

Řazení údajů

Programovací techniky

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

- Řazení = uspořádání údajů podle velikosti

- Řazení = uspořádání údajů podle velikosti
- Existuje relace úplného uspořádání na množině dat

- Řazení = uspořádání údajů podle velikosti
- Existuje relace úplného uspořádání na množině dat
- Třídění = rozdělení množiny dat na části (rozklad množiny)

- Řazení = uspořádání údajů podle velikosti
- Existuje relace úplného uspořádání na množině dat
- Třídění = rozdělení množiny dat na části (rozklad množiny)
- Existuje relace ekvivalence

- Přímé metody = realizace základního principu

- Přímé metody = realizace základního principu
- Modifikované metody = „zlepšení“ principu

- Přímé metody = realizace základního principu
- Modifikované metody = „zlepšení“ principu
- in situ = prostorová složitost je dána vstupními daty

- Přímé metody = realizace základního principu
- Modifikované metody = „zlepšení“ principu
- in situ = prostorová složitost je dána vstupními daty
- stabilita metody = data se stejnými klíči se nepřeskupují

- Přímé metody = realizace základního principu
- Modifikované metody = „zlepšení“ principu
- in situ = prostorová složitost je dána vstupními daty
- stabilita metody = data se stejnými klíči se nepřeskupují
- přímé řazení / nepřímé řazení =
přeskupují/nepřeskupují se řazené údaje

- výběrové

- výběrové
- vkládací

- výběrové
- vkládací
- rozdělovací

- výběrové
- vkládací
- rozdělovací
- slučovací

- výběrové
- vkládací
- rozdělovací
- slučovací
- Výběr je singulární případ rozdělování, vkládání je singulární případ slučování

- Přirozené: jsou schopny využít částečně seřazených dat

- Přirozené: jsou schopny využít částečně seřazených dat
- Podle časové složitosti lineární, složené logaritmické, kvadratické, horší

- Přirozené: jsou schopny využít částečně seřazených dat
- Podle časové složitosti lineární, složené logaritmické, kvadratické, horší
- Ve všech dalších příkladech: Pole P, obsazeno prvních N indexů, data jsou porovnatelná

- Princip: výběr extrému a jeho uložení do cílového pole

- Princip: výběr extrému a jeho uložení do cílového pole
- Varianta in-situ: pole má dvě části

- Princip: výběr extrému a jeho uložení do cílového pole
- Varianta in-situ: pole má dvě části
- Obecná procedura pro záměnu prvků:

```
procedure Zamena(var P: Pole; A, B: Indexy);  
  var Pom: Slozka;  
  begin if A<>B then begin  
    Pom:=P[A];  
    P[A]:=P[B];  
    P[B]:=Pom  
  end  
end;
```


- Funkce pro nalezení extrému mezi zadanými indexy:

```
function Extrem(P: Pole;  
                A, B: indexy): Indexy;  
var V, I: Indexy;  
begin V:=A;  
      for I:=A+1 to B do  
        if P[I]<P[V] then V:=I;  
      Extrem:=V  
end;
```

- Funkce pro nalezení extrému mezi zadanými indexy:

```
function Extrem(P: Pole;  
                A, B: indexy): Indexy;  
var V, I: Indexy;  
begin V:=A;  
      for I:=A+1 to B do  
        if P[I]<P[V] then V:=I;  
      Extrem:=V  
end;
```

- Vlastní řazení:

```
for I:=N downto 2 do  
  Zamena(P, Extrem(P, 1, I), I);
```

- Funkce pro nalezení extrému mezi zadanými indexy:

```
function Extrem(P: Pole;  
                A, B: indexy): Indexy;  
var V, I: Indexy;  
begin V:=A;  
      for I:=A+1 to B do  
        if P[I]<P[V] then V:=I;  
      Extrem:=V  
end;
```

- Vlastní řazení:

```
for I:=N downto 2 do  
  Zamena(P, Extrem(P, 1, I), I);
```

- Časová složitost kvadratická ($k \cdot N^2$)

- Varianta s detekcí uspořádání – přirozená

```
J:=1;
Jeste:=true;
while Jeste do begin
    Jeste:=false;
    for I:=1 to N-J do
        if P[I]>P[I+1] then begin
            Zamena(P, I, I+1);
            Jeste:=true
        end;
    inc(J)
end;
```


- Varianta s detekcí uspořádání – přirozená

```
J:=1;
Jeste:=true;
while Jeste do begin
  Jeste:=false;
  for I:=1 to N-J do
    if P[I]>P[I+1] then begin
      Zamena(P, I, I+1);
      Jeste:=true
    end;
  inc(J)
end;
```

- Přirozená, sekvenční, stabilní

- Varianta s detekcí uspořádání – přirozená

```
J:=1;
Jeste:=true;
while Jeste do begin
  Jeste:=false;
  for I:=1 to N-J do
    if P[I]>P[I+1] then begin
      Zamena(P, I, I+1);
      Jeste:=true
    end;
  inc(J)
end;
```

- Přirozená, sekvenční, stabilní
- Implementační jednoduchost

- Varianta s detekcí uspořádání – přirozená

```
J:=1;
Jeste:=true;
while Jeste do begin
  Jeste:=false;
  for I:=1 to N-J do
    if P[I]>P[I+1] then begin
      Zamena(P, I, I+1);
      Jeste:=true
    end;
  inc(J)
end;
```

- Přirozená, sekvenční, stabilní
- Implementační jednoduchost
- Časová složitost kvadratická

- Pojem hromada – uspořádaný strom, kde otec je větší než libovolný syn, mezi syny není definováno uspořádání

- Pojem hromada – uspořádaný strom, kde otec je větší než libovolný syn, mezi syny není definováno uspořádání
- Ustavení hromady – přehození prvků tak, aby platilo pravidlo hromady

- Pojem hromada – uspořádaný strom, kde otec je větší než libovolný syn, mezi syny není definováno uspořádání
- Ustavení hromady – přehození prvků tak, aby platilo pravidlo hromady
- Kořen stromu je extrém

- Pojem hromada – uspořádaný strom, kde otec je větší než libovolný syn, mezi syny není definováno uspořádání
- Ustavení hromady – přehození prvků tak, aby platilo pravidlo hromady
- Kořen stromu je extrém
- Strom je reprezentován polem, index levého syna je dvojnásobkem indexu otce, index pravého syna je následníkem levého syna

- Pojem hromada – uspořádaný strom, kde otec je větší než libovolný syn, mezi syny není definováno uspořádání
- Ustavení hromady – přehození prvků tak, aby platilo pravidlo hromady
- Kořen stromu je extrém
- Strom je reprezentován polem, index levého syna je dvojnásobkem indexu otce, index pravého syna je následníkem levého syna
- Velmi efektivní metoda

- Vlastní řazení má dva kroky: ustavení hromady

```
| for I:=N div 2 downto 1 do Sift(I, N);
```

- Vlastní řazení má dva kroky: ustavení hromady

```
| for I:=N div 2 downto 1 do Sift(I, N);
```

- Řazení – odebrání kořene a ustavení hromady

```
| for I:=N downto 2 do begin  
    Zamena(P, 1, I);  
    Sift(1, I-1)  
end;
```


- Klíčová procedura `sift`

- Klíčová procedura `sift`
- Příprava:

```
procedure Sift(L, R: Index);  
  var I, J: Index; Pom: Prvek;  
      Jeste: Boolean;  
begin I:=L;  
      J:=I*2;  
      Pom:=P[I];  
      Jeste:=J<=R;
```

```
while Jeste do begin
  if J<R then
    if P[J]<P[J+1] then J:=J+1;
    if P[I]<P[J] then begin
      P[I]:=P[J];
      I:=J;
      J:=I*2;
      Jeste:=J<=R
    end else Jeste:=false;
    P[I]:=Pom;
  end;
end;
```


- Nepřirozená, nestabilní, in-situ

- Nepřirozená, nestabilní, in-situ
- Časová složitost nejlepší = $k \cdot N \cdot \log_2 N$

- nalezení vhodného místa pro vkládaný prvek

- nalezení vhodného místa pro vkládaný prvek
- odsun následných prvků

- nalezení vhodného místa pro vkládaný prvek
- odsun následných prvků
- vložení nového prvku

- nalezení vhodného místa pro vkládaný prvek
- odsun následných prvků
- vložení nového prvku
- in-situ, nesekvenční, stabilní varianta

- nalezení vhodného místa pro vkládaný prvek
- odsun následných prvků
- vložení nového prvku
- in-situ, nesekvenční, stabilní varianta
- časová složitost $k \cdot N^2$

- nalezení vhodného místa pro vkládaný prvek
- odsun následných prvků
- vložení nového prvku
- in-situ, nesekvenční, stabilní varianta
- časová složitost $k \cdot N^2$
- pokud je hledání půlením intervalu, lze zlepšit časovou složitost o jednu třídu na lineárně logaritmickou = $k \cdot N \cdot \log_2 N$

- vkládání nových prvků do stromové struktury v uspořádání $o \leq ls, o > ps$

- vkládání nových prvků do stromové struktury v uspořádání $o \leq l_s, o > p_s$
- následné provedení operace inorder

```
procedure Vloz (var S: UkUzel; D: Prvek);  
  begin if S=nil then begin  
    new(S);  
    S^.Data:=D;  
    S^.Vlevo:=nil;  
    S^.vpravo:=nil  
  end else  
    if D<S^.Data then Vloz(S^.Vlevo,D)  
    else Vloz(S^.Vpravo,D)  
  end;
```


- Vlastní řazení:

```
for I:=1 to N do begin
    read(X);
    Vloz(S, X)
end;
Inorder(S)
```

- Vlastní řazení:

```
for I:=1 to N do begin
    read(X);
    Vloz(S, X)
end;
Inorder(S)
```

- časová složitost je $k \cdot N \cdot \log_2 N$, nestabilní, nepřirozená

- Vlastní řazení:

```
for I:=1 to N do begin
    read(X);
    Vloz(S, X)
end;
Inorder(S)
```

- časová složitost je $k \cdot N \cdot \log_2 N$, nestabilní, nepřirozená
- implementačně jednoduchá, prostorová složitost úměrná rekurzivnímu zanoření $O(S) = k \cdot \log_2 N$

- Ideální časová složitost (lineární)

- Ideální časová složitost (lineární)
- Velké omezení dat

- Ideální časová složitost (lineární)
- Velké omezení dat
- Zlepšení: logické pole, celočíselné pole

- Ideální časová složitost (lineární)
- Velké omezení dat
- Zlepšení: logické pole, celočíselné pole
- Základní princip:

```
var M: set of TypData;  
    d: TypData;  
begin M:=[];  
    while not eof do begin  
        read(d);  
        M:=M + [d];  
    end;  
    for d:=DM to HM do  
        if d in M then write(d);  
    end.
```

Řazení rozptylováním

- Hodnoty lib. typu převádíme (rozptylováním) do hodnot ordinálního typu

- Hodnoty lib. typu převádíme (rozptylováním) do hodnot ordinálního typu
- V synonymech rozptylování musí existovat uspořádání ve stejném smyslu jako v původních datech

- Hodnoty lib. typu převádíme (rozptylováním) do hodnot ordinálního typu
- V synonymech rozptylování musí existovat uspořádání ve stejném smyslu jako v původních datech
- Překročení omezení dat při řazení množinou

- Hodnoty lib. typu převádíme (rozptylováním) do hodnot ordinálního typu
- V synonymech rozptylování musí existovat uspořádání ve stejném smyslu jako v původních datech
- Překročení omezení dat při řazení množinou
- Efektivita metody závislá na rozptylovací funkci, jejíž vlastnosti (zachování uspořádání) jsou obvykle náročné na implementaci

- Jedna z nejrychlejších metod

- Jedna z nejrychlejších metod
- Rekurzivní zápis vede k větší prostorové složitosti

- Jedna z nejrychlejších metod
- Rekurzivní zápis vede k větší prostorové složitosti
- Základní algoritmus:

```
begin Rozdel;  
    if L<J then QuickSort(L, J);  
    if I<R then QuickSort(I, R)  
end;
```



```
procedure QuickSort(L, R: word);
var I, J: word;
    X: TypData;
    procedure Rozdel;
    begin I:=L; J:=R;
          X:=P[(I+J) div 2];
          repeat
            while P[I]<X do Inc(I);
            while P[J]>X do Dec(J);
            if I<=J then begin
              Zamena(P, I, J);
              Inc(I);
              Dec(J)
            end
          until I>J
    end;
end;
```


- Časová složitost: lineárně logaritmická

- Časová složitost: lineárně logaritmická
- Prostorová složitost (rekurze): logaritmická

- Časová složitost: lineárně logaritmická
- Prostorová složitost (rekurze): logaritmická
- Uspořádaná data – problematické

- Slučování dvou proudů uspořádaných dat

- Slučování dvou proudů uspořádaných dat
- Cyklické rozdělování a slučování

- Slučování dvou proudů uspořádaných dat
- Cyklické rozdělování a slučování
- Lineárně logaritmická složitost

- Slučování dvou proudů uspořádaných dat
- Cyklické rozdělování a slučování
- Lineárně logaritmická složitost
- Jeden krok slučování: práce se dvěma soubory, se dvěma frontami (seznamy)