

# 《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  $A = \{a_1, a_2, a_3, \dots, a_n\}$ 。现在考虑活动  $a_1$ ，它结束最早。一旦选择了  $a_1$ ，就必须解决一个子问题，即在不与  $a_1$  冲突的所有活动中进行会场安排。

令  $A'$  表示所有与  $a_1$  不相交的活动集合。如果找到了  $A'$  的最优解，就可以将  $a_1$  添加到这个解中，因为  $a_1$  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  $a_1$  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  $a_1$  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  $a_1$  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  $a_1$  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  $\text{numRooms} = 0$ ，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  $\text{numRooms}++$  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  $\text{numRooms}$  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  $O(n \log n)$ ，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  $O(\log k)$ ，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  $O(n \log n)$ ，因此算法总的时间复杂度是  $O(n \log n)$ 。

#### 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  $s_1, s_2, \dots, s_k$ ，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：

对于最优合并顺序(最少比较次数)，为了证明贪心选择性质，需要证明在每一步合并中，选择两个最短的序列进行合并是最优的。假设有序列长度分别为  $l_1, l_2, \dots, l_k$ ，且  $l_1 \leq l_2 \leq \dots \leq l_k$ 。如果选择  $l_1$  和  $l_2$  合并，总比较次数是  $l_1 + l_2 - 1$ ，因已知任何序列在最终合并过程中都要参与合并，所以越早使用越短序列合并将减少比较次数。使用反证法：假设有在某一更优的合并顺序，其中  $l_1$  没有与  $l_2$  合并，这说明  $l_1$  必须与某序列合并，那么因为所有序列长度都至少是  $l_2$ ，合并  $l_1$  的成本低于与  $l_2$  合并的成本，产生矛盾。因此，合并  $l_1$  和  $l_2$  是贪心选择。而对于最多比较次数同理可证，只不过每次合并最长的两个序列。

#### 最优子结构性质证明：

考虑有  $k$  个序列的集合  $S = \{s_1, s_2, \dots, s_k\}$ ，每个序列分别有  $l_1, l_2, \dots, l_k$  的长度。从序列集合  $S$  开始，选择两个长度最短的序列进行合并。假设这两个序列是  $s_i$  和  $s_j$ ，且  $l_i \leq l_j$ ，合并这两个序列比较次数为  $l_i + l_j - 1$ ，合并后得到新序列  $s_{ij}$ ，长度为  $l_i + l_j$ ，和新序列集合  $S' = (S - \{s_i, s_j\}) \cup \{s_{ij}\}$ 。现需要证明，如果  $s_{ij}$  和  $S'$  的剩余部分可以被合并成一个总比较次数最小的单序列，那么原始序列集合  $S$  包含了  $S'$  的最优解。

设  $C(S')$  是合并  $S'$  的最少比较次数， $C^*(S)$  是合并  $S$  的最少比较次数。通过贪心选择  $s_i$  和  $s_j$  得到  $s_{ij}$ ，然后得到  $C(S')$ ， $C^*(S) = C(S') + l_i + l_j - 1$ 。使用反证法：若存在另一个合并顺序，它包括选择  $s_i$  和  $s_j$ ，但仍得到了比  $C^*(S)$  更少的总比较次数  $C'(S)$ ，这说明  $C'(S) < C(S') + l_i + l_j - 1$ ，这与  $C(S')$  是  $S'$  的最优解矛盾，因为任何  $S'$  的合并都会导致至少  $C(S')$  的比较次数，且合并  $s_i$  和  $s_j$  至少添加  $l_i + l_j - 1$  的次数。因此，合并序列  $S$  的最优解一定包含对  $S'$  的最优解。

对于最多比较次数，同理可证。

### 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  $O(\log n)$  的时间复杂度, 因此, 整个算法总的时间复杂度是  $O(n \log n)$ 。

### 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  $f_1, f_2, \dots, f_n$ , 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  $p_1, p_2, \dots, p_n$ , 且  $\sum_{i=1}^n p_i = 1$ 。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  $f_i$  存放在第  $i$  道上,  $1 \leq i \leq n$ , 则检索这  $n$  个文件的期望时间是  $\sum_{1 \leq i \leq j \leq n} p_i p_j d(i, j)$ 。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  $a_i$ , 表示文件的检索概率。实际上第  $k$  个文件的检索概率应为  $a_k / \sum_{i=1}^n a_i$ 。

**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:

因为检索时间的期望是概率和距离的乘积, 所以希望概率大的文件尽可能靠近中心位置。  
假设有两文件  $f_a$  和  $f_b$ , 其检索概率为  $p_a$  和  $p_b$ , 且  $p_a > p_b$ , 并且在最优解中  $f_a$  没有放在比  $f_b$  更中心的位置, 现交换  $f_a$  和  $f_b$  的位置, 产生一个新解。  
在原始解中,  $f_a$  和  $f_b$  对期望检索时间的贡献是  $p_a d(k, a) + p_b d(k, b)$ ,  $d(k, a)$  和  $d(k, b)$  分别表示  $f_a$  和  $f_b$  与磁头起始位置的径向距离, 在新的解中,  $p_a d(k, b) + p_b d(k, a)$ , 由于  $p_a > p_b$  且  $d(k, a) > d(k, b)$ , 可以发现将  $f_a$  放在中心位置得到了一个更优解, 这与原始解为最优解相矛盾, 因此证明贪心选择性质, 概率大的文件应尽可能位于中心位置。



### 最优子结构性质证明:

假设有一个最优解, 其中文件  $f_i$  被放置在磁道  $d_i$  上, 现考虑一个子问题, 即不考虑文件  $f_i$  的情况下的最优解。在原问题的最优解中移除文件  $f_i$  后, 剩下的配置在仍然是这个子问题的最优解, 如果不是, 那么就可以改进原问题的最优解。通过调整  $f_i$  以外的文件位置来得到一个更好的解, 这与原问题最优解的假设矛盾。现, 该问题具有最优子结构性质。

### 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  $O(n \log n)$ , 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  $t_i$  ( $1 \leq i \leq n$ )。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:**

首先假设存在一个最优服务次序, 且在此次序中, 存在服务较长的顾客在服务时间较短的顾客之前得到服务。在上述假设的最优服务次序中, 找到两个相邻的顾客  $i$  和  $j$ , 且满足  $t_i > t_j$ , 现将  $i$  和  $j$  两名顾客交换服务次序, 产生一个新服务次序。对于所在  $i$  和  $j$  之后的顾客, 他们等待的时间不会改变, 但顾客  $i$  等待时间减少了  $t_j$ , 顾客  $j$  等待时间增加了  $t_i$ , 因为增加了  $t_j$ , 而顾客  $j$  等待时间减少了  $t_i$ , 因为  $t_i > t_j$ , 所以总等待时间减少, 这与假设的最优服务次序相矛盾。因此, 应将服务时间短的顾客放在服务次序的前面, 本问题具有贪心选择性质。

### 最优子结构性质证明:

考虑任何最优服务次序, 服务完前  $k$  顾客后, 剩余顾客还应按照最优服务次序来服务, 如果将前  $k$  顾客看作是一个子问题, 剩余顾客看作另一个子问题, 两个子问题的最优解组合起来, 应该为问题的最优解。假设剩余顾客的服务次序并不是最优的, 这说明经过重新排列剩余顾客可以得到一个更优解, 这样就得到了一个更好的整体服务次序, 这与最初的假设矛盾, 由此, 该问题具有最优子结构性质。

#### 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  $O(n \log n)$ , 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  $O(n \log n)$ 。

#### 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:

如果一个结点的子树中存在从根到叶子的路径长度超过  $d$ , 那么这个结点需要被删除。从叶子结点出发, 向上回溯到根结点, 检查每一条路径, 如果这条路径长度超过  $d$ , 就删除这条路径上最靠近叶子的父结点, 这是因为删除更高层的祖先结点并不会提供更短的路径, 而删除更低层的结点可能更多需要删除的结点, 由此可见, 本问题的整体最优解可以通过一系列局部最优解得到, 即具备贪心选择性质。

#### 最优子结构性质证明:

对假设问题的最优解的顶点集  $S = \{a_1, a_2, \dots, a_k\}$ ，如果选择删除顶点  $a_1$ ，那么对于剩余的森林顶点而言，最优解应为  $\{a_2, \dots, a_k\}$ ，如果其存在一个更优解  $S' = \{a'_1, a'_2, \dots, a'_k\}$ ， $k < k$ ，那么，可以用  $S'$  取代  $\{a_2, \dots, a_k\}$ ，由此可以得到一个比  $S$  更优的整体解，这与原假设矛盾，说明问题整体最优解包含其子问题的最优解。问题具有最优子结构性质。

#### 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,但又保证汽车会因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少的加油次数,但是由于汽车已经能够到达  $S$ ,并且油箱有限,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,其中包含一系列加油站,如  $\{s_1, s_2, \dots, s_k\}$ 。现考虑从某加油站  $s_i$  出发到终点的子问题,其最优解是  $\{s_i, s_{i+1}, \dots, s_k\}$ 。假设有在另外一个解  $S' = \{s_i, \dots, s_k\}$ , 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,对于任何一个子问题(即从任一加油站到终点),其解始末都必须最优,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  $k \leq n$  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:

对于任意的 $n$ 位正整数,要删成一个尽可能小的新数,应尽可能保持高位的数字小,从最高位开始向低位检查,找到第一个使得它右边数字小于它本身的数字,这个数字是应该去除的,接下来是证明:假设在某一步,存在一个更优的选择是删除一个较低位数字,设这个较低位数字为 $x$ ,删除后得到 $M$ ,设我们删除的是高位数字 $y$ ,得到的数为 $N$ ,可知 $N$ 一定小于 $M$ ,因此,删除高位上的第一个满足条件的数字总得到更小的数,证明了贪心选择的有效性。

#### 最优子结构性证明:

假设问题中为了得到最小数而删除的数的集合为 $S = \{a_1, a_2, \dots, a_k\}$ ,如果确定第一步中去除 $a_1$ ,剩下的新数为 $B$ ,若 $B$ 有子问题,则 $S' = \{a_2, \dots, a_k\}$ 应为 $B$ 的最佳解,更存在一个更优解 $S'' = \{a'_2, \dots, a'_k\}$ ,则可用 $S''$ 替代 $S'$ ,从而得到一个更优的解,这与假设矛盾,因此,对于本问题的最优解含有其子问题的最优解,即问题具有最优子结构性质。

#### 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:**



对任意正的自然数  $k \geq 2$ , 令  $h(k) = 2+3+\dots+k$ , 对任意正的自然数  $n \geq 2$ , 存在唯一的自然数  $k$  使得  $h(k) \leq n < h(k+1)$ , 令  $m = n - h(k)$ , 必有  $0 \leq m \leq k$ . 称此为  $n$  的分解方案  $A$ ; 若  $m=0$ , 则分解为  $\{2, 3, \dots, k\}$ ; 若  $1 \leq m \leq k-1$ , 则分解为  $\{2, 3, \dots, k+1\} \setminus \{k-m+1\}$ ; 若  $m=k$ , 则分解为  $\{3, 4, \dots, k+2\} \setminus \{k+1\}$ .  
 定义函数  $t(n) = \begin{cases} k & m=0 \\ k+1 & 0 < m < k \\ k+2 & m=k \end{cases}$ , 一方面,  $A$  的最大数等于  $t(n)$ , 另一方面, 总有  $t(n) > m$ , 所以  $n - t(n) < h(k)$ .  
 若  $n$  的一个分解含有 1, 则这个分解不可能是最优分解. 因为将 1 加到最大数上会得到更大乘积. 此外, 如果最大数小于  $t(n)$ , 那么可能得到各数互不相同且不含 1 的分解方案. 于是, 对任意  $n \geq 2$ , 定义函数  $g(n) = n$  按  $A$  分解的乘积, 而函数  $f(n) = n$  按最优分解的乘积.

要证明对于  $k \geq 2$ , 若  $h(k) \leq n < h(k+1)$ , 则  $f(n) = g(n)$ . 用数学归纳法证明.

首先, 易知  $h(2)=2, h(3)=5$ , 易证对  $n=2, 3, 4$ ,  $f(n)=g(n)$ , 即  $k=2$  时成立.

考虑  $k \geq 3$ ,  $\frac{f(n-j)}{f(n-j-1)} \geq \frac{k+1}{k} > \frac{t(n+1)}{t(n)} \geq \frac{j+1}{j}$  即  $f(n-j)j > f(n-j-1)(j+1)$ .

即  $f(n) = f[n-t(n)] \cdot t(n) = g[n-t(n)] \cdot t(n) = g(n)$ . 综上所述, 分解方案  $A$  是最优分解, 具备贪心选择性质和最优子结构性质.

#### 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  $O(n)$ .