

Časová a prostorová složitost algoritmů

Programovací techniky

doc. Ing. Jiří Rybička, Dr.
ústav informatiky
PEF MENDELU v Brně
rybicka@mendelu.cz

- Kromě základních vlastností lze hodnotit kvalitu algoritmů

- Kromě základních vlastností lze hodnotit kvalitu algoritmů
- Stejnou úlohu lze řešit různými přístupy

- Kromě základních vlastností lze hodnotit kvalitu algoritmů
- Stejnou úlohu lze řešit různými přístupy
- Rozdíl je zejména ve spotřebě klíčových zdrojů – paměť, čas

- Kromě základních vlastností lze hodnotit kvalitu algoritmů
- Stejnou úlohu lze řešit různými přístupy
- Rozdíl je zejména ve spotřebě klíčových zdrojů – paměť, čas
- Pojem složitosti – **funkce** závislosti na N

- Kromě základních vlastností lze hodnotit kvalitu algoritmů
- Stejnou úlohu lze řešit různými přístupy
- Rozdíl je zejména ve spotřebě klíčových zdrojů – paměť, čas
- Pojem složitosti – **funkce** závislosti na N
- Znalost složitosti algoritmu je klíčová pro vhodnou aplikaci

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech
- Složitost časová – spotřeba času v závislosti na vstupních datech

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech
- Složitost časová – spotřeba času v závislosti na vstupních datech
- Časová složitost je obvykle kritičtější (prostor si lze koupit, čas nikoliv)

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech
- Složitost časová – spotřeba času v závislosti na vstupních datech
- Časová složitost je obvykle kritičtější (prostor si lze koupit, čas nikoliv)
- Třídy složitosti – není podstatná konkrétní přesná závislost, stačí charakterizovat třídu

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech
- Složitost časová – spotřeba času v závislosti na vstupních datech
- Časová složitost je obvykle kritičtější (prostor si lze koupit, čas nikoliv)
- Třídy složitosti – není podstatná konkrétní přesná závislost, stačí charakterizovat třídu
- Sledujeme horní ohraničení $O(N)$

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech
- Složitost časová – spotřeba času v závislosti na vstupních datech
- Časová složitost je obvykle kritičtější (prostor si lze koupit, čas nikoliv)
- Třídy složitosti – není podstatná konkrétní přesná závislost, stačí charakterizovat třídu
- Sledujeme horní ohraničení $O(N)$
- Zanedbáváme implementační konstanty

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech
- Složitost časová – spotřeba času v závislosti na vstupních datech
- Časová složitost je obvykle kritičtější (prostor si lze koupit, čas nikoliv)
- Třídy složitosti – není podstatná konkrétní přesná závislost, stačí charakterizovat třídu
- Sledujeme horní ohraničení $O(N)$
- Zanedbáváme implementační konstanty
- Vyjádření třídy – jednoduchá matematická funkce

- Pojem: matematická funkce představující závislost sledovaného parametru na množství vstupních dat
- Složitost prostorová – spotřeba paměti, diskového prostoru v závislosti na vstupních datech
- Složitost časová – spotřeba času v závislosti na vstupních datech
- Časová složitost je obvykle kritičtější (prostor si lze koupit, čas nikoliv)
- Třídy složitosti – není podstatná konkrétní přesná závislost, stačí charakterizovat třídu
- Sledujeme horní ohraničení $O(N)$
- Zanedbáváme implementační konstanty
- Vyjádření třídy – jednoduchá matematická funkce
- Třídy lze uspořádat

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$
- Logaritmická $O(N) = k \log N$

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$
- Logaritmická $O(N) = k \log N$
- Lineární $O(N) = k \cdot N$

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$
- Logaritmická $O(N) = k \log N$
- Lineární $O(N) = k \cdot N$
- Lineárně logaritmická $O(N) = k \cdot N \cdot \log N$

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$
- Logaritmická $O(N) = k \log N$
- Lineární $O(N) = k \cdot N$
- Lineárně logaritmická $O(N) = k \cdot N \cdot \log N$
- Kvadratická $O(N) = k \cdot N^2$

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$
- Logaritmická $O(N) = k \log N$
- Lineární $O(N) = k \cdot N$
- Lineárně logaritmická $O(N) = k \cdot N \cdot \log N$
- Kvadratická $O(N) = k \cdot N^2$
- Kubická $O(N) = k \cdot N^3$

- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$
- Logaritmická $O(N) = k \log N$
- Lineární $O(N) = k \cdot N$
- Lineárně logaritmická $O(N) = k \cdot N \cdot \log N$
- Kvadratická $O(N) = k \cdot N^2$
- Kubická $O(N) = k \cdot N^3$
- Exponenciální $O(N) = k \cdot Z^N$

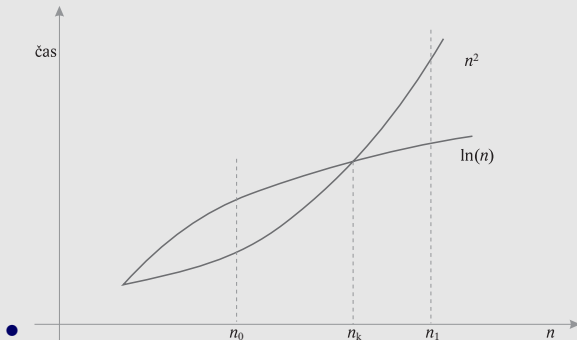
- Ve vyjádření tříd je k implementační konstanta, jejíž vliv na charakter algoritmu je nulový
- Konstantní $O(N) = k$
- Logaritmická $O(N) = k \log N$
- Lineární $O(N) = k \cdot N$
- Lineárně logaritmická $O(N) = k \cdot N \cdot \log N$
- Kvadratická $O(N) = k \cdot N^2$
- Kubická $O(N) = k \cdot N^3$
- Exponenciální $O(N) = k \cdot Z^N$
- Faktoriální $O(N) = k \cdot N!$

- Testovací množina dat je obvykle podstatně menší než reálná data

- Testovací množina dat je obvykle podstatně menší než reálná data
- Program funguje s malým množstvím dat výborně

- Testovací množina dat je obvykle podstatně menší než reálná data
- Program funguje s malým množstvím dat výborně
- Při nasazení do provozu nastávají „nečekané“ komplikace

- Testovací množina dat je obvykle podstatně menší než reálná data
- Program funguje s malým množstvím dat výborně
- Při nasazení do provozu nastávají „nečekané“ komplikace



- Metody – experimentální, analytické

- Metody – experimentální, analytické
- **Experiment** – vytvoření tabulky s údaji, kde počet hodnot nezávisle proměnné stanovujeme podle vznikajícího charakteru algoritmu

- Metody – experimentální, analytické
- **Experiment** – vytvoření tabulky s údaji, kde počet hodnot nezávisle proměnné stanovujeme podle vznikajícího charakteru algoritmu
- Možnosti praktických měření jsou omezené (stanovení přesné spotřeby času, stanovení přesné spotřeby prostoru)

- **Analýza algoritmu**

- **Analýza algoritmu**
- U prostoru – lokální statické proměnné, dynamické proměnné, rekurze

- **Analýza algoritmu**
- U prostoru – lokální statické proměnné, dynamické proměnné, rekurze
- Alokace paměti v cyklech, jejichž počet průchodů závisí na vstupních datech

- **Analýza algoritmu**
- U prostoru – lokální statické proměnné, dynamické proměnné, rekurze
- Alokace paměti v cyklech, jejichž počet průchodů závisí na vstupních datech
- Není potřeba sledovat globální statické proměnné

- **Analýza algoritmu**
- U prostoru – lokální statické proměnné, dynamické proměnné, rekurze
- Alokace paměti v cyklech, jejichž počet průchodů závisí na vstupních datech
- Není potřeba sledovat globální statické proměnné
- Časová složitost – cykly a rekurze závisující na vstupních datech

- **Analýza algoritmu**
- U prostoru – lokální statické proměnné, dynamické proměnné, rekurze
- Alokace paměti v cyklech, jejichž počet průchodů závisí na vstupních datech
- Není potřeba sledovat globální statické proměnné
- Časová složitost – cykly a rekurze závisující na vstupních datech
- Vnoření cyklů – násobení závislostí

- **Analýza algoritmu**
- U prostoru – lokální statické proměnné, dynamické proměnné, rekurze
- Alokace paměti v cyklech, jejichž počet průchodů závisí na vstupních datech
- Není potřeba sledovat globální statické proměnné
- Časová složitost – cykly a rekurze závisící na vstupních datech
- Vnoření cyklů – násobení závislostí
- Následné cykly – započítáváme jen horší složitost

- Hledání – sekvenční hledání v neuspořádané lineární struktuře (hledáme hodnotu C v poli P , v němž je naplněno N prvků)

- Hledání – sekvenční hledání v neuspořádané lineární struktuře (hledáme hodnotu C v poli P, v němž je naplněno N prvků)
- Časová složitost odvozená z algoritmu:

```
I:=1;  
while (I<=N) and (P[I]<>C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Hledání – sekvenční hledání v neuspořádané lineární struktuře (hledáme hodnotu C v poli P , v němž je naplněno N prvků)
- Časová složitost odvozená z algoritmu:

```
I:=1;  
while (I<=N) and (P[I]<>C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Cyklus při neúspěšném hledání (nejhorší případ) proběhne N -krát, složitost je tedy $O(N) = k \cdot N$ (lineární)

- Hledání – sekvenční hledání v neuspořádané lineární struktuře (hledáme hodnotu C v poli P , v němž je naplněno N prvků)
- Časová složitost odvozená z algoritmu:

```
I:=1;  
while (I<=N) and (P[I]<>C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Cyklus při neúspěšném hledání (nejhorší případ) proběhne N -krát, složitost je tedy $O(N) = k \cdot N$ (lineární)
- Uvažujme průměrně projití 75 % prvků pole, pak $O(N) = 0,75 \cdot k \cdot N = k_1 \cdot N$

- Modifikovaný algoritmus sekvenčního hledání v uspořádané struktuře:

```
I:=1;  
while (I<=N) and (P[I]<C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Modifikovaný algoritmus sekvenčního hledání v uspořádané struktuře:

```
I:=1;  
while (I<=N) and (P[I]<C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Má zcela **stejnou složitost**, i když neúspěšné hledání je detekováno dříve

- Modifikovaný algoritmus sekvenčního hledání v uspořádané struktuře:

```
I:=1;  
while (I<=N) and (P[I]<C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Má zcela **stejnou složitost**, i když neúspěšné hledání je detekováno dříve
- Uvažujme průměrné projití 50 % prvků pole. Pak $O(N) = 0,5 \cdot k \cdot N = k_2 \cdot N$

- Modifikovaný algoritmus sekvenčního hledání v uspořádané struktuře:

```
I:=1;  
while (I<=N) and (P[I]<C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Má zcela **stejnou složitost**, i když neúspěšné hledání je detekováno dříve
- Uvažujme průměrné projití 50 % prvků pole. Pak $O(N) = 0,5 \cdot k \cdot N = k_2 \cdot N$
- Zřejmě platí $k_1 > k_2$, nemění se však **charakter** algoritmu, ani časová složitost (třída)

- Modifikovaný algoritmus sekvenčního hledání v uspořádané struktuře:

```
I:=1;  
while (I<=N) and (P[I]<C)  
    do Inc(I);  
Nalezeno:=I<=N
```

- Má zcela **stejnou složitost**, i když neúspěšné hledání je detekováno dříve
- Uvažujme průměrné projití 50 % prvků pole. Pak $O(N) = 0,5 \cdot k \cdot N = k_2 \cdot N$
- Zřejmě platí $k_1 > k_2$, nemění se však **charakter** algoritmu, ani časová složitost (třída)
- Jedná se o názornou ukázkou velmi neefektivního použití algoritmu – v uspořádané struktuře je potřebné hledat v jiné třídě složitosti

- Binární vyhledávací strom (implementace dynamickou strukturou):

```
U:=Koren;  
while (U<>nil) and (U^.Data<>C)  
    do if C<U.Data then U:=U^.Vlevo  
        else U:=U^.Vpravo;  
Nalezeno:=U<>nil
```

- Binární vyhledávací strom (implementace dynamickou strukturou):

```
U:=Koren;  
while (U<>nil) and (U^.Data<>C)  
    do if C<U.Data then U:=U^.Vlevo  
        else U:=U^.Vpravo;  
Nalezeno:=U<>nil
```

- Cyklus závisí na vstupních datech, v jeho těle proběhne rozhodnutí, do kterého podstromu se vydáme

- Binární vyhledávací strom (implementace dynamickou strukturou):

```
U:=Koren;  
while (U<>nil) and (U^.Data<>C)  
    do if C<U.Data then U:=U^.Vlevo  
        else U:=U^.Vpravo;  
Nalezeno:=U<>nil
```

- Cyklus závisí na vstupních datech, v jeho těle proběhne rozhodnutí, do kterého podstromu se vydáme
- Cyklus proběhne nejhůře tolikrát, kolik je hladin stromu

- Binární vyhledávací strom (implementace dynamickou strukturou):

```
U:=Koren;  
while (U<>nil) and (U^.Data<>C)  
    do if C<U.Data then U:=U^.Vlevo  
        else U:=U^.Vpravo;  
Nalezeno:=U<>nil
```

- Cyklus závisí na vstupních datech, v jeho těle proběhne rozhodnutí, do kterého podstromu se vydáme
- Cyklus proběhne nejhůře tolikrát, kolik je hladin stromu
- V případě vyváženého stromu je počet hladin stromu o N prvcích roven $h = \log_2 N$

- Binární vyhledávací strom (implementace dynamickou strukturou):

```
U:=Koren;  
while (U<>nil) and (U^.Data<>C)  
    do if C<U.Data then U:=U^.Vlevo  
        else U:=U^.Vpravo;  
Nalezeno:=U<>nil
```

- Cyklus závisí na vstupních datech, v jeho těle proběhne rozhodnutí, do kterého podstromu se vydáme
- Cyklus proběhne nejhůře tolikrát, kolik je hladin stromu
- V případě vyváženého stromu je počet hladin stromu o N prvcích roven $h = \log_2 N$
- Časová složitost je tedy $O(N) = k \cdot \log_2 N$

- Součet řady čísel ze standardního vstupu – prostorová složitost

- Součet řady čísel ze standardního vstupu – prostorová složitost
- Iterativní algoritmus:

```
var C, Soucet: real;
begin Soucet:=0;
      while not SeekEof do begin
          read(C);
          Soucet:=Soucet+C
      end;
      writeln(Soucet)
end.
```


- Součet řady čísel ze standardního vstupu – prostorová složitost
- Iterativní algoritmus:

```
var C, Soucet: real;
begin Soucet:=0;
      while not SeekEof do begin
          read(C);
          Soucet:=Soucet+C
      end;
      writeln(Soucet)
end.
```

- Cyklus neobsahuje alokaci paměti, prostorová složitost je tedy rovna $O(N) = k$ (konstantní)

- Součet řady řešený rekurzivním algoritmem:

```
function Soucet: real;
  var C: real;
  begin if not SeekEof then begin
        read(C);
        Soucet:=C+Soucet
      else Soucet:=0
    end;
  begin writeln(Soucet)
  end.
```


- Rekurze vyvolaná v hlavním programu je závislá na počtu vstupních dat

- Rekurze vyvolaná v hlavním programu je závislá na počtu vstupních dat
- S každým zpracovaným číslem se volá funkce, na systémový zásobník se ukládá její návratová adresa a lokální proměnná C

- Rekurze vyvolaná v hlavním programu je závislá na počtu vstupních dat
- S každým zpracovaným číslem se volá funkce, na systémový zásobník se ukládá její návratová adresa a lokální proměnná C
- Prostorová složitost je lineární $O(N) = k \cdot N$

- Rekurze vyvolaná v hlavním programu je závislá na počtu vstupních dat
- S každým zpracovaným číslem se volá funkce, na systémový zásobník se ukládá její návratová adresa a lokální proměnná C
- Prostorová složitost je lineární $O(N) = k \cdot N$
- Jedná se o ukázkou neefektivního algoritmu