算法复习@onevfall

各章知识

第1章

1. 算法定义：
    - 算法是一系列解决问题的明确指令，也就是说，对于符合一定规范的输入，能够在有限时间内获得要求的输出。
    - 在有限时间内，对问题求解的一个清晰的指令序列。算法的每个输入确定了该算法求解问题的一个实例。
2. 伪代码
    - 忽略变量的声明
    - 使用缩进表示***for, if, while***等域
    - 赋值用<—,注释用//
3. **算法特点**：输入，输出，确定性，有穷性，可行性。

第2章

1. 分析框架概要
    - 算法效率包括时间效率和空间效率，算法的时间效率和空间效率都用输入规模的函数进行度量。
    - 用**算法基本操作的执行次数来度量算法的时间效率。**通过计算**算法消耗的额外存储单元的数量来度量空间效率。**
    - 在输入规模相同的情况下，有些算法的效率会有显著差异。对于这样的算法，我们需要区分最差效率、平均效率、最优效率。
    - 本框架主要关心一点，当算法的输入规模趋向于无限大时，它的运行时间（消耗的额外空间）函数的增长次数。
2. **渐近符号的含义**
3. 利用极限比较增长次数
4. 线性对数（n log n）的渐近效率类型：许多分治算法，包括合并排序和快速排序的平均效率，都属于这个类型。
5. 非递归算法效率的通用方案
    - 决定用**哪个（哪些）参数**表示**输入规模**
    - 找出算法的**基本操作**（总是在算法的最内层循环中）
    - 检查基本操作的执行次数是否**只依赖于输入规模。**即看他是否依赖于其他特性，则**最差效率、平均效率以及最优效率需要分别研究。**
    - 建立一个算法基本操作执行次数的**求和表达式。**
    - 利用**求和运算**的标准公式和法则**来解一个操作次数的闭合公式**，或者至少确定它的增长次数。（order of growth）
6. 递归算法时间效率的通用方案
    - 决定用**哪个（哪些）参数**表示**输入规模**
    - 找出算法的**基本操作**
    - **检查一下，对于相同规模的不同输入，基本操作的执行次数是否可能不同。**如果有这种可能，则必须对最差效率、平均效率、最优效率做单独研究。
    - 对于算法基本操作的执行次数，**建立一个递推关系以及相应的初始条件。**
    - **解这个递推式**，或者至少确定它的解的增长次数。

第3章——蛮力法

1. 深度优先查找（DFS）和广度优先查找（BFS）的时间效率和空间效率的探索。

第4章——减治法

1. **减治法利用了一个问题给定实例的解和同样问题较小实例的解之间的关系。**一旦建立了这样的一种关系，我们既可以**自顶至下**，也可以**自底至上**地运用这种关系。
2. 分类
    - 减一法：插入排序、拓扑排序
    - 减常因子算法：折半查找、天平选假币问题、俄式乘法、约瑟夫斯问题
    - 减可变因子算法：欧几里得算法
3. 插入排序，无论在平均情况还是最差情况下，它都是一个Θ(n^2)的算法，但在平均情况下的效率大约要比最差情况快一倍。此算法比较大的一个优势在于，对于几乎有序的数组，它的性能是很好的。
4. 拓扑排序要求按照有向图中指定的次序列出它的顶点，使得对于图中每一条边来说，边的起始顶点总是排在边的结束顶点之前。当且仅当有向图是一个**无环有向图**（不包含回路的有向图）时，该问题有解。

第5章——分治法

1. 分治法是一种一般性的算法设计技术，它**将问题的实例划分成若干个较小的实例**（最好拥有同样的规模），对这些较小的实例**递归**求解，然后**合并**这些解，以得到原始问题的解。
2. 时间效率满足T(N)=a*T(n/b) + f(n)
3. 主定理的记忆
4. 案例：
    - **合并排序**【按位置划分】
        - 时间效率**总是**为nlogn
        - 可用主定理或者**平滑法则**计算在最坏情况下键值比较次数。
        - 优点：稳定性
        - 缺点：需要线性的额外空间。
    - **快速排序**【按值大小划分】
        - 时间效率**平均情况下**为nlogn
        - 最好情况：所有分裂点都位于子数组的中点
        - 最差情况：所有分裂点都趋于极端，两个子数组一个为空，另一个仅仅比被划分的数组少一个元素，此时效率与普通排序一致。
        - 对比：合并排序中，将问题划分成两个子问题是很快的，算法的主要工作在于合并子问题的解；而在快速排序中，算法的主要工作在于划分阶段，而不需要再去合并子问题的解了。
        - 如何改进：
            - 随机选取划分元素，增强划分的对称性。
            - 当子数组足够小时，改用插入排序方法。
            - 三路划分方法，将数组分成三段，每段的元素分别小于、等于、大于中轴元素。
    - **折半查找**——针对有序数组进行查找的算法,效率为logn【这个属于减常因子算法，也属于分治法；在这一点上有联系】
    - **大整数乘法**
    - **Strassen矩阵乘法**

第6章——变治法

1. 变治法是一种基于**变换**思想，把问题变换成一种**更容易解决**的类型。
    - **实例化简—**—变换为**同样问题**地一个**更简单或者更方便**地实例（例子：预排序、高斯消去法、平衡查找树）
        - 预排序
            - 检验数组中元素的唯一性：用于排序的时间更长，占据主导，所以优化排序时间会提升效率
            - 模式计算
            - 查找问题：若在同一个列表中进行多次查找，在排序上花费的时间应该是值得的。
        - 高斯消去法
            - 效率为三次方
            - 应用：LU分解、计算矩阵的逆、计算行列式
    - **改变表现**——变换为**同样实例的不同表现**
        - **堆和堆排序【替代了优先级队列】**
            - 自底向上构造堆的效率为O(n)
            - 插入效率为O(log n),删除效率为O(log n)
            - 堆排序包括 构造堆和删除最大键的两阶段，最差情况与平均情况的时间效率都属于**Θ(n log n)。**
            - 与合并排序相比，堆排序是**在位**的，不需要任何额外的存储空间。
        - **霍纳法则**：计算多项式的算法
    - **问题化简**——变换为**另一个问题的实例**，这种问题的**算法是已知的**
        - 线性规划
        - 简化为图

第7章——回溯法和分支界限法

1. 这两种方法求解的是这种问题：随着实例规模的增长，问题的选择次数至少呈指数增长。两种算法每次都只构造解的一个分量，一旦确定当前已经做出的选择无法导出一个解，就会立即终止当前步骤。这种方法使我们能够在可接受的时间内，对NP困难问题的许多较大实例求解。
2. 无论是回溯法还是分支界限法，都把**状态空间树**作为它们的主要机制。状态空间树是一颗有根树，它的节点代表了所讨论问题的部分构造解。一旦能够确认，从和节点子孙相对应的选择中无法求得问题的解，这两种技术都会立即终止该节点。
3. 回溯法在它的大多数应用中，都按照**深度优先查找法**构造它的状态空间树。如果状态空间树的当前节点所代表的选择序列可以进一步扩展，而且不会违反问题的约束，它就会考虑下一个分量的第一个余下的合法选择。否则，这个方法就会回溯，也就是撤销部分构造解的最后一个分量，并用下一个选择来代替。
4. 分支界限法是一种算法设计技术，它强化了状态空间树的生成方法。也就是估计可能从状态空间树的当前节点中求得的最佳值，如果这个估计值不超过当前过程中已经得到的最佳解，接下来就不会再考虑该节点。
5. 回溯法例子
    - n皇后问题
    - 哈密顿回路问题
    - 子集和问题
6. 分支界限法例子
    - 分配问题
    - 背包问题
    - 装载问题

    - 旅行商问题
7. 两种解空间树
    - 子集树
        - 当给定问题是从n个元素的集合S中找出满足某种性质的子集时，相应的解空间树称为子集树，如背包问题
        - 图上表现为“一边为包含，一边为不包含”
    - 排列树：当给定问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树，如TSP问题
8.
9.

第9章 贪婪技术

1. 贪婪策略的三个特点：可行的、局部最优的、不可撤回的

- 最优解案例
    - 一些找零问题
    - 最小生成树问题
    - 单源最短路径问题
    - 哈夫曼编码
- 近似解案例
    - TSP旅行商问题
    - 0-1背包问题【对比：连续背包问题中取得最优解，但0-1背包问题中取得近似解】

1.

伪代码可关注问题

1. 分支界限法求背包问题
2. 关于装箱问题，如果最后有时间再看

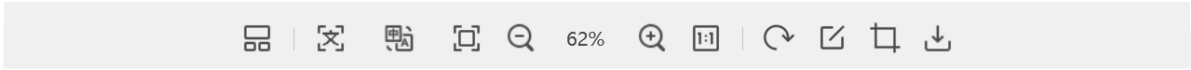
概念

1. 回溯法：算法搜索至解空间树的任一节点时，对于以**该节点为根的子树判断**是否**含有问题的解**，若没有，则**跳过**以该节点为根的子树的**检索**，**逐层**向其祖先结点**回溯。**否则，进入该子树，继续**按深度优先的策略**进行检索。这种以深度优先方式系统地搜索问题的解的算法就称为回溯法。
2. 回溯法与穷举法的区别与联系
    - 联系：都是基于**试探搜索**的方法
    - 区别：
        - 穷举法要将**一个解的各个部分全部生成后**，才检查是否满足条件，**若不满足，则直接放弃该完整解**，然后再尝试另一个可能的完整解，它并没有沿着一个可能完整解的各个部分逐步回退生成解的过程。

        - 回溯法，**一个解的各个部分是逐步生成的**，当发现当前生成的某部分不满足约束条件时，**就放弃该步所做的工作，退到上一步进行新的尝试**，而不是放弃整个解重来。
3. 分支限界法的**基本思想**：常以**广度优先**或以**最小耗费（最大收益）优先**的方式搜索问题的解空间树，**裁剪**那些**不能得到最优解的子树**以**提高搜索效率。**搜索策略是：在**扩展结点处**，先生成其所有的**儿子结点**，然后再从当前的**活结点表**中选择下一个**扩展结点**，以加速搜索的进程，在每个活结点处，**计算一个函数值**，并根据已计算的函数值，从当前活结点表中选择一个**最有利的结点**作为扩展结点，**使搜索**朝着解空间树上**有最优解的分支推进**，以便**尽快地找出一个最优解。**
4. 请简述三种求最短路径问题的算法的区别和联系：多段图法、Floyd算法和迪杰斯特拉(Dijkstra)算法。
    - 区别：
        - 多段图和Floyd是属于dp，多段图是单源到单源，而Floyd则是找所有的点之间的最短路径；迪杰斯特拉是贪心，如何贪心策略的
        - Floyd是对两个结点之间的距离通过增加可能的中间节点寻找更近的路径；多段图在多个阶段直接连接可行的路径，从前往后或从后往前地完成整个最短路径的搜索。
    - 联系：都可以求最短路径；贪心和dp都有最优子结构。
5. 动态规划：如果问题是由交叠的子问题构成的，我们就可以用动态规划技术解决它。一般来说，这样的子问题出现在对给定问题求解的递推关系中，这个递推关系中包含了相同类型的更小子问题的解。所以，从动态规划的角度，与其对交叠子问题一次又一次地求解，还不如对每个较小的子问题只求解一次并把结果记录在表中，这样就可以从表中得出原始问题的解。
6. 动态规划**思想实质**：分治思想和解决冗余——前者是指大问题分为小问题来求解， 后者是指子问题相互交叠，存在冗余。
7. **动态规划的有效性依赖于**：①最优子结构 ： ②重叠子问题
    - 最优子结构：如果问题的最优解是由其子问题的最优解来构造，则称该问题具有最优子结构
    - 重叠子问题：
8. **动态规划和贪心策略的区别和联系**：
    - **子问题上**：
      dp：**每步所做的选择**往往**依赖于子问题的解**，只有在解出相关子问题后才能作出选择
      贪心：**仅在当前状态下**作出最好选择，即 局部最优选择，然后再去**作出这个选择后产生的相应的子问题**，**不依赖于子问题的解**
    - **求解方式**：
      dp：通常以**自底向上的方式解各子问题**
      贪心：通常以**自顶向下**的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题**简化为规模更小的子问题**
9. Bellman最优性原理：最优化问题任一实例的最优解，都是由子实例的最优解构成的。
10. 动态规划的基本解题步骤：
    - 找出**最优解的性质**，并刻画其**结构特征**
    - **递归定义**最优值，写出**动态规划方程**
    - 以**自底向上或自顶向下**的方式**计算出最优值**
    - 根据计算最优值时的**记录信息**，构造**最优解**
11. 贪心选择性：每一步贪心选出来的一定是原问题的最优解的一部分。
12. TSP：n!
13. 动态规划方程那注意写初始条件

Single-source Shortest Paths vs. *Minimum Spanning Tree*

- ***Prim's vs. Dijkstra's Algorithsm***
  - ***Generate different kinds of spanning trees***
    - Prim's: a minimum spanning tree.
    - Dijkstra's : a spanning tree rooted at a given source s, such that the distance from s to every other vertex is the shortest.
  - ***Different greedy strategies***
    - Prims': Always choose the closest (to the tree) vertex in the priority queue Q to add to the expanding tree V_T.
    - Dijkstra's : Always choose the closest (to the source) vertex in the priority queue Q to add to the expanding tree V_T.
  - ***Different labels for each vertex***
    - Prims': parent vertex and the distance from the tree to the vertex..
    - Dijkstra's : parent vertex and the distance from the source to the vertex.



最终复习必看内容

1. ~~考试题再做~~
2. 作业题再过一遍!
3. ~~对于重要概念和纲领性的认知!~~
4. 动态规划、贪心放在最后再看一遍**【考前必再看】**

这道题用一个数组，我们定义一个数组dp，存放的是所有值。

需要关心的是两个点，第一，数据是连续的，第二，子段总以某一位结尾。所以想到可以用每一位来表示该位前所有子段的最大值。然后状态转移方程就是dp[i]=max(dp[i-1]+dp[i]，0)。可能还考虑那个i，j位置地存储等等。