# 数据结构与算法 II 作业 (11.3)

中国人民大学　信息学院　崔冠宇　2018202147

**T15.5-4**　Knuth 已经证明，对所有 $1 \le i < j \le n$，存在最优二叉搜索树，其根满足 $root[i, j-1] \le root[i, j] \le root[i+1, j]$。利用这一特性修改算法 OPTIMAL-BST，使得运行时间减少为 $\Theta(n^2)$。

**解:** 先给出书上的 OPTIMAL-BST 的伪代码:

```
1  OPTIMAL-BST(p, q, n)
2      e[1..n + 1, 0..n], w[1..n + 1, 0..n], root[1..n, 1..n]
3      for i = 1 to n + 1:
4          e[i, i - 1] = q[i - 1]
5          w[i, i - 1] = q[i - 1]
6      for l = 1 to n:
7          for i = 1 to n - l + 1:
8              j = i + l - 1
9              e[i, j] = +INFINITY
10             w[i, j] = w[i, j - 1] + p[j] + q[j]
11             for r = i to j:
12                 t = e[i, r - 1] + e[r + 1, j] + w[i, j]
13                 if t < e[i, j]:
14                     e[i, j] = t
15                     root[i, j] = r
16     return e and root
```

如果认为 4 ∼ 5、12 ∼ 15 行的运行时间为常数，则这种平凡的 OPTIMAL-BST 算法的总运行时间为

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n+1} \Theta(1) + \sum_{l=1}^{n} \sum_{i=1}^{n-l+1} \sum_{r=i}^{i+l-1} \Theta(1) \\
&= \Theta(1)(n+1) + \Theta(1) \sum_{l=1}^{n} \sum_{i=1}^{n-l+1} l \\
&= \Theta(1)(n+1) + \Theta(1) \sum_{l=1}^{n} (n-l+1)l \\
&= \Theta(1)(n+1) + \frac{1}{6}\Theta(1)n(n+1)(n+2) \\
&= \Theta(n^3)
\end{aligned}
$$

Knuth 在他的论文 *Optimum binary search trees* 中证明了 $root[i, j-1] \le root[i, j] \le root[i+1, j]$（见 [1]），于是算法第 11 行可改为

```
11     for r = root[i, j - 1] to root[i + 1, j]
```

即:

Cui Guanyu

```
OPTIMAL-BST(p, q, n):
    e[1..n + 1, 0..n], w[1..n + 1, 0..n], root[1..n, 1..n]
    for i = 1 to n + 1:
        e[i, i - 1] = q[i - 1]
        w[i, i - 1] = q[i - 1]
    for l = 1 to n:
        for i = 1 to n - l + 1:
            j = i + l - 1
            e[i, j] = +INFINITY
            w[i, j] = w[i, j - 1] + p[j] + q[j]
            // By D. E. Knuth
            for r = root[i, j - 1] to root[i + 1, j]:
                t = e[i, r - 1] + e[r + 1, j] + w[i, j]
                if t < e[i, j]:
                    e[i, j] = t
                    root[i, j] = r
    return e and root
```

下面计算改进后的时间复杂度。

为简便起见，记 $R(i,j) = root[i,j]$。此时算法运行时间为（注意到 $|R(i_1,j_1) - R(i_2,j_2)| \le n$）：

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n+1} \Theta(1) + \sum_{l=1}^{n} \sum_{i=1}^{n-l+1} \sum_{r=R(i,j-1)}^{R(i+1,j)} \Theta(1) \\
&= \Theta(1)(n+1) + \Theta(1) \sum_{l=1}^{n} \sum_{i=1}^{n-l+1} (R(i+1,i+l-1) - R(i,i+l-2) + 1) \\
&= \Theta(1)(n+1) + \Theta(1) \sum_{l=1}^{n} \left[ (n-l+1) + \sum_{i=1}^{n-l+1} (R(i+1,i+l-1) - R(i,i+l-2)) \right] \\
&= \Theta(1)(n+1) + \Theta(1) \sum_{l=1}^{n} \left[ (n-l+1) + \sum_{i=1}^{n-l+1} R(i+1,i+l-1) - \sum_{i=1}^{n-l+1} R(i,i+l-2) \right] \\
&= \Theta(1)(n+1) + \Theta(1) \sum_{l=1}^{n} \left[ (n-l+1) + \sum_{i=1}^{n-l+1} R(i+1,i+l-1) - \sum_{i=0}^{n-l} R(i+1,i+l-1) \right] \\
&= \Theta(1)(n+1) + \Theta(1) \sum_{l=1}^{n} [(n-l+1) + R(n-l+2,n) - R(1,l-1)] \\
&= \Theta(1)(n+1) + \Theta(1) \left[ \frac{n^2+n}{2} + \sum_{l=1}^{n} (R(n-l+2,n) - R(1,l-1)) \right] \\
&= \Theta(1)(n+1) + \Theta(1) \left[ \frac{n^2+n}{2} + \sum_{l=1}^{n} O(n) \right] \\
&= \Theta(n^2)
\end{aligned}
$$

Cui Guanyu

**T15-11** （库存规划）Rinky Dink 公司是一家制造溜冰场冰面修整设备的公司。这种设备每个月的需求量都在变化，因此公司希望设计一种策略来规划生产，需求是给定的，即它虽然是波动的，但是可预测的。公司希望设计接下来 $n$ 个月的生产计划。对第 $i$ 个月，公司知道需求 $d_i$，即该月能够销售出去的设备的数量。令 $D=\sum_{i=1}^{n} d_i$ 为后 $n$ 个月的总需求。公司雇用的全职员工，可以提供一个月制造 $m$ 台设备的劳动力。如果公司希望一个月内制造多于 $m$ 台设备，可以雇用额外的兼职劳动力，雇用成本为每制造一台机器付出 $c$ 美元。而且，如果月末有设备尚未售出，公司还要付出库存成本。保存 $j$ 台设备的成本可描述为一个函数 $h(j)$，$j=1,2,\ldots,D$，其中对所有 $1\le j\le D$，$h(j)\ge 0$，对 $1\le j\le D-1$，$h(j)\le h(j+1)$。

设计库存规划算法，在满足所有需求的前提下最小化成本。算法运行时间应为 $n$ 和 $D$ 的多项式函数。

**解:** 本题类似背包问题，可以用动态规划的方法解决。

记 $D[i]=\sum_{l=1}^{i} d_i(i=1,2,\cdots,n)$ 为前 $i$ 个月的总需求，$cost[i,j](i=0,1,\cdots,n;j=0,1,\cdots,D)$ [1] 为前 $i$ 个月共生产 $j$ 台设备的最小成本（题目要求解的问题即 $cost[n,D]$），同时记 $p[i,j](i=1,2,\cdots,n;j=0,1,\cdots,D)$ 为第 $i$ 个月共生产 $j$ 台设备的条件下，第 $i$ 个月的生产量。

再记 $COST(i,j,k)$ 为第 $i$ 个月生产 $k$ 台设备的前提下，前 $i$ 个月共生产 $j$ 台设备的最小成本，则有[2]：

$$COST(i,j,k)=\begin{cases} cost[i-1,j-k]+h(j-D[i]), & k\le m \\ cost[i-1,j-k]+h(j-D[i])+c\times(k-m), & k>m \end{cases}$$

根据题目要求，前 $i$ 个月的生产总量 $j\ge D[i]$，所以求解 $COST(i,j,k)$ 时 $D[i]\le j\le D[n]$。再根据 $k$ 的含义，有 $0\le k\le j-D[i]$，故

$$cost[i,j]=\begin{cases} COST(1,j,j), & i=1 \\ \min\limits_{0\le k\le j-D[i]} COST(i,j,k), & i>1 \end{cases}$$

$$p[i,j]=\begin{cases} j, & i=1 \\ \arg\min\limits_{0\le k\le j-D[i]} COST(i,j,k), & i>1 \end{cases}$$

可写出伪代码如下:

```
// 库存规划:
// 参数:
//      d[]: 需求数组
//      m: 每月生产台数
//      c: 附加雇用成本
//      h(): 库存成本函数
// 返回值:
//      p[i][j]: 前 i 个月共生产 j 台设备时,
```

[1] 初始化时，$cost[0,j]=0(j=0,1,2,\cdots,D)$。

[2] $i=1$ 时，应该有 $j=k$。

Cui Guanyu

```
//            第 i 月最优生产量
STOCK-PLAN(d, m, c, h()):
    n = d.size()
    // 前 i 个月的总需求
    D[1...n]
    // 最小成本、生产量
    cost[0...n, 0...D], p[0...n, 0...D]
    // 初始化 D
    D[1] = d[1]
    for i = 2 to n:
        D[i] = D[i - 1] + d[i]
    // 循环填表
    // O(n) 次
    for i = 1 to n:
        // O(D) 次
        for j = D[i] to D[n]:
            // i == 1 单独处理
            if i == 1:
                cost[i, j] = h(j - D[i])
                if j > m:
                    cost[i, j] += c * (j - m)
            // i > 1
            // 找最小 k
            else:
                cost[i, j] = +INFINITY
                // O(D) 次
                for k = 0 to j - D[i]:
                    tmp = cost[i - 1, j - k] + h(j - D[i])
                    if k > m:
                        tmp += c * (k - m)
                    if tmp < cost[i, j]:
                        cost[i, j] = tmp
                        p[i, j] = k
    return p
```

重构解的时候，只需要递归打印前面各月的计划，最后打印本月结果即可。伪代码如下：

```
// 打印解
// 参数:
```

Cui Guanyu

```
//      p[][]：上面代码的生产计划矩阵
//      i，j：月份，总生产量
// 输出：
//      生产计划序列
PrintSolution(p, i, j):
    if i > 0:
        PrintSolution(p, i - 1, j - p[i, j])
        print(p[i, j])
```

**STOCK-PLAN 算法复杂度分析**：

- 时间复杂度：STOCK-PLAN 的主要过程是填 $(n+1)\times(D+1)$ 的矩阵（$cost$、$p$），而填写每一个单元格时，循环求 $k$ 的次数是 $O(D)$ 次，所以算法运行时间 $T(n) = O(nD^2)$ [3]。
- 空间复杂度：STOCK-PLAN 内有 $D[1...n]$、$cost[0...n, 0...D]$ 以及 $p[0...D]$，共 $n + 2(n+1)(D+1)$ 个辅助单元，故空间复杂度为 $S(n) = O(nD)$。

**PrintSolution(p, n, D) 时间复杂度分析**：

PrintSolution(p, i, j) 每调用一次，$i$ 将会减一，当 $i$ 为 0 时，递归结束；在打印原题目的解时，它共会被调用 $n$ 次，运行时间是 $T(n) = O(n)$。

## 参考文献


[1] Knuth, D.E. *Optimum binary search trees. Acta Informatica 1, 14-25 (1971).* `https://doi.org/10.1007/BF00264289`


[3]虽说这个算法的时间复杂度是关于 $n$ 和 $D$ 的多项式，但实际上是伪多项式算法。

Cui Guanyu