Programovací strategie: Medián

Tomáš Maršálek

6. října 2011

1 Zadání

Najděte v dostupné literatuře nebo vymyslete co nejlepší algoritmus pro výpočet mediánu. Nezapomeňte na citaci zdrojů. Kritéria kvality v sestupném pořadí jsou: výpočetní složitost, jednoduchost a implementační nenáročnost, paměťová spotřeba.

2 Problém

Máme posloupnost N porovnatelných prvků, ve které hledáme K-tý nejmenší prvek. Speciálním případem je medián, pro který $K = N/2$. Triviálním řešením je seřazení posloupnosti a odpočítání K-tého prvku. Pro obecný případ tedy máme horní hranici $O(N \log N)$. Pokud pro danou datovou množinu existuje lepší řadící algoritmus (např. radix sort), jsme schopni najít medián v lineárním čase.

3 Obecné řešení

Algoritmus quickselect (Hoare, 1962) je velmi podobný quicksortu. S ním sdílí stejnou rozdělující proceduru a složitost nejhoršího případu $O(N^2)$, očekávaná složitost u běžného případu je ale $O(N)$ oproti $O(N \log N)$ u quicksortu.

4 Implementace

4.1 Rozdělení

Rozdělovací funkce určí prvek zvaný pivot a uspořádá prvky posloupnosti tak, že všechny prvky menší než pivot budou vlevo a všechny větší vpravo od pivotu. Pole pak vypadá následovně:

$\{ar[x] < pivot\}$	$pivot$	$\{ar[x] \geq pivot\}$

```python
def rozdelit(ar, levy, pravy, p):
        pivot = ar[p]
        i, j = levy, levy
        ar[pravy], ar[p] = ar[p], ar[pravy]
        while i <= pravy:
                if ar[i] < pivot:
                        ar[i], ar[j] = ar[j], ar[i]
                        j += 1
                i += 1
        ar[pravy], ar[j] = ar[j], ar[pravy]
        return j
```

4.2 Quickselect

Algoritmus nejprve vybere náhodný prvek jako pivot a podle něj rozdělí pole. O pivotu teď můžeme s jistotou tvrdit, že kdyby bylo pole seřázené, byl by na stejné pozici. Pokud se zadařilo a pozice pivotu se shoduje s pozicí hledaného prvku K, máme hotovo, pivot je hledaným prvkem. Pokud ne, pak rekurzivně opakujeme quickselect na tu část pole, ve které prvek definitivně bude. V případě pole o jednom prvku ($levy = pravy$) vrátíme tento prvek.

```python
def quickselect(ar, levy, pravy, K):
        if levy == pravy:
                return ar[levy]
        nahodnyIndex = random.randint(levy, pravy)
        pivotIndex = rozdelit(ar, levy, pravy, nahodnyIndex)
        if pivotIndex == K:
                return ar[pivotIndex]
        if pivotIndex < K:
                return quickselect(ar, pivotIndex + 1, pravy, K)
        else:
                return quickselect(ar, levy, pivotIndex - 1, K)
```

4.3 Optimalizace

Po odstranění koncové rekurze se kód podobá binárnímu vyhledávání.

```python
def quickselect(ar, levy, pravy, K):
        while levy != pravy:
                nahodnyIndex = random.randint(levy, pravy)
                pivotIndex = rozdelit(ar, levy, pravy, nahodnyIndex)
                if pivotIndex == K:
                        return ar[pivotIndex]
                if pivotIndex < K:
                        levy = pivotIndex + 1
                else:
                        pravy = pivotIndex - 1
        return ar[levy]
```

5 Časová náročnost

Složitost algoritmu je závislá na výběru pivotu. Nejjednodušším řešením je výběr fixního prvku, například prvku nejvíce vlevo. Pak ale při seřazeném poli složitost degeneruje na případ $O(N^2)$. Randomizovaný výběr alespoň ochrání před případem seřazeného pole, ale stále může s malou pravděpodobností nastat degenerující případ.

5.1 Ideální případ

Budeme-li předpokládat, že vybraný pivot je v každé iteraci mediánem posloupnosti, pak časová složitost bude

$$\begin{aligned}
T(N) &= T(N/2) + N \\
&= T(N/4) + N + N/2 \\
&= T(N/8) + N + N/2 + N/4 \\
&= T(N/2^i) + (N + N/2 + N/4 + \ldots + N/2^i) \\
&< O(1) + 2N \\
&= O(N)
\end{aligned}$$

Obdobně bude-li výběr pivotu v každé iteraci rozdělovat posloupnost na dvě části podle nějakého zlomku, který nezávisí na N (např. 1/2 | 1/2 nebo 75/100 | 25/100 nebo $pN \mid (1-p)N$ pro $0 < p < 1$, ale ne $(N-1)/N \mid 1/N$), bude složitost pro dostatečně velkou konstantu stále lineární

$$\begin{aligned}
T(N) &= T(pN) + N \\
&= T(p^2N) + N + pN \\
&= T(p^iN) + (N + pN + \ldots + p^iN) \\
&= O(1) + N\frac{1 - p^{1-\log_p N}}{1-p} \\
&< O(1) + N\frac{1}{1-p} \\
&= O(N)
\end{aligned}$$

5.2 Nejhorší případ

Nastane, pokud každé rozdělení rozdělí posloupnost na jeden prvek a $N - 1$ prvků

$$\begin{aligned}
T(N) &= T(N-1) + N \\
&= T(N-2) + (N) + (N-1) \\
&= T(N-3) + (N) + (N-2) + (N-3) \\
&= O(1) + \sum_{i=0}^{N}(N-i) \\
&= O(N^2)
\end{aligned}$$

Ten ale v praxi potkáme s velmi malou pravděpodobností.

6 Vylepšení

Výběr pivotu je klíčovým prvkem, který určuje složitost celého algoritmu. Existuje však metoda pro výběr pivotu takového, že zaručí složitost nejhoršího případu $O(N)$, známá jako Median of Medians nebo BFPRT (Blum, Floyd, Pratt, Rivest, Tarjan, 1973) [2].

7 Dodatek k implementaci

Program `median.py` je v jazyce python 2.7.2. Povolená vstupní data jsou pouze celá čísla. Program slouží pouze jako demonstrace, proto neplánuji rozšíření pro jiné datové typy.

7.1 Příklad použití pro bash

```
# vygenerovani zkusebnich dat
> test.data; for i in $(seq 1 10000); do echo $RANDOM >> test.data; done

# vypocet medianu vzorovych dat
cat test.data | ./median.py

# overeni vysledku (serazeni a vypsani n/2 teho radku)
cat test.data | sort -n | head -n $(($(wc -l < test.data)/2)) | tail -1
```

Reference

[1] *Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford*
Introduction to Algorithms (2nd ed.)
MIT Press and McGraw-Hill, 2001

[2] *Prof. Erik Demaine, Prof. Charles Leiserson*
Introduction to Algorithms (SMA 5503), Lecture 6: Order Statistics, Median (video) 2005
`http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-6-order-statistics-median/`

[3] *Jonathan Shewchuk*
CS 61B Lecture 32: Sorting III (video) 2006
`http://www.youtube.com/watch?v=Y6LOLpxg6Dc`