

Příklady

```
program dvojkova;  
var a:integer;  
begin  
    readln(a);  
    while a > 0 do  
        begin  
            if a mod 2 = 1 then  
                write(1)  
            else    write(0);  
            a:=a div 2;  
        end;  
end.
```

Příklad vylepšený

Při programování je hlavní **myslet, jinak si často zbytečně přiděláme práci!**

```
program dvojk;  
var a:integer;  
begin  
    readln(a);  
    while a > 0 do  
        begin  
            write(a mod 2);  
            a:=a div 2;  
        end;  
end.
```

■ Nápady

Rozklad na prvočinitele

- Nápady
- Naivní algoritmus: Hledej postupně prvočísla až do n a zkoušej jimi dělit.

Rozklad na prvočinitele

- Nápady
- Naivní algoritmus: Hledej postupně prvočísla až do n a zkoušej jimi dělit.
- Méně naivní algoritmus: Hledej prvočísla až do $\sqrt{n_i}$ (kde n_i je hodnota, kterou ještě zbývá faktorizovat).

Rozklad na prvočinitele

- Nápady
- Naivní algoritmus: Hledej postupně prvočísla až do n a zkoušej jimi dělit.
- Méně naivní algoritmus: Hledej prvočísla až do $\sqrt{n_i}$ (kde n_i je hodnota, kterou ještě zbývá faktorizovat).
- Ještě méně naivní algoritmus: Zkoušej všechna čísla až do $\sqrt{n_i}$.
Proč toto funguje?

Příklad, hledání prvočíselného rozkladu:

```
program rozklad;  
var a,i:integer;  
begin  
    i:=2;  
    readln(a);  
    while i <= a do  
    begin    if (a div i)*i = a then  
            begin  
                write(i);  
                a:=a div i;  
            end  
            else    i:=i+1;  
    end;  
end.
```

Příklad, hledání prvočíselného rozkladu vylepšený:

```
program rozklad;  
var a,i:integer;      opakujeme:boolean;  
begin   i:=2;   opakujeme:=false;  
        readln(a);  
        while i <= a do  
        begin   if (a div i)*i = a then  
                begin   if opakujeme then  
                        write('*')  
                else   opakujeme:=true;  
                write(i);  
                a:=a div i;  
        end   else   i:=i+1;  
        end;  
end.
```


Komentáře umísťujeme do složených závorek:

Příklad: {Toto je kus komentare}

Komentář může obsahovat kusy kódu, které si nejsme jisti, zda použijeme:

```
{x=sqrt(x); Tak nevim, melo to byt pro odmocninu nebo  
ne... :-}
```

- Chceme uskladnit mnoho dat stejného typu (třeba tisíc celočíselných hodnot),
- definuje se v sekci proměnných (tedy var)),
- uvádí se klíčovým slovem array, následuje definice mezí a určení typu.

- Příklad: var a: array [1..100] of integer;
 soubor:array[5..50] of string;

- Do pole přistupujeme operátorem hranatých závorek:

Příklad:

```
a[1]=10;
```

```
soubor[6]='xxx';
```

```
{Pozor na:} soubor[1]='meze!';
```

K Eratosthenovu sítu

Jak nagenarovat všechna prvočísla menší než n ?

- Naivní algoritmus (generuj a testuj),

Jak nagenarovat všechna prvočísla menší než n ?

- Naivní algoritmus (generuj a testuj),
- Méně naivní algoritmus generuj a zkoušej dělit jen již nagenarovanými prvočísky.

K Eratosthenovu sítu

Jak nagenarovat všechna prvočísla menší než n ?

- Naivní algoritmus (generuj a testuj),
- Méně naivní algoritmus generuj a zkoušej dělit jen již nagenarovanými prvočísky.
- Eratosthenes: Nagenarovuj čísla $2 \dots n$. Pro i od 2 do \sqrt{n} pokud je i prvočíslo, proškrtej všechny jeho násobky.

Erathostenovo síto

```
program prvoc_do_tis;  
var prvocisla: array[2..1000] of boolean;  
    i,j:integer;  
begin  
for i:=2 to 1000 do prvocisla[i]:=true;  
for i:=2 to 1000 do  
begin  
    if prvocisla[i] then  
        begin writeln(i,' je prvocislo');  
            j:=2;  
            while(i*j<=1000) do  
                begin  
                    prvocisla[i*j]:=false;  
                    j:=j+1;  
                end;  
        end;  
    end;  
end.
```

Algoritmus má složitost:

- Statickou – jak velký je jeho zdrojový kód v určitém jazyce (zpravidla nezajímavá),

Algoritmus má složitost:

- Statickou – jak velký je jeho zdrojový kód v určitém jazyce (zpravidla nezajímavá),
- dynamickou – jak dlouho běží.

Algoritmus má složitost:

- Statickou – jak velký je jeho zdrojový kód v určitém jazyce (zpravidla nezajímavá),
- dynamickou – jak dlouho běží.
- implicitně máme na mysli složitost **dynamickou!**

Definition

Označuje-li hodnota n délku vstupu, pak složitostí algoritmu \mathcal{A} nazveme funkci f takovou, že pro každý vstup délky n algoritmus \mathcal{A} zastaví po nejvýše $f(n)$ elementárních krocích.

Definition

Označuje-li hodnota n délku vstupu, pak složitostí algoritmu \mathcal{A} nazveme funkci f takovou, že pro každý vstup délky n algoritmus \mathcal{A} zastaví po nejvýše $f(n)$ elementárních krocích.

- Erathostenovo síto: lineární vůči testovanému číslu
- Rozklad na prvočísla: Zdánlivě kvadratický vůči dekadickému zápisu, ve skutečnosti lineární.
- Převod do dvojkové soustavy: lineární v délce binárního zápisu.
- Pozor! Složitost počítáme vůči délce zápisu čísla!
- Mínótaurus v bludišti – lineární v počtu hran grafu.
- Stabilní párování (volenkový algoritmus) – nejvýše kvadratické v počtu pánů (resp. dam).

Jelikož počítat přesnou složitost je obtížné a jelikož není zajímavé, zda pro daný vstup budeme čekat hodinu nebo dvě, ale že algoritmus doběhne, používáme přibližné odhady složitosti:

Definition

Řekneme, že:

- $f \in O(g)$, jestliže existuje c a x_0 tak, že pro každé $x > x_0$ platí $f(x) \leq cg(x)$.
- $f \in \Omega(g)$, jestliže existuje $c > 0$ a x_0 tak, že pro každé $x > x_0$ platí $f(x) \geq cg(x)$.
- $f \in \Theta(g)$, pokud $f \in O(g)$ a současně $f \in \Omega(g)$.

Příklady:

- Je $n \in O(n^2)$?
- Je $n^2 \in O(n)$?
- Je $3n^5 + 2n^3 + 1000 \in \Theta(n^5)$?
- Je $n^{1000} \in O(2^n)$?
- Je $2^n \in n^{2000}$?

Cvičení s kartami aneb jak rychle roste exponenciála.

- Složitost v nejlepším případě,

Další pojmy složitosti

- Složitost v nejlepším případě,
- složitost v průměrném případě – průměrný počet kroků algoritmu přes jednotlivé instance dané délky,

- Složitost v nejlepším případě,
- složitost v průměrném případě – průměrný počet kroků algoritmu přes jednotlivé instance dané délky,
- amortizovaná složitost – průměrný počet kroků při potenciálně nekonečně mnoha po sobě jdoucích operacích – uvažujeme nejhorší možnou posloupnost,

- Složitost v nejlepším případě,
- složitost v průměrném případě – průměrný počet kroků algoritmu přes jednotlivé instance dané délky,
- amortizovaná složitost – průměrný počet kroků při potenciálně nekonečně mnoha po sobě jdoucích operacích – uvažujeme nejhorší možnou posloupnost,
- složitost problému – řeší, jak efektivní algoritmus existovat může a jak efektivní algoritmus už existovat nemůže.

- Neseříděné pole \Rightarrow snadný horní i dolní odhad (prohledat celé pole),

- Neseříděné pole \Rightarrow snadný horní i dolní odhad (prohledat celé pole),
- seříděné pole:

- Nesetříděné pole \Rightarrow snadný horní i dolní odhad (prohledat celé pole),
- setříděné pole:
 - unární hledání (listuj jako knihou, dokud nenajdeš),

- Neseříděné pole \Rightarrow snadný horní i dolní odhad (prohledat celé pole),
- seříděné pole:
 - unární hledání (listuj jako knihou, dokud nenajdeš),
 - binární hledání (začni uprostřed, pul intervaly),

- Neseříděné pole \Rightarrow snadný horní i dolní odhad (prohledat celé pole),
- seříděné pole:
 - unární hledání (listuj jako knihou, dokud nenajdeš),
 - binární hledání (začni uprostřed, pul intervaly),
 - kvadratické vyhledávání, zobecněné kvadratické vyhledávání...

- Snadný algoritmus, snadná analýza, složitost:

- Snadný algoritmus, snadná analýza, složitost:
- $\Theta(n)$.

- Jaká je složitost algoritmu? Kdy je potřeba udělat o krok navíc?
- $\Theta \log n$.

Další příklady:

Násobení matic:

Naivní algoritmus – snadno implementovatelný, snadná analýza složitosti.

Strassenův algoritmus – obtížně implementovaný, těžká analýza složitosti, ale složitost lepší.

Coppersmith-Vinogradův algoritmus – ještě obtížněji implementovaný, ještě lepší složitost.

Hledání největší nulové podmatice:

Naivní algoritmus: $O(n^6)$

Kdo má lepší nápady?