

答：

**（1）证明**

旅行售货员回路需经过所有顶点并返回起点。对于前缀 x[1:i]，已确定的边费用为 。剩余部分需从 xi​ 出发，经过 n−i 个未完全访问的顶点。由于从每个顶点 xj​ 出发的边最小费用为 min(xj​)，剩余路径费用至少为（每个顶点至少贡献一条最小费用边）。因此，整个回路费用至少为，得证。

**（2）基于上述结论的回溯法设计与分析**

**上界函数设计**：在回溯过程中，对当前扩展节点，计算 。若 c 大于当前已知最小回路费用（初始可设为一个较大值，后续更新），则剪枝该分支。

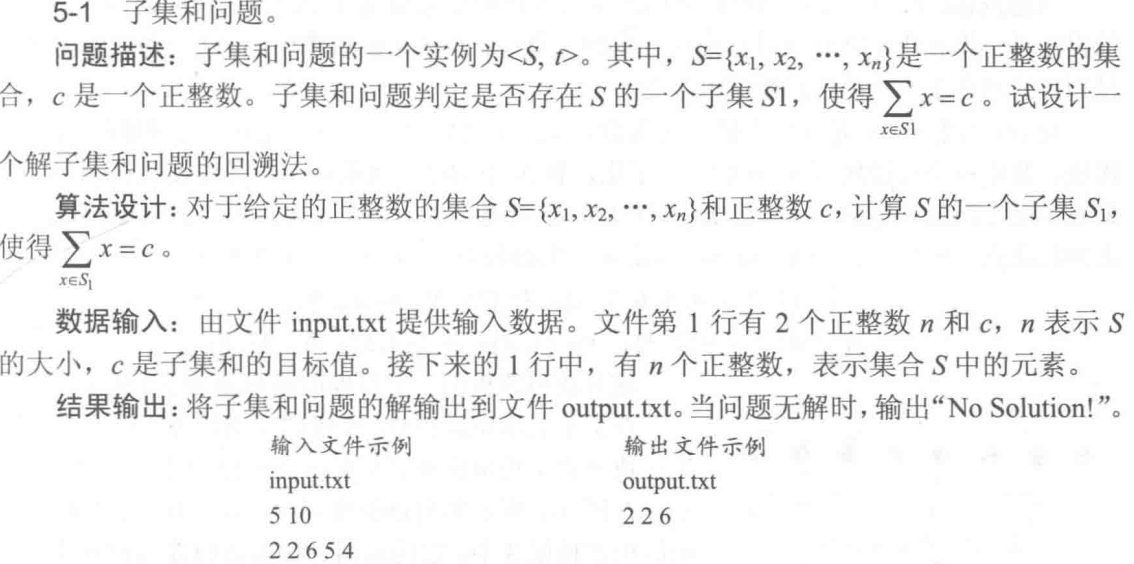
**算法步骤**：

初始化：计算每个顶点 i 的 min(i)，设初始上界 u=+∞。

回溯搜索：从起点开始，递归扩展路径。每扩展一步 xi​，计算当前路径费用与剩余 之和 c。

剪枝判断：若 c≥u，跳过该分支；否则继续搜索。若完成一个回路且费用小于 u，更新 u。

**与教材算法比较**：该方法利用更紧的下界，能更早剪枝。教材中的回溯法若未采用此下界，可能搜索更多无效分支。此方法通过 快速评估剩余路径最小代价，减少不必要的搜索，提高效率。



答：思路是通过回溯法遍历子集树，尝试所有可能的子集组合，判断是否存在和为 c 的子集，符合子集和问题的求解要求。

**回溯函数设计**：

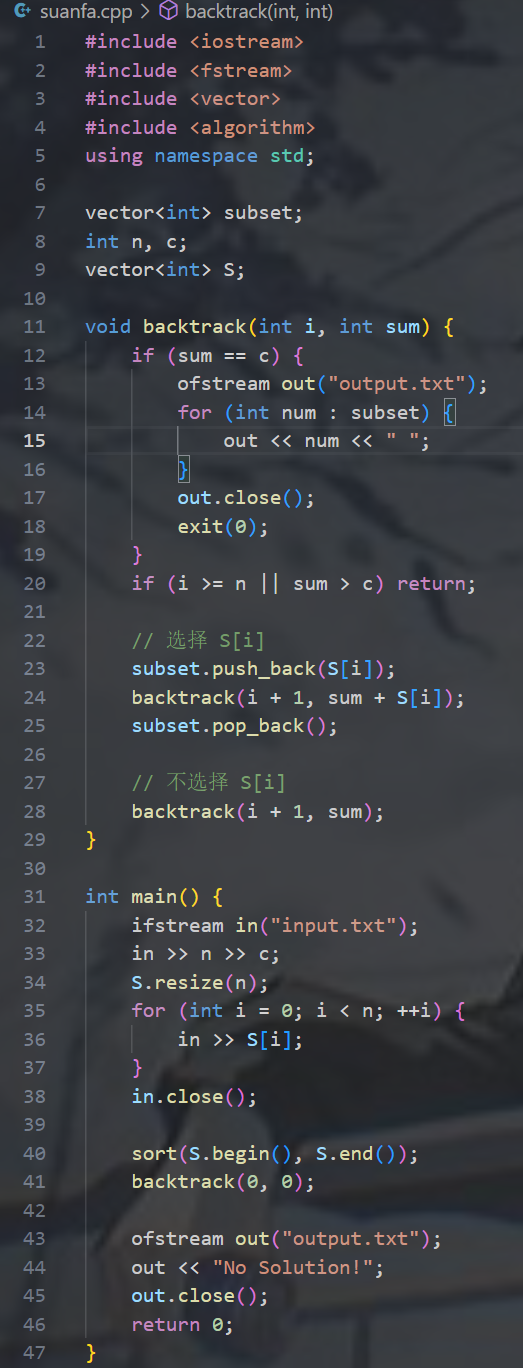
backtrack(int i, int sum)：i 是当前处理的元素索引，sum 是当前子集和。

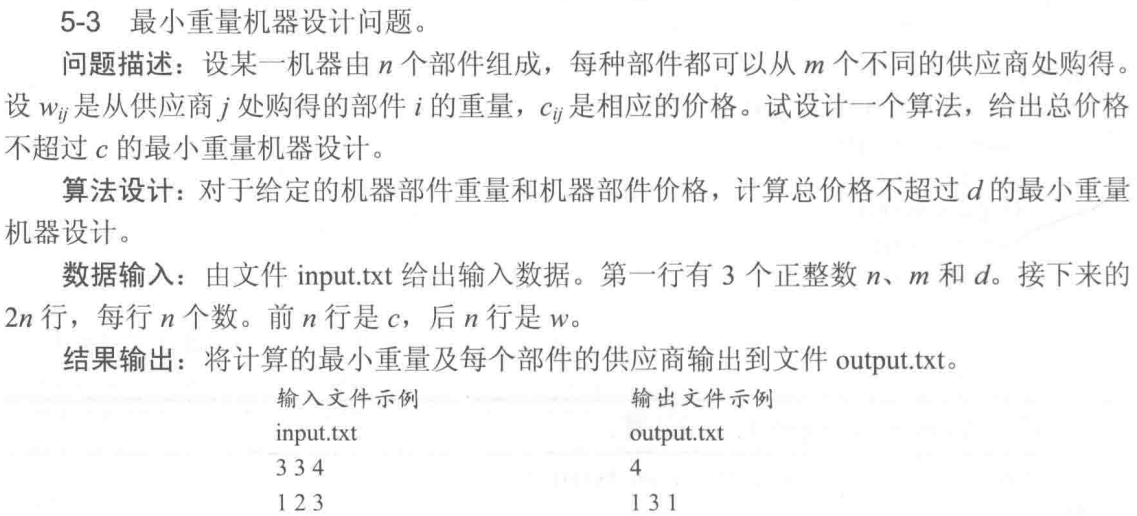
若 sum == c，找到解，输出到 output.txt 并结束程序。

若 i >= n（超出范围）或 sum > c（和超过目标），回溯。

递归处理选择 S[i] 和不选择 S[i] 两种情况。

代码如下：



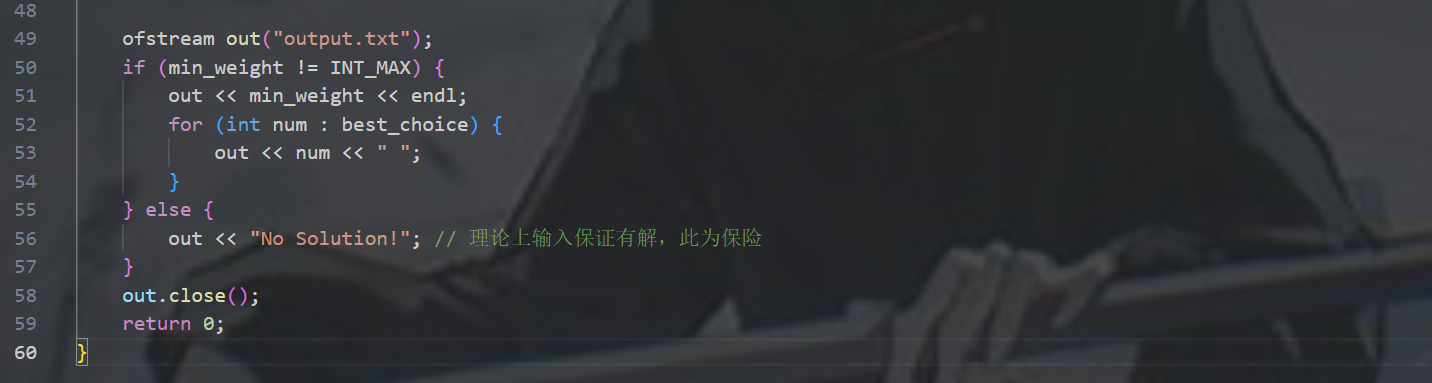


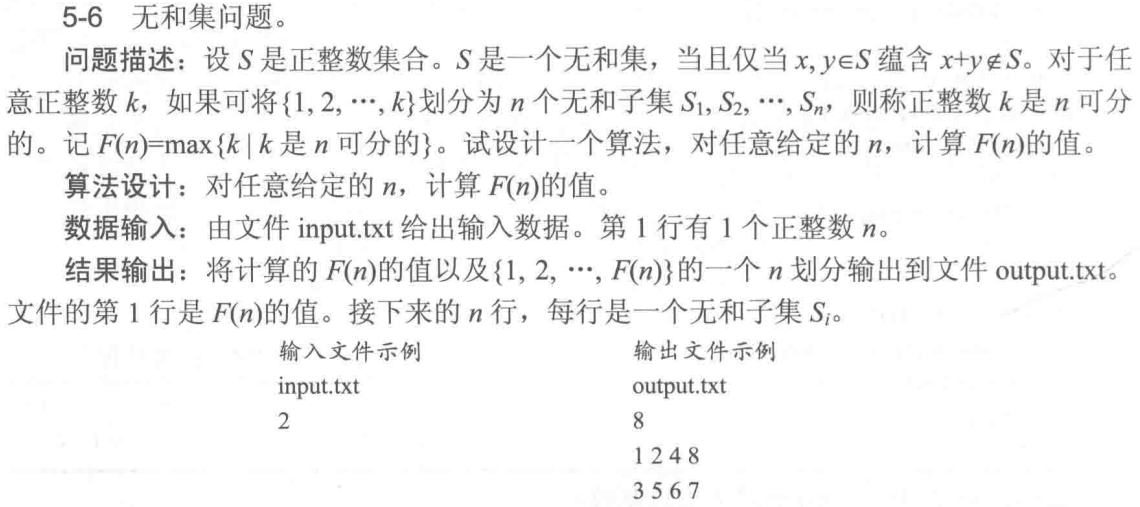


答：思路：用回溯法，就是递归地选每个部件的供应商。对于每个部件 i，遍历 m 个供应商 j。如果选了 j 后，总价格不超过 d，就继续选下一个部件。记录当前的总重量和总价格。当处理完所有 n 个部件时，检查总价格是否符合条件，如果符合且重量更小，就更新最优解。

代码如下：







答：算法思想：从1开始回溯，建立两个两个二维数组，一个存放中间值，一个存放最终的结果。

dfs(1) a[0][1] a[0][0]++ t+1

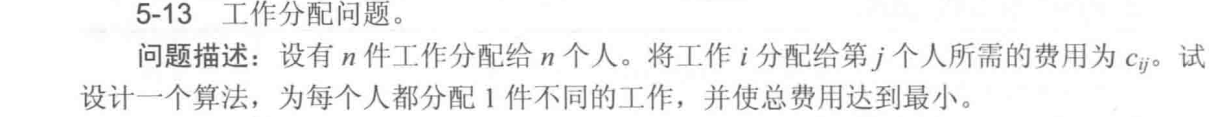
dfs(2) a[0][2] a[0][0]++ t+1

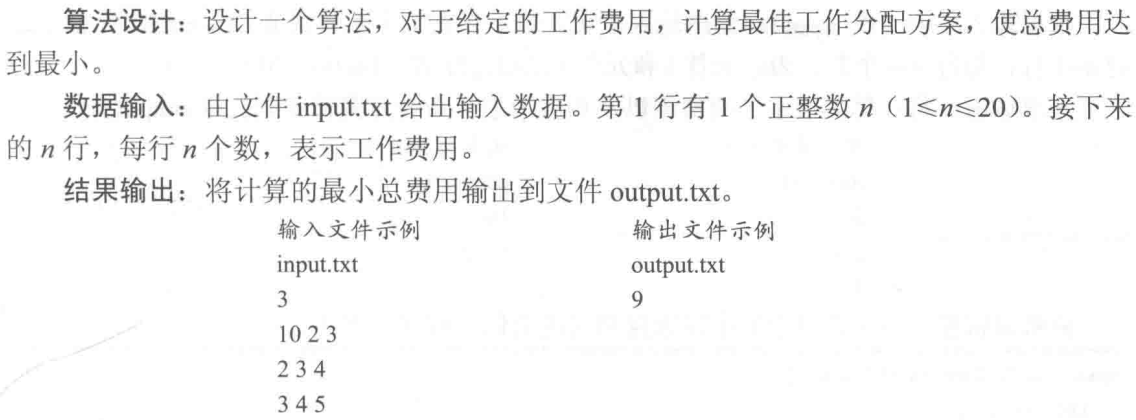
dfs(3) a[0][3]不满足，a[1][1] a[1][0]++ t+1

dfs(4) a[0][3]或者a[1][2]都满足 回溯尝试，找到一个k最大的情况

代码如下：







答：思路：读取输入的 n 和费用矩阵 c。

初始化 minCost 为一个很大的数，比如 INT\_MAX。

定义一个数组 used 来记录工作是否被分配，或者用 path 数组记录已分配的工作。

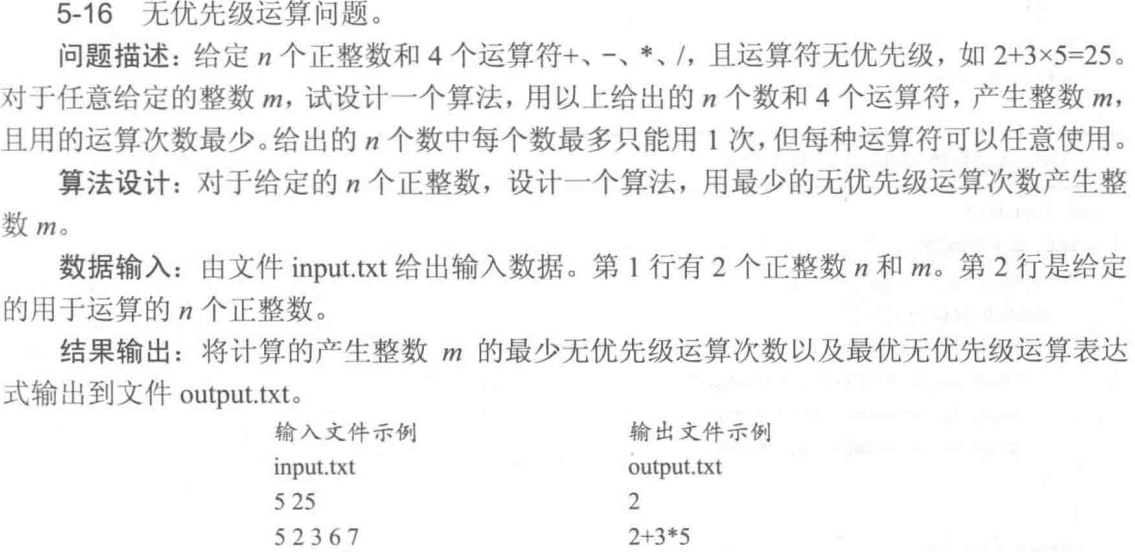
回溯函数，参数是当前处理的人的编号（比如 k），当前费用 sum，以及 path 数组。

在回溯函数中，当 k == n 时，更新 minCost。

否则，对于每个工作 i（0 到 n-1），如果未被分配，就分配给第 k 个人，递归 k+1，sum + c [k][i]。

代码如下：





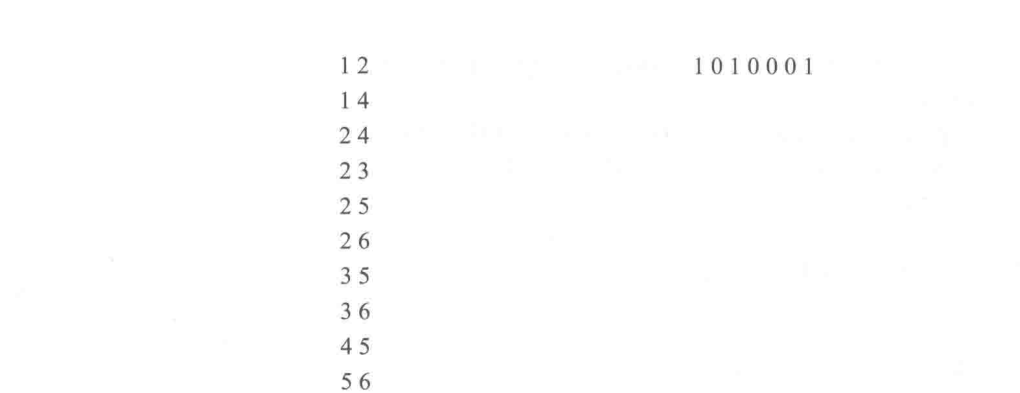
答：算法思路：使用回溯法，尝试不同的数字组合和运算。每次选择两个数字进行四种运算中的一种，得到一个新的结果，然后将这个结果加入剩余数字中，继续递归。记录运算次数，找到最小的。

代码如下：









答：算法思路：

构建仇敌关系图。可以用邻接表或者邻接矩阵。这里用邻接矩阵方便，比如 bool enemy [n+1][n+1]，记录 u 和 v 是否是仇敌。

回溯法搜索。定义一个数组 x [i] 表示第 i 个居民是否被选（1 选，0 不选）。递归过程中，尝试选或不选当前居民，同时检查是否与已选的有仇敌关系。

记录最大的人数和对应的 x 数组。

代码如下：

