Vyhledávání, třídění, složitost

IB111 Programování a algoritmizace

2010

Otrávené studny

- 8 studen, jedna z nich je otrávená
- laboratorní rozbor
  - dokáže rozpoznat přítomnost jedu ve vodě
  - je drahý
- kolik rozborů potřebujeme?

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 -1 pokud tam není)
  např. 3

Vyhledávání a logaritmus

- naivní metoda = průchod seznamu
  - lineární vyhledávání
  - pomalé (viz např. databáze s milióny záznamů)
  - jen velmi krátké seznamy
- základní rozumná metoda = půlení intervalu
  - analogicky hře s hádáním čísel
  - podívám se na prostřední člen
  - podle jeho hodnoty pokračuji v levém/pravém intervalu
  - logaritmický počet kroků (vzhledem k délce seznamu)

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.

Vyhledávání půlením intervalu (iterativně)

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.

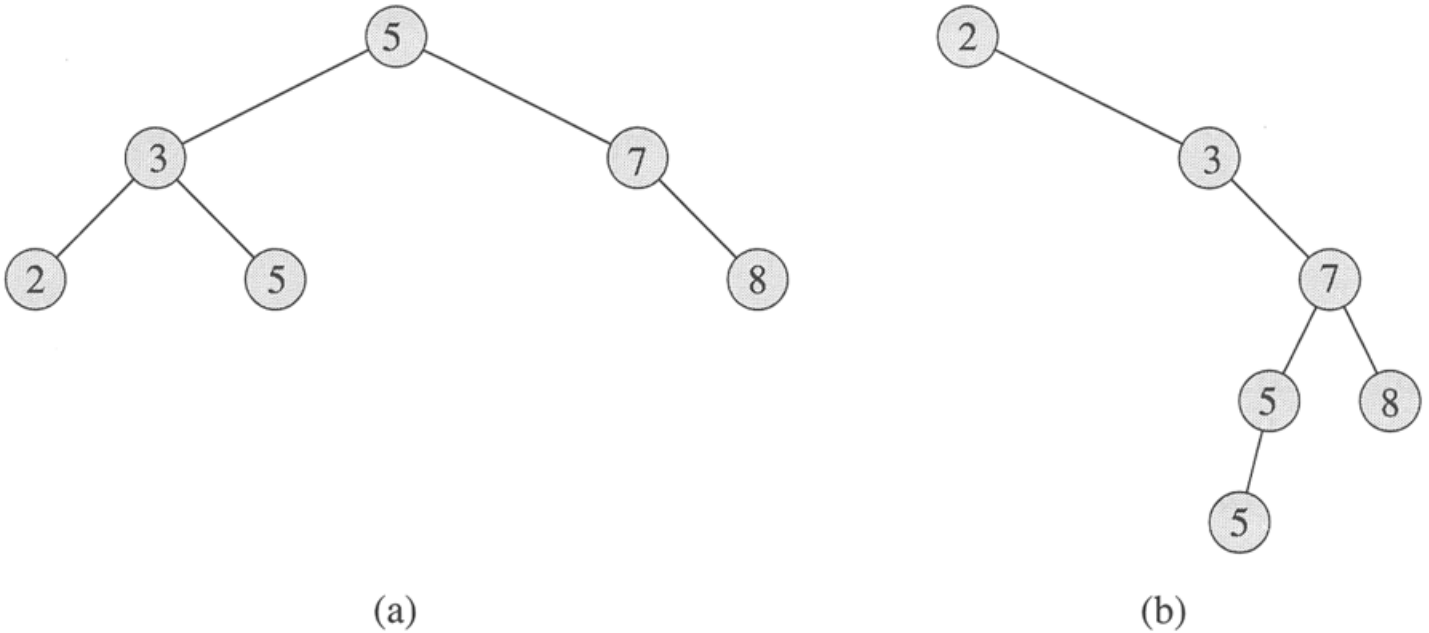
1. $low \leftarrow 1; \quad high \leftarrow n; \quad j \leftarrow 0$
2. **while** $(low \leq high)$ **and** $(j = 0)$
3. $\quad mid \leftarrow \lfloor (low + high)/2 \rfloor$
4. $\quad$ **if** $x = A[mid]$ **then** $j \leftarrow mid$
5. $\quad$ **else if** $x < A[mid]$ **then** $high \leftarrow mid -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

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

Upozornění k uvedeným algoritmům

- uvedené pseudokódy nejsou přímo v Pythonu
- hlavní rozdíl: indexování od 0 vs od 1
- záměr:
  - abyste trénovali čtení různých notací
  - abyste při přepisování kódu museli přemýšlet

Vyhledávací stromy



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

(více později)

Třídění: komentář

- terminologie: třídění, řazení
- mnoho různých algoritmů pro stejný účel
- většina programovacích jazyků má vestavěnou podporu (funkce `sort()`)

Proč se tím tedy zabýváme?

Třídění: komentář

Proč se tím tedy zabýváme?

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

Doporučený zdroj

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

- animace
- kódy
- vizualizace

Více podobných: Google $\longrightarrow$ 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 in range(n):
    print a
    for j in range(n-i-1):
        if a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]
            # paralelní přiřazení;
            # = prohození hodnot
```

invariant cyklu: `a[n-i-1..n-1]` ve finální pozici

Bubble sort: příklad běhu

```
[8, 2, 7, 14, 3, 1]
[2, 7, 8, 3, 1, 14]
[2, 7, 3, 1, 8, 14]
[2, 3, 1, 7, 8, 14]
[2, 1, 3, 7, 8, 14]
[1, 2, 3, 7, 8, 14]
```

Select sort

- třídění výběrem
- projdeme dosud nesetříděnou část pole a vybereme 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.

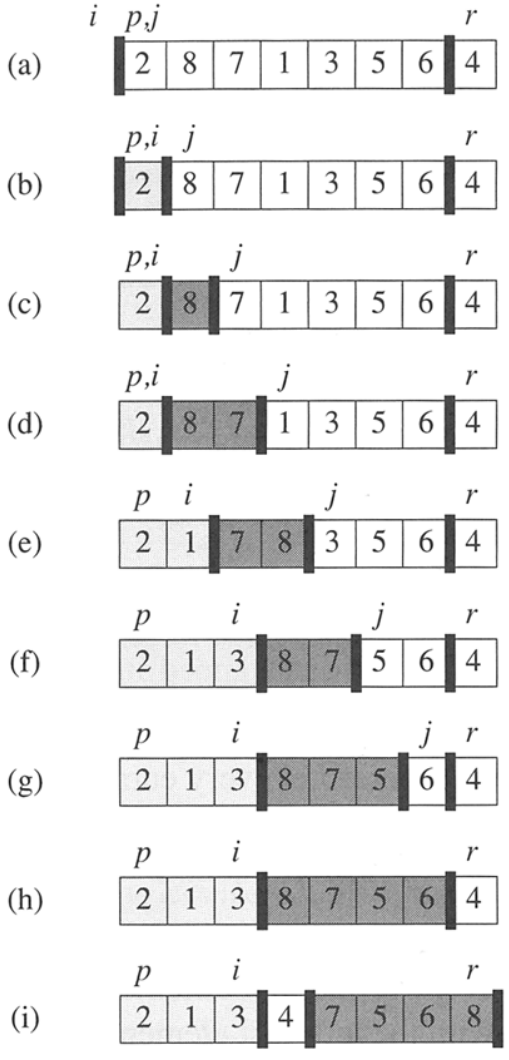
```
1. for i ← 2 to n
2.     x ← A[i]
3.     j ← i − 1
4.     while (j > 0) and (A[j] > x)
5.         A[j + 1] ← A[j]
6.         j ← j − 1
7.     end while
8.     A[j + 1] ← x
9. end for
```

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 – *quick*
  $O(n \log(n))$

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.

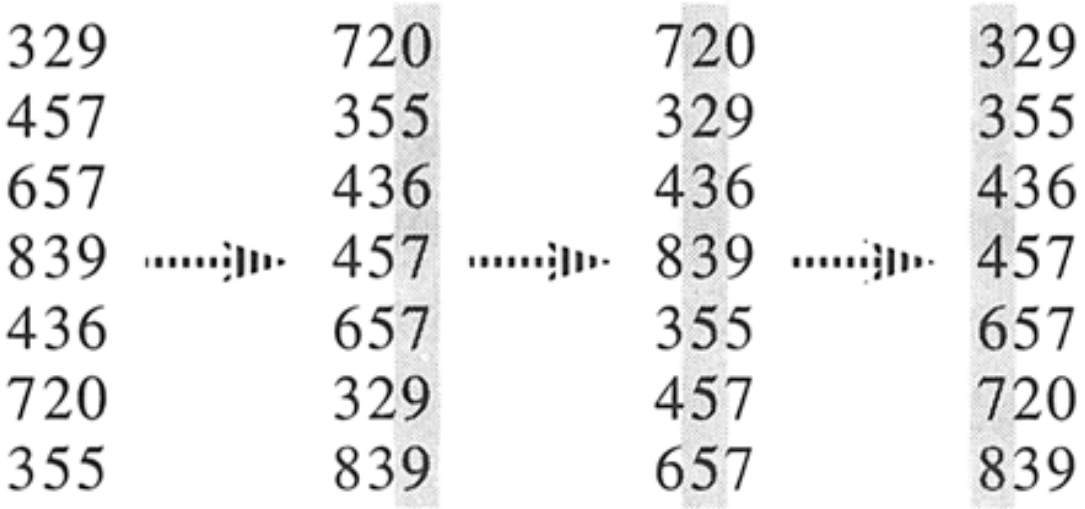
Radix sort

- 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 nejvýznamnější
- 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.

Složitost trochu podrobněji

- složitost algoritmu – jak je algoritmus výpočetně náročný
- časová, prostorová
- měříme počet operací nikoliv čas na konkrétním stroji
- vyjadřujeme jako funkci délky vstupu
- O notace – zanedbáváme konstanty
- např. $O(n)$, $O(n \log(n))$, $O(n^2)$

Ilustrace rozdílů v složitosti

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3 nsec	0.01 μ	0.02 μ	0.06 μ	0.51 μ	0.26 μ
16	4 nsec	0.02 μ	0.06 μ	0.26 μ	4.10 μ	65.5 μ
32	5 nsec	0.03 μ	0.16 μ	1.02 μ	32.7 μ	4.29 sec
64	6 nsec	0.06 μ	0.38 μ	4.10 μ	262 μ	5.85 cent
128	0.01 μ	0.13 μ	0.90 μ	16.38 μ	0.01 sec	10^{20} cent
256	0.01 μ	0.26 μ	2.05 μ	65.54 μ	0.02 sec	10^{58} cent
512	0.01 μ	0.51 μ	4.61 μ	262.14 μ	0.13 sec	10^{135} cent
2048	0.01 μ	2.05 μ	22.53 μ	0.01 sec	1.07 sec	10^{598} cent
4096	0.01 μ	4.10 μ	49.15 μ	0.02 sec	8.40 sec	10^{1214} cent
8192	0.01 μ	8.19 μ	106.50 μ	0.07 sec	1.15 min	10^{2447} cent
16384	0.01 μ	16.38 μ	229.38 μ	0.27 sec	1.22 hrs	10^{4913} cent
32768	0.02 μ	32.77 μ	491.52 μ	1.07 sec	9.77 hrs	10^{9845} cent
65536	0.02 μ	65.54 μ	1048.6 μ	0.07 min	3.3 days	10^{19709} cent
131072	0.02 μ	131.07 μ	2228.2 μ	0.29 min	26 days	10^{39438} cent
262144	0.02 μ	262.14 μ	4718.6 μ	1.15 min	7 mnths	10^{78894} cent
524288	0.02 μ	524.29 μ	9961.5 μ	4.58 min	4.6 years	10^{157808} cent
1048576	0.02 μ	1048.60 μ	20972 μ	18.3 min	37 years	10^{315634} cent

Table 1.1 Running times for different sizes of input. "nsec" stands for nanoseconds, "μ" is one microsecond and "cent" stands for centuries.

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

Složitost třídících algoritmů

n – délka vstupní posloupnosti

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

Shrnutí

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