《计算机算法设计与分析》第四章作业

姓名：任宇　　学号：33920212204567

算法实现题 4-1 会场安排问题

问题描述：假设要在足够多的会场里安排一批活动，并希望使用尽可能少的会场。设计一个有效的贪心算法进行安排。（这个问题实际上是著名的图着色问题。若将每个活动作为图的一个顶点，不相容活动间用边相连，使相邻顶点着有不用颜色的最小着色数，相当于要找的最小会场数。）

算法设计：对于给定的 k 个待安排的活动，计算使用最少会场的时间表。

数据输入：第 1 行有 1 个正整数 k，表示有 k 个待安排的活动。接下来的 k 行，每个有两个正整数，分别表示 k 个待安排的活动的开始时间和结束时间。

数据输出：输出计算的最少会场数。

答：算法设计：

首先证明问题的贪心选择性质：假设存在一个最优解，其中第一个安排的活动不是结束最早的活动，那么可以将这个活动替换为结束最早的活动，这不会影响后续的安排，因为结束的时间只会更早或者不变。这意味着，在任何最优解中，第一个活动总是结束最早的活动。由此可见，本问题具有贪心选择性质。

接着证明问题的最优子结构性质：假设有一个按结束时间排序的活动列表 A={a1，a2，a3，……，an}。现在考虑活动 a1，它结束最早。一旦选择了 a1，就必须解决一个子问题，即在不与 a1 冲突的所有活动中进行会场安排。

令 A′ 表示所有与 a1 不相交的活动集合。如果找到了 A′ 的最优解，就可以将 a1 添加到这个解中，因为 a1 和 A′ 中的所有活动都不冲突。现在证明如果 A′ 的解是最优的，加上 a1 后，得到的解仍然是最优的。

假设 S′ 是子问题 A′ 的最优解，它使用了最小的会场数 m，那么可以创建一个解 S，它包含了 a1 和 S′ 中的所有活动，如果 S 不是最优的，那么必须存在另外一个解 S″，它使用少于 m+1 个会场来安排所有 A 中的活动，但是，这说明 S″ 在排除了 a1 以后能用 m 个或更少的会场来安排 A′ 中的所有活动，这与 S′ 是 A′ 的最优解矛盾，因此如果 A′ 的解是最优的，加上 a1 后，得到的解仍然是最优的，即问题具有最优子结构性质。

进行算法设计：

1. 按照活动的结束时间对所有活动进行升序排序。
2. 初始化会场数 numRooms = 0，并创建最小堆来记录每个会场最早可用的时间。
3. 对于每个活动，检查是否有一个会场已经空闲（即该会场的最早可用时间小于或等于当前活动的开始时间）。
   - 如果有，则将当前活动安排在这个会场，并更新该会场的最早可用时间为当前活动的结束时间。
   - 如果没有，则新增一个会场，并将当前活动安排在这个新会场中，更新会场数 numRooms++ 和该会场的最早可用时间。
4. 继续处理直到所有活动都被安排完毕。
5. 输出会场数 numRooms 作为结果。

算法分析： 算法第一步主要步骤的排序所需要的时间复杂度为 O(nlogn)，遍历活动安排会场的时间复杂度是 O(n)，而对于每个活动，使用最小堆来检查最早可用的会场，这个操作的时间复杂度是 O(logk)，其中 k 是当前已使用的会场数，在最坏的情况下，遍历活动和分配会场的时间复杂度可以看做 O(nlogn)，因此算法总的时间复杂度是 O(nlogn)。

算法实现题 4-2 最优合并问题

问题描述：给定 k 个排好序的序列 s1 ， s2,……， sk ，用 2 路合并算法将这 k 个序列合并成一个序列。假设所采用的 2 路合并算法合并 2 个长度分别为 m 和 n 的序列需要 m + n -1 次比较。试设计一个算法确定合并这个序列的最优合并顺序，使所需的总比较次数最少。

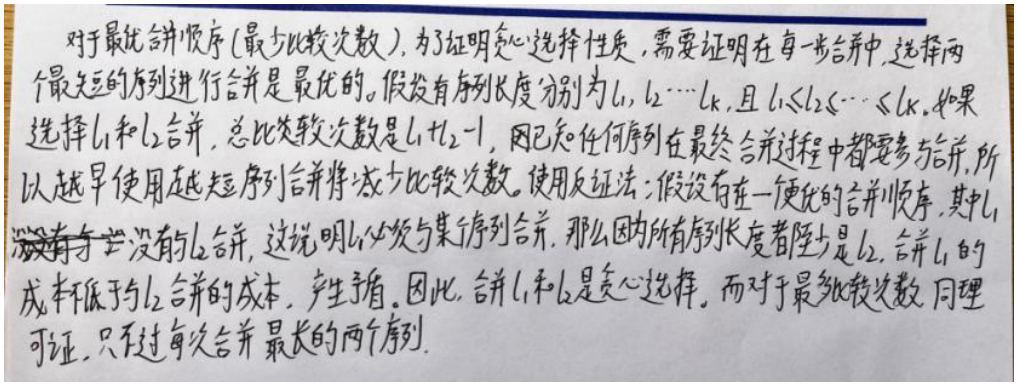
为了进行比较，还需要确定合并这个序列的最差合并顺序，使所需的总比较次数最多。

算法设计：对于给定的 k 个待合并序列，计算最多比较次数和最少比较次数合并方案。

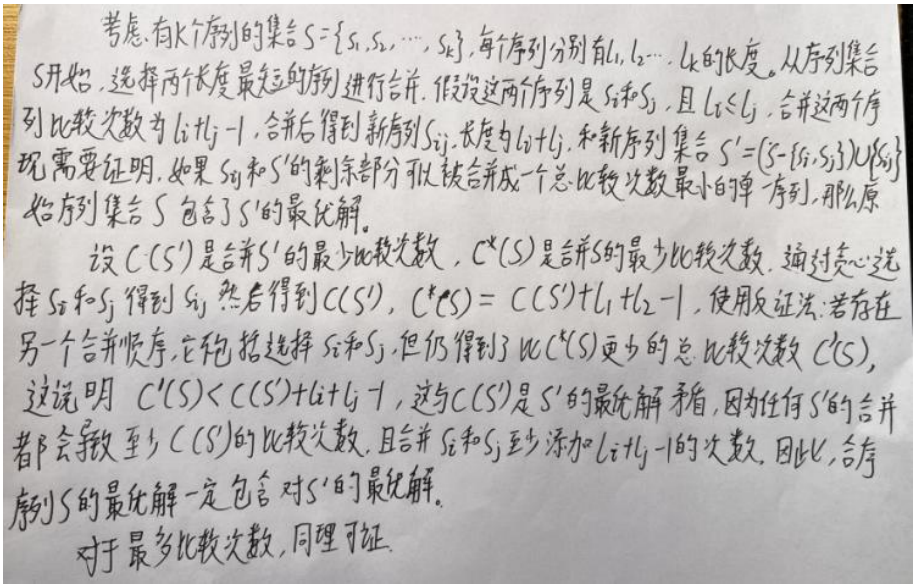
数据输入：第 1 行有 1 个正整数 k，表示有 k 个待合并序列。接下来的 1 行中，有 k 个正整数，表示 k 个待合并序列的长度。

数据输出：输出计算的最多比较次数和最少比较次数。

答：贪心选择性质证明：



最优子结构性质证明：



算法设计：

对于最少比较次数：

1. 创建一个最小堆，将所有序列长度插入最小堆中
2. 合并序列，当堆中元素数量大于 1 时执行以下操作：
   从堆中弹出两个最小元素，计算这两个序列合并的比较次数，将比较次数累加到总比较次数上，同时将合并后的序列插入到堆中。
3. 当堆中只剩下一个元素时，这个元素就代表了合并了所有序列的最终序列，此时累加的总比较次数就是最少比较次数。

对于最多比较次数：

1. 创建一个最大堆，将所有序列长度插入最大堆中
2. 合并序列，当堆中元素数量大于 1 时执行以下操作：
   从堆中弹出两个最大元素，计算这两个序列合并的比较次数，将比较次数累加到总比较次数上，同时将合并后的序列插入到堆中。
3. 当堆中只剩下一个元素时，这个元素就代表了合并了所有序列的最终序列，此时累加的总比较次数就是最多比较次数。

算法分析： 对于最小堆（最大堆），初始化建堆的时间复杂度为 O(n)，每次合并操作后，堆的大小减一，总共需要进行 n-1 次合并操作，而每次对于堆的插入操作和删除操作都需要 O(logn)的时间复杂度，因此，整个算法总的时间复杂度是 O(nlogn)。

算法实现题 4-4 磁盘文件最优存储问题

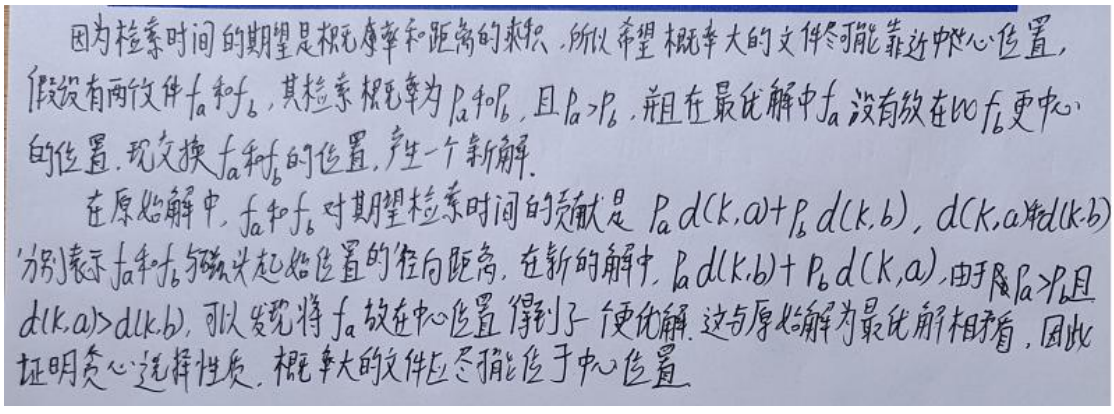
问题描述：设磁盘上有 n 个文件 f1, f2, …, fn, 每个文件占用磁盘上的 1 个磁道。这 n 个文件的检索概率分别是 p1, p2, …, pn, 且$\sum_{i=1}^{n} pi$=1。磁头从当前磁道移到被检信息磁道所需的时间可用这 2 个磁道之间的径向距离来度量。如果文件 fi 存放在第 i 道上，1≦i≦n，则检索这 n 个文件的期望时间是$\sum_{1\le i\le j\le n} pipjd(i,j)$。其中 d(i, j)是第 i 道与第 j 道之间的径向距离|i-j|。

算法设计：对于给定的文件检索概率，计算磁盘文件的最优存储方案。

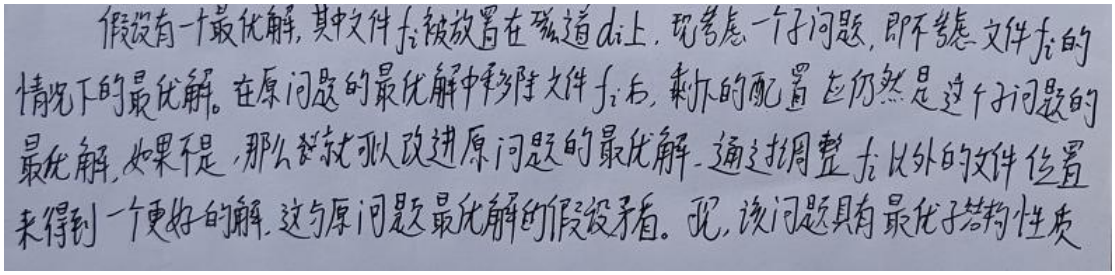
数据输入：第 1 行是正整数 n, 表示文件个数。第 2 行有 n 个正整数 ai，表示文件的检索概率。实际上第 k 个文件的检索概率应为 ak/$\sum_{i=1}^{n} ai$。

数据输出：输出计算的最小期望检索时间

答：贪心选择性质证明：



最优子结构性质证明：



算法设计：

1. 根据检索概率对文件进行降序排序。
2. 将排序后的文件检索概率存放到磁盘数组中，从中心向两边移动，即第一个元素放在中心位置，第二个和第三个元素分别位于中心位置的两边，以此类推，直到所有元素都被放入。
3. 按照题目给定的公式进行计算。

算法分析： 排序文件的检索概率的时间复杂度是 O(nlogn)，放置文件需要 O(n)的时间，而按照题目公式计算期望检索时间的时间复杂度为 $O(n^2)$，所以总的时间复杂度为 $O(n^2)$。

算法实现题 4-6 最优服务次序问题

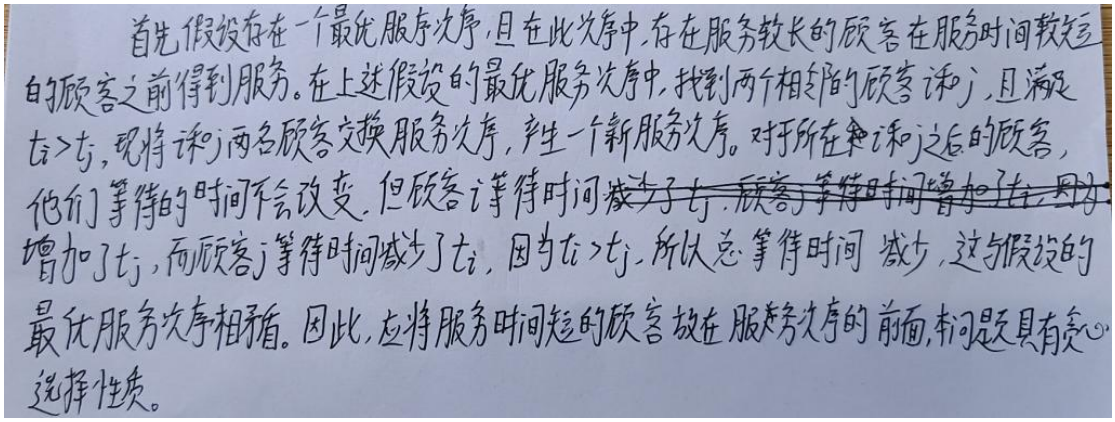
问题描述：设有 n 个顾客同时等待一项服务，顾客 i 需要服务的时间为 ti（1≤i≤n）。应如何安排 n 个顾客的服务次序才能使平均等待时间达到最小？平均等待时间是 n 个顾客等待服务时间总和除以 n。

算法设计：对于给定的 n 个顾客需要的服务时间，计算最优服务次序。

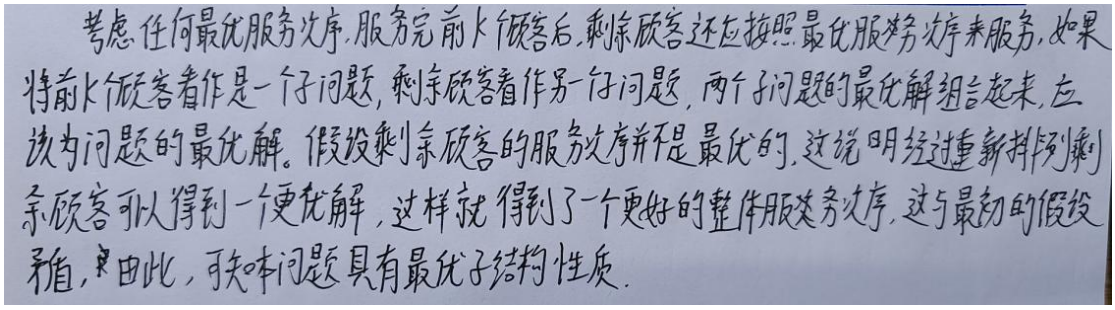
数据输入：第 1 行是正整数 n，表示有 n 个顾客。接下来的 1 行中，有 n 个正整数，表示 n 个顾客需要的服务的服务时间。

数据输出：输出计算的最小平均等待时间。

答：贪心选择性质证明：



最优子结构性质证明：



算法设计：

1. 将服务时间进行升序排序
2. 初始化一个总等待时间变量 total 为 0
3. 依次服务每个顾客，计算 total
4. 计算平均等待时间 average = total / n

算法分析： 排序的时间复杂度为 O(nlogn)，依次服务顾客并计算 total 的时间复杂度为 O(n)，因此，算法总的时间复杂度为 O(nlogn)。

算法实现题 4-8 d 森林问题

问题描述：设 T 为一带权树，树中的每个边的权都为整数。又设 S 为 T 的一个顶点的子集，从 T 中删除 S 中的所有结点，则得到一个森林，记为 T/S。如果 T/S 中所有树从根到叶子节点的路径长度都不超过 d，则称 T/S 是一个 d 森林。

1） 设计一个算法算法求 T 的最小顶点集 S，使 T/S 是 d 森林。

2） 分析算法的正确性和计算复杂性。

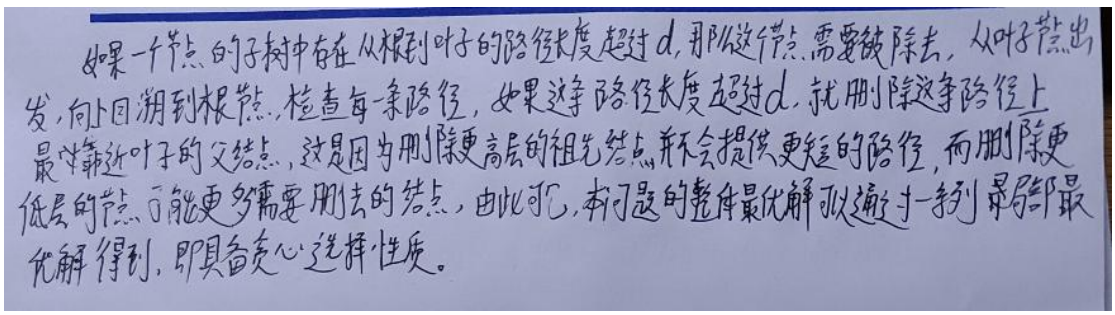
3） 设 T 中有 n 个顶点，则算法的计算时间复杂性应为 O(n)。

算法设计：对于给定的带权树，计算最小分离集 S。

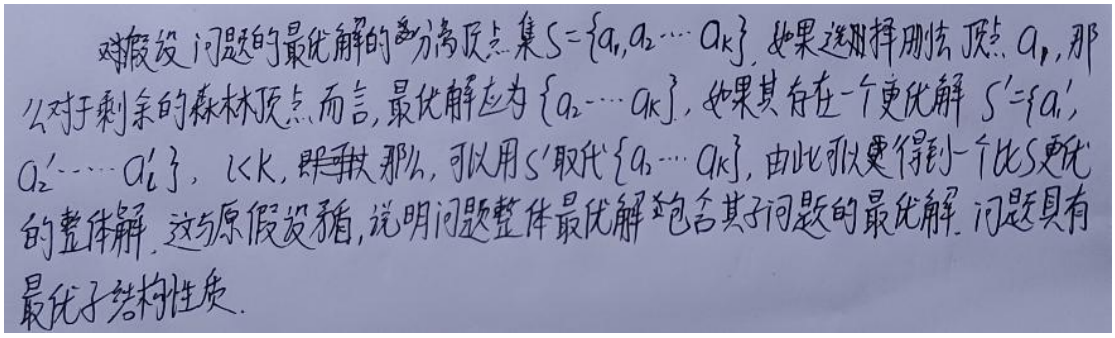
数据输入：第 1 行有 1 个正整数 n，表示给定的带权树有 n 个顶点，编号为 1，2，3，……，n。编号为 1 的顶点是树根。接下来的 n 行里，第 i+1 行描述与 i 个顶点相关联的边的信息。每行的第 1 个正整数 k 表示与该顶点相关联的边数。其后 2k 个数中，每两个数表示一条边。第 1 个数是与该顶点相关联的另一个顶点的编号，第 2 个数是边权值。K=0，表示相应的结点是叶结点。最后一行是正整数 d，表示森林中所有树的从根到叶的路长都不超过 d。

数据输出：输出计算的最小分离集 S 的顶点数。如果无法得到所要求的 d 森林则输出“No Solution!”。

答：贪心选择性质证明：



最优子结构性质证明：



算法设计：

1. 定义一个集合 S，初始为空，用来存储需要删除的顶点。
2. 从叶子节点开始，向上遍历树直到根节点。
3. 对于当前节点，如果它到任何一个叶子的路径长度大于 d，执行步骤 4，否则执行步骤 5。
4. 将当前节点添加到集合 S 中，即删除该节点。
   从树中删除当前节点及其子树。
   返回步骤 2，继续处理树的下一个节点。
5. 如果当前节点到其子节点的最长路径长度不超过 d，无需删除当前节点。
   移动到当前节点的父节点，继续执行步骤 3。
6. 重复步骤 3 至步骤 5，直到遍历完所有节点，从叶子到根节点。
7. 输出集合 S 的大小。
8. 如果在某个时刻，根节点的路径长度也大于 d，则输出“No Solution!”。

算法分析： 由于采取从底向上的遍历方式，每个结点只需要访问一次，这个步骤的时间复杂度为 O(n)，对于每个结点的操作都是常数时间，即 O(1)，因此算法总的时间复杂度是 O(n)，其中 n 是树中的结点数。

算法实现题 4-9 虚拟汽车加油问题

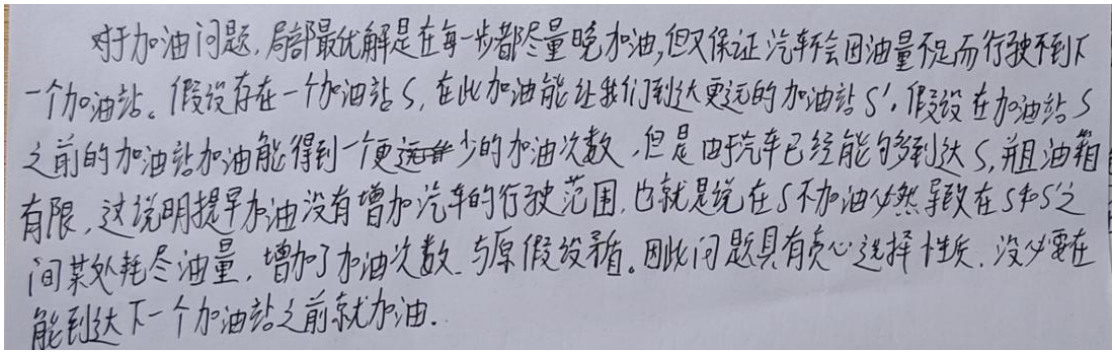
问题描述：一辆虚拟汽车加满油后可以行驶 n km。途中有若干个加油站。设计一个有效的算法，指出应在那个加油站停靠加油，使沿途加油次数最少。并证明算法能产生一个最优解。

算法设计：给定 n 和 k 个加油站位置，计算最少加油次数。

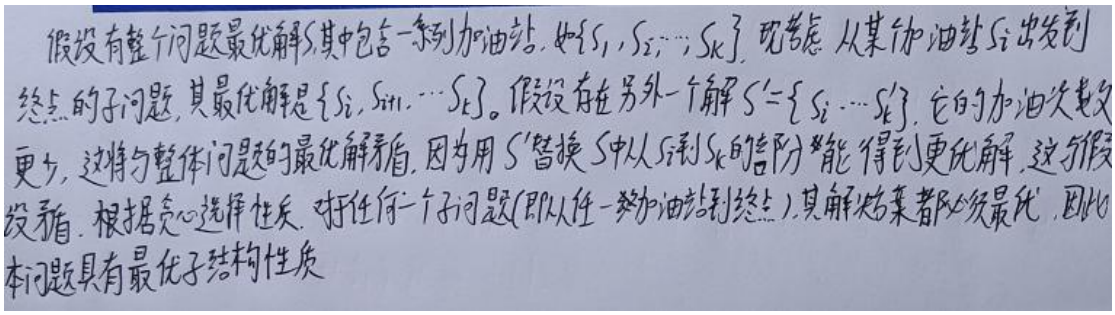
数据输入：第一行有两个整数 n 和 k，表示汽车加满油后可行驶 n km,且路途中有 k 个加油站。接下来的一行中有 k+1 个整数，表示第 k 个加油站与 k-1 个加油站之间的距离。第 0 个加油站表示处出发地，汽车已加满油，第 k+1 个加油站便是目的地。

数据输出：输出计算的最少加油次数。如果无法到达目的地，则输出“No Solution”

答：贪心选择性质证明：


对于加油问题，局部最优解是在每一步都尽量晚加油，但又保证汽车行驶回油量不足而行驶不到下一个加油站。假设存在一个加油站S，在此加油能让我们到达更远的加油站S'，假设在加油站S之前的加油站加油能得到一个更远一步的加油次数，但是由于汽车已经能够到达S，油箱有限，这说明提早加油没有增加汽车的行驶范围，也就是说在S不加油必然导致在S和S'之间某处耗尽油量，增加了加油次数，与原假设矛盾。因此问题具有贪心选择性质，没必要在能到达下一个加油站之前就加油。

最优子结构性质证明：


假设有整个问题最优解S，其中包含一系列加油站，如$\{S_1, S_2, \cdots, S_k\}$。现考虑从某个加油站$S_i$出发到终点的子问题，其最优解是$\{S_i, S_{i+1}, \cdots S_k\}$。假设存在另外一个解$S'=\{S_i \cdots S_k'\}$，它的加油次数更少，这将与整体问题的最优解矛盾，因为用S'替换S中从S_i到S_k的部分能得到更优解，这与假设矛盾。根据贪心选择性质，对任何一个子问题（即从任一类加油站到终点），其解均集都必须最优，因此本问题具有最优子结构性质

算法设计：

1. 初始化一个变量，用于记录当前油箱油量，初始值为 n
2. 初始化加油次数为 0
3. 从起点开始遍历每个加油站位置，如果能到达下一个加油站，则向前移动并更新油量，如果不能到达下一个加油站，则在这个加油站加油，并增加加油次数，接着向前移动并更新油量。
4. 如果在任何时候，不能到达下一个加油站，则输出“No Solution”
5. 如果能到到达终点，则输出加油次数。

算法分析：从起点遍历到终点，时间复杂度为 O(n)，在每次遍历中的操作的时间复杂度是 O(1)，因此算法总的时间复杂度是 O(n)，其中 n 是加油站的数量。

算法实现题 4-11 删数问题

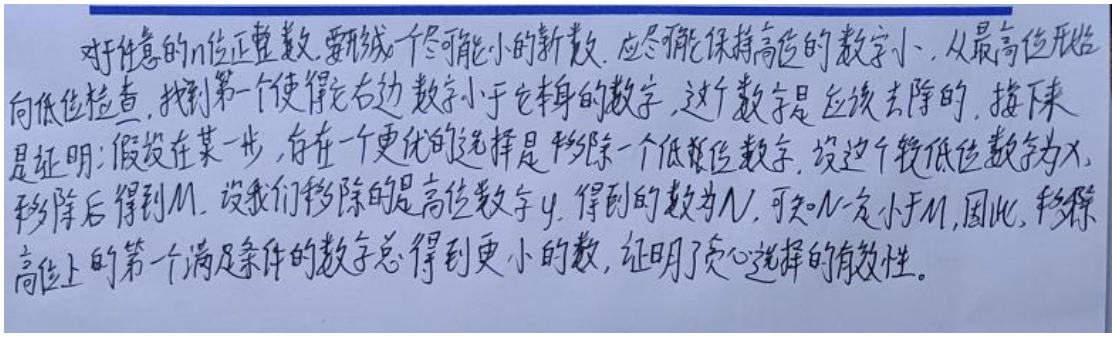
问题描述：给定 n 位正整数 a，去掉其中任意 k≤n 个数字后，剩下的按原次序排列组成一个新的正整数。对于给定的 n 位正整数 k，设计一个算法找出剩下数字组成新数最小的删数方案。

算法设计：对于给定的正整数 a，计算删除 k 个数字后得到的最小数。

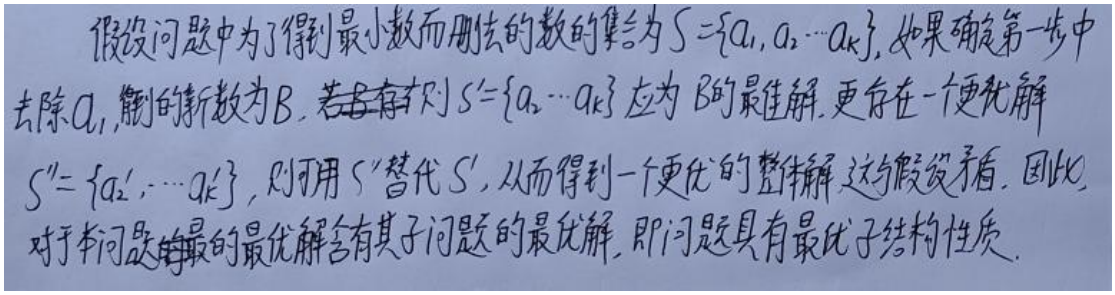
数据输入：文件的第 1 行是 1 个正整数 a。第 2 行是正整数 k。

数据输出：输出计算的最小数

答：贪心选择性质证明：



最优子结构性质证明：



算法设计：

1. 从最高位开始向低位检查数字。
2. 在任意连续的两位数字中，如果前一位大于后一位，则删除前一位。
3. 如果是数字是递增的，则删除最后一位数字。
4. 重复上述过程 k 次。
5. 最终得到的数就是删除 k 个数字后得到的最小数。

算法分析：从最高位开始向低位检查数字，这需要从头到尾遍历数字，最坏的情况下（即数字是递增的），每一位都需要检查，这是 O(n)的操作。在检查过程中，删除操作在普通数组中是 O(n)的，但如果不实际移动数组元素而是标记删除或者是使用链表，那可以视作 O(1)，上述过程需要重复 k 次，因此算法总的时间复杂度是 O(kn)。

算法实现题 4-15 最优分解问题

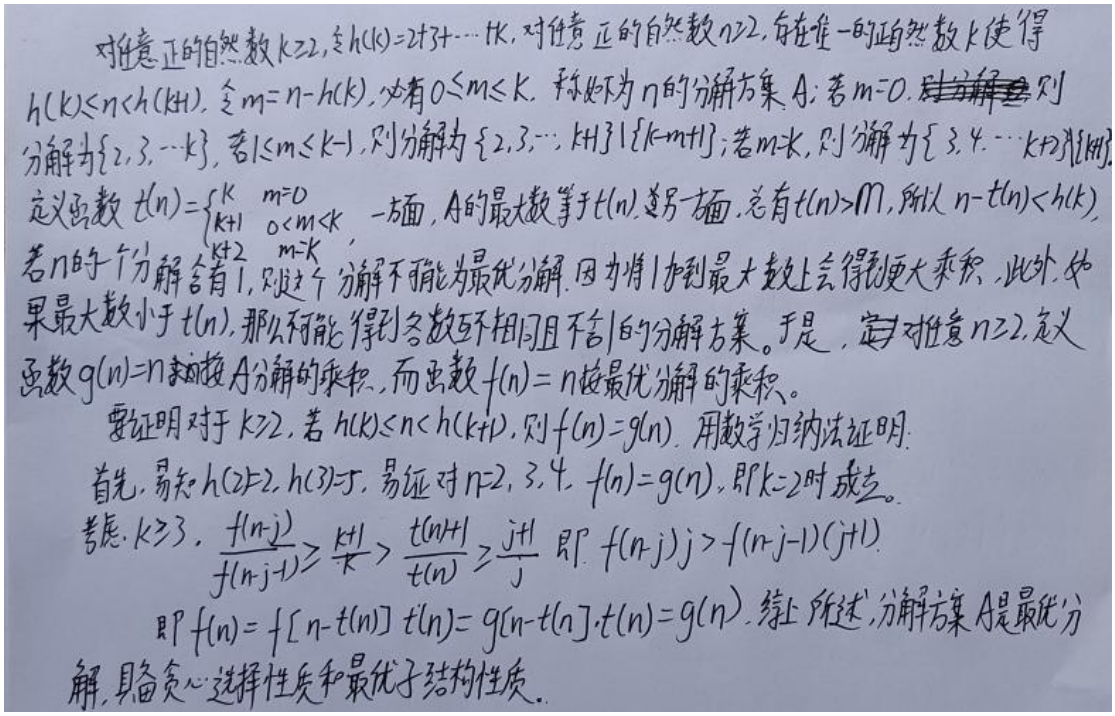
问题描述：设 n 是一个正整数。现在要求将 n 分解为若干互不相同的自然数的和，且使这些自然数的乘积最大。

算法设计：对于给定的正整数 n，计算最优分解方案。

数据输入：第 1 行是正整数 n。

数据输出：输出计算的最大乘积。

答：算法证明：



算法设计：

1. 初始化一个数组存放因数，以及变量 m，初始值为 2。
2. 如果 n=1，则直接输出 1，无需进行后续操作。
3. 进行循环，循环中 n 减去 m，将 m 加入数组，同时给 m 加 1，循环直到 n 小于 m 结束。
4. 循环结束后，处理 n：
   n=0，无需处理 ；
   0<n<m-1，则从后往前给数组中的因数加 1，进行 n 次；
   n=m-1，则先将数组中最后的因数（也是最大的因数）加 1，接着再从最后的因数开始从后往前给数组中的因数加 1，进行 n-1 次。
5. 将数组中的因数累乘得到最大乘积。

算法分析：算法的核心步骤是循环得到最初的因数以及循环处理得到最终的因数，每次循环中的操作时间复杂度是 O(1),循环的时间复杂度是 O(n)，因此，算法总的时间复杂度是 O(n)。