-138h]
00DB1750  imul        ecx,dword ptr [ebp-144h]
00DB1757  add         eax,ecx
00DB1759  jmp         wmain+38Dh (0DB175Dh)
```

- Příklad: X = (A < B) ? CONST1 : CONST2;

```
    cmp a, b                    ; Condition
    jbe L30                     ; Conditional branch
    mov eb
    jmp L31
L30:
    mov eb
L31:
```

```
__asm {
   xor ebx, ebx; //ebx=0
   cmp a, b;     //porovnání
   setge bl;     //ebx = 0 nebo 1
   //ebx = (CONST2-CONST1)*ebx + CONST1
   lea ebx, [ebx*CONST3 + CONST1];
}
```

```
code)

condition
00
CONST2
add   ebx, CONST2; ebx=CONST1 or CONST2
```

```
D:\Education\ZEP\Programy\OptimizationJMP\Release>OptimizationJMP.exe
Test1 ...done: ALU ticks / JMP ticks = 99.079%
Test2 ...done: ALU ticks / JMP ticks = 86.040%; ALU2 / JMP = 86.123%
```

Zbytečný kód

```
//zelvi grafika: '+' otocka doprava, '-' otocka doleva
//'F' postup o jednot
void zelva(const char
{
  int i = 0;
  while (LF[i] != '\0
  {
    switch(LF[i])
    {
    case '+': //otock
      break;
    case '-': //otock
      break;
    default: //osetri
      break;
    }
  }
}
```

```
//zelvi grafika: '+' otocka doprava, '-' otocka doleva
//'F' postup o jednotku dopredu
void zelva(const char* LF)
{
  int i = 0;
  while (LF[i] != '\0')
  {
    switch(LF[i])
    {
    case 'F': //osetri pohyb dopredu
      break;
    case '+': //otocka vpravo
      break;
    default: //otocka vlevo
      break;
    }
  }
}
```

Pomalý kód

- Jednorozměrné vs. vícerozměrné pole
  - *Zpracování obrázků a volumetrických dat*

```
int[][] obrazek = new int[N][N];
for (int t = 0; t < EXP; t++)
{
        for (int y = 0; y < N; y++)
        {
                for (int x = 0; x< N; x++)
                {
                        obrazek[y][x] = x;
                }
        }
}
```

```
D:\Education\PRJ2>java Pole
Uplynuly cas v ms: 1641
Uplynuly cas v ms: 1169
```

```
int[] obrazek = new int[N*N];
for (int t = 0; t < EXP; t++)
{
        for (int y = 0, yBase = 0; y < N; y++, yBase += N)
        {
                for (int x = 0, index = yBase; x< N; x++, index++)
                {
                        obrazek[index] = x;
                }
        }
}
```

Pomalý kód

Skrytá (neskrytá) přetypování

- Přetypování v cyklu (mohou být skrytá)

```
void test(int n_a, int n_b)
{
        for (int a = 0; a < n_a; a++)
        {
                for (int b = 0; b < n_b; b++)
                {
                        double value = b*(a + 1.0)*get_Value(a, b);
                }
        }
}
```

Skrytá (neskrytá) přetypování

- Jaký je rozdíl mezi:
  - float kk=a*8.0; //a je float
  - float kk=a*8.0f;

```
00414856  fld         dword ptr [a]
0041485C  fmul        qword ptr [__real@4020000000000000 (417CA8h)]
00414862  fstp        dword ptr [kk]

Poznámka: v C++ oznamuje varováním možnou ztrátu přesnosti



0041493F  fld         dword ptr [a]
00414945  fmul        dword ptr [__real@4090f5c3 (417C6Ch)]
0041494B  fstp        dword ptr [kk]

Poznámka: v C++ v precise mód je výpočet druhého totožný s tím prvním
```

Operace	Instrukce	Latence
sčítání / odčítání	FADD / FSUB	5
násobení	FMUL	7
dělení (float)	FDIV	23
dělení (double)	FDIV	38
absolutní hodnota	FABS	1
druhá odmocnina	FSQRT	58
cos / sin	FCOS / FSIN	119
$\tan^{-1}$	FPATAN	147
$\cos^{-1}$ / $\sin^{-1}$	neexistuje, počítá se přes $\tan^{-1}$ a druhou odmocninu	> 220

```
Test4 ...done: double MUL ticks / double DIV ticks = 33.143% [0.000000 0.000000]
```

Jak je to reálně