由此可看出，对于看似复杂的问题，通过转化就可变成简单的经典的动态规划问题，在问题原型的基础上，通过分析新问题与原问题的不同之处，修改状态转移方程，改变问题状态的描述和表示方式，就会降低问题规划和实现的难度，提高算法的效率。由此可见，动态规划问题中具体的规划方法将直接决定解决问题的难易程度和算法的时间与空间效率，而注意在具体的规划过程中的灵活性和技巧性将是动态规划方法提出的更高要求。

4、生物基元问题。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

图**7-2** 基元组成图

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，则其具体组成如图7-2所示。

【分析】显然很容易确定问题的状态，因为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。那么就可以设一个数组：

can：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：

can[k]:=can[k] Or (can[k-w] And (S[k-w+1...k]=Ji[I]))

其中，can[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。

至此，问题似乎已经得到解决，只需求出使can[i]为true最大的I即可。但题设中规定，字符串最长为500,000，因此状态根本无法存储，为此，需要寻找状态存贮方式，如果注意到转移方程中的can[i]只与can[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，如果想推出can[I]，最多只需要知道can[i-20]至can[i-1]的值即可，而其它的can值完全可以不去管它。这样，需要记录的can 值一下子就由500000减少到20,这时问题也就得到解决。下面我们给出问题的源程序如下：

```
{$A+,B-,D+,E+,F-,G+,I+,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+}
{$M 16384,0,655360}

{ 任务名 : PREFIX }
program PREFIX;
const
   MAX = 50 ;
   MAXLEN = 20 ;
var
  N : integer ;
  DS : array[ 1..100 ] of string ;  {各个基元串}
  DL : array[ 1..100 ] of integer ; {各个基元长度}
  F1 , F2 , F3 : text ;

  BT : array[ 0..MAX + 1 ] of byte ; {队列}

  T1 , T2 : integer ;          {队首,队尾}
  Over : boolean ;             {文件结束标志}
  CurPos , MaxPos : Longint ;   {队列首在串中的位置, 已找到的最长前缀长度}
  S : string ;                 {当前队列对应的串}
  FBuf : array[ 1..30000 ] of char ; {文件缓冲区}
procedure Readch ; {扩展队列尾部}
var C : char ;
begin
   if not Over then Readln( F2 , C ) ;
   S := S + C ;
   if C = '.' then Over := true ;
```

```
    T2 := ( T2 + 1 ) mod MAX ;
    BT[ T2 ] := 0 ;
end ;
procedure Init ; {初始化}
var I : integer ;
begin
    Over := false ;
    CurPos := 0 ;
    MaxPos := 0 ;
    S := '' ;

   {读文件}
    Assign( F1 , 'INPUT.TXT' ) ;
    Reset( F1 ) ;
    Readln( F1 , N ) ;
    for I := 1 to N do
    begin
        Readln( F1 , DL[ I ] ) ;
        Readln( F1 , DS[ I ] ) ;
    end ;
    Close( F1 ) ;
    Assign( F2 , 'DATA.TXT' ) ;
    Settextbuf( F2 , FBuf ) ;
    Reset( F2 ) ;
    {初始化队列}
    Fillchar( BT , Sizeof( BT ) , 0 ) ;
    T1 := 1 ;
    T2 := 1 ;
    for I := 1 to MAXLEN + 1 do Readch ;
    BT[ 1 ] := 1 ;
end ;

procedure GoNext ; {队首,队尾同时向后移一个字符}
begin
    T1 := ( T1 + 1 ) mod MAX ;
    Inc( CurPos ) ;
    Readch ;
    Delete( S , 1 , 1 ) ;
end ;
procedure Count ; {计算最大长度}
var
   I , J , K : integer ;
begin
    while CurPos <= MaxPos do
    begin
        if BT[ T1 ] = 1 then {如能达到该长度}
        begin
            for I := 1 to N do
             if Pos( DS[ I ] , S ) = 1 then
             begin
                 BT[ ( T1 + DL[ I ] ) mod MAX ] := 1 ; {扩展新的长度}
                 if MaxPos <  CurPos + DL[ I ] then
                   MaxPos := CurPos + DL[ I ] ;
```

```
                end ;
            end ;
            GoNext ;
        end ;
    end ;
    procedure WriteFile ; {输出}
    begin
        Assign( F3 , 'OUTPUT.TXT' ) ;
        Rewrite( F3 ) ;
        Writeln( F3 , MaxPos ) ;
        Close( F3 ) ;
    end ;
    begin
        Init ;
        Count ;
        WriteFile ;
    end
```