李青航 SA22225226

16.1-2

Algorithm 1 GREEDY-ACTIVITY-SELECTOR(s, f)

$n = s.length$
$A = \{a_n\}$
$k = n$
for $m = n - 1$ to 1 **do**
  if $f[m] <= s[k]$ **then**
    $A = A \bigcup \{a_m\}$
    $k = m$
  end if
end for
return A

令A_k是S_k的一个最大兼容活动子集，且a_j是A_k中开始时间最晚的活动若$a_j = a_m$，则已经证明a_m在S_k的某个最大兼容活动子集中。若$a_j \neq a_m$，令集合$A'_k = A_k - \{a_j\} \cup \{a_m\}$，即将$A_k$ 中的a_j替换为a_m。A'_k中的活动都是不相交的，因为A_k 中的活动都是不相交的，a_j是A_k中开始时间最晚的活动，而$s_j \leq s_m$。由于$|A'_k| = |A_k|$，因此得出结论A'_k也是 S_k的一个最大兼容活动子集，且他包含a_m，所以贪心总是安全的。

所以可以选择一个活动放入最优解，然后对剩余子问题继续求解。

16.2-2

见算法2

状态变量：$K[i][j]$表示前i件物品放入容量为j的背包的最大价值

当前容量为j，我们要考虑第i件物品能否放入？是否放入？

1.如果当前背包容量$j < v[i]$,不能放入，则$K[i][j] = K[i-1][j]$

2.如果当前背包容量$j >= v[i]$，能放入但是要比较代价

2.1 如果第i件物品不放入背包，则$K[i][j] = K[i-1][j]$

2.2 如果第i件物品放入背包，则$K[i][j] = K[i-1][j - v[i]] + w[i]$

对于01背包来说边界就是f[i][j]=0,即当i=0或者j=0时f[i][j]的值为0。

$i = 0$时，表示背包没有放入一个物品，那么此时背包的最大价值无从谈起，所以为0；$j = 0$时，表示背包的容量为0，那么此时无法放入物品，所以最大价值也为0；

Algorithm 2 0-1 Knapsack(n, W)

Initialize a new table $K[n+1, W+1]$
for $j = 1$ to W **do**
  $K[0, j] = 0$
end for
for $i = 1$ to n **do**
  $K[i, 0] = 0$
end for
for $i = 1$ to n **do**
  for $j = 1$ to W **do**
    if $j < i.weight$ **then**
      $K[i, j] = K[i-1, j]$
    end if
    $K[i, j]$ =max$(K[i-1, j], K[i-1, j - i.weight] + i.value)$
  end for
end for