动态规划的深入讨论

东北育才学校　　李刚

【关键字】　动态规划、状态

【 摘要】

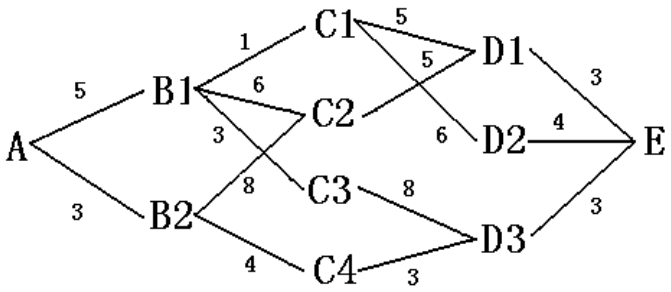
本文讨论了一种解决问题十分有效的技术——“动态规划”。它较高的解题效率一直受到很大的关注。本文首先对“动态规划”的理论基础进行了讨论。给出了一个用“动态规划”可以解决的问题的两个先决条件：“最优子结构”与“无后效性”。接着，讨论了在实际应用中的两个比较常见的问题：“动态规划”中状态的选定与存储。再通过以上问题的讨论，引出了“动态规划”的基本思维方法：“不做已经做过的工作”以及“动态规划”技术在解决问题中速度惊人的原因——“解决了查看中的冗余，达到了速度的极限”。最后，阐述了解决“动态规划”问题的一般步骤，即“思考，计划，应用”。

【正文】

一　. 引　论

在信息学竞赛中，特别是最近几年，“动态规划”作为一种解题工具，经常被提及。其应用范围愈来愈广，应用程度也愈来愈深。那么，“动态规划”究竟与其它的算法有什么差别？它有什么具体的应用价值呢？本文将对此进行讨论。

我们先通过一个具体问题认识一下“动态规划”。



（图 1）

〖**例 1**〗：图 1 中给出了一个地图，地图中每个顶点代表一个城市，两个城市间的连线代表道路，连线上的数值代表道路的长度。现在，我们想从城市 A 到达城市 E，怎样走路程最短，最短路程的长度是多少？

假设：

Dis[X]为城市 X 到 E 的最短路线的长度；（X 表示任意一个城市）

Map[I,J]表示 I，J 两个城市间的距离，若 Map[I，J]=0，则两个城市不连通。

这个问题我们可以用搜索法来做，程序很容易写出来：

```
Var
    Se:未访问的城市集合；

      Function  Long(Who:当前访问城市):Integer;  ：求当前访问城市与城市 E 的最短距离。
            Begin
                If Who=E Then  Search:=0
        Else
                      Begin
                            Min:=Maxint;
                              For I 取遍所有城市  Do
              If （Map[Who,I]>0）  And  (I In Se) Then
                Begin
                Se:=Se-[I];
                  J:=Map[Who,I]+Long(I);
                Se:=Se+[I];
                  If  J<Min Then Min:=J;
                End;
           Long:=Min;
                          End;
      End;
        Begin
```

Se:=除 A 外所有城市的集合；
Dis[A]:=Long(A);
End.

这个程序的效率如何呢？我们可以看到，每次除了已经访问过的城市外，其他城市都要访问，所以时间复杂度为$O(N!)$，这是一个“指数级”的算法，那么，还有没有更好的算法呢？

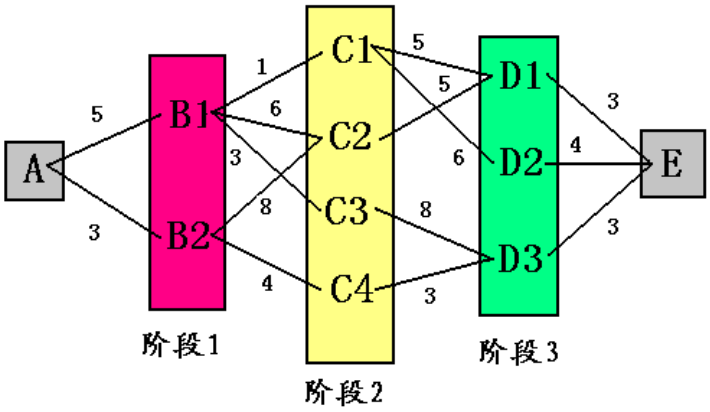
首先，我们来观察一下这个算法。在求从 B1 到 E 的最短路径的时候，先求出从 C2 到 E 的最短路径；而在求从 B2 到 E 的最短路径的时候，又求了一遍从 C2 到 E 的最短路径。也就是说，从 C2 到 E 的最短路径我们求了两遍。同样可以发现，在求从 C1、C2 到 E 的最短路径的过程中，从 D1 到 E 的最短路径也被求了两遍。而在整个程序中，从 D1 到 E 的最短路径被求了四遍，**这是多么大的一个浪费啊！**如果在求解的过程中，同时将求得的最短路径的距离“记录在案”，随时调用，**那会是多么的方便啊！**

于是，一个新的思路诞生了，即：**由后往前**依次推出每个 Dis 值，直到推出 Dis[A]为止。这个思路的确很好，但等等，究竟什么是“由后往前”呢？

所谓“后”、“前”是我们自己为城市编的序号，当两个城市 I，J 的前后顺序定为 I“前”J“后”时，必须满足这个条件：

或者 I，J 不连通，或者 Dis[I]+Map[I,J]≥Dis[J]。

因为如果 I，J 连通且 Dis[I]+Map[I,J]<Dis[J]，则说明 Dis[J]存在更优的情况，可 J 位于 I 后，就不可能推出此情况，会影响最后的解。那么，我们如何划分先后次序呢？



(图 2)

如图 2 所示。我用不同颜色给城市分阶段，可以用**阶段**表示每个城市的次序，因为**阶段**的划分有如下性质：

1：阶段 I 的取值只与阶段 I+1 有关，阶段 I 的取值只对阶段 I-1 的取值产生影响；

2：每个阶段的顺序是确定的，不可以调换任两个阶段顺序；

通过这两个性质，可以推出阶段作为“前”、“后”顺序满足刚才提出的条件，所以我们可以用**阶段**作为每个城市的次序，然后从阶段 3 倒推至阶段 1，再推出 Dis[A]。

公式：Dis[X]=Min{ Dis[Y]：Y 是下一个阶段中与 X 相连通的城市 }

注：可以把 E 看成第 4 个阶段，A 看成第 0 个阶段。

程序：

```
Dis[E]=0
 For  X=阶段 3 的    每个城市 Downto 阶段 0 的每个城市  Do
      Begin
Dis[X]:=Maxint;
For   Y=阶段 X 的下一个阶段中的每个城市  Do
 If  Dis[Y]+Map[X,Y]<Dis[X] Then
       Dis[X]:=Dis[Y]+Map[X,Y];
End。
```

这个程序的时间复杂为 $O(N^2)$，比上一个程序的复杂度 $O(N!)$ 要小得多。第二个算法就是“动态规划”算法。

二 ． 动态规划的理论基础

通过上面的例子，我们对“动态规划”有了一个初步认识，它所处理的问题是一个“多阶段决策问题”。我们现在对一些概念进行具体定义：

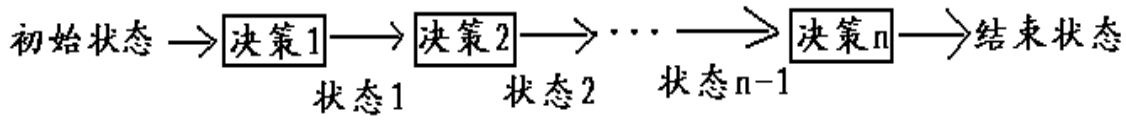
状态（State）：它表示事物的性质，是描述“动态规划”中的“单元”的量。如例 1 中的城市 A，B1，D2，E 都为单个的状态。

阶段（Stage）：阶段是一些性质相近，可以**同时处理**的状态集合。通常，一个问题可以由处理的先后次序划分为几个阶段。如例 1 中的问题由三个阶段组成，其中阶段 1 包含状态 B1，B2。实际上，阶段只是标识那些处理方法相同、处理顺序无关的状态。一个阶段既可以包含多个状态，也可以只包含一个状态。其关系很类似分子与原子的关系。

状态转移方程：是前一个阶段的状态转移到后一个的状态的演变规律，是关于两个相邻阶段状态的方程，是“动态规划”的中心。

决策（Decision）：每个阶段做出的某种选择性的行动。它是我们程序所需要完成的选择。

动态规划所处理的问题是一个“**多阶段决策问题**”，一般由初始状态开始，通过对中间阶段决策的选择，达到结束状态。这些决策形成了一个决策序列，同时确定了完成整个过程的一条活动路线（通常是求最优的活动路线）。如图 3 所示。



（图 3）

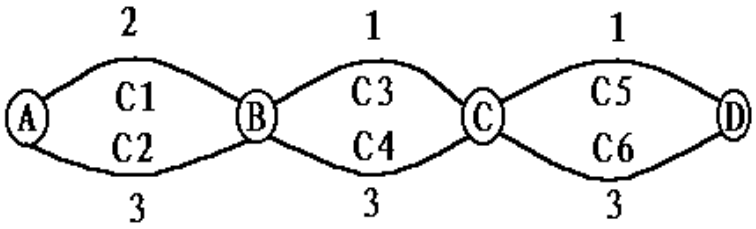
总体上来说，一个“动态规划”算法应该包含上面提到的几种基本关系与属性。

那么，什么样的“决策问题”才可以划分阶段，来用“动态规划”求解呢？

我们说一个“决策问题”只有具有以下性质时，才可以考虑划分阶段，用“动态规划”算法：

一：最优子结构

即一个问题的最优解只取决于其子问题的最优解，也就是说，非最优解对问题的求解没有影响。在例 1 中，如果我们想求 Dis[B1]的最小值，我们只需要知道 C1，C2，C3 的最小值，而不用考虑其他的路径，因为其他的非最优路径肯定对 Dis[B1]的结果没有影响。我们再来看一个问题：



（图 4）

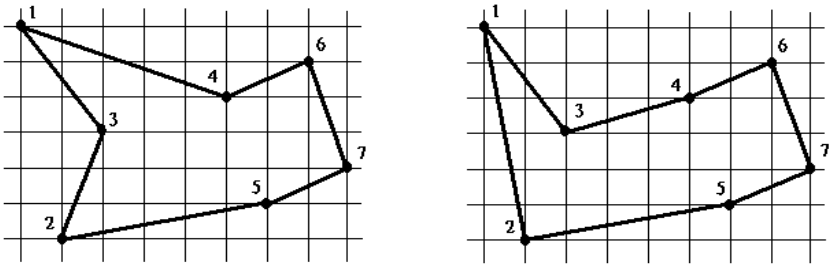
〖**例 2**〗有 4 个点，分别是 A、B、C、D，如图 4 所示，相邻两点用两条连线 C2k,C2k-1(1≤k≤3)表示两条通行的道路。连线上方的数字表示道路的长度。我们定义从 A 到 D 的所有路径中，长度除以 4 所得余数最小的路径为最优路径。求一条最优路径。

在这个题目中，我们如果还按照刚才的方法来求解就会发生错误。例如，按照例 1 的思维，A 最优取值可以由 B 的最优取值来确定，而 B 的最优取值为 0，所以 A 的最优值为 2，而实际上，路径 C1—C3—C5 可得最优值为 0，所以，B 的最优路径并不是 A 最优路径的子路径，也就是说，A 的最优取值不是由 B 的最优取值决定的，其不具有最优子结构。

由此可见，并不是所有的“决策问题”都可以用“动态规划”来解决。所以，**只有当一个问题呈现出最优子结构时，“动态规划”才可能是一个合适的侯选方法。**

二：无后效性

即一个问题被划分阶段后，阶段 I 中的状态只能由阶段 I+1 中的状态通过状态转移方程得来，与其他状态没有关系，特别是与未发生的状态没有关系，这就是无后效性。实际上，如果我们把这个问题中的状态定义成图中的顶点，两个状态之间的转移定义为边，转移过程中的权值增量定义为边的权值，则这个问题实际上就是在一个**“有向无环加权图”**中寻找两个顶点路径的问题。因为无后效性，所以没有环路（否则，无论如何划分阶段，都无法出现后效性）。



(a)　　(b)

(图 5)

排序的顺序

我

〖**例 3**〗：欧几里德货郎担问题是对平面给定的 n 个点确定一条连结各点的、闭合的游历路线问题。图 5(a)给出了七个点问题的解。Bitonic 旅行路线问题是欧几里德货郎担问题的简化，这种旅行路线先从最左边开始，严格地由左至右到最右边的点，然后再严格地由右至左到出发点，求路程最短的路径长度。图 5（b）给出了七个点问题的解。

这两个问题看起来很相似。但实质上是不同的。为了方便讨论，我将每个顶点标记了号码。由于必然经过最右边的顶点 7，所以一条路(P1—P2)可以看成两条路（P1—7）与(P2—7)的结合。所以，这个题目的状态可以用两条道路结合的形式表示。我们可以把这些状态中，两条路中起始顶点相同的状态归于一个阶段，设为阶段[P1,P2]。

那么，对于 Bitonic 旅行路线问题来说，阶段[P1,P2]如果可以由阶段[Q1,Q2]推出，则必须满足的条件就是:P1<Q1 或 P2<Q2。例如，阶段[3,4]中的道路可以由阶段[3,5]中的道路加一条边 4—5 得出，而阶段[3,5]的状态却无法由阶段[3,4]中的状态得出，因为 Bitonic 旅行路线要求必须严格地由左到右来旅行。所以如果我们已经知道了阶段[3,4]中的状态，则阶段[3,5]中的状态必然已知。因此我们可以说，Bitonic 问题满足“无后效性原则”，可以用“动态规划”算法来解决，其程序可以参见 Pro_3_1.Pas。

（图 6）

对于欧几里德货郎担问题,阶段与阶段之间没有什么必然的“顺序”。如道路{3—2—5—7,4—6—7}属于阶段[3,4]，可由属于阶段[2,4]的道路{2—5—7，4—6—7}推出；而道路{2—3—6—7，4—5—7}属于阶段[2,4]，可由属于阶段[3,4]的道路{3—6—7，4—5—7}推出。如果以顶点表示阶段，推出关系表示边，那么，阶段[3,4]与阶段[2,4]对应的关系就如图 6 所示。我们可以很清晰地看出，这两个阶段的关系是“有后效性”的。因为这个图中存在“环路”。对于这个问题是不能像上一个问题那样来解决的，事实上，这个问题是一个 NP 完全问题，其解决的时间复杂度很可能是指数级的。

所以，对于一个问题能否用“动态规划”来解决的一个十分关键的判断条件就是“它是否有后效性”。而我们在判断这个问题是否有“后效性”时，一个很有效的方法就是将这个问题的阶段作为顶点，阶段与阶段之间的关系看作有向边，判断这个有向图是否为“有向无环图”，亦即这个图是否可以进行“拓扑排序”。

通过上面的说明，我们可以总结出一些解决**“动态规划”问题**的基本方法与步骤：

1：确定问题的研究对象，即确定状态。

2：划分阶段，确定阶段之间的状态转移方程。

3：考察此问题现在可否用“动态规划”来解决：

①：考察此问题是否具有“最优子结构”。

②：考察此问题是否为“无后效性”。

4：如果发现此问题目前不能用“动态规划”来解决，则应该调整相应的定义与划分，以达到可以用“动态规划”来解决。

以上只是一般情况下的“动态规划”思维过程。一些较为简单的问题可以“按部就班”来操作，但大多数的“动态规划”问题，特别是作为信息学竞赛中的“动态规划”问题，考察的知识是多方面的，应用的技巧是灵活多变的。下面，对“动态规划”在应用中的一些重点、难点进行讨论。

三　．动态规划的实际应用

我们衡量一个算法的标准，无外乎时间、空间两项指标。“动态规划”算法的时间大多数为“多项式级”的，比起同样解决这个问题的搜索算法“指数级”的时间来说，“动态规划”的时间需要是很少的，所以我们在实际应用中，很少考虑“动态规划”算法的时间问题，而最经常考虑的是空间问题：即状态的选定与存储。

1：状态的选定：

对于一个“动态规划”算法来说，阶段的划分显得不很重要，因为阶段只是一些可以等同处理的状态的集合，我们尽可以把单个的状态定义为一个阶段。所以，状态的选定对整个问题的处理起了决定性的作用。我们选定的状态必须满足如下两点：

1：状态必须完全描述出事物的性质，两个不同事物的状态是不同的；

2：必须存在状态与状态之间的“转移方程”。以便我们可以由“初始问题”对应的状态逐渐转化为“终结问题”对应的状态。

状态是描述事物性质的量，所以我们应该以这个标准，根据题目中的具体要求来具体分析，我们来看下面一个例子：

〖**例 4**〗有一个奶牛运输公司，需在 7 个农场 A,B,C,D,E,F,G 之间运输奶牛，从 A 出发，完成任务后要回到 A。运输车每次只能运送一头奶牛，每个运输任务是由一对字母给出的，分别表示奶牛从哪个农场被运向另外一个农场。任务总数为 N(1≦N≦12)。已知这些农场之间的距离，求出一条完成所有任务的最短路线。

我们先分析这个问题的解决方法：我们的目的是完成一些任务，由于运输车每次只能运送一头奶牛，所以我们只能一个接一个地处理这些任务。由于题目求的是一条完成这些任务的最短路线，所以在完成任务的过程中，我们走的

路线都是最短路线，也就是说，一旦我们确定了完成任务的顺序 P1—P2—……—PN，那么这条路线的最短时间也就确定了。我们的目的就是确定这 N 个任务的完成顺序。所以，这个问题的研究对象就是某些任务的集合 S。那么，状态该如何定义呢？既然研究对象是集合，所以很自然地就想起了描述集合中元素关系的方法。我们可以把状态定义为一个数组（$t_1,t_2,\ldots,t_N$），每一个 t_i 或者为 0，表示任务 i 不在当前集合中；或者为 1，表示任务 i 在当前集合中。但如果这样定义状态，我们会发现，无法写出状态转移方程，因为涉及到前后两个任务的“接口”的最短路线长度。所以我们还需要一个量 x 来限定当前这个任务集合中首先要执行任务，则一个状态定义为（$x,t_1,t_2,\ldots,t_N$），其中 t_x 必为 1。那么，一个状态对应的权值就是完成这个状态描述的任务集合中任务的最短路线长度。则状态转移方程为：

$$(x,t_1,t_2,\ldots,t_N)=Min\{(y,p_1,p_2,\ldots\ldots,p_n)+Dis(x,y)\}$$

其中，y 是不等于 x 且 $t_y=1$ 的值，若 i≠x，则 $p_i=t_i$；$p_x=0$。假设 Dis(x,y)为由任务 x 的终点城市到任务 y 的起点城市的最短距离。这个转移方程的初始条件为：

（$x,t_1,t_2,\ldots,t_N$）={任务 x 的终点城市到 A 的距离}，其中 $t_x=1$，其余的 t_i 均为 0。

在实际处理中，用这样一个最大可能为 12 维的数组显然是不方便的，既然每一位只能是 0 或 1，所以我就把（$t_1,t_2,\ldots,t_N$）看作一个十进制数的二进制表示，于是，状态数组为一个二维数组，具体程序见附录中的 Pro_4_1.Pas。

从这个问题中我们可以看出，一个“动态规划”问题的状态选定实质上就是选择描述这个问题中事物的最贴切，最简洁的方法。状态的选定不是一蹴而就的，而是一个在思考过程中逐步调整，逐步完善的过程。

2：状态的存储

当状态选定后，我们面对的问题就是如何存储状态了。从理论上讲，每一个状态都应该存储两个值，一个是此状态的最优的权值，一个是决策标识值。但实际中，我们面对的“动态规划”问题很多都是状态数目十分庞大，这就要求我们在存储上必须做一定的优化才可以实现这个算法的程序化。主要的优化就是：**舍弃一切不必要的存储量。**

在一些问题中，题目给出了只在某一范围内的关系，所以我们只需存储这一范围内的状态即可。下面来看这个问题：

〖**例 5**〗一个生物体的结构可以用“基元”的序列表示，一个“基元”用一些英文字符串表示。对于一个基元集合 P，可以将字符串 S 作为 N 个基元 P1,P2…PN 的依次连接而成。问题是给定一个字符串 S 和一个基元集合 P，使 S 的前缀可由 P 中的基元组成。求这个前缀的最大长度。基元的长度最大为 20，字符串的长度最大为 500,000。例如基元集合为{A,AB,BBC,CA,BA}，字符串为 ABABACABAABCB，则最大长度为 11，具体组成见图 7。图中不同的“基元”以不同的颜色标出。

ABABACABAABCB

（图 7）

这个问题的状态十分容易确定，字符串中的每一位的性质只有两种，即：包含这一位字符的字符串前缀是否可以由基元序列构成。那么，我们就可以设一个数组：Could：Array[1..Max] Of Boolean 作为表示字符串中每一位的状态。那么，状态转移方程为：

Could[K]:=Could[K] Or (Could[K-W] And (S[K-W+1...K]=Ji[I]))

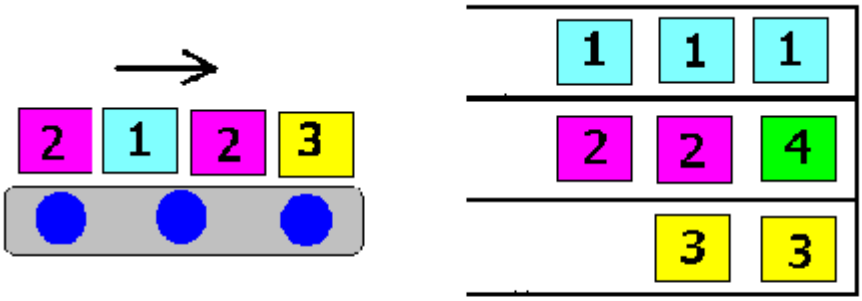
其中，Could[k]的初始值均为 False，Ji[I]表示第 I 个基元对应的字符串，其长度为 W，S[K-W+1...K]为题中给定的字符串从 K-W+1 位到 K 位的字符组成的字符串。

写到这里，这个问题似乎已经解决了，我们只需求出使 Could[I]为 True 最大的 I 即可。但只要回过头看一看问题就会发现，字符串最长为 500,000，根本无法存储，该如何是好呢？如果再仔细观察一遍转移方程，就会发现：Could[I]只与 Could[I-W]发生关系，而 W 为基元的长度，基元的长度最大为 20，也就是说，我们如果想推出 Could[I]，最多只需要知道 Could[I-20]至 Could[I-1]的值即可，其它的 Could 值我们可以不去管它。这样，我们需要记录的 Could 值一下子就由 500000 减少到 20，这个突变是巨大的。具体程序见 Pro_5.Pas。

这个问题为什么只记录很少的状态就可以呢？这是因为它特殊的转移方程与求解顺序。求解顺序是按照字符串中位置的先后而定的，而转移方程中，当前状态只与比它先求解的那 20 个状态发生关系，所以这个问题可以如此解决。

但对于大多数问题来说，这样“美丽”的“结构”不一定具备，那么我们在解决这样的问题时，只有“**处心积虑**”地削减存储量。我们来看一个例子。

〖**例 6**〗某公司运进一批箱子，总数为 **N（1≤N≤1000）** 由“传送带”依次运入，然后在仓库内至多排成 **P（1≤P≤4）** 列，（如图 8 所示）。



（图 8）

现已知运来的箱子最多为 M（**1≤M≤20**）种，想把同一种类的箱子尽量排在一起，以便美观。“美观程度”**T** 定义为：**T=Σ（每列依次看到的不同种类数）**；所谓“依次看到的不同种类数”即为：如果某一列中第 K 个箱子与第 K-1 个箱子种类不同，则“美观程度”的值加 1。求一种调动安排，使各列的“美观程度”值的和最小。

这个问题的思路如下：第 K 个箱子需要选择一列来放，如果它放在与其种类不同的某个箱子上，则它会使总的“美观程度”值加 1。也就是说，第 K 个箱子的摆放只与当前 P 列的顶端情况有关。又如果想求这个箱子摆放后的最小值，自然需要这个前（K-1）个箱子在此顶端情况时的最小值。这个多阶段决策问题同时满足最优化原理与无后效性原则，所以这个问题可以用“动态规划”的方法来解决。则状态的选定很明显，为（k，P1，P2，P3，P4），K 是当

前即将放入的箱子的编号，P1—P4 放入第 K 个箱子后各列的“顶端情况”，即放入第 K 个箱子后顶端是什么种类的箱子（假设当前有 4 列）。则阶段划分也很明显，以箱子由传送带运来的先后顺序划分阶段。那么，状态转移方程就是：

（K,P1,P2,P3,P4）=Min(K-1,Q1,Q2,Q3,Q4) +Dis}

其中，Q1—Q4 为未放入第 K 个箱子前各列的情况，Dis 为 0 或 1，取决与第 K 个箱子与其列中上一个箱子的种类有无差别。

写到这里，“动态规划”的算法各个步骤已经清楚了，似乎可以直接写程序了。但对于本题来说，存储大量的信息，是我们所不得不面对的一个问题。Pascal 语言所允许的最大可用空间为 640KB，在保护模式下，最多也不过 1000 多 KB。一个“动态规划”程序所需存储的一般有两个值，一是决策信息，二是状态的价值。对于本题，若按普通的方法，最多时为 $1000\times20^3\times(1+2)$字节,根本无法存储。必须对存储做一定的优化，舍弃一切不必要的量：

1．只记录每步的“决策信息”，不记录每步的状态的价值，这样可省下 $\frac{2}{3}$ 的空间；

2．对每步的状态来说，所对应的 P 列顶端不含有两个同一种的箱子；

3．除本次要选的列以外，其它（P-1）列顶端的排列对结果没影响，所以存储的状态应为另外（P-1）列的组合，而非排列。

这样，总的存储量可由 24000KB 减少到 1200KB，缩小了约 20 倍，在 Pascal 保护模式下完全可以操作。具体的程序见 Pro_6.Pas。

通过以上两个问题，我们可以看出，“动态规划”在实际应用中，速度几乎“不成问题”，但存储状态的“环节”却必须思考再思考，优化再优化。近几年的试题中，大多数“动态规划”问题的难点与重点就是“动态规划”中状态的选定与存储。这也是“动态规划”在实际应用中的重中之重。

四．动态规划的深入思考

以上讨论了“动态规划”的理论基础与在实际应用中可能遇到的问题。那么，“动态规划”究竟好在哪里？我们为什么要用“动态规划”呢？从上面的一些例子可以看出，“动态规划”这种方法最大优点就是节约了时间。这个“巨大的优点”可以从例 1 的解决中看出。在实际中，搜索算法与“动态规划”算法的时间差异是巨大的，表 1 是例 1 的两个算法对应程序的执行时间。

测试数据 算法	1	2	3	4	5
搜索算法	1.04s	1.42s	1.05s	7.42s	22.92s
动态规划算法	<0.01s	<0.01s	0.06s	0.02s	0.01s

（表 1）

从这张执行时间表可以看出，“动态规划”算法与搜索算法在实际应用中的运行时间差异是巨大的。从理论上讲，搜索算法的时间复杂度为$O(N!)$，“动态规划”算法的时间复杂度是$O(N^2)$，**这两个时间根本就不是一个数量级的。**指数级的时间增长是十分可怕的，远远大于多项式级的算法。所以，我们经常称多项式级的算法为一个“较好”的算法，称指数级的算法是一个“较差”的算法。

“动态规划”程序实质上是一种**“以空间代价来换取时间代价”**的技术，它在实现的过程中，不得不存储产生过程中的各种状态。所以，它的空间复杂度要大于其它的算法。以 Bitonic 旅行路线问题为例，这个问题也可以用搜索算法来解决(具体见程序 Pro_3_2.Pas)。“动态规划”的时间复杂度为$O(N^2)$，搜索算法的时间复杂度为$O(N!)$，但从空间复杂度来看，“动态规划”算法为$O(N^2)$，而搜索算法为 *O(N)*。搜索算法反而优于“动态规划”算法。那么，我们为何选择“动态规划”算法来解决例 3 呢？因为“动态规划”算法在空间上可以承受（事实上是很可以承受），而搜索算法在时间上却十分巨大，所以我们**“舍空间而取时间”**。

这个问题在例 4（即奶牛运输问题）中更加明显。例 4 问题实质上是一个求最优的“哈密尔顿（Hamilton）回路”问题。这个问题是一个 NP 问题，人们到目前为止还没有找到这个问题的“多项式级”解法。所以，我们在进行“动态规划”算法时不得不记录每一种状态后再进行递推。我们在进行递推时好像是在进行着一个“多项式级”的算法。实质上，记录的状态数目已经是“指数级”的了。我们用的“动态规划”算法从理论上讲，与普通搜索算法的时间复杂度差不多，而空间复杂度却远远高于普通的搜索算法（“动态规划”算法的空间复杂度为$O(2^N)$，搜索算法的空间复杂度为 *O(N)*）。那么，为什么用“动态规划”方法解决此问题要比搜索方法快呢？（快多少可参见下边的表 2）多用的那部分空间究竟干什么了？这两个问题的答案就是我们用“动态规划”技术的根本性目的：**“解决冗余”**。

我们在用搜索算法的过程中，做了一些已经做过的工作。以例 4 为例：假设目前待解决的任务集合为[1,2,3,4]，我们如果选定头两个解决的任务为 1,2，那么还需要从后两个任务 3,4 中选择解决顺序。而如果选定前两个解决的任务为 2,1（1,2 与 2,1 不同，解决的次序不同），那么又得重新解决一遍工作“从后两个任务 3,4 中选择解决顺序”，**这是多么大的一个浪费呀！**这句话在文章开头分析例 1 时就提过了一遍，这里“旧话重提”意义在于揭示“动态规划”这种技术的基本思维：**“不做已经做过的工作。”**

对于这些做过的工作，我们已经知道了它们的结果，在第二次遇到这个工作时，完全没有必要再做一遍，只需把上次做的结果拿过来应用即可。这种“懒惰”的想法正是人类千百年来一直研究问题的根本态度。我们平时见到的数学、物理等自然学科中的一些定理、定律，从本质上来讲，这些不都是“工作记录”吗？我们学习这些定理、定律的目的，一方面是学习思考方法，另一方面不就是为了下一回 碰到这些问题时直接解决吗？所以说，“动态规划”技术实质上就是**这种想法**在计算机领域一种延续，一种“自然流露”。

解决问题的目的是下一次不解决问题。用句中国的古话，我们最后希望的是：在解决问题时能够“以不变应万变”，因为“万变不离其宗”，而这个“宗”问题，**我们曾经解决过，这次直接“读入数据、读入结果”即可。**

明白了这个道理，那么我们解决问题的能力就会更上一个台阶了。对于例4，我们尽可以用搜索算法，完全可以同样再用那么一个记录所有状态的数组辅助搜索，即一旦发现这个搜索任务已经搜索过了，则直接读入最优值即可。这样，冗余没有了，这个算法的时间自然会和“动态规划”算法的时间一样“优秀”了（甚至更加“优秀”，这是搜索中加进了“半路剪枝”的缘故）。具体见表2。

测试数据 算法	1	2	3	4	5
动态规划算法	**0.11S**	**0.09s**	**0.22s**	**0.22s**	**0.60s**
搜索算法	**1.58S**	**1.90s**	**5.27s**	**6.58s**	**7.09s**
改进后的 搜索算法	**0.06S**	**0.01s**	**0.05s**	**0.07s**	**0.16s**

到

（表 2）

从这里，我们可以发现：**搜索慢，慢在冗余，减掉冗余后，我们的工作就简单了许多。**我们先不考虑可以“半路剪枝”的因素，即搜索过程中所有的工作都有可能产生我们需要的结果。那么这些工作就是“**不得不做**”的，是每一种算法“必做的”，“不可缺少的”，也是解决这个总问题的“工作极限”。既然没有多做任何工作，那么我们的解决这个问题的速度又怎么可能不快呢？

这就是“动态规划”速度惊人的根本原因：**除非另换思路，否则你将做的工作不会少于这个极限工作量——“不重复地考察每一种可能状态”。**

五．总结

通过以上的讨论，我们知道，“动态规划”是一种效率很高的技术。**其执行时间很少的原因就是它把已经做过的工作结果存储起来。**大多数的“

多阶段决策问题”几乎都存在“动态规划”算法，只不过有些问题必须记录其所有的状态而使空间感到紧张，结果不得不放弃“动态规划”算法。

时间问题对于“动态规划”算法来说，几乎不成问题，因为“动态规划”算法已经达到了“时间极限”。但空间问题却是应用“动态规划”技术所面临的最大问题。状态存储量的巨大是应用“动态规划”算法最头疼的问题，也自然成了“信息学”竞赛中考察“动态规划”方面的一个问题点。如何选定状态，如何优化状态存储，更是“仁者见仁，智者见智”的问题。最后，我总结了解决“动态规划”问题的大概步骤：

1. **思考（Think）：**想出解决问题的方法，这是建立在理解问题、分析问题的基础上的，通过仔细的“琢磨”问题，认真的辨别各个思路的优劣，选择一个最有效、最恰当的思路，来解决这个问题。
2. **计划（Design）：**确定“动态规划”的各个条件，状态、阶 段、决策、转移方程、边界条件。这一步是整个全局的“统筹规划”。奠定了这个问题的“动态规划”算法基础以及应用条件。
3. **操作(Operate)：**将算法转化为程序。结合计算机及编程环境条件，具体解决“状态的存储”问题，以至解决整个问题。这一步是通过人与计算机“通力合作”而完成的。

通过以上的步骤，一个“动态规划”程序就应该完成了。

“动态规划”归根到底只是一种技术，需要在不断的应用中得到完善与发展。我们只有通过不断的应用与思考，才可以清楚它的结构、领悟它的内涵，从而真正掌握这门技术。

【附录】

本文共有 6 个例题，正文中的题目均为简写，这里给出每个例题的详细叙述与程序解答。

注：题目中提到的图片均已在正文中出现。编号同正文。

〖例 1〗

题目

图 1 给出了一个线路网络，两点之间连线上的数字表示两点间的距离（或费用），试求一条由 A 到 E 的旅行路线，使总距离（或总费用）最小。

说明

这个问题出自《信息学奥林匹克竞赛指导——组合数学的算法与程序设计》。正文中为了比较搜索算法与动态规划算法的效率，给出了两个程序：Pro_1_1.Pas 与 Pro_1_2.Pas。以及五个测试数据：Data_1_i.Txt 与其相应的答案：Sol_1_i.Txt，在 Data 子目录中。

程序

```
Program Pro_1_1;                    {用搜索算法解决例 1 的程序}

Const
  Max         =300;                                  {最多城市数}
  Inputfile       ='Input.Txt';                      {输入文件}
```

```
  Outputfile      ='Ouptut.Txt';                    {输出文件}
  Big             =1000000;                         {最大整数}

Type
  Maps            =Array [1..Max] Of Integer;       {地图类型说明}

Var
    Se          :Set Of Byte;                       {一个记录还未访问过的城市的集合}
    Map         :Array[1..Max] Of ^maps;            {地图数组}
    Qiu,Pr      :Array[1..Max] Of Byte;             {最短路径的数组}
    F           :Text;                              {文件变量}
    Lp          :Integer;                           {最短路径的城市数}
    N,M         :Integer;                           {输入的数据}
    Min         :Longint;                           {最短路径的长度}


 Procedure Init;                                    {初始化过程}
 Var
   I,J,K,W      :Integer;
  Begin
      Assign(F,Inputfile);
      Reset(F);
          Readln(F,N,M);
          For I:=1 To N Do                          {读入数据}
             Begin
                New(Map[I]);
                Fillchar(Map[I]^,Sizeof(Map[I]^),0);
             End;
          For I:=1 To M Do
             Begin
                Readln(F,J,K,W);
                Map[J]^[K]:=W;
             End;
      Close(F);
  End;

 Procedure Search(Dep:Byte;Al:Longint);             {寻找最短路径}
 Var
    I           :Byte;
    Begin
       If Al>=Min Then Exit;                        {砍枝条件}
       If Qiu[Dep-1]=N Then                         {如果比当前还小，记录}
          Begin
             Min:=Al;
             Pr:=Qiu;
             Lp:=Dep-1;
          End
       Else
         For I:=2 To N Do                           {访问所有没访问过的城市}
          If (I In Se) And (Map[Qiu[Dep-1]]^[I]>0) Then
            Begin
               Exclude(Se,I);
               Qiu[Dep]:=I;
                    Search(Dep+1,Al+Map[Qiu[Dep-1]]^[Qiu[Dep]]);
               Include(Se,I);
            End;
    End;
```

```
 Procedure Print;                                          {输出数据}
 Var
   I          :Integer;
  Begin
     Assign(F,Outputfile);
     Rewrite(F);
        Writeln(F,Min);
        For I:=1 To Lp Do
           Write(Pr[I],' ');
     Close(F);
  End;

Begin
   Init;                                                   {输入数据}
   Se:=[2..N];                                             {各项值初始化}
   Qiu[1]:=1;
   Min:=Big;
   Search(2,0);                                            {寻找最短路径}
   Print;                                                  {输出}
End.
```

Program Pro_1_2; **{用“动态规划”的方法解决例 1}**

```
Const
   Max          =300;                                     {最多的城市数}
   Inputfile       ='input.Txt';                          {输入文件名}
   Outputfile      ='output.Txt';                         {输出文件名}
   Big          =1000000;                                 {最大整数}

Type
   Maps         =Array[1..Max] Of Integer;                {地图类型说明}

Var
    Se          :Set Of Byte;                                    {未访问过的城市集合}
    Map          :Array[1..Max] Of ^maps;                 {地图变量}
    Dis          :Array[1..Max] Of Longint;               {某城市到终点城市的最短距离}
    Fr           :Array[1..Max] Of Byte;                  {动态规划的标识数组}
    Bo           :Array[1..Max] Of Boolean;               {访问过城市标识}
    N,M          :Integer;                                {输入数据}
    F            :Text;                                   {文件变量}

 Procedure Init;                                                    {初始化过程}
 Var
   I,J,K,W      :Integer;
  Begin
      Assign(F,Inputfile);                                  {读入数据}
      Reset(F);
          Readln(F,N,M);
           For I:=1 To N Do
               Begin
                  New(Map[I]);
                  Fillchar(Map[I]^,Sizeof(Map[I]^),0);
               End;
          For I:=1 To M Do
               Begin
                  Readln(F,J,K,W);
                  Map[J]^[K]:=W;
```

```
                End;
      Close(F);
  End;

 Procedure Main;                                          {“动态规划”递推过程}
   Var
     I,J,Who    :Integer;
     Min        :Longint;                                 {当前最小值}

     Begin
       Dis[N]:=0;                                                {初始化动态规划数
组}
       Who:=N;
       Fillchar(Fr,Sizeof(Fr),0);
       Fillchar(Bo,Sizeof(Bo),True);
       Bo[N]:=False;
       While Who<>1 Do
        Begin
          For I:=1 To N Do                                 {利用“状态转移方程”递推结果}
           If Map[I]^[Who]>0 Then
            If (Fr[I]=0) Or (Dis[I]>Dis[Who]+Map[I]^[Who]) Then
              Begin
                Dis[I]:=Dis[Who]+Map[I]^[Who];
                Fr[I]:=Who;
              End;
           Min:=Big;
           For I:=1 To N Do
            If Bo[I] And (Fr[I]>0) And (Dis[I]<Min) Then
              Begin
                Who:=I;
                Min:=Dis[I];
              End;
           Bo[Who]:=False;
        End;
     End;

  Procedure Print;                                            {输出结果}
  Var
   I          :Integer;
   Begin
     Assign(F,Outputfile);
     Rewrite(F);
       Writeln(F,Dis[1]);
       I:=1;
       While I<>N Do
       Begin
           Write(F,I,' ');
           I:=Fr[I];
       End;
       Writeln(F,N);
     Close(F);
   End;

Begin
     Init;                          {输入}
     Main;                  {求解}
     Print;                 {输出}
End.
```

〖例 2〗

题目

有 4 个点，他们分别是 A、B、C、D，如图 4 所示，相邻两点有两条连线 C2k,C2k-1(1≦k≦3)表示两条通行的道路，连线上的数字表示道路的长度。

我们定义从 A 到 D 的所有路径中，长度除以 4 所得的余数最小的路径是最优路径。

说明

本题目出自《信息学奥林匹克》杂志 1998 年 1-2 期。

〖例 3〗

题目

欧几里德货郎担问题是对平面上的给定的 N 个点确定一条连结各点的、闭合的游历路线的问题。图 5(a)给出了七个点问题的解。这个问题的一般形式是 NP 完全的，故其解需要多于多项式的时间

J.L.Bentiley 建议通过只考虑 **bitonic 旅行路线**来简化问题。这种旅行路线即从最左点开始，严格地由左至右直到最右点，然后严格地由右至左直至出发点。图 5(b)示出了同样七个点的最短 **bitonic** 游历。在这个例子中，一个多项式时间的算法是可能的。

请描述一个确定最优 **bitonic** 游历的 $O(N^2)$ 时间的算法。你可以假设任何两个节点的 x 坐标都不相同。

说明

本问题出自《计算机数据结构和实用算法大全》。本文中阐述了这个问题的两个解法：“动态规划”解法与搜索解法，程序如下：

程序

```
Program Pro_3_1;                                    {例 3 的动态规划解法}
Const
  Inputfile     ='input.Txt';                       {输入文件名}
  Outputfile    ='output.Txt';                      {输出文件名}
  Max           =100;                               {最多点的数目}
  Big           =10000;                             {最大整数值}

Var
  F                   :Text;                                        {文件变量}
  Po            :Array[1..Max,1..2] Of Integer;     {记录每个点坐标的数组}
  Dis           :Array[1..Max,1..Max] Of Real;      {记录动态规划中状态的权值}
  N                   :Integer;                                     {点的总数}

 Procedure Init;                                                  {初始化过程}
 Var
   I            :Integer;
   Begin
     Assign(F,Inputfile);                                        {读入数据}
     Reset(F);
       Readln(F,N);
        For I:=1 To N Do
          Readln(F,Po[I,1],Po[I,2]);
     Close(F);
   End;
```

```
 Function Len(P1,P2:Integer):Real;                              {求两个点之间的距离}
   Begin
     Len:=Sqrt(Sqr(Po[P1,1]-Po[P2,1])+Sqr(Po[P1,2]-Po[P2,2]));
   End;

 Procedure Main;                                                {动态规划过程}
  Var
    I,J,K          :Integer;
    Now                    :Real;                              {当前最小值}
   Begin
      Dis[N,N]:=0;                                             {初始化动态规划数
组}
      For I:=N-1 Downto 1 Do
        Begin
          Dis[I,N]:=Len(I,I+1)+Dis[I+1,N];
          Dis[N,I]:=Dis[I,N];
        End;

      For I:=N-2 Downto 1 Do                                   {递推最小值}
        For J:=N-1 Downto I+1 Do
        Begin
          If I+1<J Then Now:=Dis[I+1,J]+Len(I,I+1)
           Else
             Begin
               Now:=Big;
               For K:=J+1 To N Do
                 If Dis[K,J]+Len(I,K)<Now Then
                    Now:=Dis[K,J]+Len(I,K);
             End;
          For K:=J+1 To N Do
            If Dis[I,K]+Len(J,K)<Now Then
                 Now:=Dis[I,K]+Len(J,K);
          Dis[I,J]:=Now;
          Dis[J,I]:=Now;
        End;
   End;

 Procedure Print;                                                {输
出过程}
   Var
      X1,X2,I,D              :Integer;
      Now                          :Real;                      {当
前最小值}
      P                            :Array[1..2] Of Integer;    {打印数组的
长度}
      Pr                                 :Array[1..2,1..Max] Of Byte;    {打
印数组}
      Ok                           :Boolean;

        Procedure Change;                                      {交
换两个数的值}
          Var
            G                    :Integer;
            Begin
               D:=3-D;
               G:=X1;X1:=X2;X2:=G;
            End;
```

```
    Begin
      Assign(F,Outputfile);
      Rewrite(F);                                                    {输
出数据}
      Now:=Big;
       For I:=2 To N Do                                              {求
最短的路径值}
          If Dis[1,I]+Len(1,I)<Now Then
            Begin
              Now:=Dis[1,I]+Len(1,I);
              X2:=I;
            End;
        X1:=1;
        Writeln(F,Now:0:2);
        P[1]:=1;P[2]:=1;                                             {打印数组初
始化}
        Pr[1,1]:=X1;Pr[2,1]:=X2;
        D:=1;
        While (X1<>N) And (X2<>N) Do
          Begin
            Ok:=True;                                                {根据动态规
划递推规则构造打印数组}
             If X1+1<X2 Then
               Begin
                 If Dis[X1+1,X2]+Len(X1,X1+1)=Dis[X1,X2] Then
                     Begin
                         Inc(X1);
                         Inc(P[D]);
                         Pr[D,P[D]]:=X1;
                         Ok:=False;
                     End;
               End
              Else
               Begin
                 I:=X2+1;
                 While (I<=N) And ( Dis[I,X2]+Len(X1,I)<>Dis[X1,X2]) Do
                         Inc(I);
                 If I<=N Then
                    Begin
                       Ok:=False;
                       X1:=I;
                       Inc(P[D]);
                       Pr[D,P[D]]:=X1;
                       Change;
                    End;
               End;
             If Ok Then
               Begin
                   I:=X2+1;
                   While Dis[X1,I]+Len(I,X2)<>Dis[X1,X2] Do
                         Inc(I);
                   X2:=I;
                   Inc(P[3-D]);
                   Pr[3-D,P[3-D]]:=X2;
               End;
          End;
        While Pr[D,P[D]]<>N Do
          Begin
```

```
                Inc(P[D]);
                Pr[D,P[D]]:=Pr[D,P[D]-1]+1;
            End;

        For I:=1 To P[1] Do                                          {输出打印数
组}
            Write(F,Pr[1,I],' ');
        Writeln(F);
        For I:=1 To P[2] Do
            Write(F,Pr[2,I],' ');
        Close(F);
    End;

Begin
  Init;                 {初始化}
  Main;                 {动态规划递推最短路径}
  Print;                {输出结果}
End.
```

Program Pro_3_2; **{例 3 的搜索解法}**

```
Const
  Inputfile     ='input.Txt';                    {输入文件名}
  Outputfile    ='output.Txt';                   {输出文件名}
  Max           =100;                            {最多点的数目}
  Big           =10000;                          {最大整数值}

Var
  F             :Text;                           {文件变量}
  Po            :Array[1..Max,1..2] Of Integer   {记录各个点的坐标}
  Qiu,Pr        :Array[1..Max] Of Byte;          {记录搜索过程的数组}
  Lpr           :Integer;                        {打印数组的长度}
  N             :Integer;                        {点的数目}
  Min           :Real;                           {最短路径的长度}

 Procedure Init;                                 {初始化过程}
 Var
   I            :Integer;
   Begin
     Assign(F,Inputfile);
     Reset(F);                                   {读入数据}
       Readln(F,N);
        For I:=1 To N Do
          Readln(F,Po[I,1],Po[I,2]);
     Close(F);
   End;

 Function Len(P1,P2:Integer):Real;               {求两点之间距离的函数}
   Begin
     Len:=Sqrt(Sqr(Po[P1,1]-Po[P2,1])+Sqr(Po[P1,2]-Po[P2,2]));
   End;

 Procedure Search(Dep,Fr,Last:Byte;Al:Real);     {搜索最短路径}
   Var
     I,P:Byte;
     Rs:Real;
       Procedure Did;                            {将第 2 条路径的长
度加入当前的长度}
```

```
        Var
          K:Byte;
          Begin
              K:=Fr+1;
              Rs:=0;
              P:=Last;
              While K<>I Do
                 Begin
                    Rs:=Rs+Len(P,K);
                    P:=K;
                    Inc(K);
                 End;
            Rs:=Rs+Len(Fr,I);
          End;
    Begin
        If Al>=Min Then Exit;
        If Fr=N Then
          Begin
             If Al+Len(Last,N)<Min Then                    {如果更优，记录}
               Begin
                  Min:=Al+Len(Last,N);
                  Pr:=Qiu;
                  Lpr:=Dep-1;
               End;
          End
        Else
          For I:=Fr+1 To N Do                                {访问每一个点}
            Begin
              Qiu[Dep]:=I;
              Did;
              Search(Dep+1,I,P,Al+Rs);
            End
    End;


Procedure Print;                                             {输出数据}
  Var
    I                    :Byte;
    Se                   :Set Of Byte;                       {记录未输出的点}
    Begin
       Assign(F,Outputfile);
       Rewrite(F);
         Writeln(F,Min:0:2);
         Se:=[1..N];
         For I:=1 To Lpr Do
           Begin
              Write(F,Pr[I],' ');
              Se:=Se-[Pr[I]];
           End;
         Se:=Se+[1,N];
         Writeln(F);
         For I:=1 To N Do
           If I In Se Then
              Write(F,I,' ');
      Close(F);
    End;

Begin
  Init;                          {输入数据}
```

```
  Min:=Big;                    {最小值初始化}
  Search(1,1,1,0);             {搜索最短路径}
  Print;                       {输出结果}
End.
```

〖例 4〗

题目

有一个奶牛运输公司，设在农场 A 中，它有一个用来在七个农场 A，B，C，D、E、F、G 之间运输奶牛的运输车。七个农场之间的距离由下表给出：

	B	C	D	E	F	G
A	56	43	71	35	41	36
B	43	54	58	36	79	31
C	71	58	30	20	31	58
D	35	36	20	38	59	75
E	41	79	31	59	44	67
F	36	31	58	75	67	72

每天早晨，运输公司都要决定运输奶牛的顺序，使得总路程最少。下面是规则：

1：运输车总是从位于农场 A 的公司总部出发，而且当所有的运输任务完成之后还要返回总部；

2：运输车一次只能运送一头奶牛；

3：每个运输任务是由一对字母给出的，分别代表奶牛将从哪个农场被运向另一个农场。

你的任务是写一个程序，对一组给定的任务，求出一条完成所有任务的最短路线（从农场 A 出发，最后返回农场 A）。

输入格式：任务存放在一个名为“Delvr.Txt”的文件中。文件的第一行是任务的总数 N(1≤N≤12)，从第二行开始，每个任务由一对被一个空格各开的字母给出，第一个字母代表奶牛所在的农场，第二个字母代表奶牛将被运向的农场。你的程序运行时间应不超过 60 秒。

输入举例：

```
5
F C
G B
B D
A E
G A
```

输出格式：在屏幕上打印出最短路线的长度，并显示运输车经过的农场的顺序。

输出举例：

```
368
A E F C G B D G A
```

说明

本题出自《信息学奥林匹克》杂志 1998 年 1-2 期。在正文中，这个题目有三种算法：“动态规划”算法、搜索算法、改进后的搜索算法，依次对应的程序是 Pro_4_1.Pas、Pro_4_2.Pas、Pro_4_3.Pas。还有五个测试数据，输入、输出文件为 Data_4_i.Txt.、Sol_4_i.Txt，在 Data 子目录中。

程序

```
Program Pro_4_1;                                    {例 4 的动态规划算法}
Const
  Map:Array['A'..'G','A'..'G'] Of Byte=             {每两个农场之间的距离}
      ((0,56,43,71,35,41,36),
       (56,0,54,58,36,79,31),
       (43,54,0,30,20,31,58),
       (71,58,30,0,38,59,75),
       (35,36,20,38,0,44,67),
       (41,79,31,59,44,0,72),
       (36,31,58,75,67,72,0));

  Inputfile         ='d1.Inp';                    {输入文件名}
  Outputfile        ='sol_4_1.Txt';               {输出文件名}

Type
    Kus             =Array[0..4095] Of Word;      {权值数组类型说明}
    Dirs            =Array[0..4095] Of Byte;      {标记数组类型说明}
    Ses             =Set Of 1..12;                  {集合类型}

Var
    F               :Text;                          {文件变量}
    N               :Integer;                                {任务数目}
    Wu              :Array[0..12,1..2] Of Char;   {记录任务的数组}
    Dis             :Array[0..12,0..12] Of Integer;  {每两个任务的连接费用}
    Ku              :Array[1..12] Of ^kus;          {动态规划中记录权值的数组}
    Dir             :Array[1..12] Of ^dirs;         {动态规划的标记数组}

  Procedure Init;                                                {初始化过
程}
  Var
     I              :Integer;
     Ch             :Char;
   Begin
      Assign(F,Inputfile);
      Reset(F);
          Readln(F,N);
          For I:=1 To N Do                                       {读入数据}
              Readln(F,Wu[I,1],Ch,Wu[I,2]);
      Close(F);
   End;

  Procedure Prepare;                                             {准备过程}
   Var
      I,J           :Integer;
      Begin
         Wu[0,1]:='A';Wu[0,2]:='A';                              {求出每两个任务的
连接费用}
        For I:=0 To N Do
          For J:=0 To N Do
```

```
            Dis[I,J]:=Map[Wu[I,2],Wu[J,1]];
      End;

  Procedure Main;                                                    {动态规划
过程}
  Var
     Last,I,K,What      :Word;
     S              :Ses;

       Function Num(S:Ses):Word;                                     {将集合转化为整数的函数}
         Var
           X          :Word Absolute S;
           Begin
               Num:=X Div 2;
           End;

       Procedure Did(Dep,From:Byte;S:Ses);                          {为动态规划中记录权值的数组
赋值}
          Var
           I          :Byte;
          Begin
            If Dep>K Then
               Begin
                  For I:=1 To N Do
                   If (I In S) And (Ku[I]^[Num(S-[I])]+Dis[What,I]<Ku[What]^[Num(S)]) Then
                      Begin                                         {如果更小，改变现有权值与表示变量}
                         Ku[What]^[Num(S)]:=Ku[I]^[Num(S-[I])]+Dis[What,I];
                         Dir[What]^[Num(S)]:=I;
                      End;
               End
            Else
               For I:=From+1 To N Do
                  If I<>What Then
                     Did(Dep+1,I,S+[I]);
          End;

   Begin
      For I:=1 To N Do                                              {初始化动态规划记录数组}
         Begin
            New(Ku[I]);
            New(Dir[I]);
            Fillchar(Ku[I]^,Sizeof(Ku[I]^),$ff);
         End;
      For I:=1 To N Do
            Ku[I]^[0]:=Dis[I,0];

      For K:=1 To N-1 Do                                            {为动态规划数组赋值}
          For What:=1 To N Do
            Did(1,0,[]);

      S:=[1..N];
      K:=60000;
      For I:=1 To N Do
         If Dis[0,I]+Ku[I]^[Num(S-[I])]<K Then
            Begin
              K:=Dis[0,I]+Ku[I]^[Num(S-[I])];
              What:=I;
            End;
      For I:=1 To N Do
```

```
        Inc(K,Map[Wu[I,1],Wu[I,2]]);

    Assign(F,Outputfile);                                    {输出结果}
    Rewrite(F);
      Writeln(F,K);
      Write(F,'A ');
      Last:=0;
      For I:=1 To N Do
        Begin
          If Wu[Last,2]<>Wu[What,1] Then Write(F,Wu[What,1],' ');
              Write(F,Wu[What,2],' ');
          Exclude(S,What);
          Last:=What;
          What:=Dir[What]^[Num(S)];
        End;
       If 'A'<>Wu[Last,2] Then Write(F,'A');
    Close(F);
  End;


Begin
  Init;              {初始化}
  Prepare;           {准备}
  Main;              {计算}
End.
```

Program Pro_4_2; **{例 4 的搜索算法}**

```
Const
  Map:Array['A'..'G','A'..'G'] Of Byte=                  {每两个农场之间的距离}
       ((0,56,43,71,35,41,36),
        (56,0,54,58,36,79,31),
        (43,54,0,30,20,31,58),
        (71,58,30,0,38,59,75),
        (35,36,20,38,0,44,67),
        (41,79,31,59,44,0,72),
        (36,31,58,75,67,72,0));

  Inputfile      ='d1.Inp';                                  {输入文件名}
  Outputfile      ='data_4_1.Txt';                          {输出文件名}


Var
   F            :Text;                                      {文件变量}
   N            :Integer;                                   {任务的数目}
   Min          :Word;                                      {最短距离}
   Wu           :Array[0..12,1..2] Of Char;          {记录任务的数组}
   Dis          :Array[0..12,0..12] Of Integer;      {记录每两个任务的连接权值}
   Qiu,Pr         :Array[0..14] Of Byte;                {搜索中的记录数组}
   Se           :Set Of Byte;                               {未完成的任务的集合}

  Procedure Init;                                                {初始化过程}
  Var
    I          :Integer;
    Ch          :Char;
  Begin
     Assign(F,Inputfile);
     Reset(F);
```

```
        Readln(F,N);                                                    {读入数据}
        For I:=1 To N Do
        Readln(F,Wu[I,1],Ch,Wu[I,2]);
    Close(F);
  End;

 Procedure Prepare;                                                     {准备过程}
  Var
    I,J          :Integer;
    Begin
      Wu[0,1]:='A';Wu[0,2]:='A';                                        {求出每两个任务的
连接权值}
       For I:=0 To N Do
         For J:=0 To N Do
           Dis[I,J]:=Map[Wu[I,2],Wu[J,1]];
    End;

 Procedure Search(Dep:Byte;Al:Word);                                    {搜索最优得路径}
 Var
    I            :Byte;
   Begin
     If Al>=Min Then Exit;
     If Dep=N+1 Then
       Begin
            If Dis[Qiu[N],0]+Al<Min Then
              Begin
                Min:=Dis[Qiu[N],0]+Al;                          {如果更优,记录之}
                Pr:=Qiu;
              End;
       End
     Else
      For I:=1 To N Do                                                  {访问每个未访问
的农场}
        If I In Se Then
          Begin
            Qiu[Dep]:=I;
            Exclude(Se,I);
              Search(Dep+1,Al+Dis[Qiu[Dep-1],I]);
            Include(Se,I);
          End;
   End;

  Procedure Print;                                                      {输出结
果}
  Var
       I          :Byte;
   Begin
     For I:=1 To N Do
      Inc(Min,Map[Wu[I,1],Wu[I,2]]);
     Assign(F,Outputfile);
     Rewrite(F);
        Writeln(F,Min);
        Write(F,'A ');
        For I:=1 To N Do
          Begin
            If Wu[Pr[I-1],2]<>Wu[Pr[I],1] Then Write(F,Wu[Pr[I],1],' ');
            Write(F,Wu[Pr[I],2],' ');
          End;
        If 'A'<>Wu[Pr[N],2] Then Write(F,'A');
```

```
    Close(F);
  End;

Begin
  Init;                          {初始化}
  Prepare;                       {准备}
  Se:=[1..12];                   {初始化各项搜索应用值}
  Qiu[0]:=0;
  Min:=60000;
  Search(1,0);                   {搜索}
  Print;                         {输出}
End.
```

Program Pro_4_3; **{例 4 改进后的搜索算法}**

```
Const
  Map:Array['A'..'G','A'..'G'] Of Byte=          {每两个农场的距离}
        ((0,56,43,71,35,41,36),
         (56,0,54,58,36,79,31),
         (43,54,0,30,20,31,58),
         (71,58,30,0,38,59,75),
         (35,36,20,38,0,44,67),
         (41,79,31,59,44,0,72),
         (36,31,58,75,67,72,0));

  Inputfile      ='d1.Inp';              {输入文件名}
  Outputfile     ='data_4_1.Txt';        {输出文件名}

Type
  Kus          =Array[0..4095] Of Word;  {记录数组类型}
  Ses          =Set Of 1..12;            {集合类型}

Var
  F            :Text;                    {文件变量}
  N            :Integer;                 {任务的数目}
  Min          :Word;                    {最短路径的长度}
  Big,I        :Word;                    {应用变量}
  Wu           :Array[0..12,1..2] Of Char;  {记录任务的数组}
  Dis          :Array[0..12,0..12] Of Integer;  {记录每两个任务的连接权值}
  Ku           :Array[1..12] Of ^kus;    {记录最优权值的数组}
  Qiu          :Array[1..13] Of Byte;    {搜索中记录路径的数组}
  Se           :Ses;

  Procedure Init;                                {初始化过程}
  Var
    I          :Integer;
    Ch         :Char;
  Begin
    Assign(F,Inputfile);
    Reset(F);
      Readln(F,N);
      For I:=1 To N Do                           {读入数据}
      Readln(F,Wu[I,1],Ch,Wu[I,2]);
    Close(F);
  End;

  Procedure Prepare;                             {准备过程}
```

```
Var
  I,J          :Integer;
  Begin
    Wu[0,1]:='A';Wu[0,2]:='A';                          {求出每两个任务的连接权值}
    For I:=0 To N Do
      For J:=0 To N Do
        Dis[I,J]:=Map[Wu[I,2],Wu[J,1]];

      For I:=1 To N Do                                  {初始化记录数组}
        Begin
          New(Ku[I]);
          Fillchar(Ku[I]^,Sizeof(Ku[I]^),$ff);
        End;
      Big:=Ku[1]^[1];
  End;

Function Num(S:Ses):Word;                               {将集合转化为整数的函数}
  Var
      X       :Word Absolute S;
    Begin
      Num:=X Div 2;
    End;

Procedure Search(Dep:Byte;Al:Word);                     {搜索过程}
   Var
      I          :Byte;
      D          :Word;
  Begin
    If Al>=Min Then Exit;
    If Ku[Qiu[Dep-1]]^[Num(Se)]<Big Then
      Begin                                             {如果已经做过此工作,则直
接读值}
        If Al+Ku[Qiu[Dep-1]]^[Num(Se)]<Min Then
          Min:=Al+Ku[Qiu[Dep-1]]^[Num(Se)];
      End
    Else
      If Dep>N Then
        Begin                                           {边界时直接计算}
          Ku[Qiu[Dep-1]]^[Num(Se)]:=Dis[Qiu[Dep-1],0];
          If Al+Ku[Qiu[Dep-1]]^[Num(Se)]<Min Then
              Min:=Al+Ku[Qiu[Dep-1]]^[Num(Se)];
        End
     Else
        Begin
          D:=Big;                                       {搜索每个位访问点}
          For I:=1 To N Do
           If I In Se Then
            Begin
              Exclude(Se,I);
              Qiu[Dep]:=I;
              Search(Dep+1,Al+Dis[Qiu[Dep-1],Qiu[Dep]]);
              If Ku[I]^[Num(Se)]+Dis[Qiu[Dep-1],Qiu[Dep]]<D Then
              D:=Ku[I]^[Num(Se)]+Dis[Qiu[Dep-1],Qiu[Dep]];  {记录工作结果}
              Include(Se,I);
            End;
          Ku[Qiu[Dep-1]]^[Num(Se)]:=D;
        End;
  End;
```

```
  Procedure Print;                                                    {输出结果}
  Var
    I,J,Last        :Integer;
    D               :Word;
   Begin
     Se:=[1..N];
     J:=1;
     While  Dis[0,J]+Ku[J]^[Num(Se-[J])]<>Min Do
       Inc(J);
    D:=Min;
     For I:=1 To N Do
       Inc(D,Map[Wu[I,1],Wu[I,2]]);
     Assign(F,Outputfile);
     Rewrite(F);
        Writeln(F,D);
        Write(F,'A ');
        Dec(Min,Dis[0,J]);
        Last:=0;

       For I:=1 To N Do
         Begin
           If Wu[J,1]<>Wu[Last,2] Then
              Write(F,Wu[J,1],' ');
           Write(F,Wu[J,2],' ');
           If I<>N Then
             Begin
               Last:=J;
               Exclude(Se,J);
               J:=1;
               While (Not (J In Se)) Or (Ku[J]^[Num(Se-[J])]+Dis[Last,J]<>Min) Do
                        Inc(J);
               Dec(Min,Dis[Last,J]);
             End;
         End;
       If Wu[J,2]<>'A' Then Write(F,'A');
     Close(F);
   End;

Begin
 Init;                                          {输入}
 Prepare;                                         {准备}
 Se:=[1..N];                                    {初始化搜索中应用的各项值}
 Min:=Big;
  For I:=1 To N Do
    Begin
      Se:=[1..N]-[I];
      Qiu[1]:=I;
      Search(2,Dis[0,I]); {搜索}
    End;
  Print;                          {输出}
End.
```

〖**例 5**〗

题目

一些生物体的复杂结构可以用其基元的序列表示，而一个基元用一个大写英文字母字符串表示。生物学家的一个问题就是将一个这样的长序列分解为基元（字符串）的序列。对于给定的基元的集合 P，如果可以从中选出 N 个基元

P1，P2，P3，…，Pn，将它们各自对应的字符串依次连接后得到一个字符串S，称S可以由基元集合P。在从P中挑选基元时，一个基元可以使用多次，也可以不用。例如，序列aBaBaCaBaaB可以由基元集合{a，aB，Ba，Ca，BBC}构成。

字符串的前K个字符称为该字符串的前缀，其长度为K。请写一个程序，对于输入的基元集合P和字符串T，求出一个可以由基元集合P构成的字符串T的前缀，要求该前缀的长度尽可能长，输出其长度。

输入数据：有两个输入文件INPUT.TXT和DATA.TXT。INPUT.TXT的第一行是基元集合P中基元的数目N（1≦N≦100），随后有2N行，每两行描述第一基元，第一行为该基元的长度L(1≦L≦20)。随后一行是一个长度为L的大写英文字符串，表示该基元。每个基元互不相同。

文件DATA.TXT描述要处理的字符串T，每一行行首有一个大写英文字母，最后一行是一个字符”.”，表示字符串结束。T的长度最小为1，最大不超过500,000。

输出数据：文件名为OUTPUT.TXT。只有一行，是一个数字，表示可以由P构成的T的最长前缀的长度。

说明

本问题是第八届国际奥林匹克信息学竞赛试题。程序在下面。样例输入、输出为Data_5.Txt，Input_5.Txt，Sol_5.Txt。在子目录Data下。

程序

```
Program Pro_5;                                  {例 5 的动态规划解法}

Const
  Datafile     ='Data.Txt';                     {文本输入文件名}
  Inputfile    ='Input.Txt';                    {基元输入文件名}
  Outputfile   ='Output.Txt';                   {输出文件名}

Type
  Jis        =Record                            {基元记录类型说明}
                Lst:Byte;                       {基元长度}
                St:String[20];                  {基元内容}
              End;
Var
  F          :Text;                  {文件变量}
  Ji         :Array[1..100] Of Jis;          {基元记录数组}
  Ch         :Array[0..21] Of Char;          {字符记录数组}
  Bo         :Array[0..21] Of Boolean;       {记录是否为前缀的数组}
  N          :Integer;                       {基元数目}
  Tot        :Longint;                       {总的字符数}
  Big        :Longint;                       {最长的前缀长度}

  Procedure Init;                                                {初始化过程}
   Var
     I       :Integer;
    Begin
      Assign(F,Inputfile);
      Reset(F);
        Readln(F,N);                                             {读入每个基元}
        For I:=1 To N Do
          Begin
            Readln(F,Ji[I].Lst);
```

```
                Readln(F,Ji[I].St);
            End;
        Close(F);
    End;

Procedure Main;                                          {求最长前缀的过程}
  Var
     I,J       :Integer;

     Procedure See;                                      {查看此位是否为前缀}
       Var
          I     :Byte;
          X,Y    :Integer;
       Begin
          Bo[J]:=False;
          For I:=1 To N Do
             Begin                                       {考察每个基元}
               X:=J;Y:=Ji[I].Lst;
               While (Y>0) And (Ji[I].St[Y]=Ch[X]) Do
                  Begin
                     Dec(X);
                     Dec(Y);
                     If X<0 Then X:=21;
                  End;
               If (Y=0) And (Bo[X]) Then
                  Begin
                     Bo[J]:=True;
                     Exit;
                  End;
             End;
       End;
  Begin
    Assign(F,Datafile);
    Reset(F);
    Big:=0;                                                  {边读入边判定}
    Tot:=0;
    Bo[0]:=True;
    J:=0;
     While (Tot-Big<20) And (Ch[J]<>'.') Do          {如果超过 20 位不连续或读完文件,结束}
       Begin
          Inc(Tot);Inc(J);
          If J>21 Then J:=0;
          Readln(F,Ch[J]);
          If Ch[J]<>'.' Then
             Begin
                See;
                If Bo[J] Then Big:=Tot;
             End;
       End;
     Close(F);
  End;

Procedure Print;                                         {输出数据}
  Begin
    Assign(F,Outputfile);
    Rewrite(F);
    Writeln(F,Big);
    Close(F);
  End;
```

```
Begin
  Init;          {输入}
  Main;          {动态规划求最长前缀}
  Print;    {输出}
End.
```

〖例 6〗

题目

某公司运进一批箱子，总数为 N（1≤N≤1000）由“传送带”依次运入，然后在仓库内至多排成 P（1≤P≤4）列，此排列过程由一台“调度机器”完成（如图 8 所示）。

现在已知运来的箱子最多为 M（1≤M≤20）种，老板自然想让这些箱子排列的尽可能“漂亮”，即把同一种类的箱子尽量排在一起，以便美观。根据老板的爱好，他的“美观程度”T 定义为：

T=Σ（每列依次看到的不同种类数）；

所谓“依次看到的不同种类数”即为：若第 K 个与第 K-1 个不同种类，则单记一类。

如：1 2 3 1 这一列的“依次看到的不同种类数”为 4；

1 2 2 3 3 1 2 这一列的“依次看到的不同种类数”为 5；

即 T 越小，就越美观，老板就越满意。现已知对每列箱子的个数不做限制，你可以将运来的箱子排在一列，也可以分 P 列排列。但由于是传送带运输，所以箱子是有次序的，即若 A 箱子在 B 箱子前运来，则若 A、B 排在一列，A 必在 B 的前边。

现在请你为“调度机器”编一段程序，使其根据“传送带”发来的箱子信息进行调度，使其排列为 P 列后的“美观程度”数最小。

输入要求：输入文件名为“INPUT.TXT”。内容有两行，第一行有三个数，依次为 N，M，P。第二行有 N 个数，为传送带运来的箱子种类。第 K 个数为第 K 个箱子的种类。

输出要求：输出文件名为“OUTPUT.TXT”。内容为两行，第一行为“美观程度”数 T，第二行有 N 个数，为一种调度方案，即第 K 个数表示第 K 个箱子将去往那一列。

说明

这个问题是我自己编的。程序在下面，样例输入、输出为 Data_6.Txt、Sol_6.Txt 在 Data 子目录中。

程序

```
Program  Pro_6;                                          {例 6 的动态规划解法}
Const  Fin                ='Input.txt';                  {输入文件名}
       Fou                ='Output.txt';                 {输出文件名}
Type   Kus                =Array[0..1160] Of Byte;    {记录数组的类型说明}
       Zis                =Array[0..1160] Of Integer;
Var    Ku       :Array[1..1000] Of ^kus;    {记录决策的数组}
       Shu,Shu1   :Array[1..1000] Of Byte;
       Zi       :Array[0..1] Of Zis;        {记录状态的数组}
       Use1     :Array[0..20,0..20,0..20] Of Integer; {记录三个数的组合的数组}
       Use2     :Array[0..1160,1..3] Of Byte; {将一个数转化为三个数组合的数组}
       Total,I,M,N,P,N1   :Integer;                    {总数与各种数据}
       F            :Text;                      {文件变量}
```

```
{--Init--}
Procedure Init;                                                    {初始化}
  Var I,J:Integer;
    Begin
      Assign(F,Fin);
      Reset(F);
        Readln(F,N1,M,P);
        For I:=1 To N1 Do
          Read(F,Shu1[I]);
      Shu[1]:=Shu1[1];
      N:=1;
      For I:=2 To N1 Do
        If Shu1[I]<>Shu1[I-1] Then
            Begin Inc(N);Shu[N]:=Shu1[I];End;
      Close(F);
    End;
{---Diduse----}
Procedure Diduse;                                                  {将三个数组合的情况转化
为一个序数与其对应}
 Var Qs    :Array[1..3] Of Byte;
       Stop  :Byte;
       I,J,K  :Byte;
         {==Serch==}
    Procedure Serch(Depth,From:Byte);                              {搜索所有的组合情况}
      Var I:Byte;
       Begin
        If Depth=Stop+1 Then
          Begin
            Inc(Total);
            Use2[Total,1]:=Qs[1];Use2[Total,2]:=Qs[2];
            Use2[Total,3]:=Qs[3];
            Use1[Qs[1],Qs[2],Qs[3]]:=Total;Use1[Qs[1],Qs[3],Qs[2]]:=Total;
            Use1[Qs[2],Qs[1],Qs[3]]:=Total;Use1[Qs[2],Qs[3],Qs[1]]:=Total;
            Use1[Qs[3],Qs[1],Qs[2]]:=Total;Use1[Qs[3],Qs[2],Qs[1]]:=Total;
          End
        Else
         For I:=From To M-1 Do
          Begin
            Qs[Depth]:=I;
            Serch(Depth+1,I+1);
          End;
       End;
  Begin
     Total:=0;
     Fillchar(Qs,Sizeof(Qs),0);
     For I:=0 To M Do
       For J:=0 To M Do
         For K:=0 To M Do
           Use1[I,J,K]:=-1;
     Use2[0,1]:=0;Use2[0,2]:=0;Use2[0,3]:=0;
     Use1[0,0,0]:=0;
     For Stop:=1 To P-1 Do
        Serch(1,1);
  End;
{-----Main-------}                                                  {主过程}
Procedure Main;
  Var I,J,K,W1,W2,W3    :Integer;
        Q               :Array[1..4] Of Byte;
  Begin
```

```
  For I:=1 To N Do
    Begin New(Ku[I]);End;
  Fillchar(Zi,Sizeof(Zi),0);                                  {动态规划初始化}
  Fillchar(Ku[1]^,Sizeof(Ku[1]^),0);
  Zi[0][0]:=1;
  Fillchar(Q,Sizeof(Q),0);
  For I:=2 To N Do                                            {动态规划求值}
   Begin
     For J:=0 To Total Do
       If Zi[0][J]<>0 Then
         Begin
           W1:=0;
           For K:=1 To 3 Do
                Begin Q[K]:=Use2[J,K];
                      If Q[K]>=Shu[I-1] Then Inc(Q[K]);
                      If Q[K]=Shu[I] Then W1:=K;
                End;{End For K}
           Q[4]:=Shu[I-1];
           If W1<>0 Then
              Begin
                W2:=Q[4];Q[4]:=Q[W1];Q[W1]:=W2;
                For K:=1 To 3 Do
                     If Q[K]>Shu[I] Then Dec(Q[K]);
                W3:=Use1[Q[1],Q[2],Q[3]];
                If (Zi[1][W3]=0) Or (Zi[1][W3]>Zi[0][J]) Then
                     Begin
                       Zi[1][W3]:=Zi[0][J];
                       Ku[I]^[W3]:=Shu[I];
                     End;
              End {End If}
            Else
              Begin
                  For K:=1 To 4 Do
                    If Q[K]>Shu[I] Then Dec(Q[K]);
                  For K:=1 To 4 Do                {分四种情况决策}
                      Begin
                        Case K Of
                          1:W1:=Use1[Q[2],Q[3],Q[4]];
                          2:W1:=Use1[Q[1],Q[3],Q[4]];
                          3:W1:=Use1[Q[1],Q[2],Q[4]];
                          4:W1:=Use1[Q[1],Q[2],Q[3]];
                        End;
                      If W1<>-1 Then
                        If (Zi[1][W1]=0) Or (Zi[1][W1]>Zi[0][J]+1) Then
                           Begin
                             Zi[1][W1]:=Zi[0][J]+1;
                             W2:=Q[K];
                             If W2>=Shu[I] Then Inc(W2);
                             Ku[I]^[W1]:=W2;
                           End;{End If}
                        End;{End For K}
                  End;{End For Else};
           End;{End For J}
         Zi[0]:=Zi[1];
         Fillchar(Zi[1],Sizeof(Zi[1]),0);
    End;{End For I}
 End;
 {-----Print---}
Procedure Print;                                              {打印结果}
```

```
  Var Pr,Pr1            :Array[1..1000] Of Integer;
      Now,Q             :Array[1..4] Of Byte;
      I,J,K,Ps,W1,W2    :Integer;
  Begin
    Ps:=1001;
     For J:=0 To Total Do
       If (Zi[0][J]<>0) And (Zi[0][J]<Ps) Then
           Begin Ps:=Zi[0][J];K:=J;End;
    Assign(F,Fou);
    Rewrite(F);
      Writeln(F,Ps);
      Now[2]:=Use2[K,1];
      Now[3]:=Use2[K,2];
      Now[4]:=Use2[K,3];
      For I:=2 To 4 Do
        If Now[I]>=Shu[N] Then Inc(Now[I]);
      Now[1]:=Shu[N];
      Pr[N]:=1;
      For K:=N-1 Downto 1 Do  {分阶段打印}
       Begin
         J:=1;
         While Now[J]<>Shu[K+1] Do
               Inc(J);
         W2:=0;
         For I:=1 To 4 Do
          If I<>J Then
            Begin
              Inc(W2);Q[W2]:=Now[I];
              If Q[W2]>=Shu[K+1] Then Dec(Q[W2]);
            End;
         W1:=Use1[Q[1],Q[2],Q[3]];
         Now[J]:=Ku[K+1]^[W1];
         For I:=1 To 4 Do
          If Now[I]=Shu[K] Then
            Pr[K]:=I;
       End;{End For K}
       J:=1;
       Pr1[1]:=Pr[1];
       For I:=2 To N1 Do
         Begin
           If Shu1[I]<>Shu1[I-1] Then Inc(J);
           Pr1[I]:=Pr[J];
         End;
     For I:=1 To N1 Do
       Write(F,Pr1[I],' ');
    Close(F);
  End;
 {Main}                  {主程序}
Begin
  Init;                  {初始化}
  Diduse;                {准备}
  Main;                  {动态规划过程}
  Print;                 {输出}
End.
```

注：以上的程序均是在 PASCAL 7.0 环境下调试通过。

【参考书目】

①：《青少年国际和全国信息学（计算机）奥林匹克竞赛指导——组合数学算法与程序设计》　　（吴文虎 王建德编著，清华大学出版社）
②：《信息学奥林匹克》杂志 1998 年 1-2 期。
③：《计算机数据结构和实用算法大全》　　（谋仁 主编，北京希望电脑公司）
④：《国际国内奥林匹克信息学（计算机）1996 年竞赛试题解析》　（吴文虎　王建德编著，北京大学出版社）