Vysoká škola báňská – Technická univerzita Ostrava
Fakulta stavební

Přednáška z předmětu: Algoritmizace inženýrských výpočtů

Téma č.4: Metody pro třídění souboru prvků

doc. Ing. Martin Krejsa, Ph.D.

Obsah

Kapitola 1

Metody pro třídění souboru prvků

Cíle

Kapitola studentům přiblíží:

- základy algoritmu pro třídění, které najdou uplatnění v mnoha inženýrských aplikacích,
- dvojný cyklus `for`,
- procedury pro čtení dat z textového souboru i jejich zápis do diskového souboru.

Obsah
3. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

1.1. Třídící algoritmy

Třídící (řadící) algoritmy najdou uplatnění v celé řadě inženýrských úloh, např. při zpracování experimentálně získaných dat či měření. Souhrnně jsou uvedeny např. v [1].

1.1.1. Bublinkové třídění (Bubble Sort)

Bublinkové řazení (anglicky *Bubblesort*, česky též *řazení záměnou*) je implementačně jednoduchý řadicí algoritmus. Název algoritmu vyjadřuje průběh zpracování. Při představě, že řazená čísla jsou přirovnány k bublinkám, které stoupají k vodní hladině s různou rychlostí podle své velikosti, pak řazené prvky s větší hodnotou „probublávají“ směrem ke konci vzestupně řazeného seznamu. Algoritmus opakovaně prochází posloupnost n prvků, přičemž porovnává každé dva sousedící prvky, a pokud nejsou ve správném pořadí, prohodí je. Porovnávání prvků běží do té doby, dokud není seznam seřazený.

Algoritmus je univerzální (pracuje na základě porovnávání dvojic prvků), pracuje lokálně (nevyžaduje pomocnou paměť), je stabilní (prvkům se stejným klíčem nemění vzájemnou polohu) a patří mezi přirozené řadicí algoritmy (částečně seřazený seznam zpracuje rychleji než neseřazený).

Obsah
4. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

Vstup : $\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}$
Výstup: $\mathbf{x}$

for $j \leftarrow 1, 2, 3, \ldots, n-1$ **do**
  for $i \leftarrow n-1, n-2, \ldots, j+1, j$ **do**
    if $x_i > x_{i+1}$ **then**
      $c \leftarrow x_i$
      $x_i \leftarrow x_{i+1}$
      $x_{i+1} \leftarrow c$
    end
  end
end

Algoritmus 1: Algoritmus třídění typu Bubble sort

Výpočetní postup je schématicky znázorněn algoritmem 1 a jeho přepis do programovacího jazyka systému MATLAB může vypadat následovně:

```
function y=bubblesort(x);
n=length(x);
for j=1:n-1
  for i=n-1:-1:j
    if x(i)>x(i+1);
      c=x(i);
      x(i)=x(i+1);
      x(i+1)=c;
    end
```

Obsah
5. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

```
  end
end
y=x;
```

Pro praktické účely je však postup neefektivní, využívá se hlavně pro výukové účely či v nenáročných aplikacích. Tento algoritmus řazení je jedním z nejpomalejších, oproti jiným algoritmům se stejnou složitostí vyžaduje velké množství zápisů do paměti.

Poznámka 1.1. Z hlediska použití cyklů typu `for` lze podotknout, že algoritmus obsahuje dvojný cyklus s řídícími proměnnými i a j. K seřazení pole o n prvcích bude zapotřebí $n-1$ provedení vnějšího cyklu, zatímco vnitřní cyklus se postupně provede $(n-1), (n-2), (n-3), \ldots, 2, 1$ krát. Koncová hodnota řídící proměnné j tedy není konstantní, ale závisí na hodnotě řídící proměnné i. Celkový počet operací při tomto typu třídění bude $\frac{n^2-n}{2}$, což řádově odpovídá složitosti úlohy s přibližným počtem výpočetních operací n^2.

Obsah
6. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

1.1.2. Třídění přímým výběrem minima (Select Sort)

Třídění přímým výběrem minima neboli *Selection sort* (zkráceně *Selectsort*) je jednoduchý algoritmus uspořádávání. Pro svou jednoduchou implementaci bývá často používán pro uspořádávání malých množství dat. Pro větší objem dat se používají algoritmy s nižší časovou složitostí.

Algoritmus postupně prochází tříděný vektor a pro daný prvek vektoru hledá minimum ze zbývajících prvků. Postup lze popsat následovně:

1. nalezne se prvek s nejmenší hodnotou v posloupnosti n hodnot,
2. zamění se s prvkem na první pozici, takže se zde nachází prvek s nejmenší hodnotou,
3. zbytek posloupnosti se uspořádá opakováním těchto kroků pro zbylých $n - 1$ prvků.

Algoritmus *Selectsortu* tedy vychází z myšlenky, že pokud se řadí pole od nejmenšího prvku k největšímu, tak na první pozici posloupnosti bude prvek s nejmenší hodnotou, za ním prvek s nejmenší hodnotou ze zbytku pole atd., čili stačí pouze postupně vybírat nejmenší prvky z neseřazené části pole a umísťovat je na konec seřazené části.

Obsah
7. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

Vstup : $\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}$
Výstup: $\mathbf{x}$

for $i \leftarrow 1, 2, 3, \ldots, n-1$ **do**
  for $j \leftarrow i+1, i+2, i+3, \ldots, n$ **do**
    if $x_i > x_j$ **then**
      $c \leftarrow x_i$
      $x_i \leftarrow x_j$
      $x_j \leftarrow c$
    end
  end
end

Algoritmus 2: Algoritmus metody třídění přímým výběrem minima

Schématické znázornění řadícího výpočtu je obsaženo v algoritmu 2. Následuje jeho interpretace v programovacím jazyce systému MATLAB:

```
function y=selectsort(x);
n=length(x);
for i=1:n-1
  for j=i+1:n
    if x(i)>x(j);
      c=x(i);
      x(i)=x(j);
      x(j)=c;
```

Obsah
8. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

```
    end
  end
end
y=x;
```

Poznámka 1.2. Z hlediska použití cyklu `for` lze podobně jako v předchozím případě konstatovat, že algoritmus obsahuje dvojný cyklus s řídícími proměnnými i a j a složitost úlohy zůstává s přibližným počtem výpočetních operací n^2.

1.1.3. Třídění přímým vkládáním (Insert Sort)

Třídění přímým vkládáním neboli *Insertion sort* (zkráceně *Insertsort*) je jednoduchý řadicí algoritmus založený na porovnávání. Algoritmus výpočtu pracuje tak, že prochází prvky postupně a každý další nesetříděný prvek zařadí na správné místo do již setříděné posloupnosti. Je to jeden z nejrychlejších algoritmů. Je sice pomalejší než pokročilé algoritmy typu *Quicksort* nebo *Shellsort* (viz dále), ale zato má jiné výhody:

- jednoduchá implementace,
- efektivní na malých množinách,
- efektivní na částečně seřazených množinách,
- efektivnější než předchozí algoritmy (Selectsort, Bubblesort),
- řadí stabilně (nemění vzájemné pořadí prvků se stejnými klíči),
- jedná se o tzv. online algoritmus, neboť umožňuje řadit data tak, jak vstupují do výpočtu.

Obsah
9. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

Algoritmus zatřiďuje prvky z pravé části posloupnosti přímo do části levé, kde se utváří vzestupně seřazený seznam hodnot. Jediný rozdíl oproti předchozímu algoritmu třídění `Selectsort` tkví v tom, že tento způsob manipuluje při řazení přímo s jednotlivými prvky a ne jenom s jejich indexy, takže lze výpočetní postup využít např. při postupném načítání dat ze souboru.

Vstup : $\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}$
Výstup: **x**

for $i \leftarrow 2, 3, \ldots, n-1, n$ **do**
  $c \leftarrow x_i$
  for $j \leftarrow i-1, i-2, i-3, \ldots, 1$ **do**
    if $x_j > x_{j+1}$ **then**
      $x_{j+1} \leftarrow x_j$
      $x_j \leftarrow c$
    end
  end
end

Algoritmus 3: Algoritmus metody třídění přímým vkládáním

Schématické znázornění výpočtu je uvedeno v algoritmu 3 a jeho přepis do programovacího jazyka systému MATLAB je následující:

```
function y=insertsort(x);
n=length(x);
for i=2:n
```

Obsah
10. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

```
  c=x(i);
  for j=i-1:-1:1
    if x(j)>x(j+1);
      x(j+1)=x(j);
      x(j)=c;
    end
  end
end
y=x;
```

Řadící algoritmy Quicksort a Shellsort, které jsou uvedeny v dalším výkladu, jsou již z kategorie pokročilých. Jejich uvedení v tomto učebním textu má za cíl nahlédnout do pozadí kvalitních výpočetních postupů.

Obsah

11. strana ze 22

Zavřít dokument

Konec

Celá obrazovka/Okno

1.1.4. Rychlé (rekurzivní) řazení (Quick Sort)

Rychlé (rekurzivní) řazení do tříd neboli zkráceně *Quicksort* je jeden z nejrychlejších známých algoritmů řazení založených na porovnávání prvků. Algoritmus vymyslel v roce 1962 Sir Charles Antony Richard Hoare. Jeho průměrná časová složitost je pro algoritmy této skupiny nejlepší možná $(n \cdot \log n)$, v nejhorším případě (kterému se ale v praxi jde obvykle vyhnout) může být jeho časová náročnost opět i n^2.

Základní myšlenkou *Quicksortu* je rozdělení řazené posloupnosti čísel na dvě přibližně stejné části. V jedné části jsou čísla větší a ve druhé menší, než je hodnota jednoho ze zvolených prvků posloupnosti, nazývaného **pivot**. Pokud je pivot zvolen optimálně, jsou obě části posloupnosti přibližně stejně velké. Obě části se pak řadí samostatně.

Důležitým aspektem pro výkonnost tohoto výpočetního postupu je tedy volba pivota. Používanou volbou pivota bývá tzv. fixní prvek (první nebo poslední) a náhodný prvek. Fixní prvek má problém na částečně uspořádaných polích, případně na polích s nějakou strukturou (kde nedochází k optimálnímu dělení problému a složitost narůstá až k n^2).

Schématicky lze výpočetní postup *Quicksortu* popsat takto:

```
procedure quicksort(seznam_hodnot)
if length(seznam_hodnot) <= 1
  return
pivot = náhodný prvek ze seznamu_hodnot

Rozdělení seznam_hodnot do 3 částí
  seznam1 = {prvky menší než pivot}
  seznam2 = {pivot}
  seznam3 = {prvky větší než pivot}
seznam_hodnot = quicksort(seznam1) + seznam2 + quicksort(seznam3)
```

Obsah
12. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

Algoritmus třídění *QuickSort* vychází z tzv. rekurzivního přístupu.

Rekurze je často používaná technika v matematice a informatice. Termín je pravděpodobně odvozen z latinského slovesa *recurso* (vrátit se) nebo *recursus* (návrat, zpětný běh). V programování představuje rekurze opakované volání stejné funkce (funkce vyvolává „sama sebe“), v takovém případě se hovoří o rekurzivní funkci. Nedílnou součástí rekurzivní funkce musí být ukončující podmínka určující, kdy se má vnořování zastavit. Jelikož bývá nejčastějším zdrojem chyb, je třeba ji navrhnout dostatečně robustním způsobem a prověřit veškeré možné situace jejího chodu.

Při menších velikostech problému je *Quicksort* poměrně pomalý, protože velmi pravděpodobně nedojde rozdělení pole na ideální poloviny. Z tohoto důvodu se *Quicksort* používá pouze u velkých polí, pro řešení malých polí bývá výrazně rychlejší třídění pomocí *Insertsortu* nebo *Shellsortu* (viz dále).

Jedna z možností, jak rekurzivní způsob řazení pomocí algoritmu Quicksort v prostředí systému MATLAB naprogramovat, může vypadat například takto:

```
function y=quicksort(x)
n=length(x);
if(n<=1);y=x;return;end;
if(n==2)
  if(x(1)>x(2))
     x=[x(2); x(1)];
  end
  y=x;
  return;
end
```

Obsah

13. strana ze 22

Zavřít dokument

Konec

Celá obrazovka/Okno

```
m=fix(n/2);
pivot=x(m);
Mensi=find(x<pivot);
if(isempty(Mensi))
  ind=find(x>pivot);
  if(isempty(ind));y=x;return;end;
  pivot=x(ind(1));
  Mensi=find(x<pivot);
end
Vetsi=find(x>=pivot);
y=[quicksort(x(Mensi));quicksort(x(Vetsi))];
```

Poznámka 1.3. Uvedený algoritmus předpokládá, že řazený vektor je ve sloupcového tvaru. Z tohoto důvodu je nutné vyvolávat m-funkci s vektorem v příslušném tvaru, např:

```
A=[76 5 44 90 59 63 4 1 28 57];
B=quicksort(A')
```

Poznámka 1.4. V této m-funkci byly použity funkce programu MATLAB `fix` pro operaci celočíselné dělení a funkce `find`, která vrací vektor prvků posloupnosti hodnot x, které splňují podmínku obsaženou ve vstupním parametru. Logická funkce `isempty` testuje obsah proměnné (vektoru) v parametru. V případě, že je proměnná prázdná, výsledkem funkce je logická hodnota PRAVDA (tedy hodnota 1). Příkaz `return` nepřeruší běh výpočtu, jako např. příkaz `break`, pouze ukončí činnost vyvolané funkce (u rekurzivních výpočtů to neznamená konec výpočtu, ale návrat do té části algoritmu, ze které byla funkce vyvolaná).

Obsah
14. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

1.1.5. Shellovo řazení (Shell Sort)

Shellovo řazení (zkráceně *Shellsort*) nebo také řazení se snižujícím se přírůstkem je řadicí algoritmus podobný *Insertsortu*, který objevil a v roce 1959 publikoval Donald Shell. Časová složitost *Shellsortu* je přibližně rovna $n^{\frac{3}{2}}$ a je z algoritmů složitosti typu n^2 nejvýkonnější.

Obecným problémem řadících algoritmů je zatřídění prvků do opačné části pole, než je jejich původní umístění. Tyto prvky musí v běžných kvadratických algoritmech „procestovat“ postupně celé pole. Přístup *Shellsortu* je v tomto ohledu jiný, protože neporovnává sousední prvky, ale v každém svém kroku prvky v určité vzdálenosti (tato vzdálenost se v každém kroku snižuje, až se zmenší na 1). Tímto způsobem se zajistí nejrychlejší způsob, jak prvky zařazené na špatné straně pole přesunout na jejich správnou pozici.

Základním problémem algoritmu je volba ideální vzdálenosti pro porovnávání jednotlivých prvků. Donald Shell původně navrhoval, aby se při porovnávání začínalo s hodnotou mezery $\frac{n}{2}$, kde n je velikost pole, a vzdálenost mezi porovnávanými prvky se tedy vždy půlila. Tento přístup má nevýhodu, protože prvky na sudých a lichých místech se porovnávají až v posledním kroku algoritmu. Dalšími pokusy s volbou mezery byla například volba posloupnosti $2 \cdot k - 1$ (Hibbard) se složitostí $n^{\frac{3}{2}}$ nebo posloupnost $9 \cdot 4^i - 9 \cdot 2^i$ (Sedgewick) se složitostí $n^{\frac{4}{2}}$, případně Fibonacciho posloupnost umocněná na dvojnásobek zlatého řezu. Nejlepší výsledky lze získat s posloupností $1, 4, 10, 23, 57, 132, 301, 701, 1750$ nebo $mezera \cdot 2, 2$, jejímž autorem je Marcin Ciura.

Výpočet s využitím algoritmu *Shellsortu* lze naprogramovat v m-funkci např. následujícím způsobem:

```
function y=shellsort(x)
n=length(x);
mezera=floor(n/2);
```

```
while mezera>0
  for j=mezera:n
    for i=j-mezera:-mezera:1
      if x(i+mezera)>=x(i)
        break;
      else
        c=x(i);
        x(i)=x(i+mezera);
        x(i+mezera)=c;
      end
    end
  end
  mezera=floor(mezera/2);
end
y=x;
```

Poznámka 1.5. V m-funkci byla použita funkce `floor`, která zaokrouhluje na celé číslo směrem k $-\infty$, tj. např. pro `floor(-0.4)` vychází výsledek -1.

1.2. Práce s textovým souborem

Činnost třídících algoritmů lze doplnit načítáním číselných hodnot z textového souboru do pole vektoru v systému MATLAB, které se pak může setřídit některým z popsaných algoritmů. Naopak zápis dat do textového souboru může sloužit pro ukládání výsledků výpočtu, příp. pro přípravu dat.

Obsah
16. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

Poznámka 1.6. Zápis dat do textového souboru lze provádět např. s využitím příkazů systému MATLAB s názvy `fopen`, `fprintf` a `fclose`. Jejich použití je zřejmé z následujícího příkladu.

Příklad 1.7. Do textového souboru s názvem `exp.txt` uložte tabulku s hodnotami funkce e^x pro $x = 0; 0,1; 0,2; \ldots; 1,0$.

Řešení. Úlohu lze provést např. s pomocí následujícího sledu příkazů:

```
x=0:.1:1;
y=[x;exp(x)];
fid=fopen('exp.txt','w');
fprintf(fid,'%6.2f %12.8f\n',y);
fclose(fid);
type exp.txt
```

Ve funkci `fopen` je přitom nutno specifikovat druhým parametrem mód přístupu k souboru, jehož název obsahuje první parametr. K základním možnostem patří parametr **'r'** pro otevření souboru pro čtení a **'w'** pro zápis (v takovém případě se případně vymaže obsah již existujícího souboru).

Výsledný obsah textového souboru s názvem `exp.txt` se provede příkazem `type`. Výpis by pak měl vypadat následovně:

```
 0.00   1.00000000
 0.10   1.10517092
 0.20   1.22140276
 0.30   1.34985881
 0.40   1.49182470
 0.50   1.64872127
```

Obsah
17. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

```
0.60   1.82211880
0.70   2.01375271
0.80   2.22554093
0.90   2.45960311
1.00   2.71828183
```

▲

Obsah

18. strana ze 22

Zavřít dokument

Konec

Celá obrazovka/Okno

Příklad 1.8. Vytvořte textový soubor s názvem `hodnoty.txt`, do kterého zapíšete 10000 náhodně vygenerovaných čísel s rozsahem v rozmezí 0 až 1000000 pomocí funkce `randi`.

Řešení. Správné vyvolání příkazu pro vygenerování určeného počtu náhodných čísel v předem stanoveném rozsahu hodnot má tvar:

```
randi(1000000,1,10000)
```

Pokud se zkombinuje i se zápisem do textového souboru, pak může být posloupnost povelů např.:

```
clc;
clear;
x=randi(1000000,1,10000);
fid=fopen('hodnoty.txt','w');
fprintf(fid,'%7g\n',x);
fclose(fid);
type hodnoty.txt
```

▲

Poznámka 1.9. Funkce `randi` není obsažena v programu Octave, kde je nutno použít funkci `unidrnd`.

Obsah

19. strana ze 22

Zavřít dokument

Konec

Celá obrazovka/Okno

Příklad 1.10. Načtěte obsah vytvořeného souboru `hodnoty.txt` do proměnné A pomocí příkazu `dlmread`.

Řešení. Příkaz je velice jednoduchý s tvarem:

```
A=dlmread('hodnoty.txt')
```

Soubor hodnoty.txt ovšem musí obsahovat pouze číselné hodnoty. ▲

Poznámka 1.11. Obecněji lze operaci z předchozího příkladu provést následujícím způsobem:

```
clc;
clear;
fid=fopen('hodnoty.txt');
k=0;
while feof(fid)==0
    cti=fscanf(fid,'%f',1);
    if ~isempty(cti)
       k=k+1;
       A(k)=cti;
    end
end
fclose(fid);
A'
```

Uvedená m-funkce obsahuje funkci `feof`, která vrací hodnotu PRAVDA, tedy 1, v případě, že se při čtení textového souboru dostal program na jeho konec (v opačném případě vrací hodnotu NEPRAVDA čili 0).

Obsah
20. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

Příklad 1.12. Setřiďte obsah proměnné A z předchozího příkladu pomocí třídících algoritmů, jenž byly vysvětleny v této kapitole.

Poznámka 1.13. Pokud je zapotřebí sledovat strojový čas výpočtu, je možné s výhodou použít dvojici funkcí `tic` a `toc`, např. tímto způsobem:

```
clc
tic;
B=bubblesort(A);
toc
```

Výpis o trvání výpočtu pak může vypadat např.:

```
Elapsed time is 1.790291 seconds.
```

Příklad 1.14. Upravte výpočetních postupů řadících algoritmů tak, aby výsledkem byl seznam sestupně setříděných hodnot.

Poznámka 1.15. Pro kontrolu správnosti práce vytvořených m-funkcí lze využít příkazy systému MATLAB s názvy `sort` (tříd vektory a matice vzestupně i sestupně) a `issorted` (vrací hodnotu PRAVDA, tedy 1, pro setříděný vektor či matici, v opačném případě je výslednou hodnotou NEPRAVDA, čili 0), ve kterých vstupní parametr obsahuje předmětný vektor či matici.

Obsah
21. strana ze 22
Zavřít dokument
Konec
Celá obrazovka/Okno

Literatura

[1] *Algoritmus*. Webové stránky zaměřené na tvorbu algoritmů. [on-line]. <http://www.algoritmy.net>. (Citováno na s 4.)

Obsah

22. strana ze 22

Zavřít dokument

Konec

Celá obrazovka/Okno