数据结构与算法概述

本篇是数据结构与算法的先导篇，目的是帮大家认识数据结构和算法的世界，具体的数据结构和算法会在之后的讲义详细进行讲述。

数据结构与算法是什么？

作为一个程序员，我们会写各种各样的代码。但是不管是什么功能，只要我们对其拆解得足够细，你会发现其都是**数据结构 + 算法**，其重要程度可见一斑。那么数据结构与算法究竟是什么呢？

数据结构就是数据的存储形式，算法就是对数据的一系列操作。

我们先来看数据结构。本质上来说，数据结构就是一堆数据，这些数据内部是结构化的。这里有两个要点：

- 一堆数据。 也就是说不是单一的值。比如一个字符，一个数字等。我们所说的数据结构通常就是一堆数据，或者说一系列数据。
- 结构化。那么一堆数据怎么组织呢？一个个紧邻排就是数组，用一个指针指向下一项就是链表等等。

接下来，我们看一下什么是算法。

前面说**算法就是对数据的一系列操作**。 而这些操作从本质上来说就是**增，删，查**，我们无时无刻不在与这些东西打交道。而算法的学习就需要我们稳稳地抓住这三点。当你明确了这三点之后，你会发现数据结构就是手到渠成的事情了，也就是说数据结构是为了算法服务的。当你的算法分析好了，数据结构自然也会到位。

比如我们想从一个城市出发到达另外一个城市。显然，我们有很多方法。比如乘坐火车，飞机，自己开车等等。就算交通工具确定了，那么路线也有很多种。因此我们有无数种方法可选，那么哪一种性价比最高？

首先我们需要对问题进行定义。比如什么是性价比，再比如究竟有哪些交通工具和路线，每种的时间和金钱花费是多少等等。接下来，我们可以对问题进行抽象，使用计算机来解决。而使用计算机帮助我们解决这个问题编写的代码，我们就可以称之为算法。 显然这是一种狭义的算法定义。更为广义的算法指的是解决问题的方法，只不过对我们做题和理解帮助不大罢了。使用这种算法，我需要存储一些中间数据，**那么如何组织这些中间数据**以使得性能最优呢？这就是数据结构的话题了。

最后总结一下：一般而言，我们遇到一个算法问题，可以大致分为如下三个步骤。

- 第一步是建立算法模型。
- 第二步则是根据算法模型分析我们需要对数据进行哪些操作（增删改）。

- 第三步，我们需要分析哪些操作最频繁，对算法的影响最大。最后，我们需要根据第三步的分析结果选择合适的数据结构。

可以看出，我们的分析过程是一层一层，逐步递进的。每一步都需要前一步分析的结果。如果你碰到的问题都严格按照我的这个思维模型进行的话，久而久之，你会逐步培养起自己的算法思维。这个是最最重要的，大家一定要把算法思维的培养放到学习算法中最高的位置。

虽然实际工程中用到算法的地方有很多，但是 99% 的情况并不需要我们设计一个新的算法，而是使用一个已有的经典算法。而我们《91 天学算法》要解决的就是**学习并在实际中使用这些经典算法，而不是让你自己设计新的算法**。

希望大家在接下来的章节，可以用这个思维模型去练习。

这是 91 天学算法的先导篇。里面不会深入讲解一些知识点，而是从大的方向上描绘数据结构与算法的蓝图。除此之外，也有一些参与活动必须知道的东西。

数据结构算法面面观

狭义的算法指的是经典的具体算法，广义的算法指的是解决问题的方法。

比如 math.sqrt 就是一种广义的算法， 其具体的算法可以是牛顿迭代法，二分法，也可以是暴力法等。

在这里，我们往往关注的是狭义的算法， 并且是那些被反复验证的经典的算法思想。

研究这些经典算法很重要，它不仅可以帮我们解决实际的问题，锻炼思维，而且还可以帮助我们交流， 在这个层面上来说，其作用类似设计模式。比如我跟你说这道题用二分法就行了，你如果也恰好明白什么是二分，就可能知道我的意思。而如果你不知道二分， 我只能这样解释你才可能明白”用两个指针，分别指向数组的头部和尾部，然后计算中间位置，通过比较目标元素和中间位置的关系移动两个指针。。。“。

而数据结构是计算机存储、组织**数据**的方式，指相互之间存在一种或多种**特定关系**的数据元素的集合。要想深刻理解其数据结构，一定要结合算法。 因为任何数据结构都是为了实现某一个或者多个算法的，数据结构是为算法所服务的。 就好像你要做红烧鱼需要鱼，你做可乐鸡翅需要鸡翅。 当你用到特定算法的话，自然会用到特定的数据结构。

我们知道，内存的物理表现实际上就是一系列连续的内存单元，每一个内存单元的大小是固定的，这也是内存支持随机访问的本质原因。那么应该怎么存储和检索以及修改内存呢（增删查）？这就是数据结构研究的话题。

上面提到的是内存的物理结构，我们也可以基于这个物理结构拓展逻辑结构。比如，上面提到物理内存是固定大小连续存储的，那我可不可以将一段大的数据存储在不连续的空间呢？当然可以，这需要你自己处理，当你尝试去处理的时候，实际上就涉及到了逻辑结构。你所知道的，以及我们即将研究的全部都是逻辑结构。

不同的逻辑结构存储实际上有不同的特点，我们需要结合实际的场景分析。要想拥有具体问题具体分析的能力，我们首先要对基本的数据结构要特别清楚。包括他们的特点， 适用场景，不适合场景等。我的建议是先过一遍数据结构， 有个大致印象，然后研究具体算法。之后结合算法回头继续复习数据结构.

例子

我们公司前台可以代收快递。但是当我们取快递的时候，需要我们在一个事先写了我们报告信息的表格上签字。表格大概是这样的：

时间	名字	快递公司	单号	签字
2020-09-09	西法	SF	sf298001	西法
2020-09-10	张三	SF	sf133990	

由于是来一个快递，快递小哥就会在表格增加一个记录，因此时间那一列是升序排列的。

如果你某一天要来公司取你的快递，那么你肯定想要快速找到你的包裹所在的行。一个好的方式则是先根据日期检索，确定大概范围之后再根据快递公司找，由于一个快递小哥通常会一次登记很多快递，因此会出现连续的同一个快递公司的常见现象，这样你可以快速定位到你的快递公司那几行数据，最后再根据你的单号来最终确认即可。

我们来分析一下上面的问题。 实际上，上面的例子查询的次数要远远大于插入。而查询的过程除了时间（只有时间是有序的）可以二分，其他条件则不可以。如果我只有一个快递，而和我同天的快递很多，那么要快速找到自己的快递就不那么容易。

如果相同名字的包裹放在一起，这样我们不是就可以像查字典一样快速定位到自己的包裹了么？

确实如此，如果名字还能按照拼写排序，我们也可利用查字典的方式快速定位。

假设我们真的按照名字拼写顺序进行组织，**并且名字是挨着写的**，即张三后面直接跟着另外一个人，**中间没有空行**。大概是这样的：

```
张三
张四
xxx
xxx
xxx
```

这样的话假设又来了一个张三的快递，我们就需要擦掉张四那一行，写上张三的信息。然后将擦掉的张四的信息重写到下一行。由于这样做会将下面的 xxx 给覆盖，因此我们需要执行同样的操作。 如果张四下面人数很多这个效率可想而知。

怎么解决呢？

其实我们也可以给公司的所有人准备一个表格，每一个人的快递都写到对应的表格。这就对快递员（写入）有了新的要求。并且会更加浪费表格（空间），但是相应地查询速度会更快。

但是这样还有一个问题。如果公司有很多人都不寄快递到公司呢？为他们也准备表格就显得多余。 另外我们需要给每个人都准备同样大小的表格么？这都是我们需要考虑的。 实际上，我们可以取舍一下，比如给所有名字 L 开头的用三张表，给所有名字 B 开头的一张表。

> 我们假设公司 L 开头的人比较多， 是 B 开头的大约三倍。

计算机也是类似，不同的存储有不同的特点。有好处也有坏处，我们要做的就是根据实际情况选择合适的数据结构。

数据结构与算法就是研究在什么情况下**应该如何高效组织和操作数据**的一门学科。请大家务必记住这一句话。

逻辑结构

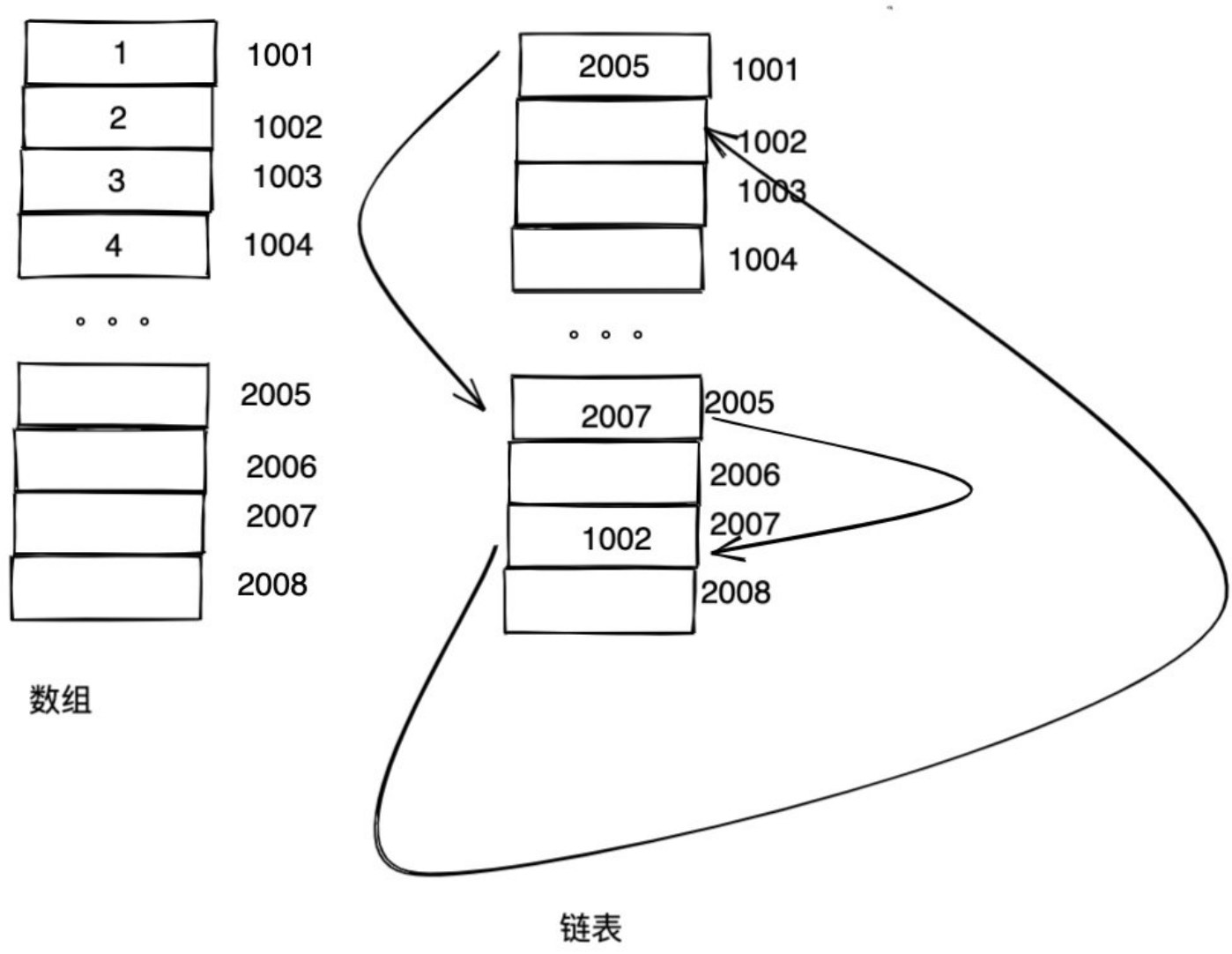
我们平常说的数据结构都指的是逻辑结构。比如数组，链表，二叉树等等，这个我们会在基础篇做详细介绍。这里我们就简单带大家了解一下就行了。

数组用来表示连续的内存空间，而链表通常用来表示不连续的内存空间。**这个连续性指的是在内存中存放的数据是否物理上连续**。

连续我们比较好理解，毕竟内存本来就是连续的。那么如何理解不连续的内存空间呢？

其实，我们只需要区分数据域和指针域即可。数组的话，我们可以根据内存的物理地址来索引，比如 A[0] 就是数组第一项，A[12] 就是数组第 13 项。

由于数组中的元素大小固定，因此我们可以通过基址 + 偏移量的方式实现随机访问。链表则不行，因为其下个元素并不一定是紧邻的，我们无法通过上面的基址和偏移量算出来。因此需要多一个指针域，来表示其下一个元素的内存单元位置。 不难看出， 链表相比数组，会增加额外的空间负担。有什么好处呢？好处则是增加或删除变得容易，这个我们留到基础篇详细谈谈。



如上图是一段物理内存。 如果我在连续内存中存值，并通过内存编号访问那就是数组。而如果我在内存中除了存值，还**多存**一个内存编号，用于寻找下一个数据，那就是链表。

内存还是那个内存， 通过不同的**逻辑抽象**就是两种数据结构了。当然这个解释比较粗糙，但是却可以帮助你认识本质。

总结

本节我们学习了数据结构和算法的关系，以及狭义和广义的算法。

对于算法而言本讲义的全部内容都是围绕狭义的算法展开，帮助大家理解经典的算法思想，并将这些思想串联起来进行综合运用。

对于数据结构而言本讲义的全部内容都是围绕逻辑结构而已，逻辑结构只不过是我们使用物理结构的一种抽象表示而已。基于数组和链表这两种基本的逻辑结构，拓展了无数的丰富的数据结构，这些数据结构都是为了解决特定问题产生的，因此理解数据结构一定要结合算法。接下来基础篇的章节，我们会带你剖析常见的数据结构，准备好了么？

最后给大家两个学习建议：

1. 无关联，不学习

学习的过程中要善于和已有的知识建立联系，简单来说就是**无关联，不学习**。我们的讲义也会时不时和其他知识进行串联，帮助大家加深理解。为什么有一些人每次看讲义都会得到不一样的知识，不断有顿悟提升的感觉？这就是他们不断地和自己已有的知识建立联系的结果，这些已有的知识也可能是刚刚才建立的。因此我建议大家也采用同样的方法，不断和自己已有的知识进行关联，这样学习效率会很高。

在舒适区边缘，一点一点向外扩展！ 祝大家收获自己满意的 offer，达到自己满意的算法能力。

2. 知行合一

道理，原理我都懂，但就是写不出来咋回事？ 实际上这是人性使然，有句话说的挺有道理“脑袋： 我会了。手：不！你不会！”。 如果你不经过足够的练习是无法获得足够的刺激使得神经元产生**强连接**。练习（刷题）就是给予足够刺激使神经细胞产生练习的过程。因此一定量的练习是必须的。

> 学习的本质从生物学上讲就是大脑中的神经细胞建立联系

而枯燥的练习容易让人产生抵触心理。这里西法分享一个自己的小技巧 - 延迟激励。我会给予自己一些小激励，比如这个月我刷 100 道题，我就在月末给自己 1000 元的激励，我可以任意支配。 另外一些国外的留学生还流行一句话，大概意思是现在你刷一道题，未来就多赚多少美金。这本质也是一种延迟激励。这时大家就像一只 donkey， 前面有一个胡萝卜吊着你一样 😄 。

上一页 下一页