

算法48: MATRIX-MULT-DP(DimeList[0..n])

```

1 for low := n - 1 downto 1 do
2   for high := low + 1 to n do
3     if high - low = 1 then
4       bestCost := 0 ;
5       bestLast := -1 ;
6     else
7       bestCost := +∞ ;
8     for k := low + 1 to high - 1 do
9       a := cost[low][k] ;
10      b := cost[k][high] ;
11      c := multCost(DimeList[low], DimeList[k], DimeList[high]) ; /* 最后一次
相乘的代价为  $d_{low} \cdot d_k \cdot d_{high}$  */
12      if a + b + c < bestCost then
13        bestCost := a + b + c ;
14        bestLast := k ;
15      cost[low][high] := bestCost ;
16      last[low][high] := bestLast ;
17 EXTRACT-ORDER() ;
18 return cost[0][n] ;

```

算法49: EXTRACT-ORDER()

```

1 multNext := 0 ;
2 Initialize the queue queMultOrder to store the multiplication order ;
3 EXTRACT(0, n) ;
4 subroutine EXTRACT(low, high)
5 if high - low > 1 then
6   k := last[low][high] ;
7   EXTRACT(low, k) ;
8   EXTRACT(k, high) ;
9   queMultOrder.push(k) ;

```

内部合并
low ————— high

右多遍历。因为 才能保证 low, high, 所以内部计算先做，先压入队列。

13.3 动态规划的关键特征

通过对矩阵链相乘问题的深入分析，我们可以归纳动态规划策略的三个关键要素：

- 重叠子问题。正如 13.1 节所讨论的，减少子问题的重复计算是使用动态规划策略的

算法 48: MATRIX-MULT-DP(DimeList[0..n])

```
1 for low := n - 1 downto 1 do
2   for high := low + 1 to n do
3     if high - low = 1 then
4       bestCost := 0 ;
5       bestLast := -1 ;
6     else
7       bestCost := +∞ ;
8     for k := low + 1 to high - 1 do
9       a := cost[low][k] ;
10      b := cost[k][high] ;
11      c := multCost(DimeList[low], DimeList[k], DimeList[high]) ; /* 最后一次
          相乘的代价为  $d_{low} \cdot d_k \cdot d_{high}$  */
12      if a + b + c < bestCost then
13        bestCost := a + b + c ;
14        bestLast := k ;
15      cost[low][high] := bestCost ;
16      last[low][high] := bestLast ;
17 EXTRACT-ORDER() ;
18 return cost[0][n] ;
```

算法 49: EXTRACT-ORDER()

```
1 multNext := 0 ;
2 Initialize the queue queMultOrder to store the multiplication order ;
3 EXTRACT(0, n) ;
4 subroutine EXTRACT(low, high)
5 if high - low > 1 then
6   k := last[low][high] ;
7   EXTRACT(low, k) ;
8   EXTRACT(k, high) ;
9   queMultOrder.push(k) ;
```

内部分裂
low high

*右多遍历。因为才执行 low, high。
所以内部的计算先做，先压入队列。*
