

1 动态规划的基本步骤

- 描述最优解的结构。
- 递归定义最优解的值。
- 按自底向上的方式计算最优解的值。

2 装配线调度问题

2.1 问题描述

现在有两条装配线，编号为 i ，每一条装配线有 n 个装配站。编号为 j ，所以将装配线 i 的第 j 个装配站表示为 $s_{i,j}$ 。每个装配站的时间都是不同的。我们把在装配站 $s_{i,j}$ 上所需的装配时间记为 $a_{i,j}$ 。零件进入装配线 i 所需要的时间记为 e_i ，离开装配线所需要的时间记为 x_i 。

工厂经理可以将部分完成的零件从一条装配线移到另一条装配线上，把已经通过装配站 $s_{i,j}$ 的零件从装配线 i 移动到另一个装配线的时间为 $t_{i,j}$ 。

现在的问题是，需要确定在装配线 1 和装配线 2 中选择哪些站，使得完成一个零件的时间最短。

2.2 问题解决

2.2.1 描述最优解的结构

对于装配线调度问题，一个问题的最优解包含了子问题的一个最优解，这种性质称为最优子结构。

我来解释一下这种性质。现在我们要找通过装配站 $s_{i,j}$ 的最快路线。零件可能来自装配站 $s_{1,j-1}$ 也可能来自装配站 $s_{2,j-1}$ 。不管是来自哪个装配站，零件通过装配站 $s_{1,j-1}$ 或 $s_{2,j-1}$ 时，都需要保证它是经过最快路线的。这样一来，求解一个问题的最优解之前，我们可以先求出它的子问题的最优解。

这样一来，我们就可以利用子问题的最优解来构造原问题的一个最优解。现在我们先求出通过 $s_{1,j}$ 的最优路线：

- 求出装配站 $s_{1,j-1}$ 的最快路线，加上 $a_{1,j}$ 后得到总时间。
- 求出装配站 $s_{2,j-1}$ 的最快路线，加上 $t_{1,j-1}$ 和 $a_{1,j}$ 得到总时间。
- 对比两种方案的总时间，选择时间较短的路线为最佳路线。

2.2.2 递归定义最优解的值

令 $f_i[j]$ 表示一个零件从起点到装配站 $s_{i,j}$ 的最快时间。令 f^* 表示完成一个零件所需要的最少时间。很容易知道有下列等式关系成立：

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2) \quad (1)$$

通过子问题求解原问题，我们容易有下列等式成立：

$$\begin{aligned} f_1[j] &= \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) \\ f_2[j] &= \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) \end{aligned} \quad (2)$$

而且问题的初始条件也是容易得到的：

$$\begin{aligned} f_1[1] &= e_1 + a_{1,1} \\ f_2[1] &= e_2 + a_{2,1} \end{aligned} \quad (3)$$

2.2.3 按自底向上的方式计算最优解的值

根据上一小节的公式，很容易就能编写出相应的伪代码。

```

1  FASTEST_WAY(a, t, e, x, n)
2      f1[1] = e1 + a1[1]
3      f2[1] = e2 + a2[1]
4      for j = 2 to n
5          // 求f1[j]的最优路线
6          // l1[j]用于存放上一个装配站
7          if f1[j-1]+a1[j] <= f2[j-1]+t2[j-1]+a1[j]
8              f1[j] = f1[j-1] + a1[j]
9              l1[j] = 1
10         else
11             f1[j] = f2[j-1] + t2[j-1] + a1[j]
12             l1[j] = 2
13         // 求f2[j]的最优路线
14         // l2[j]用于存放上一个装配站
15         if f2[j-1]+a2[j] <= f1[j-1]+t1[j-1]+a2[j]
16             f2[j] = f2[j-1] + a2[j]
17             l2[j] = 1
18         else
19             f2[j] = f1[j-1] + t1[j-1] + a2[j]
20             l2[j] = 2
21     if f1[n]+x1 <= f2[n]+x2
22         f = f1[n] + x1
23         l = 1
24     else
25         f = f2[n] + x2
26         l = 2
27     // 之后可以通过l、l1与l2中存放的路线构造出最快路线

```