Giải Thuật Lập Trình

Nơi tổng hợp và chia sẻ những kiến thức liên quan tới giải thuật nói chung và lý thuyết khoa học máy tính nói riêng.

‹ Chọn phần tử lớn thứ k của mảng--Median Selection • Chia để trị -- divide and conquer ›

Sắp xếp-- quicksort and merge sort

May 28, 2015 in Uncategorized | 2 comments

Trong bài này chúng ta sẽ áp dụng phương pháp **chia để trị (http://www.giaithuatlaptrinh.com/?p=48) (divide and conquer)** trong bài toán sắp xếp. Hai phương pháp sắp xếp đặc trưng dựa trên chia để trị là **Quick Sort** và **Merege Sort**.

Problem: Cho một mảng gồm n phần tử $A[1, 2, \ldots, n]$, sắp xếp các phần tử của mảng theo thứ tự tăng dần.

Quick Sort

Quick Sort được phát minh bởi Tony Hoare (http://en.wikipedia.org/wiki/Quicksort) 1959. Quick Sort là một thuật toán sắp xếp khá nhanh trong thực tế, mặc dù trong trường hợp tồi nhất là $O(n^2)$. Phiên bản randomized của Quicksort (http://www.giaithuatlaptrinh.com/?p=950) có thời gian **kì vọng (expected)** là $O(n \log n)$, với hằng số đằng sau O khá nhỏ. Điều đó giải thich một phần vì sao Quick Sort cũng hay được dùng trong thực tế.

Tư tưởng của quick sort khá đơn giản và gồm hai bước chính:

1. Bước 1: chọn một phần tử của mảng (bất kì), $A[p]$, gọi là pivot, và phân mảng thành hai phần: phần 1 gồm những phần tử nhỏ hơn hoặc bằng $A[p]$ và phần 2 gồm những phần tử lớn hơn $A[p]$.
2. Bước 2: gọi đệ quy sắp xếp phần 1 và phần 2

Dưới đây là giả mã của thuật toán. Hàm PARTITION$(A[1, 2, \ldots, n], p)$ chia mảng A thành hai phần với pivot được chọn là $A[p]$, và trả lại vị trí mới của $A[p]$ trong mảng sau khi đã chia.

```
QuickSort(A[1, 2, ..., n]):
    if(n > 1)
        Choose a pivot element A[p]
        r ← Partition(A[1, 2, ..., n], p)
        QuickSort(A[1, 2, ..., r − 1])
        QuickSort(A[r + 1, 2, ..., n])
```

Dưới đây là code C của giả mã trên:

+ expand source (#)

Giả mã của thuật toán chia mảng với phần tử chọn (pivot) là $A[p]$, trả lại vị trí mới cuả phần tử có giá trị $A[p]$ trong mảng.

```
PARTITION(A[1, 2, ..., n], p):
    swap A[p] ↔ A[n]
    i ← 0
    j ← n
    while i < j
        repeat i ← i + 1 until (i ≥ i or A[i] ≥ A[n])
        repeat i ← j − 1 until (i ≥ i or A[j] ≥ A[n])
        if i < j
            swap A[i] ↔ A[j]
    swap A[i] ↔ A[n]
    return i
```

Dưới đây là code C của giả mã trên. Có đôi chút khác biệt khi ta thực thi bằng C vì mảng bắt đầu từ phần tử 0 và mình dùng while (do đã quen) thay vì do while để thực hiện giả mã repeat.

+ expand source (#)

Code đầy đủ bằng C sẽ được cho ở cuối bài.

Phân tích thời gian

Nhìn vào giả mã cho thuật toán thực hiện PARTITION$(A[1, 2, \ldots, n], p)$: dễ thấy thời gian thực hiện là $O(n)$. Như vậy, thời gian tính của Quick Sort là:

$$T(n) = T(r-1) + T(n-r) + O(n)$$

Ở đây r phụ thuộc vào việc chọn phần tử pivot $A[p]$. Trong trường hợp tồi nhất, trong mỗi lần đệ quy, phần tử pivot chi mảng thành một phần có 1 phần tử và phần còn lại có n-2 phần tử. Như vậy thời gian là $T(n) = T(n-2) + O(n) \Rightarrow T(n) = O(n^2)$ (xem tại đây (http://www.giaithuatlaptrinh.com/?p=22) để biết tại sao). Như vậy các bạn có thể ghi nhớ, trong trường hợp tồi nhất, Quick Sort có thời gian chạy là $O(n^2)$.

Trong trường hợp lí tưởng, phần tử pivot chọn luôn chia mảng thành hai phần xấp xỉ như nhau. Như vậy ta có $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$. Như đã nói đến trong bài median selection (http://www.giaithuatlaptrinh.com/?p=35) n, trong những trường hợp không biết chọn phần tử nào là pivot tốt, cách đầu tiên có thể nghĩ tới là chọn **ngẫu nhiên** một phần tử của mảng làm pivot, với xác suất như nhau. Với Quick Sort, chọn ngẫu nhiên sẽ cho ta thuật toán với thời gian kì vọng (expected) là $O(n \log n)$. Chi tiết sẽ thảo luận trong phần thuật toán ngẫu nhiên.

Cách khác là có thể áp dụng thuật toán median selection (http://www.giaithuatlaptrinh.com/?p=35) với thời gian tuyến tính để chọn pivot. Về mặt lí thuyết, thời gian tính là $O(n \log n)$ như hằng số trong biểu thức O sẽ khá cao. Nếu thực thi thực tế sử dụng cách này sẽ rất chậm.

Merge Sort

Merge Sort được phát minh bởi John von Neumann (http://en.wikipedia.org/wiki/Merge_sort) vào năm 1945 là một thuật toán sắp xếp luôn đảm bảo thời gian chạy tồi nhất là $O(n \log n)$ dựa trên kĩ thuật chia để trị. Ý tưởng cơ bản của Merge Sort gồm hai bước:

1. Bước 1: chia mảng thành hai phần với kích thước (xấp xỉ) bằng nhau, sau đó gọi đệ quy sắp xếp trên mỗi phần. Chú ý ở đây khác với Quick Sort, ta không dùng pivot.
2. Bước 2: trộn (merge) hai phần đã sắp xếp ở bước 1 thành một mảng.

Dưới đây là giả mã của thuật toán. Hàm MERGE($A[1,2,\ldots,n], m$) trộn hai mảng con $A[1,2,\ldots,m]$, $A[m+1,m+2,\ldots,n]$ đã sắp xếp thành mảng $A[1,2,\ldots,n]$ đã xắp xếp.

```
MERGESORT(A[1,2,...,n]):
    if n > 1
        m ← ⌊n/2⌋
        MERGESORT(A[1,2,...,m])
        MERGESORT(A[m+1,2,...,n])
        MERGE(A[1,2,...,n],m)
```

Code của giả mã bằng C:

```
+ expand source (#)
```

Giả mã của hàm MERGE($A[1,2,\ldots,n], m$)

```
MERGE(A[1,2,...,n]):
    i ← 1; j ← m + 1
    for k ← 1 to n
        if j > n
            B[k] ← A[i]; i ← i + 1
        else if i > m
            B[k] ← A[j]; j ← j + 1
        else if A[i] > A[j]
            B[k] ← A[i]; i ← i + 1
        else
            B[k] ← A[j]; j ← j + 1
    for k ← 1 to n
        A[k] ← B[k]
```

Code của giả mã bằng C:

```
+ expand source (#)
```

Phân tích thời gian
Nhìn vào giả mã ta có thể thấy hàm MERGE có thể thực hiện trong thời gian tuyến tính $O(n)$. Như vậy thời gian thực hiện của Merge Sort là

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n)$$

Để đơn giản, ta có thể bỏ dấu phần nguyên trên và dưới. Như vậy thời gian thực hiện của Merge Sort là
$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$.

Code: merge sort and quick sort (http://www.giaithuatlaptrinh.com/wp-content/uploads/2015/05/merge-quick-sort.tar.gz)

Tham khảo

Bài viết dự chủ yêu trên notes của Jeff Erickson
http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/01-recursion.pdf
(http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/01-recursion.pdf)

Tài liệu tham khảo liên quan:
[1] Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2001) [1990]. ***Introduction to Algorithms (2nd ed.)*** . MIT Press and McGraw-Hill. ISBN 0-262-03293-7

Chia để trị -- divide and conquer (http://www.giaithua...p=48)
May 29, 2015
In "binary search"

Chọn thế nào là tốt? -- How to avoid bad cases? (http://www.giaithua...p=950)
March 23, 2016
In "chocolate-bar-breaking"

Mảng hậu tố-- Suffix Array (http://www.giaithua...p=488)
September 6, 2015
In "Rabin Karp"

Tags: divide and conquer, merge sort, quick sort, sorting

2 comments

Comments feed for this article

Trackback link: http://www.giaithuatlaptrinh.com/wp-trackback.php?p=41

hoahuongduong on June 20, 2016 at 4:53 pm

Mình có một nhận xét thế này đúng ko ạ: giả sử đầu vào là dãy có n số bằng nhau hết, n rất lớn, lúc đấy thời gian sắp xếp của Q - sort luôn là O(n^2) cho dù có sử dụng thuật chọn trung vị đi nữa.

Reply

Hùng Lê on June 20, 2016 at 5:12 pm

Nhận xét của bạn hoàn toàn đúng. Có nhiều cách để giải quyết trường hợp này. Ví dụ khi bạn so sánh pivot với một phần tử bằng nó, bạn sử dụng một counter để đếm xem bao nhiêu phần tử bằng với pivot mà ta đã so sánh, rồi quyết định đưa phần tử này sang trái hay sang phải. Cách này không làm tăng thời gian tiệm cận của Quicksort. Thông thường, để phân tích đơn giản, ta sẽ giả sử các phần tử luôn khác nhau.

Reply

Reply

Your email address will not be published. Required fields are marked *

Your comment

Name *

Email *

Website

Post Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.