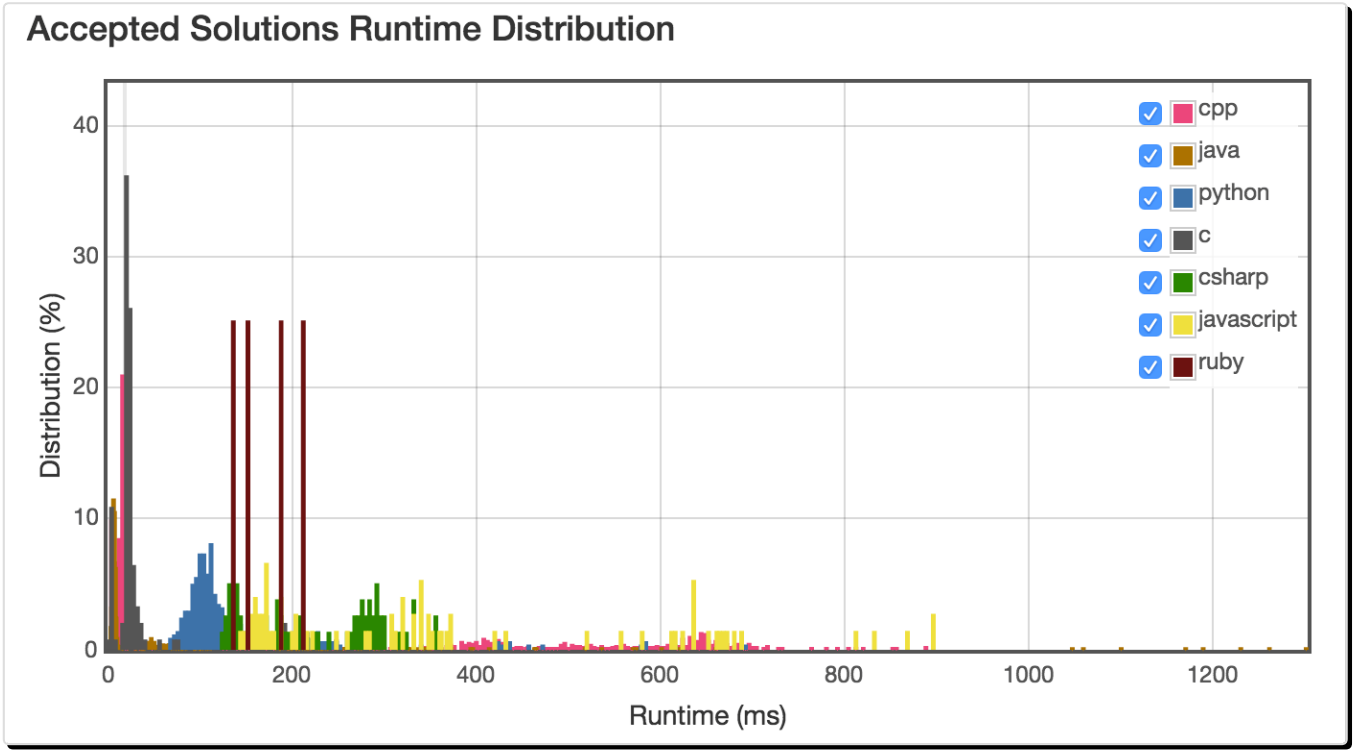
Just For Fun

享受快乐的时光

7月 24 2016 • 算法

LeetCode 刷题指南（一）：为什么要刷题

虽然刷题一直饱受诟病，不过不可否认刷题确实能锻炼我们的编程能力，相信每个认真刷题的人都会有体会。现在提供在线编程评测的平台有很多，比较有名的有 hihocoder，LintCode，以及这里我们关注的 LeetCode。



LeetCode 是一个非常棒的 OJ（Online Judge）平台，收集了许多公司的面试题目。相对其他 OJ 平台而言，有着下面的几个优点：

- 题目全部来自业内大公司的真实面试
- 不用处理输入输出，精力全放在解决具体问题上
- 题目有丰富的讨论，可以参考别人的思路
- 精确了解自己代码在所有提交代码中运行效率的排名
- 支持多种主流语言：C/C++，Python, Java
- 可以在线进行测试，方便调试

下面是我刷 LeetCode 的一些收获，希望能够引诱大家有空时刷刷题目。

问题：抽象思维

波利亚用三本书：《How To Solve It》、《数学的发现》、《数学与猜想》）来试图阐明人类解决问题的一般性的思维方法，总结起来主要有以下几种：

- 时刻不忘未知量。即时刻别忘记你到底想要求什么，问题是什么。（动态规划中问题状态的设定）
- 试错。对题目这里捅捅那里捣捣，用上所有的已知量，或使用所有你想到的操作手法，尝试着看看能不能得到有用的结论，能不能离答案近一步（回溯算法中走不通就回退）。
- 求解一个类似的题目。类似的题目也许有类似的结构，类似的性质，类似的解方案。通过考察或回忆一个类似的题目是如何解决的，也许就能够借用一些重要的点子（比较 Ugly Number 的三个题目：263. Ugly Number， 264. Ugly Number II， 313. Super Ugly Number）。
- 用特例启发思考。通过考虑一个合适的特例，可以方便我们快速寻找出一般问题的解。
- 反过来推导。对于许多题目而言，其要求的结论本身就隐藏了推论，不管这个推论是充分的还是必要的，都很可能对解题有帮助。

刷 LeetCode 的最大好处就是可以锻炼解决问题的思维能力，相信我，如何去思考本身也是一个需要不断学习和练习的技能。

此外，大量高质量的题目可以加深我们对计算机科学中经典数据结构的深刻理解，从而可以快速用合适的数据结构去解决现实中的问题。我们看到很多ACM大牛，拿到题目后立即就能想出解法，大概就是因为他们对于各种数据结构有着深刻的认识吧。LeetCode 上面的题目涵盖了几乎所有常用的数据结构：

- Stack：简单来说具有后进先出的特性，具体应用起来也是妙不可言，可以看看题目 32. Longest Valid Parentheses。
- Linked List：链表可以快速地插入、删除，但是查找比较费时（具体操作链表时结合图会简单很多，此外要注意空节点）。通常链表的相关问题可以用双指针巧妙的解决，160. Intersection of Two Linked Lists 可以帮我们重新审视链表的操作。

- Hash Table：利用 Hash 函数来将数据映射到固定的一块区域，方便 O(1) 时间内读取以及修改。37. Sudoku Solver 数独是一个经典的回溯问题，配合 HashTable 的话，运行时间将大幅减少。
- Tree：树在计算机学科的应用十分广泛，常用的有二叉搜索树，红黑书，B+树等。树的建立，遍历，删除相对来说比较复杂，通常会用到递归的思路，113. Path Sum II 是一个不错的开胃菜。
- Heap：特殊的完全二叉树，“等级森严”，可以用 O(nlogn) 的时间复杂度来进行排序，可以用 O(nlogk) 的时间复杂度找出 n 个数中的最大（小）k个，具体可以看看 347. Top K Frequent Elements。

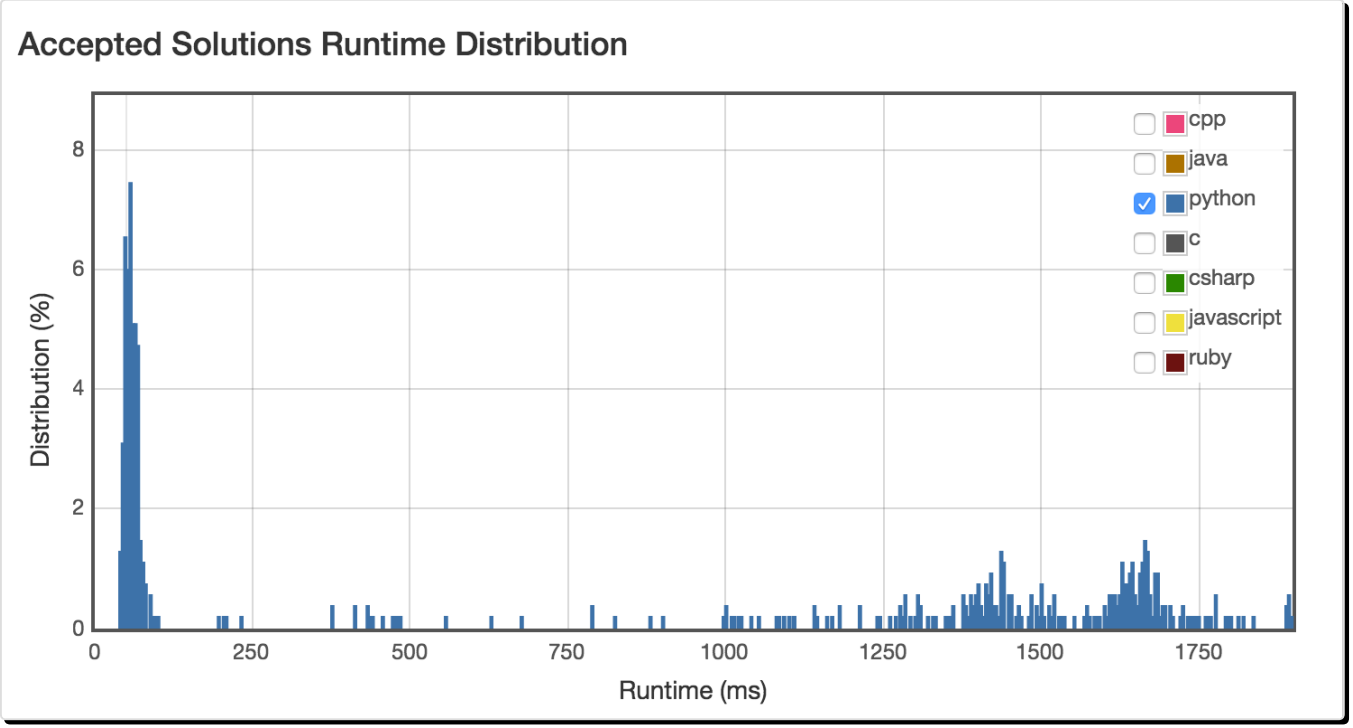
算法：时间空间

我们知道，除了数据结构，具体算法在一个程序中也是十分重要的，而算法效率的度量则是时间复杂度和空间复杂度。通常情况下，人们更关注时间复杂度，往往希望找到比 O(n^2) 快的算法，在数据量比较大的情况下，算法时间复杂度最好是O(logn)或者O(n)。计算机学科中经典的算法思想就那么多，LeetCode 上面的题目涵盖了其中大部分，下面大致来看下。

- 分而治之：有点类似“大事化小、小事化了”的思想，经典的归并排序和快速排序都用到这种思想，可以看看 Search a 2D Matrix II 来理解这种思想。
- 动态规划：有点类似数学中的归纳总结法，找出状态转移方程，然后逐步求解。 309. Best Time to Buy and Sell Stock with Cooldown 是理解动态规划的一个不错的例子。
- 贪心算法：有时候只顾局部利益，最终也会有最好的全局收益。122. Best Time to Buy and Sell Stock II 看看该如何“贪心”。
- 搜索算法（深度优先，广度优先，二分搜索）：在有限的解空间中找出满足条件的解，深度和广度通常比较费时间，二分搜索每次可以将问题规模缩小一半，所以比较高效。
- 回溯：不断地去试错，同时要注意回头是岸，走不通就换条路，最终也能找到解决问题方法或者知道问题无解，可以看看 131. Palindrome Partitioning。

当然，还有一部分问题可能需要一些数学知识去解决，或者是需要一些位运算的技巧去快速解决。总之，我们希

望找到时间复杂度低的解决方法。为了达到这个目的，我们可能需要在一个解题方法中融合多种思想，比如在 300. Longest Increasing Subsequence 中同时用到了动态规划和二分查找的方法，将复杂度控制在O(nlogn)。如果用其他方法，时间复杂度可能会高很多，这种题目的运行时间统计图也比较有意思，可以看到不同解决方案运行时间的巨大差异，如下：



当然有时候我们会牺牲空间换取时间，比如在动态规划中状态的保存，或者是记忆化搜索，避免在递归中计算重复子问题。213. House Robber II 的一个Discuss会教我们如何用记忆化搜索减少程序执行时间。

语言：各有千秋

对一个问题来说，解题逻辑不会因编程语言而不同，但是具体coding起来语言之间的差别还是很大的。用不同语言去解决同一个问题，可以让我们更好地去理解语言之间的差异，以及特定语言的优势。

速度 VS 代码量

C++ 以高效灵活著称，LeetCode 很好地印证了这一点。对于绝大多数题目来说，c++ 代码的运行速度要远远超过 python 以及其他语言。和 C++ 相比，Python 允许我们用更少的代码量实现同样的逻辑。通常情况下，Python程序的代码行数只相当于对应的C++代码的行数的三分之一左右。

以 347 Top K Frequent Elements 为例，给定一个数组，求数组里出现频率最高的 K 个数字，比如对于数组 [1,1,1,2,2,3]，K=2 时，返回 [1,2]。解决该问题的思路比较常规，首先用 hashmap 记录每个数字的出现频率，然后可以用 heap 来求出现频率最高的 k 个数字。

如果用 python 来实现的话，主要逻辑部分用两行代码就足够了，如下：

```
num_count = collections.Counter(nums)
return heapq.nlargest(k, num_count, key=lambda x: num_count[x])
```

当然了，要想写出短小优雅的 python 代码，需要对 python 思想以及模块有很好的了解。关于 python 的相关知识点讲解，可以参考这里。

而用 C++ 实现的话，代码会多很多，带来的好处就是速度的飞跃。具体代码在这里，建立大小为 k 的小顶堆，每次进堆时和堆顶进行比较，核心代码如下：

```
// Build the min-heap with size k.
for(auto it = num_count.begin(); it != num_count.end(); it++){
 if(frequent_heap.size() < k){
    frequent_heap.push(*it);
 }
 else if(it->second >= frequent_heap.top().second){
    frequent_heap.pop();
```

```
        frequent_heap.push(*it);
    }
}
```

语言的差异

我们都知道 c++ 和 python 是不同的语言，它们有着显著的区别，不过一不小心我们就会忘记它们之间的差别，从而写出bug来。不信？来看 69 Sqrt(x)，实现 int sqrt(int x) 。这题目是经典的二分查找（当然也可以用更高级的牛顿迭代法），用 python 来实现的话很容易写出 AC 的代码。

如果用 C++ 的话，相信很多人也能避开求中间值的整型溢出的坑： int mid = low + (high - low) / 2; ，于是写出下面的代码：

```
int low = 0, high = x;
while(low <= high){
    // int mid = (low+high) / 2,  may overflow.
    int mid = low + (high - low) / 2;
    if(x>=mid*mid && x<(mid+1)*(mid+1)) return mid;
    else if(x < mid*mid)  high = mid - 1;
    else low = mid + 1;
}
```

很可惜，这样的代码仍然存在整型溢出的问题，因为mid*mid 有可能大于 INT_MAX ，正确的代码在这里。当我们被 python 的自动整型转换宠坏后，就很容易忘记c++整型溢出的问题。

除了臭名昭著的整型溢出问题，c++ 和 python 在位运算上也有着一点不同。以 371 Sum of Two Integers 为例，不用 +, - 实现 int 型的加法 int getSum(int a, int b) 。其实就是模拟计算机内部加法的实现，很明显是一

个位运算的问题，c++实现起来比较简单，如下：

```
int getSum(int a, int b) {
    if(b==0){
        return a;
    }
    return getSum(a^b, (a&b)<<1);
}
```

然而用 python 的话，情况变的复杂了很多，归根到底还是因为 python 整型的实现机制，具体代码在这里。

讨论：百家之长

如果说 LeetCode 上面的题目是一块块金子的话，那么评论区就是一个点缀着钻石的矿山。多少次，当你绞尽脑汁终于 AC，兴致勃发地来到评论区准备吹水。结果迎接你的却是大师级的代码。于是，你高呼：尼玛，竟然可以这样！然后闭关去思考那些优秀的代码，顺便默默鄙视自己。

除了优秀的代码，有时候还会有直观的解题思路分享，方便看看别人是如何解决这个问题的。@MissMary 在“两个排序数组中找出中位数”这个题目中，给出了一个很棒的解释：Share my o(log(min(m,n)) solution with explanation，获得了400多个赞。

你也可以评论大牛的代码，或者提出改进方案，不过有时候可能并非如你预期一样改进后代码会运行地更好。在 51. N-Queens 的讨论 Accepted 4ms c++ solution use backtracking and bitmask, easy understand 中，@binz 在讨论区中纳闷自己将数组 vector（取值非零即一）改为 vector 后，运行时间变慢。@prime_tang 随后就给出建议说最好不要用 vector，并给出了两个 StackOverflow 答案。

当你逛讨论区久了，你可能会有那么一两个偶像，比如@StefanPochmann。他的一个粉丝 @agave 曾经问

StefanPochmann 一个问题：

> Hi Stefan, I noticed that you use a lot of Python tricks in your solutions, like “v += val,” and so on… Could you share where you found them, or how your learned about them, and maybe where we can find more of that? Thanks!

StefanPochmann 也不厌其烦地给出了自己的答案：

> @agave From many places, though I’d say I learned a lot on CheckiO and StackOverflow (when I was very active there for a month). You might also find some by googling python code golf.

原来大神也是在 StackOverflow 上修炼的，看来需要在 为什么离不开 StackOverflow 中添加一个理由了：因为 StefanPochmann 都混迹于此。

类似这样友好，充满技术味道的讨论，在 LeetCode 讨论区遍地都是，绝对值得我们去好好探访。

成长：大有益处

偶尔会听旁边人说 XX 大牛 LeetCode 刷了3遍，成功进微软，还拿了 special offer！听起来好像刷题就可以解决工作问题，不过要知道还有刷5遍 LeetCode 仍然没有找到工作的人呢。所以，不要想着刷了很多遍就可以找到好工作，毕竟比你刷的还疯狂的大有人在（开个玩笑）。

不过，想想前面列出的那些好处，应该值得大家抽出点时间来刷刷题了吧。

更多阅读

跟波利亚学解题
为什么我反对纯算法面试题
聊聊刷题
如何看待中国学生为了进 Google、微软等企业疯狂地刷题？
LeetCode 编程训练
国内有哪些好的刷题网站?

↑

欣赏此文？支持一下吧

#python #思考

Comments

NEWER

从零开始搭建论坛（一）：Web服务器与Web框架

OLDER

为什么离不开 Stackoverflow