华中科技大学

**软件工程·《算法设计与分析》**

**课**

**程**

**结**

**业**

**汇**

**报**

**材**

**料**

**姓名：粟锦**

**班级： 软工1803班**

**学号：U201817053**

**华中科技大学 软件学院**

# 目录

[目录 - 2 -](#_Toc415731055)

[引言 - 3 -](#_Toc415731056)

算法概念与检验标准 [- 3 -](#_Toc415731057)

[排序算法总结 - 4 -](#_Toc415731058)

[分治思想 - 5 -](#_Toc415731059)

[贪婪算法 - 6 -](#_Toc415731060)

[动态规划 - 7 -](#_Toc415731061)

[图论算法总结 - 8 -](#_Toc415731062)

[总结 - 10 -](#_Toc415731063)

引言

在软件行业中，检验一个程序员优秀与否的一个重要标准便是看他对于解决问题的算法的了解程度。算法对于程序员编程有着至关重要的作用，有了解决某一问题的算法思想，程序员才可以最大化地利用已有的资源编程出令人满意的目标应用。与此同时，对算法的学习可以极大地培养程序员的思维逻辑，锻炼思考能力。

在这篇结业报告材料中，我将根据自身的理解，结合老师在上课中所讲授的知识，总结《算法设计与分析》这门专业课程的主要内容。

算法概念与检验标准

在介绍具体的算法思想之前，有必要先简单描述一下算法的具体含义，以及评价一个算法好坏的判断依据。

算法，是为了求解特定问题而给出的一组指令序列。这里的指令序列，并不是特定的语言中特定的代码（例如C语言、JAVA语言）。算法是一个抽象概念，是解决问题的一种思想，而基于算法思想来实现的代码才是算法的具象化。我们可以将指令序列理解为达成一个目标要进行的所有过程。例如：我们要在纸上画一个小人，那么算法便是指导我们如何画一个小人的过程分解。我们先画小人的脑袋、再画躯干、最后画上四肢。可以看出来，这种指导过程并没有教我们绘画的具体细节，也没有规定我们作画的工具。我们完全可以使用铅笔画这个小人，也可以使用水彩笔、钢笔等来进行作画，而这就类似于我们选择实现算法的语言（C语言、JAVA、PYTHON等）。

了解了算法的含义后，我们可能会困惑：算法与算法之间有好坏之分吗？如果要解决某一个问题，我们有两种算法来指导解决这个问题，我们怎么确定选哪一个算法更好呢？这里便需要引入时间复杂度和空间复杂度的概念：

时间复杂度：[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95" \t "https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E6%80%A7/_blank)的时间复杂度是一个[函数](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E6%80%A7/_blank)，它定性描述该算法的运行时间。

空间复杂度：空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度。

引入了这两个概念，我们就可以知道，越是时间复杂度低与空间复杂度低的算法，它对于目标的实现效果可能就越好。当然，有时候鱼与熊掌不可兼得，时间复杂度低的算法，空间复杂度可能会很高。而空间复杂度低的算法往往时间复杂度会更高。这便需要程序员根据实际情况作出取舍：是为了运行速度更快而占用更多的空间开销，还是为了节省内存而选择花费更多的运行时间。

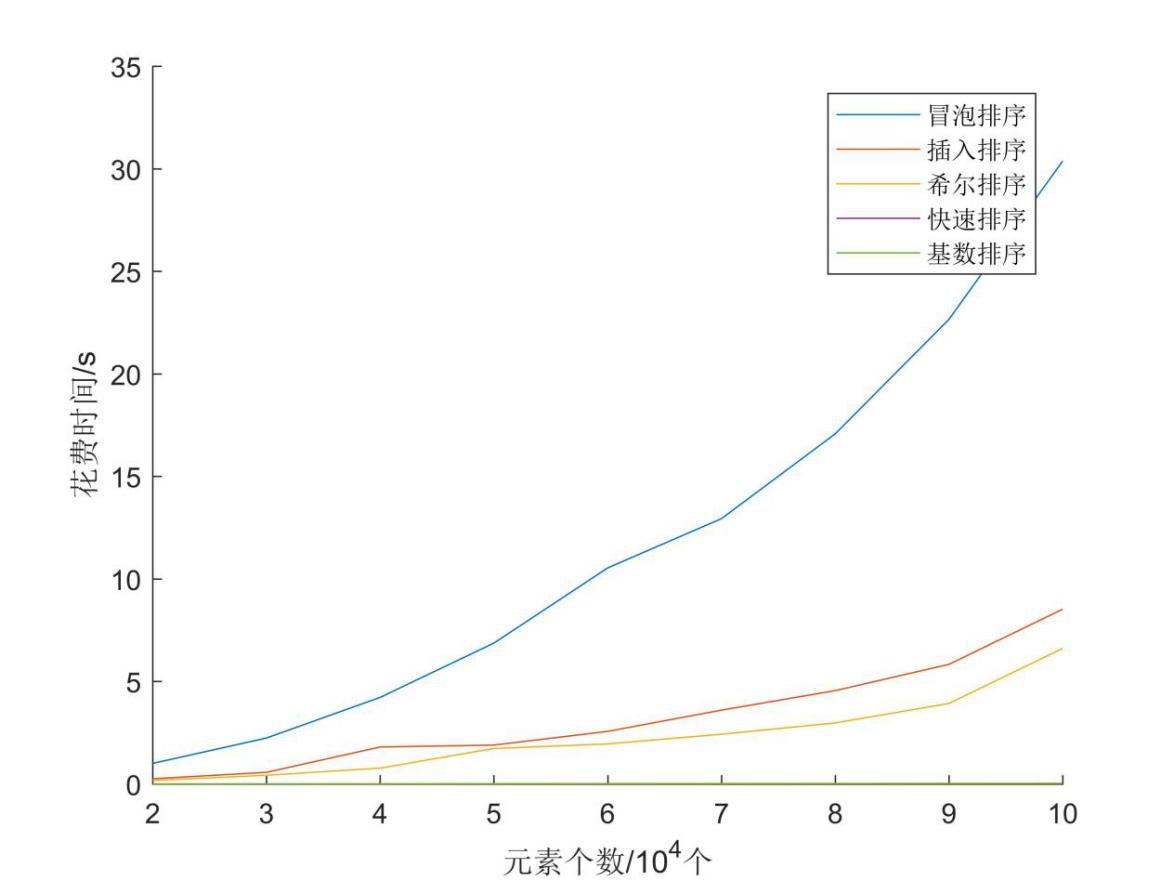
排序算法总结

在我看来，排序问题是非常经典的体现出算法优劣的一种问题模型。从最开始的完全基于比较的冒泡排序、选择排序，到通过分治思想提出的快速排序，再到不再依赖比较进行的基数排序、桶排序等等。在人们思维的不断拓展下，不断有着更加优秀的算法被提出，作为供后人引用的经典思想。在这里，考虑到各种排序算法的思想我们都已了然于心，我并不打算再花篇幅讲解每个算法的原理，而是根据算法的发展进程来分析时间复杂度的变化。

考虑到排序算法中排序数组的随机性，在考虑排序算法的时间复杂度时，我们往往不会选择算法的最优时间复杂度与最坏时间复杂度作为评判依据，而是以平均时间复杂度作为判断标准。

算法的精进是以平均时间复杂度的减小来判断的。在冒泡排序和选择排序中，平均时间复杂度都是O(n^2)。希尔在选择排序的基础上通过对选择间隔的更改降低了平均时间复杂度至O(n(^1.3-2))。随后发明的快速排序、堆排序等平均时间复杂度降低至O(nlogn)。随着定理的证明：基于比较的排序算法平均时间复杂度不可能低于O(nlogn)。快速排序这样的平均时间复杂度为O(nlogn)的算法被冠以“最优排序算法”的称号。

但是，人总是精益求精，不会满足于已有的成绩，而是不断钻研，不断精进。当人们发现基于比较的排序算法平均时间复杂度无论如何都有无法突破的下限时，他们开始转变思想：如何设计出一个不需要进行比较就能够完成排序的更好的算法呢？桶排序和基数排序应运而生。基数排序通过对数字的每一位进行判断来分类，桶排序通过简单直接的判断将数字放入相应桶中，都巧妙地避开了数字之间的比较。而最终的结果也是令人欢欣的：基数排序的平均时间复杂度是O(nlogm)，而桶排序的平均时间复杂度是O(m+n)。虽然这并不意味着在任何过程中这两种算法都是最好的（要依实际问题情况而定）。但是这也以为这排序算法有了新的突破，是思想上的一次重大进步。



排序算法花费时间示意图

排序算法的进化不会停止，在以后的发展中，也许我们还会遇见更加优秀的算法思想，保持生生不息的思考活力，是科技进步的重要动力。

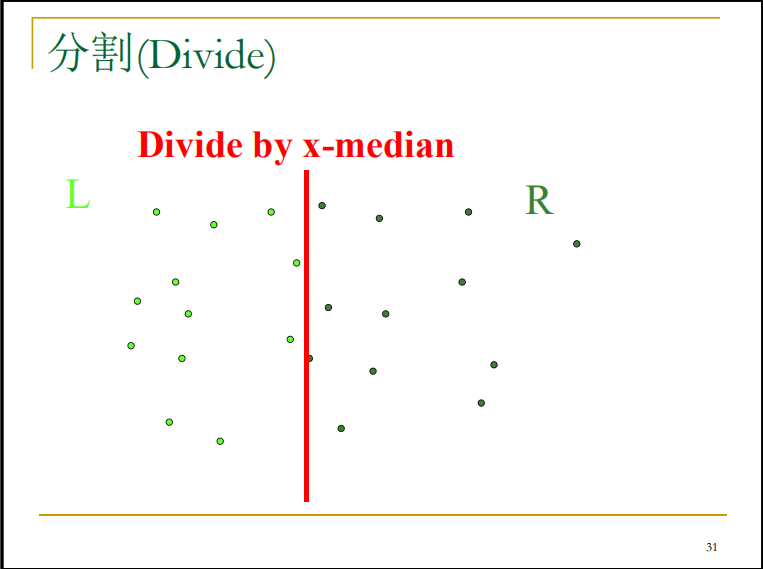
分治思想

分治思想是解决许多问题的有效思想。其基本原理是将原本的问题分解成若干个独立的相同性质的子问题，然后将这些子问题的解进行整合，从而求得原问题的解。以下以最近点对问题进行阐述。

最近点对问题：给定一个二维空间上的点集，每个点包含x坐标和y坐标，求在这个点集中两点之间的最短距离。

问题分解过程：

我们可以将二维空间分成两个部分，即意味着将点集分成两个部分。然后分别独立求出这两个点集中各自的两点之间的最短距离。可以知道，这是将原问题分解成了相同性质的两个子问题的解。



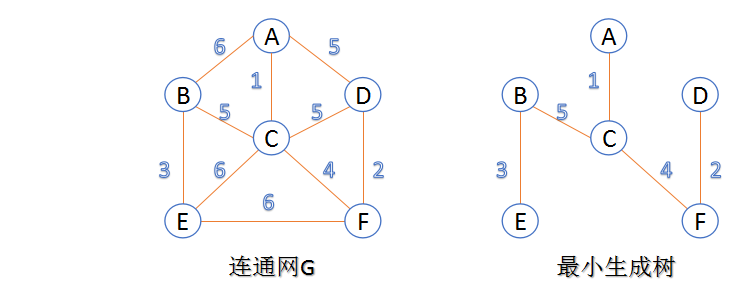
第二步，进行整合。当得到了两个独立的子问题的解之后，需要进行整合过程，从根据子问题的解来获得原问题的解。在本题中，我们可以知道原问题的解可能有两种情况，一种是原问题的解就是子问题的解之一，另一种就是最近距离的两个点分别在两个点集中。此时，我们要做的就是求出两个点集中的点的最近距离，再与子问题的解进行对比，得到的最小值就是原问题的解。

算法并不关心编程人员的具体实现，而是只给出解决某一类型问题的思路。对于这一题来说，如何求出最近距离是算法并不关心的，因此具体实现也没有给出（可以有很多方法实现）。我们可以知道的是，利用分治思想，我们将原问题不断地递归分解直到无法进行分解（递归有终止条件）再不断整合，最终就得到了原问题的解。

在我们以后的编程生涯中，遇到的难题将层出不穷且千变万化，单一的编程代码无法解决所有问题，而分治思想如同我们手中制作钥匙的模板，只要我们稍加改进转化，就能制作成解开各种变化的难题的钥匙。

贪婪算法

贪婪算法是我们不可或缺的解决问题的经典算法之一。在有些问题中，我们需要找到关于某个答案的最优解。有时寻找最优解的过程是困难且繁琐的，这既可能是编程难度过大，也可能是占用资源过多。而这时，我们也许不需要执着于求出最优的解，可以根据贪婪算法求出次优解来代替（在一些情况下，贪婪算法求得的解也是最优解）。贪婪算法并不着眼于全局最优，而是查找过程细分成多步，每一步都查找局部最优解，最后将局部最优解整合在一起。在这里，我们以最小生成树的kruskal算法为例介绍贪婪算法的具体实现。



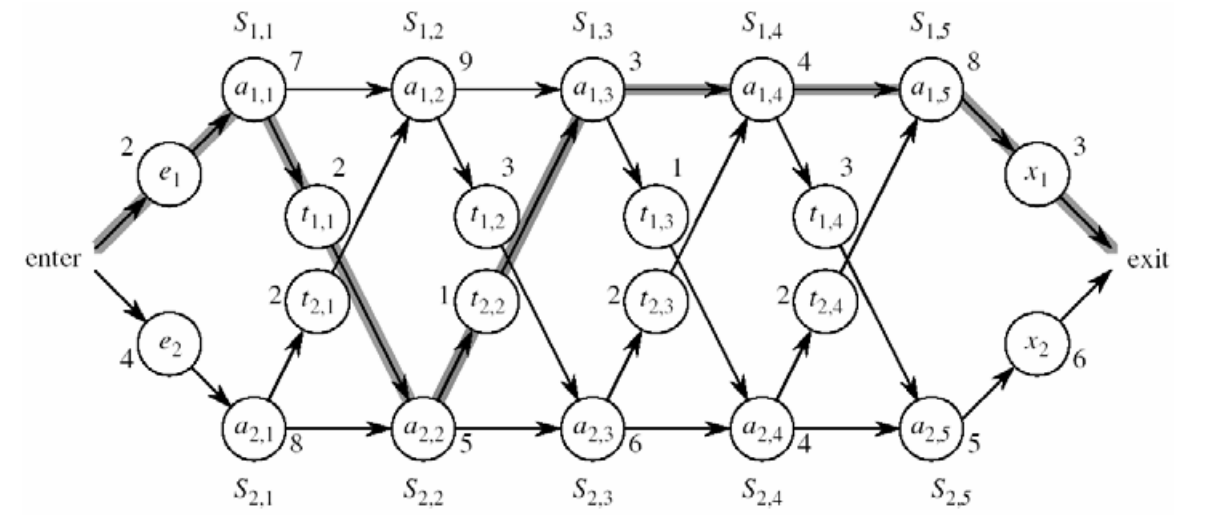
可以看到，当我们要寻找一张连通图的最小生成树时，我们希望最小生成树的每一条边权重都尽可能的小。那么使用贪婪算法便是很好的一种解法（在本示例中可以证明贪婪算法求得的解是最优解）。

因为连通图有6个节点，可以知道最小生成树只有5条边，我们根据贪婪算法来逐次为6个节点连边。在每一次加边时，我们将图中最小权重的边加上去，这便是贪婪的思想。因为最小生成树所有边的权重之和要最小，所以我们直接将最小权重的边加上去即可。在加上当前最小权重的边之后我们再检验目前图中是否产生了环，如果产生了环说明当前边不可加，于是抛弃掉这条边并选择接下来权重最小的边添加。在进行完5次操作后形成的图便是最小生成树了。

贪婪算法在很多地方都有着应用。其优势在于简化计算的同时还能够保持较高的准确性，掌握之后对于许多问题的解决往往能够达到事半功倍的效果。

动态规划

动态规划思想与分治思想类似，也需要将原问题进行分解然后导出一个递推公式。但动态规划与分治不同的是，分治思想往往通过递归的方式求解子问题的解之后再进行整合，而动态规划是一般通过自底向上的方式求解，从最小的子问题开始求解，较大子问题的解可以通过查询较小子问题的解并加以运算求出。接下来以装配线排程问题来阐释。



从图中可以看出，一个节点可以通过1号线或者2号线到达下一个节点，反之，一个节点的上一个节点可以来自1号线或者2号线。由此我们可以作出这样两种假设：到达终点的上一个节点来自1号线、到达终点的上一个节点来自2号线。那么最短距离就变成了：

min{enter~a(1,5)+a(1,5)~exit ; enter~a(2,5)+a(2,5)~exit}。

这样便将原问题分成了更小的一个子问题，并且通过查询子问题的解再稍加计算可以得出更大问题的解。在导出一个递推公式之后，我们只需要自底向上的进行计算，将计算结果存储在一个表格之中供之后查询即可。

合理的使用动态规划可以避免循环调用递归，从而节省系统开销。

图论算法总结

图论是数学的一个分支。它以图为研究对象。图论中的图是由若干给定的点及连接两点的线所构成的图形，这种图形通常用来描述某些事物之间的某种特定关系，用点代表事物，用连接两点的线表示相应两个事物间具有这种关系。图论对于科学研究的重要性不言而喻，诞生于图论中的许多算法也需要我们熟悉并掌握。

单源最短路径：

单源最短路径讨论的是在一张有向图中，以某一个节点为起点，求得该节点到其他任意节点的最短距离。对于这个问题，Bellman-Ford 算法可以很好地解决。Bellman-Ford 算法是建立在缩短法的基础上的。所谓缩短法，即对边(u, v)的缩短，是检查能否通过顶点u，改善已有的到达v的最短路径)。如果满足 d[v] > d[u] + w(u, v)，则可以改进抵达v的最短路径。而Bellman-Ford算法则是对每条边进行节点-1次缩短，因为最短路径不可能含有回路，所以假设有n个节点，那么总起点到任意一个节点的最短距离最多只经过n-1条边，因此只需要进行n-1次缩短即可。

除了Bellman-Ford算法，另外一个非常有效的算法便是Dijkstra’s算法。该算法是从起点开始，每次更新起点到各个节点的距离，并在下一次从最短距离的那个节点开始，重复更新过程直到将所有节点都遍历完。

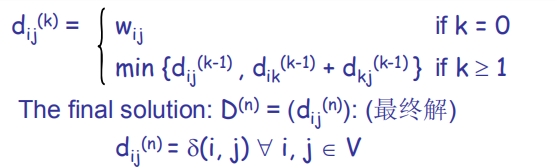
全成对最短路径：

全成对最短路径讨论的是在一张有向图中，任意一个节点到其他任意节点的最短距离，因此表示方法通常是一个距离矩阵D（n,n）,其中第i行第j列的元素代表的就是第i个节点到第j个节点的最短距离。解决这类问题比较基础的方法是延伸法。设最开始的距离矩阵D（n,n）记录的是每个节点到任意其他节点的初始距离（无法达到则为∞）。延伸法则是对距离矩阵进行变形，有

QQ截图20200506155642

在对每个元素进行n-2次延伸后得到的距离矩阵便是最短距离矩阵。因为最短路径最多有n-1条边，因此只需要进行n-2次延伸。

基于延伸法的全程对最短路径算法的时间开销是O(n^4)，较为耗时，而相比之下Floyd-warshall算法更加快速。

Floyd-warshall算法考虑从i节点到j节点途中经过的节点编号，设dij (k)是从i到j 的路径的权值, 中间节点来自集合{1, 2, …, k}。根据递推公式可得

由此只需同样对距离矩阵进行相应的更新，便可以得到最短距离矩阵，而时间花销为O(n^3)。

总结

迄今为止，我们已经学过了许多的算法，在解决大部分问题上已经有了些许经验。但是这还远远不够，算法只是的浩瀚程度远远超出我们的想象。正所谓“学海无涯，当精益求精”，在只看见了算法领域冰山一角的我们的面前，是数不胜数的深邃思想在等待我们发掘。《算法设计与分析》课程时间虽短，但它带给我的影响却深远厚重。在今后的日子里，我也会带着在课上学到的知识，不断功课难题，大步流星地前进！