Třídění a vyhledávání

Obsah

1. Úvod
2. Třídění
   - Bubble sort
   - Select sort
   - Insert sort
   - Quick sort
   - Merge sort
   - Radix sort
3. Vyhledávání
   - Vyhledávání v poli
   - Vyhledávací stromy
4. Shrnutí

Komentář k přednášce

- mnoho různých algoritmů pro stejný účel – třídění
- většina programovacích jazyků má vestavěnou podporu (funkce `sort()`)

Proč se tím tedy zabýváme?

Komentář k přednášce

Proč se tím tedy zabýváme?

1. tradice
2. ilustrace algoritmického myšlení, technik návrhu algoritmů
3. ilustrace drobné změny algoritmu s velkým dopadem na rychlost programu
4. hezky se to vizualizuje a vysvětluje

Doporučený zdroj

`http://www.sorting-algorithms.com/`

- animace
- kódy
- vizualizace

Více podobných: Google → sorting algorithms

Třídění: problém

- vstup: posloupnost (přirozených) čísel
  např. 8, 2, 14, 3, 7, 9
- výstup: setříděná posloupnost
  např. 2, 3, 7, 8, 9, 14

Co vy na to?

- zkuste vymyslet
  - třídící algoritmus
  - co nejvíce různých principů
  - co nejefektivnější algoritmus
- možná inspirace: jak třídíte karty?

Složitost

n – délka vstupní posloupnosti

	počet operací
jednoduché algoritmy	$O(n^2)$, „kvadratická“
složitější algoritmy	$O(n \log(n))$

Bubble sort

- probublávání vyšších hodnot nahoru
- srovnávání a prohazování sousedů

```
for i = 1:n
  for j = n:i+1
    if a[j] < a[j-1],
        swap a[j,j-1]
  //invariant: a[1..i] in final position
```

Select sort

- třídění výběrem
- projdeme dosud nesetříděnou část pole a výbereme nejmenší prvek
- nejmenší prvek zařadíme na aktuální pozici (výměnou)

Select sort

Algorithm 1.4 SELECTIONSORT
Input: An array $A[1..n]$ of n elements.
Output: $A[1..n]$ sorted in nondecreasing order.

```
1. for i ← 1 to n − 1
2.     k ← i
3.     for j ← i + 1 to n      {Find the ith smallest element.}
4.         if A[j] < A[k] then k ← j
5.     end for
6.     if k ≠ i then interchange A[i] and A[k]
7. end for
```

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

Insert sort

- zatřiďování
- prefix pole udržujeme setříděný
- každou další hodnotu zařadíme tam, kam patří
- „třídění karet“

Insert sort

Algorithm 1.5 INSERTIONSORT
Input: An array $A[1..n]$ of n elements.
Output: $A[1..n]$ sorted in nondecreasing order.

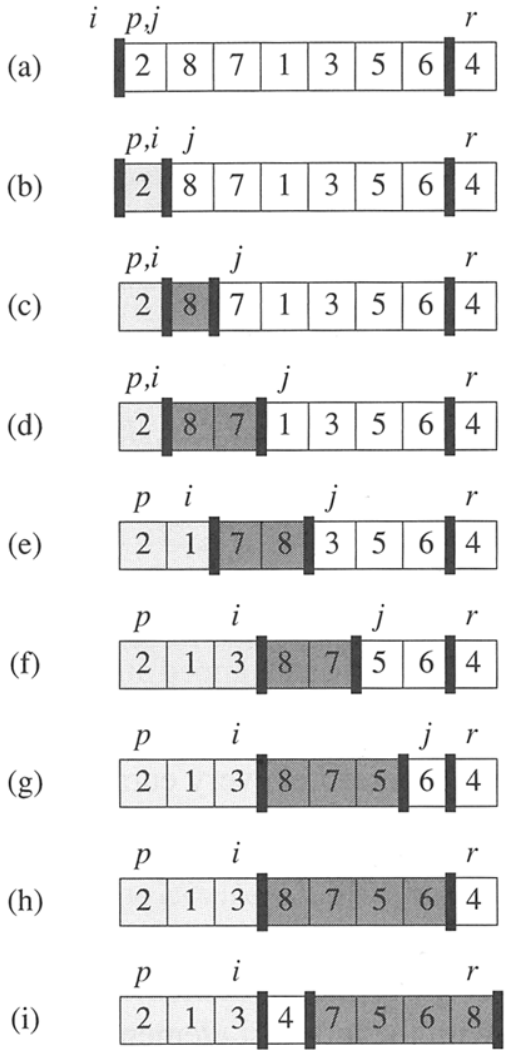
```
1. for i ← 2 to n
2.     x ← A[i]
3.     j ← i − 1
4.     while (j > 0) and (A[j] > x)
5.         A[j + 1] ← A[j]
6.         j ← j − 1
7.     end while
8.     A[j + 1] ← x
9. end for
```

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

Quick sort

- rekurzivní algoritmus
- vybereme „pivota“ a pole rozdělíme na dvě části:
  - větší než pivot
  - menší než pivot
- obě části pak nezávisle setřídíme (rekurzivně pomocí quick sortu)

Quick sort ilustrace



T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

Quick sort

- pokud máme smůlu při výběru pivota, tak je stejně pomalý jako předchozí
- v průměrném případě je rychlý ($O(n \log(n))$ –*quick*

Merge sort

- rekurzivní algoritmus
- pole rozdělíme na dvě poloviny a ty setřídíme (merge sort)
- ze setříděných polovin vyrobíme jedno setříděné pole – „zipování“

Merge sort

Algorithm 6.3 MERGESORT
Input: An array $A[1..n]$ of n elements.
Output: $A[1..n]$ sorted in nondecreasing order.

1. $mergesort(A, 1, n)$

Procedure $mergesort(low, high)$

1. **if** $low < high$ **then**
2. $\quad mid \leftarrow \lfloor (low + high)/2 \rfloor$
3. $\quad mergesort(A, low, mid)$
4. $\quad mergesort(A, mid + 1, high)$
5. $\quad$ MERGE $(A, low, mid, high)$
6. **end if**

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

Operace Merge

Algorithm 1.3 MERGE

Input: An array $A[1..m]$ of elements and three indices p, q and r, with $1 \leq p \leq q < r \leq m$, such that both the subarrays $A[p..q]$ and $A[q+1..r]$ are sorted individually in nondecreasing order.

Output: $A[p..r]$ contains the result of merging the two subarrays $A[p..q]$ and $A[q+1..r]$.

1. **comment:** $B[p..r]$ *is an auxiliary array.*
2. $s \leftarrow p; \quad t \leftarrow q+1; \quad k \leftarrow p$
3. **while** $s \leq q$ **and** $t \leq r$
4. **if** $A[s] \leq A[t]$ **then**
5. $B[k] \leftarrow A[s]$
6. $s \leftarrow s+1$
7. **else**
8. $B[k] \leftarrow A[t]$
9. $t \leftarrow t+1$
10. **end if**
11. $k \leftarrow k+1$
12. **end while**
13. **if** $s = q+1$ **then** $B[k..r] \leftarrow A[t..r]$
14. **else** $B[k..r] \leftarrow A[s..q]$
15. **end if**
16. $A[p..r] \leftarrow B[p..r]$

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

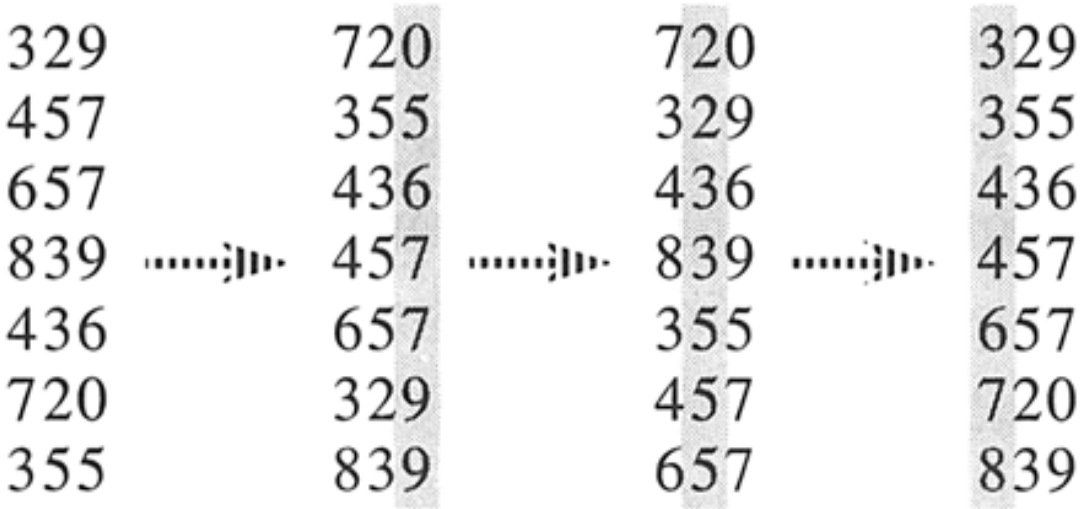
Radix sort

- předchozí algoritmy využívají pouze operaci porovnání dvou hodnot
- aplikovatelné na cokoliv, co lze porovnávat, žádné další předpoklady
- *s doplňujícími předpoklady můžeme dostat nové algoritmy* (obecný princip)

Radix sort

- aplikovatelné na (krátká) čísla
- postupujeme od nejméně významné cifry k nejvýznamější
- setřídíme pole podle dané cifry = rozdělení do 10 „kyblíčků“ (jednoduché, rychlé)

Radix sort ilustrace



T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

Vyhledávání: hra

- myslím si přirozené číslo X mezi 1 a 1000
- povolená otázka: „Je X menší než N?“
- kolik otázek potřebujete na odhalení čísla?
- (kolik předem formulovaných otázek potřebujete?)

Vyhledávání: problém

mnoho různých variant, základní verze:

- vstup: setříděná posloupnost čísel + dotaz (číslo)
  např. 2, 3, 7, 8, 9, 14 + dotaz 8
- výstup: index hledaného čísla v posloupnosti (případně 0 pokud tam není)
  např. 4

Vyhledávání a logaritmus

zjednodušeně platí:

- základní metoda: půlení intervalu
- rozumné vyhledávání – logaritmický počet kroků (vzhledem k délce seznamu)
- lineární vyhledávání – jen velmi krátké seznamy

Vyhledávání půlením intervalu

Algorithm 1.2 BINARYSEARCH
Input: An array $A[1..n]$ of n elements sorted in nondecreasing order and an element x.
Output: j if $x = A[j], 1 \leq j \leq n$, and 0 otherwise.

1. $low \leftarrow 1; \quad high \leftarrow n; \quad j \leftarrow 0$
2. **while** $(low \leq high)$ **and** $(j = 0)$
3. $\quad mid \leftarrow \lfloor (low + high)/2 \rfloor$
4. $\quad$ **if** $x = A[mid]$ **then** $j \leftarrow mid$
5. $\quad$ **else if** $x < A[mid]$ **then** $high \leftarrow mid - 1$
6. $\quad$ **else** $low \leftarrow mid + 1$
7. **end while**
8. **return** j

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

Vyhledávání půlením intervalu (rekurzivně)

Algorithm 6.2 BINARYSEARCHREC
Input: An array $A[1..n]$ of n elements sorted in nondecreasing order and an element x.
Output: j if $x = A[j], 1 \leq j \leq n$, and 0 otherwise.

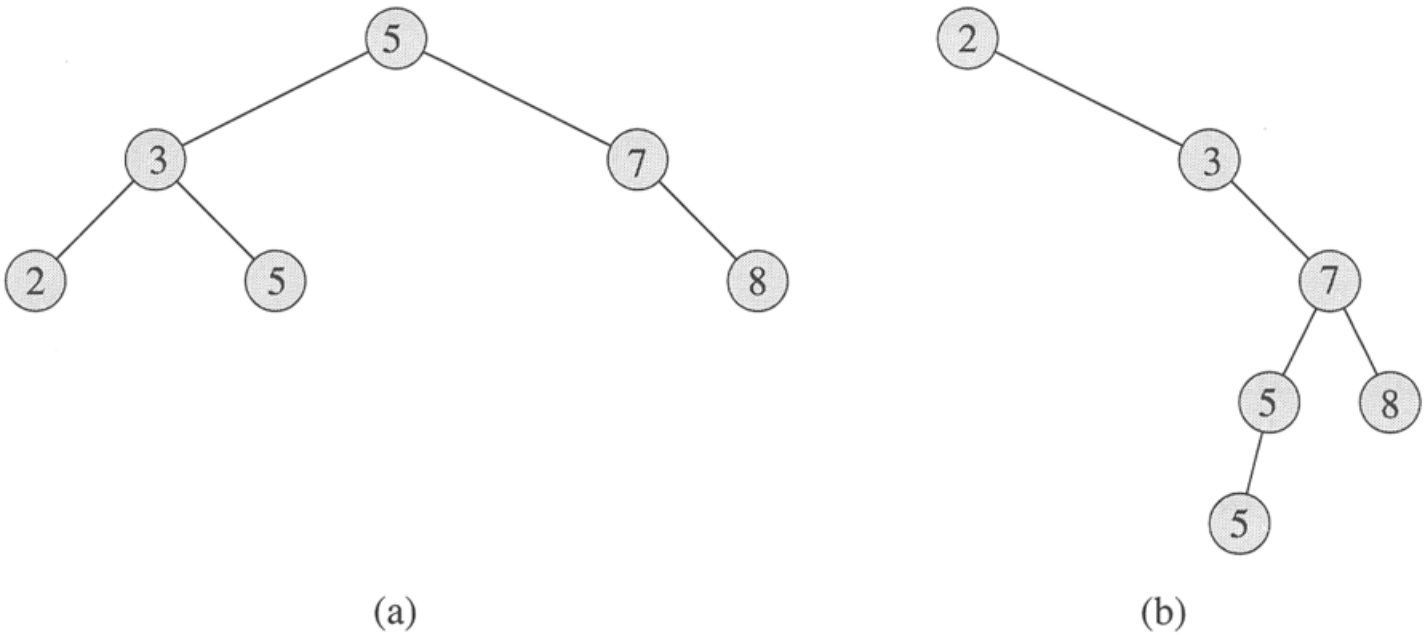
1. *binarysearch*$(1, n)$

Procedure *binarysearch*$(low, high)$

1. **if** $low > high$ **then return** 0
2. **else**
3. $\quad mid \leftarrow \lfloor (low + high)/2 \rfloor$
4. $\quad$ **if** $x = A[mid]$ **then return** mid
5. $\quad$ **else if** $x < A[mid]$ **then return** *binarysearch*$(low, mid - 1)$
6. $\quad$ **else return** *binarysearch*$(mid + 1, high)$
7. **end if**

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

Vyhledávací stromy



T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

Shrnutí

- třídící algoritmy:
  - jednoduché (kvadratické): bubble, selection, insertion
  - složitější (n log n, rekurzivní): quick sort, merge sort
- vyhledávání: půlení intervalu, rekurze