# 算法第七次作业

**10205101530-赵晗瑜**

## 15.5-3

**15.5-3** 假设 OPTIMAL-BST 不维护表 $w[i, j]$，而是在第 9 行利用公式(15.12)直接计算 $w(i, j)$，然后在第 11 行使用此值。如此改动会对渐近时间复杂性有何影响？

OPTIMAL-BST的时间复杂度仍为$\theta(n^3)$

```
OPTIMAL-BST(p,q,n)
1  let e[1..n+1,0..n],w[1..n+1,0..n],and root[1..n,1..n]be new tables
2  for i=1 to n+1
3      e[i,i-1]=q_{i-1}
4      w[i,i-1]=q_{i-1}
5  for l=1 to n
6      for i = 1 to n-l+1
7          j = i+l-1
8          e[i,j]=∞
9          w[i,j]=w[i,j-1]+p_j+q_j
10         for r =i to j
11             t =e[i,r-1]+e[r+1,j]+w[i,j]
12             if t<e[i,j]
13                 e[i,j]=t
14                 root[i,j]=r
15 return e and root
```

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \qquad (15.12)$$

用公式15.12直接计算$w(i,j)$的时间复杂度是$\theta(j-i)$，因为10-14行循环的复杂度也是$\theta(j-i)$，它不会影响OPTIMAL-BST的运行时间复杂度，仍为$\theta(n^3)$

## 16.1-2

**假定我们不再一直选择最早结束的活动，而是选择最晚开始的活动，前提仍然是与之前选出的所有活动均兼容。描述如何利用这一方法设计贪心算法，并证明算法会产生最优解。**

定义所有的活动全集为集合$A = a_0, a_1, ..., a_n$,并假设集合中的每个活动，按照开始时间降序排列。

定义$S(i,j)$为这样一个集合的集合：$S(i,j)$中的每个集合描述了从时刻 i 开始到时刻 j 结束，可安排的兼容活动的最多活动。

显然$S(i,j)$可能有多个元素，每个元素都是一互相等价的，即他们可安排的活动的数量是相同的。
数学表述为：

$$S(i,j) = a_1, a_2..a_k, a_2, a_3...a_m...$$

为了直观，暂且称每个$S(i,j)$中的元素为元素集合。

设$a_m$是$S(i,j)$中所有元素集合中，最后一个开始的活动,$f(a_m)$表示其开始的时间

$$f(a_m) = max\{f(a_k) : a_k \in S(i,j)\}$$

选择$a_m$后，$S(i,j)$将划分为两部分$S(i,m)$和$S(m,j)$

## 1.显然，子问题$S(m,j)$为空，问题会缩减为$S(i,m)$

假设$S(m,j)$不为空集，那么存在一个活动$a_k$,其开始时间$f(a_k) > f(a_m)$，且$a_k$是$S(m,j)$的元素，这与题设$a_m$是$S(i,j)$中最后一个开始的活动相矛盾。

## 2.活动am在某个S(i,j)等价的子集中存在

设活动$a_k$是所有活动集合A中最后一个开始的活动。那么如果$a_m = a_k$,则说明$a_k$确实在$S(i,j)$中的某个元素集合中出现。

如果$a_m! = a_k$,那么对于包含$a_k$的那个元素集合，假设为$S_k = ......a_k$

因为$f(a_m) > f(a_k)$，即$a_m$的开始时间晚，那么我们显然可以用$a_m$替换$a_k$,这样得到的新集合$S_m = ......a_m$，仍然是兼容的，即是$S(i,j)$的元素子集,与$S_k$等价。

由以上推理，如果我们每次选取最后一个开始，且与之前选中的活动兼容的活动，那么最终一定构成某个最佳解中。
这个贪心算法是成立的。

## 16.1-5

考虑活动选择问题的一个变形：每个活动 $a_i$ 除了开始和结束时间外，还有一个值 $v_i$。目标不再是求规模最大的兼容活动子集，而是求值之和最大的兼容活动子集。也就是说，选择一个兼容活动子集 $A$，使得 $\sum_{a_k \in A} v_k$ 最大化。设计一个多项式时间的算法求解此问题。

Let **dp[i]** be the maximum total value before time i,

$$dp[i] = \max(dp[i-1], \max_{f_j \leq i} dp[s_j] + v_j)$$

```
def activity_selection(s, f, v):
    dp = {}
    n = len(s)
    last = None
    for i in sorted(list(set(s + f))):
        if last is None:
            dp[i] = 0
        else:
            dp[i] = last
            for j in range(n):
                if f[j] <= i:
                    dp[i] = max(dp[i], dp[s[j]] + v[j])
        last = dp[i]
    return last
```

算法运行时间为：

$$\Theta(n^2)$$