Aplikovaná informatika

Podklady předmětu
Aplikovaná informatika
pro akademický rok 2013/2014
Radim Farana

5

Obsah

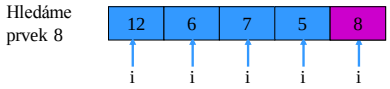
- Algoritmy vyhledávání,
  - sekvenční prohledávání,
  - binární prohledávání.
- Algoritmy třídění,
  - definice problému,
  - rozdělení algoritmů,
  - hodnocení výkonnosti.
- Algoritmy interního třídění.
  - stabilní algoritmy,
  - nestabilní algoritmy.

Definice datové struktury

- Data tvoří záznamy (množiny prvků).
- Jeden z prvků je klíč třídění.
- Datový typ klíče je vždy jednoduchý.
- Klíč je neoddělitelný od zbytku záznamu.
- Záznamy jsou v paměti organizovány do pole záznamů, externě do sekvenčního souboru – obecně do seznamu.
- V algoritmu zjednodušeně pracujeme s prvkem pole jako by byl sám klíčem.

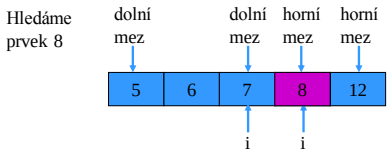
Vyhledávání

- Lineární prohledávání (seznam dat není setříděn), hledá od začátku do konce
  - počet porovnání maximálně *n*.



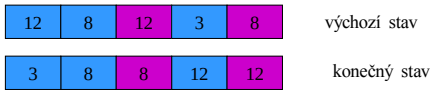
Vyhledávání

- Binární prohledávání (seznam dat je setříděn), hledá prvek vždy v polovině vzdálenosti krajních prvků skupiny
  - počet porovnání vždy $\lceil \log_2 n \rceil$



Vlastnosti algoritmů třídění

- Stabilní algoritmy – zachovávají původní vzájemnou polohu prvků stejné hodnoty klíče
  - důležité pro realizaci složených klíčů.



- Nestabilní algoritmy – vzájemná poloha prvků stejné hodnoty klíče se může změnit.

Hodnocení výkonnosti algoritmů

- Operace ovlivňující významně algoritmus:
  - Porovnání hodnot – závisí na datovém typu, náročné zejména u textových datových typů
  - Přiřazení prvků – závisí na velikosti záznamu (množství dat souvisejících s klíčem).
- Sledují se odděleně.
- Základní případy:
  - setříděný seznam (teoreticky bez nároků),
  - náhodný seznam (průměrné nároky),
  - opačně setříděný seznam (teoreticky max. nároky)

Rozdělení algoritmů podle výkonnosti

- Přirozené algoritmy – náročnost roste od setříděného přes náhodný k opačně setříděnému seznamu.
- Ostatní algoritmy nejsou přirozené.
- Rozdělení nemá na vlastní náročnost nebo aplikaci algoritmů žádný vliv.

Rozdělení algoritmů podle přístupu

- Interní algoritmy (vnitřní)
  - data tvoří pole záznamů v paměti,
  - přímý přístup ke všem záznamům,
  - označení pole a (1 To n).
- Externí algoritmy (vnější)
  - data jsou uložena v externím (sekvenčním souboru),
  - čtení soubor je možné jen jednosměrně,
  - máme k dispozici větší počet externích souborů,
  - čtení/zápis do souboru je možno střídat.

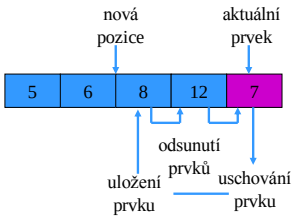
Interní algoritmy třídění

- Stabilní algoritmy třídění
  - Algoritmus přímého vkládání (Insertion Sort)
  - Algoritmus přímého výběru (Selection Sort)
  - Algoritmus přímé výměny (Bubble Sort)
  - Algoritmus třídění přetřásáním (Shake Sort)
- Nestabilní algoritmy třídění
  - Třídění zmenšováním kroku (Shell Sort)
  - Třídění rozdělováním (Quick Sort)
  - Třídění hromadou (Heap Sort) – též haldou

Přímé vkládání (Insertion Sort)

Myšlenka: ze seznamu postupně vybíráme prvky a zařazujeme je na správné místo v již setříděné části seznamu. Překážející část seznamu je odsunuta dozadu aby uvolnila místo.

Příklad:



Přímé vkládání (Insertion Sort)

Algoritmus:

```
for i = 2 to n                  pro všechny prvky mimo prvního
     v = a(i)                   uschování tříděného prvku
     j = i
     while j>1 and                  pokud je předchozí prvek menší
           a(j-1) > v
          a(j) = a(j-1)         posun prvku
          j = j-1
     end while
     a(j) = v                   uložení tříděného prvku
end for
```

Přímé vkládání (Insertion Sort)

Analýza algoritmu:

krok (*i*)	setříděný		opačně setříděný	
	porovnání	přiřazení	porovnání	přiřazení
2	1	2	1	2+1
3	1	2	2	2+2
4	1	2	3	2+3
5	1	2	4	2+4
…	…	…	…	…
n	1	2	*n*-1	2+*n*-1

Přímé vkládání (Insertion Sort)

Analýza algoritmu:

opačně setříděný	
porovnání	přiřazení
1	2+1
2	2+2
3	2+3
4	2+4
…	…
n -1	2+*n* -1

Součet aritmetické řady:

$a_2 = 1,\ a_n = n\text{-}1,\ n\text{-}1$ prvků

$$S_a = \frac{a_2 + a_n}{2}(n-1) = \frac{1+n-1}{2}(n-1) = \frac{n^2 - n}{2}$$

$a_2 = 3,\ a_n = n+1,\ n\text{-}1$ prvků

$$S_a = \frac{a_2 + a_n}{2}(n-1) = \frac{n+4}{2}(n-1) = \frac{n^2 + 3n - 4}{2}$$

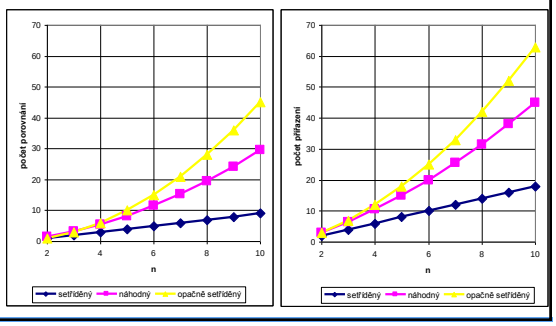
Přímé vkládání (Insertion Sort)

Složitost:

seznam	porovnání	přiřazení
setříděný	$n-1$	$2(n-1)$
náhodný	$\frac{n^2+n-2}{4}$	$\frac{n^2+9n-10}{4}$
opačně setříděný	$\frac{n^2-n}{2}$	$\frac{n^2+3n-4}{2}$

Algoritmus je přirozený

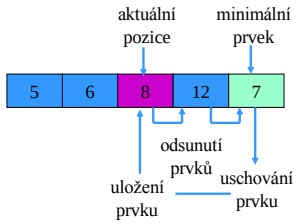
Přímé vkládání (Insertion Sort)



Přímý výběr (Selection Sort)

Myšlenka: ze seznamu postupně vybíráme prvky od nejmenšího (prvního z nejmenších) a zařazujeme na aktuální místo v seznamu (překážející část seznamu je odsunuta dozadu).

Příklad:



Přímý výběr (Selection Sort)

Algoritmus:

```
for i = 1 to n-1                pro všechny prvky mimo poslední
     min = i                    aktuální prvek je minimum
     for j = i+1 to n           pro zbytek souboru prvků
          if a(j) < a(min)      najdi první minimum
               min=j
          end if
     end for
     v=a(min)                   uschování minima
     for j = min to i+1 step -1
         a(j) = a(j-1)          odsunutí prvků dozadu
     end for
     a(i)=v                     uložení minima
end for
```

Přímý výběr (Selection Sort)

Analýza algoritmu:

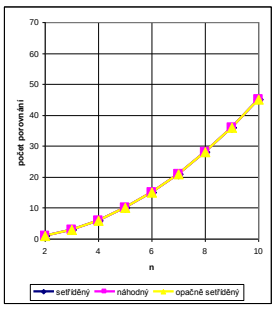
krok (i)	setříděný		opačně setříděný	
	porovnání	přiřazení	porovnání	přiřazení
1	n-1	2	n-1	2+n-1
2	n-2	2	n-2	2+n-2
3	n-3	2	n-3	2+n-3
4	n-4	2	n-4	2+n-4
…	…	…	…	…
n-1	1	2	1	2+1

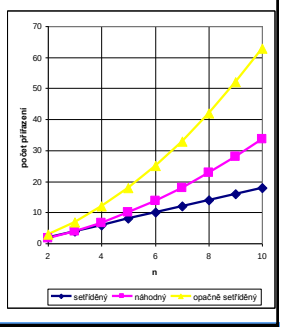
Přímý výběr (Selection Sort)

Složitost:

seznam	porovnání	přiřazení
setříděný	$\frac{n^2-n}{2}$	$2(n-1)$
náhodný	$\frac{n^2-n}{2}$	$\frac{n^2+4n-5}{4}$
opačně setříděný	$\frac{n^2-n}{2}$	$\frac{n^2+3n-4}{2}$

Přímý výběr (Selection Sort)





Přímý výběr (Selection Sort) upravený

Algoritmus:

```
for i = 1 to n-1                    pro všechny prvky mimo poslední
     min = i                        aktuální prvek je minimum
     for j = i+1 to n               pro zbytek souboru prvků
          if a(j) < a(min)          najdi první minimum
               min=j
          end if
     end for
     v=a(min)
     a(min)=a(i)                    Pozor pouhá výměna prvků
     a(i)=v                         způsobuje nestabilitu algoritmu
end for
```


Přímý výběr (Selection Sort) upravený

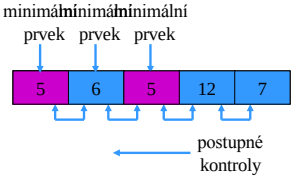
Složitost:

seznam	porovnání	přiřazení
setříděný	$\frac{n^2-n}{2}$	$3(n-1)$
náhodný	$\frac{n^2-n}{2}$	$3(n-1)$
opačně setříděný	$\frac{n^2-n}{2}$	$3(n-1)$

Přímá výměna (Bubble Sort)

Myšlenka: postupně porovnáváme sousední prvky od posledního k prvnímu, pokud nejsou ve správném pořadí, prohodíme je. Tak se nejmenší prvek dostane na začátek seznamu.

Příklad:



Přímá výměna (Bubble Sort)

Algoritmus:

```
for i = 2 to n                          n-1 cyklů
     for j = n to i step -1             n-i+1 kroků
          if a(j-1) > a(j)              nesprávné pořadí
               v = a(j-1)
               a(j-1) = a(j)            prvky vyměnit
               a(j) = v
          end if
     end for
end for
```


Přímá výměna (Bubble Sort)

Analýza algoritmu:

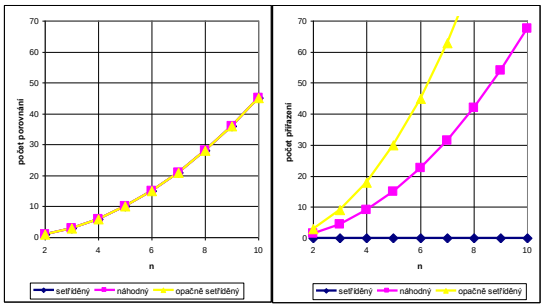
krok (i)	setříděný		opačně setříděný	
	porovnání	přiřazení	porovnání	přiřazení
2	n-1	0	n-1	3(n-1)
3	n-2	0	n-2	3(n-2)
4	n-3	0	n-3	3(n-3)
5	n-4	0	n-4	3(n-4)
…	…	…	…	…
n	1	0	1	3

Přímá výměna (Bubble Sort)

Složitost:

seznam	porovnání	přiřazení
setříděný	$\frac{n^2-n}{2}$	0
náhodný	$\frac{n^2-n}{2}$	$\frac{3(n^2-n)}{4}$
opačně setříděný	$\frac{n^2-n}{2}$	$\frac{3(n^2-n)}{2}$

Přímá výměna (Bubble Sort)



Přímá výměna (Bubble Sort) upravený

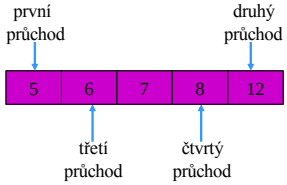
Algoritmus:

```
i = 2; zmena = True
while i < n and zmena=True          opakuj při změně
      zmena = False
      for j = n to i step -1        n-i+1 kroků
            if a(j-1) > a(j)        kontrola pořadí
                  v = a(j-1)        prvky vyměnit
                  a(j-1) = a(j)
                  a(j) = v
                  zmena = True
            end if
      end for
      i = i + 1
end while
```


Přetřásání (Shake Sort)

Myšlenka: u třídění přímou výměnou se nejrychleji zařazují malé prvky. Střídáním směru (spolu s kontrolou provedení výměny) dosáhneme snížení počtu průchodů algoritmem.

Příklad:



Přetřásání (Shake Sort)

Algoritmus:

```
el = 2; r = n; k = n
repeat
        for j = r to el step -1
                if a(j-1) > a(j)
                        swap(a(j), a(j-1))
                        k = j
                end if
        end for
        el = k+1
        for j = el to r
                if a(j-1) > a(j)
                        swap(a(j), a(j-1))
                        k = j
                end if
        end for
        r = k-1
until el>r
```

```
swap( a, b )
        pom = b
        b = a
        a = pom
end swap
```

el = levý okraj
r = pravý okraj
k = pomocná
j = počítadlo

Přetřásání (Shake Sort)

Složitost:

seznam	porovnání	přiřazení
setříděný	$n-1$	0
náhodný	$\frac{n^2-n}{4}$	$\frac{3(n^2-n)}{4}$
opačně setříděný	$\frac{n^2-n}{2}$	$\frac{3(n^2-n)}{2}$

Zmenšování kroku (Shell Sort)

Myšlenka: Shellův třídící algoritmus spočívá ve vytváření tzv. setřídění posloupnosti s krokem ***h***. Vezmeme tedy každý ***h***-tý prvek v seznamu a provedeme na takto vzniklé množině prvků třídění metodou přímého vkládání. Pak vezmeme každý ***h+1*** prvek a provedeme další setřídění atd.
Tím získáme setříděnou posloupnost s krokem ***h***. Uvedeným způsobem se nám podaří vytvořit shluky prvků s blízkými klíči a tím zmenšit počet přesunů prvků na velké vzdálenosti.
h - setříděná posloupnost se pak znovu přetřídí s menším krokem ***h***.
Pro ***h = 1*** dojde tedy k vyvolání standardního algoritmu třídění přímým vkládáním.

Zmenšování kroku (Shell Sort)

Algoritmus:

Jednotlivé kroky jsou uloženy v poli h(1 To t)

```
for m = 1 to t
      k = h(m)
      for i = k+1 to n
            v = a(i)
            j = i - k
            while (v < a(j)) and (j >= k)
                  a(j + k) = a(j)
                  j = j - k
            end while
            a(j + k) = v
      end for
end for
```

Složitost:

uvádí se $O(n^{1,2}) \div O(n^{1,25})$

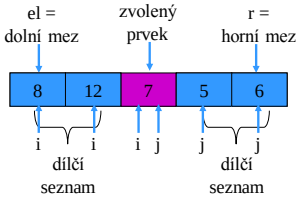
Zmenšování kroku (Shell Sort)

- Problém určení postupných kroků
  - Dobré výsledky dosahuje posloupnost:
    $h_i = 1$ pro $i = 1$
    $h_i = 3\ h_{i-1} + 1$ pro $i \geq 2$
    tedy posloupnost čísel: 1, 4, 13, 40, 121, ...
  - Nebo také posloupnost:
    $h_i = 1$ pro $i = 1$
    $h_i = 2\ h_{i-1} + 1$ pro $i \geq 2$
    tedy posloupnost čísel: 1, 3, 7, 15, 31, ...

Třídění rozdělováním (Quick sort)

Myšlenka: původní seznam rozdělíme na dva disjunktní seznamy, které necháme setřídit samostatně. Dílčí seznamy získáme tak, že zvolíme jeden prvek, pak před něj umístíme všechny prvky menší a za něj větší. Tím se současně určí správná pozice zvoleného prvku.

Příklad:



Třídění rozdělováním (Quick Sort)

Algoritmus:

```
QuickSort (el, r)
    if (r > el)
        i = partition(el, r)
        QuickSort(el, i-1)
        QuickSort(i+1, r)
    end if
end QuickSort
```

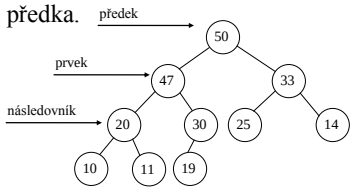
```
partition (el, r)
    v = a(r)
    i = el-1
    j = r
    repeat
        repeat
            i = i+1
        until a(i)>=v
        repeat
            j = j-1
        until (a(j)<=v) OR (j=el)
        swap(a(i), a(j))
    until j<=i
    t = a(j)
    a(j) = a(i)
    a(i) = a(r)
    a(r) = t
    partition = i
end partition
```


Třídění rozdělováním (Quick Sort)

- Problém volby dělicího prvku
  - u setříděné posloupnosti je krajní prvek nejhorší,
  - obvykle se volí $x = a((l + r) \text{ div } 2)$.
- Problém hloubky zásobníku
  - až $2n$ volání je nutno uložit v nejhorším případě,
  - řešením je úprava algoritmu pro ukládání jen jedné dvojice a okamžité zpracování druhé.
- Složitost algoritmu
  - Počet přesunů pro nejhorší případ n^2
  - Počet přesunů pro průměrný případ $\frac{n}{6}$

Třídění hromadou (Heap Sort)

- Hromada (heap) je úplný binární strom, který splňuje podmínku hromady: klíč v každém uzlu má hodnotu menší nebo rovnu klíčové hodnotě jeho předka.



Třídění hromadou (Heap Sort)

- V paměti počítače však bude hromada reprezentována pomocí pole

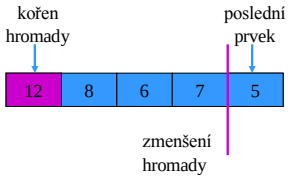
1	2	3	4	5	6	7	8	9	10
50	47	33	20	30	25	14	10	11	19

- rodičovský uzel je vždy na pozici ***(j* div 2)**
- uzel potomka je vždy na pozici ***2j*** [levý] a **(2*j* + 1)** [pravý]

Třídění hromadou (Heap Sort)

Myšlenka: Vybudujeme hromadu z daných prvků, které mají být tříděny. Odebereme prvek umístěný v kořeni (největší), vyměníme ho s posledním prvkem a snížíme počet uzlů ve stromu o 1. Tím jsme porušili podmínku hromady a hromadu je nutné přebudovat.

Příklad:



Třídění hromadou (Heap Sort)

Algoritmus:

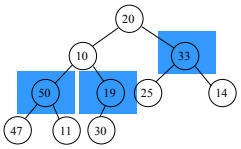
n - počet prvků umístěných na hromadě,
m - počet prvků, které se mají setřídit,
a - pole obsahující prvky hromady.

```
HeapSort
      n = m
      for  k = (m div 2) to 1 step -1
            downheap(k)
      end for
      repeat
            swap(a(1), a(n))
            n = n - 1
            downheap(1)
      until n <= 1
end HeapSort
```

```
downheap (k)
      v = a(k)
      while k <= (n div 2)
            j = 2 * k
            if j < n
                  if a(j) < a(j+1)
                        j = j + 1
                  end if
            end if
            if v >= a(j)
                  goto konec_cyklu
            end if
            a(k) = a(j)
            k = j
      end while
konec_cyklu:  a(k) = v
end downheap
```

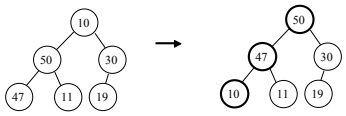
Třídění hromadou (Heap Sort)

Příklad:

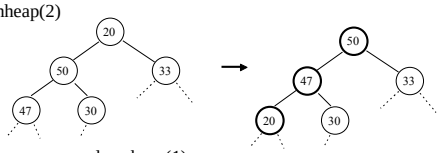


m = 10, funkci downheap(i) voláme pro uzly 5, 4, 3, 2, 1
ostatní jsou koncové uzly

Třídění hromadou (Heap Sort)



downheap(2)



downheap(1)

Třídění hromadou (Heap Sort)

- Analýza algoritmu na dvě části:
  - Inicializaci a vytvoření hromady – $n/2$ volání downheap(), nejhůře $\log_2 n$ - $\log_2 i$ přesunů pro i =1, 2, ..., $n/2$, tedy méně než $\frac{n}{2}\log_2 n$ přesunů.
  - Vlastní třídění vyžaduje n-1 volání downheap(1) vyžadující $\log_2 m$ přesunů, k tomu výměna prvků.
  - Celkový počet přesunů: $\boldsymbol{O(n \cdot \log_2 n)}$.

Speciální algoritmy

- Třídění souboru s malým počtem hodnot klíče, které se často opakují:
  - Třídění distribučním čítáním (Distributing Couting).
- Aplikace algoritmů externího třídění pro třídění interní:
  - Dvoucestné slučování (Merge Sort).

Distribuční čítání (Distributing Couting)

Myšlenka: klíče jsou v rozsahu **<0, (*m* - 1)>**, kde ***m*** je malé číslo. Vypočteme frekvence výskytu jednotlivých klíčů a určíme diskrétní distribuční funkci. Na základě vypočtení distribuční funkce utřídíme prvky seznamu.

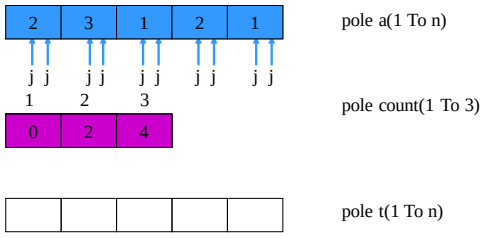
K realizaci potřebujeme pole count (0 To m-1) celých čísel, které využijeme pro výpočet frekvence výskytu klíčů a následně pro výpočet diskrétní distribuční funkce.

Distribuční čítání (Distributing Couting)

Algoritmus:

```
REM - vytvoreni pocitadla vyskytu jednotlivych klicu
for j = 0 to m - 1
      count (j) = 0
end for
REM - vypocet frekvencni funkce
for j = 1 to n
      count (a (j)) = count (a (j)) + 1
end for
REM - vypocet distribucni funkce
for j = 1 to m - 1
      count (j) = count (j - 1) + count (j)
end for
REM - vlastní setrideni, pole t je pomocne pole
for j = n to 1 step -1
      t ( count (a (j))) = a(j)
      count (a (j)) = count (a (j)) -1
end for
```

Distribuční čítání (Distributing Couting)



Distribuční čítání (Distributing Couting)

- Algoritmus neobsahuje žádné porovnání.
- Počet přiřazení do pole diskrétní distribuční funkce = $m+n+m+n = 2m+2n$ přiřazení, pro malé m tedy jen $2n$ přiřazení. Jedná se přitom o jednoduchá celá čísla.
- Počet přiřazení do pole prvků = n.
- Vyžaduje se ale dvojnásobné množství paměti.
- Algoritmus je stabilní.