《算法设计与分析》第4次作业答案

姓名：XXX　　　　学号：XXXXXXXX

算法分析题

题目1：给定n个物品，物品价值分别为$P_1, P_2, ..., P_n$，物品重量分别为$W_1, W_2, ..., W_n$，背包容量为M。每种物品可部分装入到背包中。输出$X_1, X_2, ..., X_n$，$0 \le X_i \le 1$，使得$\sum_{1 \le i \le n} P_i X_i$最大，且$\sum_{1 \le i \le n} W_i X_i \le M$。试设计一个算法求解该问题，答案需包含以下内容：证明该问题的贪心选择性，描述算法思想并给出伪代码。

答：

算法思想：对于该问题，我们可以用贪心算法解决。首先对于每个物品，计算其单位重量的价值，即$C_i = P_i / W_i$。然后再按照物品单位价值C_i进行从大到小的排序，再按照先放单位价值高的物品进背包的原则进行物品放置，当背包的剩余容量大于当前最大单位价值的物品的重量时，则将该物品全部放入；若小于当前最大单位价值的物品的重量，则放入该物品直到背包填满。

贪心选择性：将尽可能多的单位重量价值最高的物品装入背包，直到包装满为止，对于最后一件物品可能不能完全装入，只能装入一部分。

证明：假设物品$1, 2, ..., n$已按照单位价值排序,先证明最优解中包括物品 1，然后假设最优解中包括物品$1, 2..., k$, 且$M - \sum_{i=1}^{k} > 0$, 证明最优解中包括物品$k+1$。具体如下：

1、当M $\geqslant W_1$ 时，如果最优解中不包括物品 1(或只部分装入物品 1)，则可以用物品 1 替换其他任意物品直至完全装入物品1。当M $< W_1$时，若最优解中不包括重量为M的物品 1，则可以将整个背包放满物品 1。两种情况下，新解的价值均大于等于最优解。因此，物品1在最优解中。

2、假设最优解中包括物品$1, 2, ..., k$，构造子问题:背包容量为$M' = M - \sum_{i=1}^{k} W_i > 0$, 可选物品为$k+1, k+2, ..., n$。此时，物品$k+1$ 是子问题中单位价值最高的物品，故子问题的最优解中包含物品$k+1$。根据最优子结构的性质，原问题的最优解中也包含物品$k+1$。

伪代码：

Algorithm 1 贪心算法求解部分背包问题

Input: $P_1, P_2, ..., P_n$; $W_1, W_2, ..., W_n$; M;
Output: $X_1, X_2, ..., X_n$;

```
for i ← 1 to n do
    X_i ← 0
end for
for i ← 1 to n do
    C_i ← P_i/W_i
end for
```

将物品按照C_i从大到小排序，排序结果为$S(i)$。表示物品i 的单位价值排在第$S(i)$位。
for $i \leftarrow 1$ to n and $M > 0$ **do**
    if $M \geq W_{S(i)}$ **then**
        $X_{S(i)} \leftarrow 1$
        $M \leftarrow M - W_{S(i)}$
    else
        $X_{S(i)} \leftarrow M/W_{S(i)}$
        $M \leftarrow 0$
    end if
end for
return X;

题目2：假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。对每个孩子 i，都有一个胃口值 g_i，这是能让孩子们满足胃口的饼干的最小尺寸；并且每块饼干 j，都有一个尺寸 s_j。如果 $s_j \geq g_i$，我们可以将这个饼干 j 分配给孩子 i ，这个孩子会得到满足。你的目标是尽可能满足越多数量的孩子，并输出这个最大数值。注意：你可以假设胃口值为正。一个小朋友最多只能拥有一块饼干。试设计一个算法求解该问题，答案需包含以下内容：证明该问题的贪心选择性，描述算法思想并给出伪代码。示例如下：
输入: [1,2], [1,2,3]
输出: 2
解释: 你有两个孩子和三块小饼干，2个孩子的胃口值分别是1,2。你拥有的饼干数量和尺寸都足以让所有孩子满足。所以你应该输出2。

答：

算法思想：因为胃口值最小的孩子最容易吃饱，所以先满足了这个孩子之后，我们再采取同样的策略，考虑剩下孩子里胃口值最小的孩子，直到没有满足条件的饼干存在。因此这里的贪心策略是，给剩余孩子里最小胃口值的孩子分配最小的能饱腹的饼干。我们首先将孩子和饼干分别按由小到大进行排序，这样我们就可以从胃口值最小的孩子和大小最小的饼干出发，计算有多少个饼干-孩子对可以满足条件。

贪心选择性：给剩余孩子里最小胃口值的孩子分配最小的能饱腹的饼干。

证明：假设每个孩子的胃口值和饼干的大小都已经按照从小到大的顺序排好了序。先证明最优解中包括孩子 1，然后假设最优解中包括孩子$1, 2..., k$, 且饼干还有,再证明最优解中包括孩子$k+1$。具体如下：

1、如果最优解中不包括孩子 1，则可以用孩子 1 替换其他任意一个孩子，而且使得剩余的大份的饼干数量会更多。这种情况下，新解的价值一定大于等于最优解。因此最优解里面包含孩子1。

2、假设前k个小孩已经选择好饼干，第$k+1$个小孩去选饼干，他如果没有选择饼干，而第$k+2$个小孩拿到了饼干，那么把$k+2$的饼干给$k+1$，一样可以满足$k+1$这个小孩，且拿到饼干的小孩总数没有变，所以$k+1$小孩去选饼干必然会得出一个最优解。

伪代码：

Algorithm 2 贪心算法求解分饼干问题

Input: $g_1, g_2, ..., g_n$; $s_1, s_2, ..., s_n$;

Output: 能满足的孩子的数量

Sort($g_1, g_2, ..., g_n$);
Sort($s_1, s_2, ..., s_n$);
$child \leftarrow 0$;
$cookie \leftarrow 0$;
while $child < children.length$ and $cookie < cookies.length$ **do**
    if $g_{child} \leq s_{cookie}$ **then**
        $child \leftarrow child + 1$;
    end if
    $cookie \leftarrow cookie + 1$;
end while
return $child$;

题目3：给定一个区间的集合，找到需要移除区间的最小数量，使剩余区间互不重叠。可以认为区间的终点总是大于它的起点, 另外像区间 [1,2] 和 [2,3] 这样边界相互“接触”的区间，可以认为没有相互重叠。试设计一个算法求解该问题，答案需包含以下内容：证明该问题的贪心选择性，描述算法思想并给出伪代码。示例如下：

输入: [[1,2], [2,3], [3,4], [1,3]]

输出: 1

解释: 移除 [1,3] 后，剩下的区间没有重叠。

答：

算法思想：该问题本质上等同于活动安排问题，找到需要移除区间的最小数量，我们只需要留住最大数量的区间即可。针对此类问题，我们既可以考虑按照区间的起点进行排序，也可以考虑按照区间的终点进行排序。对于这道题，我们按照区间的终点进行排序。然后对排好序的区间进行遍历。在遍历的过程中，若没有区间重叠，就不移除任何区间。如果存在重叠，就直接删除掉当前区间。

贪心选择性：按照区间的终点进行递增排序，终点小的先放入结果集合中。

证明：假设所有区间已经按照终点排好了序。那我们首先证明存在最优解包含区间1，然后假设最优解包含区间k，并证明也包含区间$k+1$。具体如下：

1、任取最优解 A，A 中区间按终点递增排序。如果 A 的第一个区间为 j，且 $j \neq 1$，用1替换 A 的区间 j 得到解A’，即A’=$(A - \{j\}) \cup \{1\}$。由于区间1的终点小于区间j的终点，所以新解A’一定优于A,因此最优解一定包含区间 1。

2、算法执行到第k步，选择了区间 $i_1 = 1, i_2, ..., i_k$，根据归纳假设存在最优解A包含$i_1 = 1, i_2, ..., i_k$，A中剩下区间选自集合S',其中S'=$\{i | i \in S, i.start \geq k.end\}$, A=$\{i_1 = 1, i_2, ..., i_k\} \cup B$, 其中B是S’的最优解，若不然，S’的最优解为B*，B* 的区间比B多，那么B* $\cup \{i_1 = 1, i_2, ..., i_k\}$是S的最优解，且比A的区间多，与A的最优性矛盾!

伪代码：

Algorithm 3 贪心算法求解区间覆盖问题

Input: $intervals\ set$;

Output: 需要移除的区间数量;

```
if intervals.length = 0 then
    return 0;
end if
Sort(intervals);
end ← intervals[0].end;
count ← 1
for i ← 1 to intervals.length do
    if intervals[i].start ≥ end then
        end ← intervals[i].end;
        count ← count + 1;
    end if
end for
return intervals.length − count;;
```