Part5-QuickSort - Algorithm

Completed on	@2020/03/18
Key videos	https://www.coursera.org/learn/algorithms-divide-conquer/lecture/xUd8B/partitioning-around-a-pivot https://www.coursera.org/learn/algorithms-divide-conquer/lecture/QCLVL/choosing-a-good-pivot
Note	A thorough study and correctness proof to QuickSort Algorithm

Note

▼ Overview

QUICKSORT - THE ALGORITHM (Part V). One of the greatest algorithms ever, and our first example of a randomized algorithm. These lectures go over the pseudocode --- the high-level approach, how to partition an array around a pivot element in linear time with minimal extra storage, and the ramifications of different pivot choices --- and explain how the algorithm works.

QUICKSORT - THE ANALYSIS (Part VI). These lectures prove that randomized QuickSort (i.e., with random pivot choices) runs in O(n log n) time on average. The analysis is as elegant as the algorithm itself, and is based on a "decomposition principle" that is often useful in the analysis of randomized algorithms. Note that there are some accompanying lecture notes for this part (available for download underneath each video). Also, it may be helpful to watch the first probability review video (below) before watching this sequence.

▼ Quicksort: Overview

- 快排算法纵览

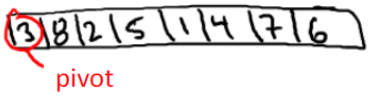
- Definitely a “greatest hit” algorithm
- Prevalent in practice
- Beautiful analysis
- $O(n \log n)$ time “on average”, works in place
  – i.e., minimal extra memory needed

- 主要思想：选定一个枢纽，是的大于/小于它的数分别放置于两端

Partitioning Around a Pivot

Key Idea : partition array around a pivot element.

-Pick element of array



-Rearrange array so that

-Left of pivot => less than pivot

-Right of pivot => greater than pivot

Note : puts pivot in its “rightful position”.

- 两个特点（two cool facts）
  1. Linear O(n) time, no extra memory
  2. Reduce problem size
- 工作流程

QuickSort (array A, length n)
-If n=1 return
-p = ChoosePivot(A,n)
-Partition A around p
-Recursively sort 1st part
-Recursively sort 2nd part

▼ Correctness of Quicksort [Review - Optional]

- 归纳法证明算法的正确性

Let P(n) = assertion parameterized by positive integers n.

For us : P(n) is "Quick Sort correctly sorts every input array of length n"

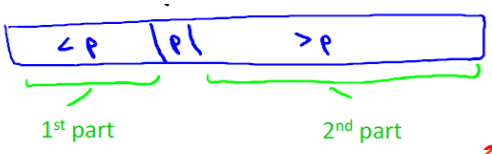
How to prove P(n) for all n >= 1 by induction :

1. [base case] directly prove that P(1) holds.
2. [inductive step] for every n>=2, prove that:
   If P(k) holds for all k<n, then P(n) holds as well.

INDUCTIVE HYPOTHESIS

- 由于整个算法可以分而治之成为子问题，所以P(k)都成立 ⇒ P(n)都成立

Recall : QuickSort first partitions A around some pivot p.



1st part 2nd part

Note : $k_1, k_2 < n$

Note : pivot winds up in the correct position.
Let k_1, k_2 = lengths of 1st, 2nd parts of partitioned array.

Using $P(k_1)$, $P(k_2)$

By inductive hypothesis : 1st, 2nd parts get sorted correctly by recursive calls. So after recursive calls, entire array correctly sorted.

▼ The Partition Subroutine

- 两种实现方法，一种是需要额外的空间O(n)，另外一种是In-place Implementation

In-Place Implementation

Assume : pivot = 1st element of array
[if not, swap pivot <--> 1st element as preprocessing step]

High – Level Idea :

-Single scan through array
- invariant : everything looked at so far is partitioned

- 伪代码：找到第一个元素为pivot，然后从第二个元素开始于pivot比较，所大于pivot不变，若小于，storeIndex += 1，并且于storeIndex对调位置，最后pivot再置换到中间去！

```
Partition (A,l,r)                    [ input corresponds to A[l...r]]
        - p:= A[l]
        - i:= l+1
        - for j=l+1 to r
                - if A[j] < p                [if A[j] > p, do nothing ]
                   -swap A[j] and A[i]
                   - i:= i+1
        - swap A[l] and A[i-1]
```

- 复杂度分析：仅有比较与对调运算这两个关键步骤！

Running time = O(n), where n = r – l + 1 is the length of the input (sub) array.

Reason : O(1) work per array entry.

Also : clearly works in place (repeated swaps)

▼ Choosing a Good Pivot

- pivot元素的重要性 ⇒ 极大影响整体运算时间！
- case analysis

1. 遇到已经排序好的数组 ⇒ O(n2)

Suppose we implement QuickSort so that ChoosePivot always selects the first element of the array. What is the running time of this algorithm on an input array that is already sorted?

- ○ Not enough information to answer question
- ○ $\theta(n)$
- ○ $\theta(n \log n)$
- ○ $\theta(n^2)$ ← 1st n/2 terms are all at least n/2

Recurse on these

empty　　Length n-1 (still sorted)

Reason :

Runtime : $\geq n + (n-1) + (n-2) + \ldots + 1$
$= \theta(n^2)$

2. pivot正好为中止 ⇒ O(nlogn)

Suppose we run QuickSort on some input, and, magically, every recursive call chooses the median element of its subarray as its pivot. What's the running time in this case?

- ○ Not enough information to answer question
- ○ $\theta(n)$
- ○ $\theta(n \log n)$
- ○ $\theta(n^2)$

Reason : Let T(n) = running time on arrays of size n.

Because pivot = median
choosePivot
partition

Then : $T(n) \leq 2T(n/2) + \theta(n)$
$=> T(n) = \theta(n \log n)$ $[like \ \ MergeSort]$

- 如何选择一个好的pivot? ⇒ 随机化选择 ⇒ 整体数值位于数组25到75百分位的都可以达到理想的高效运算

Random Pivots

Key Question : how to choose pivots ? BIG IDEA : RANDOM PIVOTS!

That is : in every recursive call, choose the pivot randomly.
(each element equally likely)

Hope : a random pivot is "pretty good" "often enough".

Intuition : 1.) if always get a 25-75 split, good enough for O(nlog(n)) running time. [this is a non-trivial exercise : prove via recursion tree]
2.) half of elements give a 25-75 split or better

Q : does this really work ?

Tim Roughgarden

- 平均运算时间！O(nlogn)

Average Running Time of QuickSort

QuickSort Theorem : for every input array of length n, the average running time of QuickSort (with random pivots) is O(nlog(n)).

Note : holds for every input. [no assumptions on the data]
- recall our guiding principles !
- "average" is over random choices made by the algorithm (i.e., the pivot choices)

References

- Quick Sort

快速排序
https://zh.wikipedia.org/wiki/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F

Quicksort
https://en.wikipedia.org/wiki/Quicksort