把握本质，灵活运用——动态规划的深入探讨

浙江省萧山中学　来煜坤

【关键字】 动态规划　构思　实现

【摘要】 本文讨论了动态规划这一思想的核心内容和其基本特点，探讨了动态规划思想的适用范围，动态规划子问题空间和递推关系式确立的一般思路。通过例子说明在子问题确立过程中的一些问题的解决办法：通过加强命题或适当调节确定状态的变量等手段帮助建立动态规划方程，通过预处理使动态规划的过程容易实现等。接着，分析动态规划实现中可能出现的空间溢出问题及一些解决办法。总结指出，动态规划这一思想，关键还在于对不同的问题建立有效的数学模型，在把握本质的基础上灵活运用。

一、引言

动态规划是一种重要的程序设计思想，具有广泛的应用价值。使用动态规划思想来设计算法，对于不少问题往往具有高时效，因而，对于能够使用动态规划思想来解决的问题，使用动态规划是比较明智的选择。

能够用动态规划解决的问题，往往是最优化问题，且问题的最优解(或特定解)的局部往往是局部问题在相应条件下的最优解，而且问题的最优解与其子问题的最优解要有一定的关联，要能建立递推关系。如果这种关系难以建立，即问题的特定解不仅依赖于子问题的特定解，而且与子问题的一般解相关，那么，一方面难以记录下那么多的“一般解”，另一方面，递推的效率也将是很低的；此外，为了体现动态规划的高时效，子问题应当是互相重叠的，即很多不同的问题共享相同的子问题。(如果子问题不重叠，则宜使用其它方法，如分治法等。)

动态规划一般可以通过两种手段比较高效地实现，其一是通过自顶向下记忆化的方法，即通过递归或不递归的手段，将对问题最优解的求解，归结为求其子问题的最优解，并将计算过的结果记录下来，从而实现结果的共享；另一种手段，也就是最主要的手段，通过自底向上的递推的方式，由于这种方式代价要比前一种方式小，因而被普遍采用，下面的讨论均采用这种方式实现。动态规划之所以具有高时效，是因为它在将问题规模不断减小的同时，有效地把解记录下来，从而避免了反复解同一个子问题的现象，因而只要运用得当，较之搜索而言，效率就会有很大的提高。

动态规划的思想，为我们解决与重叠子问题相关的最优化问题提供了一个思考方向：通过迭代考虑子问题，将问题规模减小而最终解决问题。适于用动态规划解决的问题，是十分广泛的。动态规划的思想本身是重要的，但更重要的是面对具体问题的具体分析。要分析问题是否具备使用动态规划的条件，确定使用动态规划解题的子问题空间和递推关系式等，以及在(常规)内存有限的计算机上实现这些算法。下面分别就构思和实现两个方面进一步探讨动态规划这一思想。

二、动态规划解题的构思

当我们面对一个问题考虑用动态规划时，十分重要的一点就是判断这个问题能否用动态规划高效地解决。用动态规划构思算法时，往往要考虑到这个问题所涉及到的子问题(子问题空间)，以及如何建立递推式，并最终实现算法。其实，这些过程往往是交织在一起的，子问题空间与递推关系本身就是紧密相联的，为了有效地建立起递推关系，有时就要调整子问题空间；而根据大致确定的子问题空间又可以启发我们建立递推关系式。而能否最终用一个递推关系式来联系问题与其子问题又成了判断一个问题能否使用动态规划思想解决的主要依据。因而孤立地来看这其中的每一部分，硬把思考过程人为地分成几个部分，是困难的，也是不必要的。而且动态规划这种思想方法，没有固定的数学模型，要因题而异，因而也就不可能归纳出一种“万能”的方法。但是对大多数问题而言，还是能够有一个基本的思考方向的。

首先，要大致分析一个问题是否可能用动态规划解决。如果一个问题难以确定子问题，或问题与其子问题的特殊解之间毫无关系，就要考虑使用其它方法来解决(如搜索或其它方法等)。做一个大概的判断是有必要的，可以防止在这上面白花时间。通常一个可以有效使用动态规划解决的问题基本上满足以下几方面的特性：

1、 子问题的最优解仅与起点和终点(或有相应代表意义的量)有关而与到达起点、终点的路径无关。
2、 大量子问题是重叠的，否则难以体现动态规划的优越性。

下面以 IOI’97 的“字符识别”问题为例进行分析一般情况下动态规划思路的建立。

IOI’97 的字符识别问题，题目大意是：在 FONT.DAT 中是对□(空格)、A—Z 这 27 个符号的字形说明。对每一个符号的字符点阵，用 20 行每行 20 个“0”或

者“1”表示。在另一个输入文件中，描述了一串字符的点阵图象(共 N 行)，但字符可能是“破损的”，即有些 0 变成了 1，而有些 1 变成了 0。每行固定也为 20 个“0”或“1”，但每一个字符对应的行可能出现如下情形：

- 仍为 20 行，此时没有丢失的行也没有被复制的行；
- 为 19 行，此时有一行被丢失了；
- 为 21 行，此时有一行被复制了，复制两行可能出现不同的破损。

要求输出，在一个假定的行的分割情况下，使得“0”与“1”的反相最少的方案所对应的识别结果(字符串)。

在初步确定这个问题可以用动态规划思想解决之后，我认为可以考虑用数学的方法(或观点)来刻划这个问题，比如通常的最优化问题(这也是动态规划解决的主要问题)，总会有一个最优化的标准，动态规划要通过递推来实现，就要求分析确定这个状态所需要的量。比如字符识别问题，在问题规模下相当于求 N 行的一种分割与对应方法，因而很自然地，考虑前几行就成了一个确定状态的量。最优的标准题中已经给出，即在某种假设(包括分割方法与对应识别方法)下，使得“0”与“1”反相数最少。如果把这个度量标准看作一个函数，这实际上就是一个最优化函数(指标函数)，最优化函数的值依赖于自变量，即确定状态的量。自变量的个数(这里是一个，即行数，考虑前几行之意)，要因题而异，关键是要有效地确定状态，在这种状态下，因保证靠这些量已经能够确定最优化函数的值，即最优化函数在这些量确定的时候理论上应有确定的值，否则量是不够的或要使用其它量来刻划，而即使能够完全确定，但在建立递推关系式时发生困难，也要根据困难相应调整确定最优化函数值的自变量。而反过来，如果设定了过多的量来确定最优化函数值，那么，动态规划的效率将会大大下降，或者解了许多不必要解的子问题，或者将重叠子问题变成了在这种自变量条件下的非重叠子问题，从而大大降低效率，甚至完全失去动态规划的高效。在这个例子中，对于前 L 行，此最优化函数显然有确定的值。

动态规划的递推的一种重要思想是将复杂的问题分解为其子问题。因而确定子问题空间及建立递推关系式是十分重要的。根据确定最优化函数值的自变量，往往对子问题空间有着暗示的作用。通常，通过对最接近问题的这步进行倒推，可以得到这个问题规模减小一些的子问题，不断这样迭代考虑，就往往能够体会到问题的子问题空间。而在这个过程中，通过这种倒推分析，也比较容易得出这种递推关系。需要指出，这仅仅是对一些题目解题思考过程的总结，不同的题目原则上仍应区别对待。比如字符识别问题，考虑 n 行该最优化函数值时，注意到最终一定是按照字符分割与识别的，因而最后一个字符或者是 19 行，或者是 20

行，再或者是 21 行，仅这样三种可能情况，依次考虑这三种分割方法，对于切割下来的这一段对应于一个字符，对于每一种切割方案，当然应该选择最匹配的字符(否则，如果不使用反相情况最少的字符作为匹配结果而导致全局的最优，那么只要在这一步换成反相情况最少的字符，就得到比假定的“最优”更优的结果，从而导致矛盾)。在去除一个字符后，行数有所减少，而这些行去匹配字符显然也应当使用最优的匹配(可以用反证法证明，与前述类似)，于是得到一个与原问题相似(同确定变量，同最优化标准)但规模较小的子问题，与此同时子问题与原问题的递推关系事实上也得到了建立：

f[i]:=min{Compare19[i-19+1]+f[i-19],Compare20[i-20+1]+f[i-20], Compare21[i-21+1]+f[i-21]}

f[i]表示对前 i 行进行匹配的最优化函数值；

Compare19[i]、Compare20[i]、Compare21[i]分别表示从 i 行开始的 19 行、20 行、21 行与这三种匹配方式下最接近的字符的反相的“0”与“1”的个数。

初始情况，f[0]=0，对于不可能匹配的行数，用一个特殊的大数表示即可。当然，本题的问题主要还不在于动态规划的基本思考上(这里只是通过这个例子，讲一下对于不少动态规划问题的一种基本的思考方向)，还有数学建模(用 2 进制表示 0、1 串)等(源程序见附录中的程序 1)。

有时虽然按上述思路得出的确定状态的量已经能够使最优化函数具有确定的值，但是在建立递推关系时发生困难，通过引入新的变量或调整已有变量，也是一条克服困难的途径。比如，NOI’97 的一题“积木游戏”，题目大意是：

积木是一个长方体，已知 N 个有编号的积木的三边(a、b、c 边)长，要求出用 N 块中的若干块堆成 M(1≤M≤N≤100)堆，使总高度最大的高度值，且满足：

- 第 K 堆中任意一块的编号大于第 K+1 堆中任意一块积木的编号；
- 任意相邻两块，下面的块的上表面要能包含上面的那块的下表面，且下面的块的编号要小于上面积木的编号。

因为题目要求编号小的堆的积木编号较大，这不太自然，在不改变结果的前提下，把题目改作编号小的堆的积木编号较小，这显然不会影响到最终的高度和，而且，此时每一种合理的堆放方法可看作，按编号递增的顺序选择若干积木，按堆编号递增的顺序逐堆放置，每堆中积木依次在前一个上面堆放而最终形成一种堆放方案。使用上面一般的分析方法，很容易看出，考虑前 i 个木块放置成前 j 堆，这样，i、j 两个量显然能够确定最优函数的值，然而递推关系却不易直接建立，稍作分析就会发现，问题主要出在第 i 块到底能否堆放到其子问题(i-1,j 作变量确定的状态)的最优解方案的最后一堆上。如果考虑增加该序列最后一块

的顶部的长与宽的(最小)限制这两个变量，建立递推关系并不困难，然而，很明显，递推过程中大量结果并未被用到，这就人为地扩大了子问题空间，不仅给存储带来麻烦，而且效率也很低。其实，建立递推需要的仅仅是在子问题解最后一堆顶部能否容纳当前积木块，而题中可能产生的这种限制性的面最多仅有3*100+1(无限制)=301 种情况，这样在多引入一个“最后一堆顶部能够容纳下第k 种面的要求”这个量后，递推关系只要分当前块另起一堆、当前块加在前一堆上(如果可能的话)和当前块不使用这三种情况就可以了。(源程序参见所附程序2)

此外，有些问题可能会出现仅靠这种调整递推关系仍难以建立，这时，通过增加其它量或函数来建立递推关系式也是一种思考方向(类似于数学归纳法证明时的“加强命题”)。因为，用动态规划解题的一个重要特征是通过递推，而递推是利用原有结果得到新结果的过程。如果在理论上可以证明，一个难以直接实现递推的问题可以通过引入新的递推关系，同时将两者解决，这看起来把问题复杂化了，而实际上由于对于每一步递推，在增加了解决的问题的同时也增加了条件（以前解决的值），反而使递推容易进行。举例说明，IOI’98 中的“多边形”一题，大意如下：

有一个多边形（N 边形），顶点上放整数，边上放“+”或“*”，要寻找一种逐次运算合并的顺序，通过 N-1 次运算，使最后结果最大。

如果单纯考虑用 MAX[I,L]，从 I 开始进行 L 个运算所得的最大值，则难以实现递推，而根据数学知识，引入了 MIN[I,L]为从 I 开始进行 L 个运算所得的最小值，在进行递推时，却能够有效地用较小的 I，L 来得到较大时的结果，从而事实上同时解决了最小值与最大值两个问题。

递推关系式如下：(考虑 I 从 1 到 N,L 从 1 到 N-1)

考虑 t(最后一步运算位置)从 0 到 L-1：

如果最后一步运算为“+”则：

min(i,L)=最小值{min(i,t)+min((i+t+1-1) mod N+1,L-t-1)}

max(i,L)=最大值{max(i,t)+max((i+t+1-1) mod N+1,L-t-1)}

如果最后一步运算为“*”则：

min(i,L)=最小值{min(i,t)*min((i+t+1-1) mod N+1,L-t-1),
min(i,t)*max((i+t+1-1) mod N+1,L-t-1),
max(i,t)*min((i+t+1-1) mod N+1,L-t-1),
max(i,t)*max((i+t+1-1) mod N+1,L-t-1)}

max(i,L)=最大值{min(i,t)*min((i+t+1-1) mod N+1,L-t-1),

min(i,t)*max((i+t+1-1) mod N+1,L-T-1)
max(i,t)*min((i+t+1-1) mod N+1,L-t-1),
max(i,t)*max((i+t+1-1) mod N+1,L-t-1)}

(源程序见附录中的程序 3)

此外，动态规划通过递推来实现，因而问题与子问题越相似，越有规律就越容易进行操作。因而对于某些自身的阶段和规律不怎么明显的问题，可以通过一个预处理，使其变得更整齐，更易于实现。例如，ACM'97 亚洲赛区/上海区竞赛一题“正则表达式(Regular Expression)的匹配”问题，题目大意是：

正则表达式是含有通配符的表达式，题目定义的广义符有：

- . 表示任何字符
- [c1-c2] 表示字符 c1 与 c2 间的任一字符
- [^c1-c2] 表示不在字符 c1 与 c2 间的任一字符
- * 表示它前面的字符可出现 0 或多次
- + 表示它前面的字符可出现一次或多次
- \ 表示它后面的字符以一个一般字符对待。

对一个输入串，寻找最左边的与正则表达式匹配的串(相同条件下要最长的)。这里如果不作预处理，则有时一个广义符可对应多个字符，有时又是多个广义符仅对应一个字符，给系统化处理带来很多麻烦。因而有必要对正则表达式进行标准化，使得或者某个结点仅对应一个字符，或者用一特殊标记表明它可以重复多次。定义记录类型：

```
NodeType=Record
  StartChar: Char; {开始字符}
  EndChar: Char;  {结束字符}
  Belong:  Boolean {是否属于}
  Times:  Boolean; {False: 必须一次; True:可以多次，也可以不出现}
End;
```

对输入数据预处理之后，建立递推关系就不太困难了。用 Pro[i,j]表示前 i 个正则表达式结点对以第 j 个字符为终点的子串的匹配情况(True/False)，对于为 True 的情况，同时指明此条件下最先的开始位置。如果第 i 个正则表达式结点是仅出现一次的，那么，如果它与第 j 个字符不匹配，则该值为 False，否则，它与 Pro[i-1,j-1]相同。(初始时 Pro[0,x]=True)。如果它是可重复多次的，那么它可以被解释成 0 个或多个字符。在它自身与相应位置的 0 个或多个字符匹配的条件下依次考虑这些可能情况，只要其中含 True，则 Pro[i,j]为 True，同时记录下这

些达到 True 的情况中起点最先的。按此递推，直到 i 达到结点个数。(源程序见所附程序 4)

三、动态规划实现中的问题

动态规划解决问题在有了基本的思路之后，一般来说，算法实现是比较好考虑的，但有时也会遇到一些问题，而使算法难以实现。动态规划思想设计的算法从整体上来看基本都是按照得出的递推关系式进行递推，这种递推，相对于计算机来说，只要设计得当，效率往往是比较高的，这样在时间上溢出的可能性不大，而相反地，动态规划需要很大的空间以存储中间产生的结果，这样可以使包含同一个子问题的所有问题共用一个子问题解，从而体现动态规划的优越性，但这是以牺牲空间为代价的，为了有效地访问已有结果，数据也不易压缩存储，因而空间矛盾是比较突出的。另一方面，动态规划的高时效性往往要通过大的测试数据体现出来（以与搜索作比较），因而，对于大规模的问题如何在基本不影响运行速度的条件下，解决空间溢出的问题，是动态规划解决问题时一个普遍会遇到的问题。

对于这个问题，我认为，可以考虑从以下一些方面去尝试：

一个思考方向是尽可能少占用空间。如从结点的数据结构上考虑，仅仅存储必不可少的内容，以及数据存储范围上精打细算(按位存储、压缩存储等)。当然这要因题而异，进行分析。另外，在实现动态规划时，一个我们经常采用的方法是用一个与结点数一样多的数组来存储每一步的决策，这对于倒推求得一种实现最优解的方法是十分方便的，而且处理速度也有一些提高。但是在内存空间紧张的情况下，我们就应该抓住问题的主要矛盾。省去这个存储决策的数组，而改成在从最优解逐级倒推时，再计算一次，选择某个可能达到这个值的上一阶段的状态，直到推出结果为止。这样做，在程序编写上比上一种做法稍微多花一点时间，运行的时效也可能会有一些(但往往很小)的下降，但却换来了很多的空间。因而这种思想在处理某些问题时，是很有意义的。

但有时，即使采用这样的方法也会发现空间溢出的问题。这时就要分析，这些保留下来的数据是否有必要同时存在于内存之中。因为有很多问题，动态规划递推在处理后面的内容时，前面比较远处的内容实际上是用不着的。对于这类问题，在已经确信不会再被使用的数据上覆盖数据，从而使空间得以重复利用，如果能有效地使用这一手段，对于相当大规模的问题，空间也不至于溢出。（为了求出最优方案，保留每一步的决策仍是必要的，这同样需要空间。）一般地说，这种方法可以通过两种思路来实现。一种是递推结果仅使用 Data1 和 Data2 这样两个数组，每次将 Data1 作为上一阶段，推得 Data2 数组，然后，将 Data2 通过

复制覆盖到 Data1 之上，如此反复，即可推得最终结果。这种做法有一个局限性，就是对于递推与前面若干阶段相关的问题，这种做法就比较麻烦；而且，每递推一级，就需要复制很多的内容，与前面多个阶段相关的问题影响更大。另外一种实现方法是，对于一个可能与上 N 阶段相关的问题，建立数组 Data[0..N]，其中各项即为与原 Data1/Data2 相同的内容。这样不采用这种内存节约方式时对于下标 K 的访问只要对应成对下标 K mod (N+1)的访问，就可以了。与不作这种处理的方法相比，对于程序修改的代码很少，速度几乎不受影响(用电脑做 MOD 运算是很快的)，而且需要保留不同的阶段数也都能很容易实现。这种手段对不少题目都适用，比如：NOI'98 的“免费馅饼”，题目大意是：

有一个舞台，宽度 W 格(1≤W≤99 的奇数),高度 H 格(1≤H≤100)，游戏者在时刻 0 时位于舞台正中，每个单位时间可以从当时位置向左移 2 格、向左移 1 格、保持不动、向右移 1 格或者向右移 2 格，每个馅饼会告知初始下落的时间和位置以及下落速度(1 秒内下移的格子数)和分值。仅在某 1 秒末与游戏者位于同一格内的馅饼才被认为是接住的。求一种移动方案，使得分最大。注意：馅饼已按初始下落时间排序。

从问题来看，想到动态规划并不是很困难的。但是，题中规定初始下落时间从 0 到 1000，而且考虑下落到最后可能时间要到 1100 左右,而宽度可达 99，以时间-位置作为状态决定因素进行递推，速度不会慢，但如果采用初始数据经预处理后的结果(即在何时到何地可得多少分的描述数组)用一个数组，动态规划递推用一个数组，记录每步决策用一个数组，因得分题中未指出可能的大小，如果采用前两个 Longint 型，最后一个 Shortint 型，所须内存约为 1100*99*9 字节，即约 957KB，这显然是不可能存得下的。但是注意到在进行递推时，一旦某一个(时间，位置)对应的最大分值一确定，这个位置的原始数据就不再有用，因而两者可以合二为一，从而只要 1100*99*5 字节，即约 532KB。这样对于题目规模的问题就勉强可以解决了。 当然，如果更进一步思考，其实这个问题中递推是仅与上一个时间有关的，而馅饼实际上仅使用了当前位置的值。由于初始下落时间已经排序，那么当读到初始下落时间晚于当前处理时间时，就不必马上读入。为了避免重复和无规律地读盘和内存开销过大，只要记录下当前之后约 100 个时间单位内的情况就可以了，使用前面所说的循环使用内存的方法，只要 101*99*4+99*2*2=40392 字节，不到 40KB，而对于每一个时间仅需 99 个 shortint 存储决策即可,就算把问题规模提高到 3000 或者 4000 个时间单位也能顺利出解。(源程序见附录中的程序 5)

当采用以上方法仍无法解决内存问题时，也可以采用对内存的动态申请来使绝大多数测试点能有效出解(而且，使用动态内存还有一点好处，就是在重复

使用内存而进行交换时，可以只对指针进行交换，而不复制数据)，这在竞赛中也是十分有效的。

四、总结

动态规划是一种重要的程序设计思想。但是，由于它没有确定的算法形式，因而也就有较大的灵活性，但它的本质却具有高度的相似性。所以，学习和使用这一思想方法，关键在于在理解和把握其本质的基础上灵活运用。本文虽然谈到了一些思想方法，但这些仅是对一些较普遍问题的讨论，针对具体问题进行具体分析建立数学模型才是最重要而关键之处。

【参考资料】

1、吴文虎、王建德 《实用算法的分析与程序设计》电子工业出版社 ISBN 7-5053-4402-1/TP.2036

2、吴文虎、王建德 《青少年国际和全国信息学(计算机)奥林匹克竞赛指导——组合数学的算法与程序设计》清华大学出版社 ISBN 7-302-02203-8/TP.1060

3、 NOI'97、NOI'98、IOI'97、IOI'98 试题，ACM'97 亚洲赛区试题

【程序】

程序 1　IOI'97 字符识别

```
{“字符识别”的基本动态规划方程已在正文中说明，这里补充说明一下本题提高速度的关键——错位比较时提高效率。}
{注意到少一行与多一行时的比较，虽然可能出现错位，但每一行仅有与邻近的两行比较的可能，}
{先把可能的比较记录下来，再累计从端点到某一位置的非错位时反相数之和与错位时反相数之和，}
{考虑 20 种情况，仅需一重循环(不考虑比较一行的子程序内的循环)即可，效率得到很大提高}
program Character_Recognition; {“字符识别”程序}
const
```

```
  cc:array [1..27] of char=(' ','A','B','C','D','E','F','G','H','I','J','K','L',
                            'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'); {字符常量}
var
  f,f1,f2:text; {文件变量}
  font:array [1..540] of longint; {记录字形的数组,一个 longint 数表示 20 位 2 进制数，下同}
  dd:array [1..1200] of longint; {待分析的点阵数据}
  str:string; {读入的串}
  i,j,k:integer; {辅助变量}
  t:word;
  ff:integer;
  bin:array [1..20] of longint; {2 的幂}
  pro:array [0..1200] of word; {动态规划数组}
  sta:array [0..1200] of byte; {每步分析的最优字符序号}
  bf:array [0..1200] of word; {每步分析的上一个字符的终点}
  pf:array [1..21,0..1] of word; {错位比较时用}
  n:integer;

procedure getnum(var l:longint); {String->longint 转换}
var
  i:integer;
begin
  l:=0;
  for i:=1 to 20 do
    if str[i]='1' then inc(l,bin[i]);
end;

function compare0(a,b:longint):byte; {比较 a 表示的行与 b 表示的行的反相个数}
var
  k:byte;
  i:integer;
begin
  a:=a xor b;
  k:=0;
  for i:=1 to 20 do
    if a and bin[i]<>0 then inc(k);
  compare0:=k
end;


function compare20(k:integer; var ff:integer):word; {比较 20 行的最优结果}
var
  i,j,t,s:word;
  best:word; {当前最优}
begin
```

```
  ff:=0;
  best:=maxint;
  for t:=1 to 27 do {考虑 27 个字符}
  begin
    j:=0;
    for i:=1 to 20 do
    begin
      s:=compare0(font[(t-1)*20+i],dd[i+k-1]); {比较一行}
      inc(j,s); {累计差别}
    end;
    if j<best then begin best:=j; ff:=t end; {如果更优，记录之}
  end;
  compare20:=best;
end;

function compare19(k:integer; var ff:integer):word; {返回与 k 开始 19 行最接近的字符与差别值}
var
  i,j,t,s:word;
  best:word;
  l1,l2:array [0..20] of word;
  bb,fx:integer;
begin
  ff:=0;
  best:=maxint;
  for t:=1 to 27 do {考虑 27 个字符}
  begin
    j:=0;
    fillchar(l1,sizeof(l1),0);
    fillchar(l2,sizeof(l2),0);
    for i:=1 to 19 do
      for j:=0 to 1 do
        pf[i,j]:=compare0(font[(t-1)*20+i+j],dd[k+i-1]);
        {记录 19 行中第 i 行与对应字形中第 i 行、第 i+1 行的差别}
    l1[1]:=pf[1,0]; {l1[i]为破损字形前 i 行与标准字形前 i 行匹配的差别} {}
    for i:=2 to 19 do
      l1[i]:=l1[i-1]+pf[i,0];
    l2[19]:=pf[19,1]; {l2[i]为破损字第 i 行之后与标准字形第 i+1 行之后的字形匹配的差别}
    for i:=18 downto 1 do
      l2[i]:=l2[i+1]+pf[i,1];
    bb:=maxint;
    for i:=1 to 20 do {20 种缺少方式}
      if l1[i-1]+l2[i]<bb then bb:=l1[i-1]+l2[i]; {记录最少的}
    if bb<best then begin best:=bb; ff:=t end; {如果该字符较匹配，改进 BEST}
```

```
  end;
  compare19:=best
end;

function compare21(k:integer; var ff:integer):word;
{返回与第 k 行开始的 21 行最匹配的字形与差别}
var
  i,j,t,s:word;
  best:word;
  l1,l2:array [0..22] of word;
  bb,fx:integer;
begin
  ff:=0;
  best:=maxint;
  for t:=1 to 27 do {考虑 27 个字形}
  begin
    j:=0;
    fillchar(l1,sizeof(l1),0);
    fillchar(l2,sizeof(l2),0);
    fillchar(pf,sizeof(pf),0);
    for i:=1 to 21 do
      for j:=0 to 1 do
        if not ((i=21) and (j=0)) and not ((i=1) and (j=1)) then
pf[i,j]:=compare0(font[(t-1)*20+i-j],dd[k+i-1]);
          {用破损字形第 i 行与标准字形第 i 行、第 i-1 行比较，记录差别}
    l1[1]:=pf[1,0]; {l1[i]为前 i 行与标准前 i 行匹配的差别}
    for i:=2 to 20 do
      l1[i]:=l1[i-1]+pf[i,0];
    l2[22]:=0; {l2[i]为第 i 行开始的内容与标准从 i-1 行开始的内容进行匹配的差别}
    for i:=21 downto 2 do
      l2[i]:=l2[i+1]+pf[i,1];
    bb:=maxint;
    for i:=1 to 20 do {比较 20 种方式}
      if l1[i-1]+l2[i+1]+pf[i,0]<bb then bb:=l1[i-1]+l2[i+1]+pf[i,0];
    if bb<best then begin best:=bb; ff:=t end;
  end;
  compare21:=best
end;



begin {主程序}
  assign(f,'Font.Dat');
  reset(f);
```

```
assign(f1,'Image.dat');
reset(f1);
assign(f2,'Image.out');
rewrite(f2); {文件关联}

readln(f,k); {读入行数 540}
bin[1]:=1;
for i:=2 to 20 do
  bin[i]:=bin[i-1]*2; {生成 bin 数组，为 2 的幂}

for i:=1 to k do
begin
  readln(f,str);
  getnum(font[i]); {string->longint 转换}
end;

read(f1,n); {读入分析文件的各行}
for i:=1 to n do
begin
  readln(f1,str);
  getnum(dd[i]); {string->longint 转换}
end;

fillchar(pro,sizeof(pro),255); {初值 65535}
fillchar(sta,sizeof(sta),0); {每步分析出的最优字符}
fillchar(bf,sizeof(bf),255); {每步分析出的最优切割}
pro[0]:=0;
sta[0]:=0;
bf[0]:=0;
for i:=19 to n do {考虑 19 行至 n 行}
begin
  if (i>=19) and (pro[i-19]<>65535) then {如果切去一个 19 行字符后的情况可能}
  begin
    t:=compare19(i-19+1,ff); {比较从 i-19+1 起 19 行与标准结果最接近的结果与差别}
    if t+pro[i-19]<pro[i] then {如果更优，更新动态规划数组}
    begin
      pro[i]:=t+pro[i-19];
      sta[i]:=ff;
      bf[i]:=i-19;
    end;
  end;
  if (i>=20) and (pro[i-20]<>65535) then {如果切去一个 20 行字符后的情况可能}
  begin
    t:=compare20(i-20+1,ff); {比较从 i-20+1 起 20 行与标准结果最接近的结果与差别}
```

```
        if t+pro[i-20]<pro[i] then {如果更优，更新动态规划数组}
        begin
          pro[i]:=t+pro[i-20];
          sta[i]:=ff;
          bf[i]:=i-20;
        end;
      end;
      if (i>=21) and (pro[i-21]<>65535) then {如果切去一个 21 行字符后的情况可能}
      begin
        t:=compare21(i-21+1,ff); {比较从 i-21+1 起 21 行与标准结果最接近的结果与差别}
        if t+pro[i-21]<pro[i] then {如果更优，更新动态规划数组}
        begin
          pro[i]:=t+pro[i-21];
          sta[i]:=ff;
          bf[i]:=i-21;
        end;
      end;
    end;

    str:='';
    i:=n;
    if pro[n]=65535 then begin writeln(f2,'No answer.'); close(f1); close(f2); halt end; {如果无解}
    while i<>0 do {倒推求解}
    begin
      str:=cc[sta[i]]+str;
      i:=bf[i];
    end;
    writeln(f2,str); {输出}
    close(f1);
    close(f2)
end.
```

程序 2　NOI’97 积木游戏

```
{说明：}
{为防止出现内存溢出，程序采用逐级递推的方式。}
program Toy_Bricks_Game; {积木游戏}
type
  jmtype=array [0..100] of ^node;
  {动态规划过程中仅保留与当前最接近的一个阶段的情况}
  {这是存储一个阶段的量的指针类型}
  node=array [0..300] of longint;
var
```

```
  f1,f2:text; {文件变量}
  size:array [0..300,1..2] of word; {各个出现的面的记录,0 对应无面积要求}
  num:integer; {记录的不同面数}
  n,m:integer; {积木数、堆成的堆数}
  i,j,k,t:integer; {辅助变量}
  p,q,r:jmtype; {递推阶段指针数组，每个保留一个阶段（K 值），从 p 到 q，r 用于交换}
  aa:array [1..100,1..3] of word; {边长数据}
  ss:array [1..100,1..3] of word; {3 个面对应的面序号，对同样长、宽的面作了优化}

procedure sort(i:integer); {对第 i 个长方体的三边长排序}
var
  j,k,t:integer;
begin
  for j:=1 to 2 do
    for k:=j+1 to 3 do
      if aa[i,j]>aa[i,k] then
      begin
        t:=aa[i,j];
        aa[i,j]:=aa[i,k];
        aa[i,k]:=t
      end;
end;

function add(a1,a2:integer):integer; {在面标记中增添当前面，并返回相应的序号}
var
  i,j:integer;
begin
  for i:=1 to num do
    if (a1=size[i,1]) and (a2=size[i,2]) then begin add:=i; exit end;
  inc(num);
  size[num,1]:=a1;
  size[num,2]:=a2;
  add:=num;
end;

procedure preprocess; {预处理，将要处理的面记入}
var
  i,j,k:integer;
begin
  num:=0;
  for i:=1 to n do
  begin
    sort(i);
    ss[i,1]:=add(aa[i,1],aa[i,2]);
```

```
      ss[i,2]:=add(aa[i,1],aa[i,3]);
      ss[i,3]:=add(aa[i,2],aa[i,3]);
   end;
end;

procedure check(ii,nn,hh:integer);  {检查用 ii 积木的 nn 放置方式，是否有效}
var
   g,h:integer;
begin
   if (p[ii-1]^[0]>=0) and (p[ii-1]^[0]+hh>=q[ii]^[j]) then q[ii]^[j]:=p[ii-1]^[0]+hh; {考虑另成一
堆}
   if (p[ii-1]^[ss[ii,nn]]>=0) and (q[ii-1]^[ss[ii,nn]]+hh>=q[ii]^[j]) then
q[ii]^[j]:=q[ii-1]^[ss[ii,nn]]+hh;
   {考虑放在前一堆上}
end;

begin
   assign(f1,'ToyBrick.in');
   reset(f1);
   assign(f2,'ToyBrick.out');
   rewrite(f2); {文件关联、打开}
   readln(f1,n,m); {读入 N、M 值}
   for i:=1 to n do {读入边长}
      readln(f1,aa[i,1],aa[i,2],aa[i,3]);

   size[0,1]:=0;
   size[0,2]:=0;
   preprocess; {生成各种待处理的面}

   for i:=0 to n do {动态内存初始化}
   begin
      new(p[i]);
      new(q[i]);
      fillchar(q[i]^,sizeof(q[i]^),0);
   end;

   for t:=1 to m do {连续递推 M 阶段,分成 T 堆}
   begin
      r:=q;
      q:=p;
      p:=r; {交换 P、Q}
      for i:=0 to n do fillchar(q[i]^,sizeof(q[i]^),255); {Q 初始化}
      for i:=t to n do {考虑 T 个到 N 个积木}
      begin
```

```
        for j:=0 to num do {考虑最后“输出”的面的约束条件}
        begin
          q[i]^[j]:=q[i-1]^[j]; {当前积木不用}
          if (aa[i,1]>=size[j,1]) and (aa[i,2]>=size[j,2]) then check(i,1,aa[i,3]);
          if (aa[i,1]>=size[j,1]) and (aa[i,3]>=size[j,2]) then check(i,2,aa[i,2]);
          if (aa[i,2]>=size[j,1]) and (aa[i,3]>=size[j,2]) then check(i,3,aa[i,1]);
          {如果当前积木的某方向放置可以满足此要求，考虑按此方向放置该块作为新的一堆的底或加在前一堆上（如果可能）}
        end;
      end;
  end;
  writeln(f2,q[n]^[0]); {输出答案}
  close(f1);
  close(f2)
end.
```

程序 3　IOI’98 多边形

```
program Polygon; {“多边形”程序}
var
  f1,f2:text; {输入、输出文件变量}
  n:integer; {顶点个数}
  data:array [1..50] of integer; {原始数据-顶点}
  sign:array [1..50] of char; {原始数据-运算符}
  i,j,k,l:integer; {辅助变量}
  t,s,p:integer; {辅助变量}

  ans:set of 1..50; {可能达到最大值的第一次移动的边的序号}
  best:integer; {当前最优解}
  min,max:array [1..50,0..50] of integer;
  {动态规划表格，min[i,l]表示从第 i 个顶点开始，经过 l 个符号按合理运算所得的结果的最小值；max 与之类似，但为最大值}
  first:boolean; {首次输出标志}

procedure init; {初始化，读入原始数据}
var
  i:integer;
  ch:char;
begin
  readln(f1,n);
  for i:=1 to n do
  begin
```

```
      repeat
        read(f1,ch);
      until ch<>' ';
      sign[i]:=ch; {sign[i]位于 data[i]与其后顶点间}
      read(f1,data[i]);
    end;
end;


begin
  {文件关联、打开}
  assign(f1,'Polygon.in');
  reset(f1);
  assign(f2,'Polygon.out');
  rewrite(f2);

  {初始化}
  init;

  {赋初值}
  best:=-maxint-1;
  ans:=[];
  fillchar(max,sizeof(max),0);
  fillchar(min,sizeof(min),0); {数组初始化}
  for j:=1 to n do
  begin
    max[j,0]:=data[j];
    min[j,0]:=data[j];
  end; {初值是不经过运算(l=0)的值}

  for l:=1 to n-1 do {考虑长度由 1 到 n-1}
    for k:=1 to n do {考虑起始点从 1 到 n}
    begin
      max[k,l]:=-maxint-1;
      min[k,l]:=maxint;
      for t:=0 to l-1 do {考虑分开前半部分经过的运算数}
      begin
        case sign[(k+t+1-1) mod n+1] of {考虑分开处的符号}
          't': {为加法}
      begin
            if max[k,t]+max[(k+t+1-1) mod n+1,l-t-1]>max[k,l] then
max[k,l]:=max[k,t]+max[(k+t+1-1) mod n+1,l-t-1];
            {最大值更新}
            if min[k,t]+min[(k+t+1-1) mod n+1,l-t-1]<min[k,l] then
```

```
min[k,l]:=min[k,t]+min[(k+t+1-1) mod n+1,l-t-1];
                {最小值更新}
              end;
              'x': {为乘法}
              begin
                for p:=1 to 4 do
                begin
                  case p of
                    1: s:=max[k,t]*max[(k+t+1-1) mod n+1,l-t-1];
                    2: s:=max[k,t]*min[(k+t+1-1) mod n+1,l-t-1];
                    3: s:=min[k,t]*max[(k+t+1-1) mod n+1,l-t-1];
                    4: s:=min[k,t]*min[(k+t+1-1) mod n+1,l-t-1]; {考虑四个乘积}
                  end;
                  if s>max[k,l] then max[k,l]:=s;
                  if s<min[k,l] then min[k,l]:=s; {更新最大最小值}
                end;
              end;
            end;
          end;
        end;
  for i:=1 to n do
    if max[i,n-1]>best then begin best:=max[i,n-1]; ans:=[i] end
    else if max[i,n-1]=best then include(ans,i); {更新全局的最大值}

  writeln(f2,best); {输出最大值}
  first:=true;
  for i:=1 to n do
    if i in ans then
    begin
      if first then first:=false
      else write(f2,' ');
      write(f2,i);
    end;
  writeln(f2); {输出首次被移动的边}
  close(f1);
  close(f2) {关闭文件}
end. {结束}
```

程序 4　ACM'97 亚洲区/上海赛题 正则表达式匹配

```
program Expression_Match; {正则表达式匹配程序}
type
  datatype=record {预处理数据类型}
```

```
    st,ed:char; {起始、结束字符}
    md:0..1; {重复方式 0：一次；1：0 或多次}
    mt:0..1; {匹配方式 0：不包含为匹配；1：包含为匹配}
  end;
var
  f1,f2:text; {文件变量}
  s1,s2:string; {正则表达式串、待匹配串}
  str:string;
  len:integer; {正则表达式预处理后的“长度”}
  dd:array [1..80] of datatype; {预处理结果}
  pro:array [0..80,0..80] of boolean; {动态规划数组}
  fr:array [0..80,0..80] of byte;
  {FR[i,j]表示以第 j 个字符为尾的与前 i 项正则表达式匹配的最前端的字符位置}
  i,j,k,l:integer; {辅助变量}
  ok:boolean; {找到标记}
  ha:boolean;
  ans:integer;
  bt,bj:integer; {当前最优值的开始位置、长度}
procedure preprocess; {预处理，生成规划的“正则表达式”表示}
var
  i,j,k:integer;
  ch,c:char;
begin
  i:=0;
  j:=0;
  while i<length(s1) do
  begin
    inc(i);
    case s1[i] of
      '.':
      begin {处理“.”}
        inc(j);
        dd[j].md:=0;
        dd[j].mt:=1;
        dd[j].st:=#0;
        dd[j].ed:=#255;
      end;
      '*': dd[j].md:=1; {处理“*”}
      '+': {处理“+”}
      begin
        inc(j);
        dd[j]:=dd[j-1];
        dd[j].md:=1;
      end;
```

```
      '\': {处理“\”}
      begin
        inc(i);
        inc(j);
        dd[j].md:=0;
        dd[j].mt:=1;
        dd[j].st:=s1[i];
        dd[j].ed:=s1[i];
      end;
      '[': {处理“[”}
      begin
        inc(i);
        inc(j);
        dd[j].md:=0;
        dd[j].mt:=1;
        if s1[i]='^' then begin inc(i); dd[j].mt:=0 end; {如果含“^”}
        if s1[i]='\' then inc(i);
        dd[j].st:=s1[i];
        inc(i,2);
        if s1[i]='\' then inc(i);
        dd[j].ed:=s1[i];
        inc(i);
      end
      else
      begin {处理一般字符}
        inc(j);
        dd[j].st:=s1[i];
        dd[j].ed:=s1[i];
        dd[j].mt:=1;
        dd[j].md:=0;
      end;
    end;
  end;
  len:=j
end;


begin
  assign(f1,'Match.in');
  reset(f1);
  assign(f2,'Match.out');
  rewrite(f2); {文件关联、打开}

  while true do
```

```
begin
  readln(f1,s1);
  if s1='end' then break; {如果为 end 串，跳出}
  readln(f1,s2);
  preprocess; {预处理}
  ok:=false; {标记未找到}
  fillchar(pro,sizeof(pro),false);
  fillchar(pro[0],sizeof(pro[0]),true);
  fillchar(fr,sizeof(fr),0);
  bt:=maxint;
  bj:=0;
  for i:=0 to length(s2) do {赋初值}
    fr[0,i]:=i+1;
  for i:=1 to len do {分析前 i 项正则表达式}
    for j:=1 to length(s2) do {分析前 j 个字符}
    begin
      if dd[i].md=0 then {如果最后一个是一般字符}
        if (dd[i].mt=0) xor (s2[j] in [dd[i].st..dd[i].ed]) {如果匹配}
        then
        begin
          pro[i,j]:=pro[i-1,j-1]; {与去掉这个字母后的结果一致}
          if pro[i,j] then fr[i,j]:=fr[i-1,j-1]; {如果为真，设置起始点}
        end
        else pro[i,j]:=false {最后一个不匹配，则整个不匹配}
      else
      begin
        ha:=false;
        for k:=j downto 0 do {考虑前 i-1 项正则表达式与前若干项字符串匹配情况}
        begin
          if pro[i-1,k] then {如果某个为真}
          begin
            pro[i,j]:=true; {表示匹配}
            if (fr[i,j]=0) or (fr[i,j]>fr[i-1,k]) then fr[i,j]:=fr[i-1,k];
            {如果起点较早，更新之}
          end;
          if not ((dd[i].mt=0) xor (s2[k] in [dd[i].st..dd[i].ed]))
          {如果不匹配，则不考虑再退一格的情况}
          then begin ha:=false; break end;
        end;
      end;
      if (i=len) and pro[i,j] and (fr[i,j]<=bt) then
      {如果发现更好的，更新当前最优值}
      begin
        ok:=true;
```

```
          bt:=fr[i,j];
          bj:=j-bt+1;
        end;
      end;
    if ok then
      if  bj<>0  then  writeln(f2,copy(s2,1,bt-1),  '(',  copy(s2,bt,bj),  ')',
copy(s2,bt+bj,length(s2)-bt-bj+1)) {如果找到，输出}
      else writeln(f2,'()',s2)
      {对于某些理论上讲可以与空串匹配的正则表达式，应看作与第一个字符前的空串匹
配，这里作了专门处理}
    else writeln(f2,s2); {否则输出原串}
  end;
  close(f1);
  close(f2)
end.
```

程序 5　NOI'98 免费馅饼

```
{说明：}
{动态规划方程：}
{ ans[t,j]= max ( ans[t-1,j+k] )  (其中，k=-2,-1,0,1,2，且相应位置可达)}
program Pizza_For_Free {Time Limit:3000};
{免费馅饼 将允许时间扩大到 3000 秒，分值限制在 longint 范围内}
type
  link=^linetype; {指向记录一个时间单位决策数组的指针类型}
  linetype=array [1..99] of shortint; {记录一个时间单位决策的数组}
var
  f1,f2:text; {输入、输出文件变量}
  w,h:integer; {宽度、高度}
  dd:array [0..100] of array [1..99] of longint; {循环使用的数组，用于表示对应时刻、位置的
得分值}
  pro:array [0..3100] of link; {记录决策信息的动态数组}
  dt:array [0..1,1..99] of longint; {循环使用的动态规划数组}
  i,j,k,t:integer; {辅助变量}
  bt,bj:integer; {最大状态记录}
  best:longint; {最大值记录}
  maxt:integer; {最大时间}
  ans:array [1..3100] of shortint; {倒推答案时用的暂存区}
  num:integer;
  pp:longint;
  _t,_j,_v:integer; {暂存初始时间、位置、速度}
  _fz:longint; {暂存分值}
  _saved:boolean; {是否暂存标志}
```

```
procedure try_reading; {读入原始数据的过程，其中使用了分段读取的方法。读入初始下落时间不大于 t 的馅饼}
var
  t0,i,j,v:integer; {初始下落时间}
  fz:longint; {分值}
begin
  fillchar(dd[(t+100) mod 101],sizeof(dd[(t+100) mod 101]),0);
  {t 及以后时间读入的块，至迟在 t+99 时间即进入可接收状态，可对 t+100(循环观点下)单元清 0，以便下次使用}
  if _saved then {如果上次有预存}
  begin
    if _t>t then exit; {如果预存结果仍不被使用到，则返回，否则用预存结果记录新馅饼}
    _saved:=false;
    if (h-1) mod _v=0 then
    begin
      inc(dd[(_t+(h-1) div _v) mod 101,_j],_fz);
      if (_t+(h-1) div _v)>maxt then maxt:=_t+(h-1) div _v;
    end;
  end;
  if eof(f1) then exit; {文件结束就返回}

  while true do {不断读取，直到初始下落时间大于当前处理时间}
  begin
    if eof(f1) then exit;
    readln(f1,t0,j,v,fz);
    if t0>t then
    begin
      _v:=v;
      _t:=t0;
      _j:=j;
      _fz:=fz;
      _saved:=true;
      exit {暂存返回}
    end;
    if (h-1) mod v<>0 then continue; {标记时间-位置与得到的分值}
    inc(dd[(t0+(h-1) div v) mod 101,j],fz);
    if t0+(h-1) div v>maxt then maxt:=t0+(h-1) div v;
  end;
end;



begin {主程序}
```

```
assign(f1,'INPUT.TXT');
reset(f1);
assign(f2,'OUTPUT.TXT');
rewrite(f2); {文件关联、打开}

readln(f1,w,h); {读入宽、高}
for i:=0 to 3100 do {动态内存初始化}
begin
  new(pro[i]);
  fillchar(pro[i]^,sizeof(pro[i]^),0);
end;
for i:=0 to 1 do
  for j:=1 to 99 do
    dt[i,j]:=-maxlongint-1;
{赋初值}
dt[0,(1+w) div 2]:=0; {-maxlongint-1 表示该点不可达}
fillchar(dd,sizeof(dd),0);
t:=0;
maxt:=0;
_saved:=false;
try_reading;
best:=0;
bt:=0;
bj:=(w+1) div 2;
best:=dd[0,(w+1) div 2];

while true do
begin
  inc(t); {考虑下一个时间}
  try_reading; {读入数据}
  if eof(f1) and (t>maxt) and not _saved then break; {如果没有新的数据，跳出}
  for j:=1 to w do {考虑各个位置}
  begin
    dt[t mod 2,j]:=-maxlongint-1;
    for k:=-2 to 2 do {考虑 5 种移动方式}
      if (j+k>0) and (j+k<=w) and (dt[1-t mod 2,j+k]>-maxlongint-1) {如果可能}
      and (dt[1-t mod 2,j+k]+dd[t mod 101,j]>dt[t mod 2,j]) then {而且有效}
      begin
        dt[t mod 2,j]:=dt[1-t mod 2,j+k]+dd[t mod 101,j]; {更新当前最优值}
        pro[t]^[j]:=k;
      end;
    if dt[t mod 2,j]>best then {如果到目前最优，更新之}
    begin
      best:=dt[t mod 2,j];
```

```
            bt:=t;
            bj:=j;
          end;
      end;
   end;

   writeln(f2,best); {输出最大值}
   num:=0;
   j:=bj;
   for t:=bt downto 1 do {倒推求解}
   begin
      ans[t]:=-pro[t]^[j];
      inc(j,pro[t]^[j]);
   end;
   for t:=1 to bt do
      writeln(f2,ans[t]);
   close(f1);
   close(f2)
end.
```

论文附录： 本文所引题目详细内容

1、字符识别 (IOI'97)

Character Recognition

This problem requires you to write a program that performs character recognition.

Details:

Each ideal character image has 20 lines of 20 digits. Each digit is a '0' or a '1'. See Figure 1a for the layout of character images in the file.

The file FONT.DAT contains representations of 27 ideal character images in this order:

□abcdefghijklmnopqrstuvwxyz

where □ represents the space character.

The file IMAGE.DAT contains one or more potentially corrupted character images. A character image might be corrupted in these ways:

- at most one line might be duplicated (and the duplicate immediately follows)
- at most one line might be missing
- some '0' might be changed to '1'
- some '1' might be changed to '0'.

No character image will have both a duplicated line **and** a missing line. No more than 30% of the '0' and '1' will be changed in any character image in the evaluation datasets.

In the case of a duplicated line, one or both of the resulting lines may have corruptions, and the corruptions may be different.

Task:

Write a program to recognise the sequence of one or more characters in the image provided in file IMAGE.DAT using the font provided in file FONT.DAT.

Recognise a character image by choosing the font character images that require the smallest number of overall changed '1' and '0' to be corrupted to the given font image, given the most favourable assumptions about duplicated or omitted lines. Count corruptions in only the least corrupted line in the case of a duplicated line. All characters in the sample and evaluation images used are recognisable one-by-one by a well-written program. There is a unique best solution for each evaluation dataset.

A correct solution will use precisely all of the data supplied in the IMAGE.DAT input file.

Input:

Both input files begin with an integer N ($19 \leq \quad \leq 1200$) that specifies the number of lines that follow:

N
(digit1)(digit2)(digit3) … (digit20)
(digit1)(digit2)(digit3) … (digit20)
…

Each line of data is 20 digits wide. There are no spaces separating the zeros and ones.

The file FONT.DAT describes the font. FONT.DAT will always contain 541 lines. FONT.DAT may differ for each evaluation dataset.

Output:

Your program must produce a file IMAGE.OUT, which contains a single string of the characters recognised. Its format is a single line of ASCII text. The output should not contain any separator characters. If your program does not recognise a particular character, it must output a '?' in the appropriate position.

Caution: the output format specified above overrides the standard output requirements specified in the rules, which require separator spaces in output.

Scoring:

The score will be given as the percentage of characters correctly recognised.

SEE OTHER SIDE FOR SAMPLES.

Sample files:

Incomplete sample showing the *beginning* of FONT.DAT (space and 'a').

FONT.DAT
540 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000

Sample IMAGE.DAT, showing an 'a' corrupted

IMAGE.DAT
19 00000000000000000000 00000000000000000000 00000000000000000000 00000011100000000000

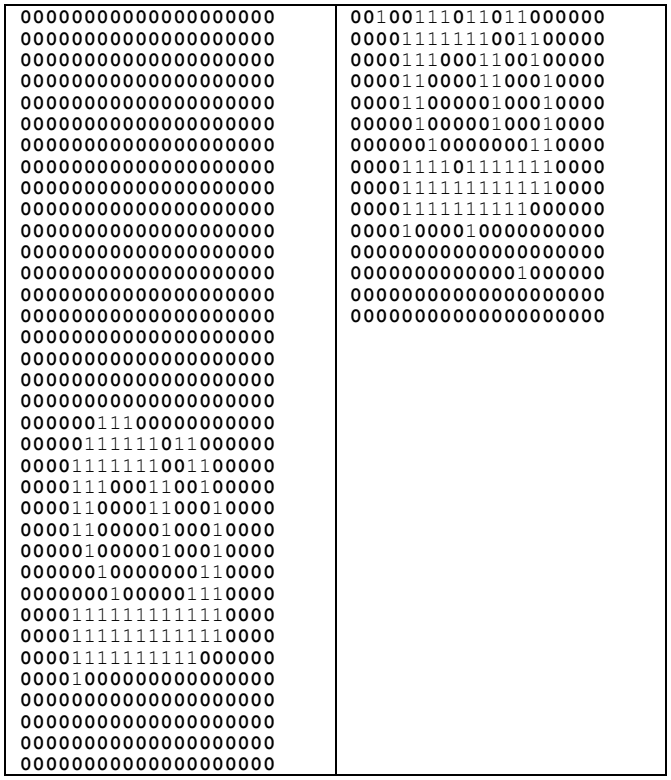


Figure 1a　　Figure 1b

Sample output:

IMAGE.OUT	Explanation
A	Recognised the single character ‘A’

Figure 2

中文：

这个题目需要你编写一个字符识别程序。

具体内容：每一个假设的字符图像(字符点阵)有 20 行，每行有 20 个“0”或“1”的数字。

FONT.DAT 文件中有 27 个按照下列顺序排列的字符图像：

□abcdefghijklmnopqrstuvwxyz

□在这里表示空格符。

文件 IMAGE.DAT 包含有一个或者多个被破损的字符图像，一个字符图形可能通过以下几种途径被破损：

- 至多有一行被复制(复制的行紧接其后)
- 至多有一行丢失
- 有些“0”可能变成“1”
- 有些“1”可能变成“0”

字符图像不会同时有一行被复制而同时又丢失一行，在测试数据中，任何一个字符图像弄反“0”和“1”的比例不超过 30%。

在行被复制的情况中，复制行和被复制行都可能破损，但破损的情形可能是

不同的。

任务：用 FONT.DAT 提供的字体对 IMAGE.DAT 文件中的一个或者多个字符序列进行识别。

在一种自己最满意的有关“行”被复制或丢失的假设下，根据实际字符图像和标准字符图像的比较，以“0”和“1”发生错误的总数越少越好为条件来识别给定的字符图像，题中所给的样例字符图像都会被一个好的程序所识别，对于一个被测数据组，有一个唯一的最佳解。

正确解应该准确使用由输入文件 IMAGE.DAT 所提供的所有行数。

输入：两个输入文件都由整数 N(19≤N≤1200)开始，该整数指出下面的行数。

N
(digit1)(digit2)(digit3) … (digit20)
(digit1)(digit2)(digit3) … (digit20)
…

每一行的数据都有 20 个码，码和码之间没有空格。

文件 FONT.DAT 描述字体。FONT.DAT 总是包含 541 行。每次 FONT.DAT 都可能是不同的。

输出：你的程序必须生成一个 IMAGE.OUT 文件。它应该包含一串识别出的字符。它的格式是一行 ASCII 码。输出结果不应含有任何分隔符，如果你的程序识别不出一个字符，则在相应的位置显示“?”。

警告：上述输出格式不遵守在规则中规定的在输出的结果中留出空格的规定。

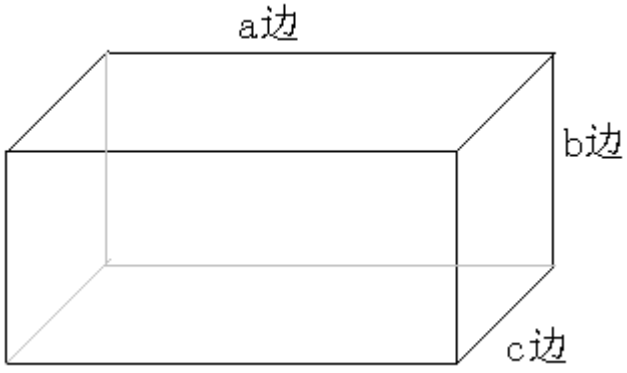
计分：根据正确识别出的字符的比例确定所得分数。

文件举例：(略，参见上面的英文试题)

2、积木游戏 (NOI’97)

SERCOI 最近设计了一种积木游戏。每个游戏者有 N 块编号依次为 1，2，3，4，…的长方体积木，它的三条不同的边分别称为“a 边”、“b 边”、“c 边”，如下图所示：

游戏规划如下：



1、 从 N 块积木中选出若干块，并将它们分成 M(1<=M<=N)堆，称为第 1 堆，第 2 堆，第 3 堆，…。每堆至少有 1 块积木，并且第 K 堆中任意一块积木的编号要大于第 K+1 堆中任意一块积木的编号(2<=K<=M)。

2、 对于每一堆积木，游戏者要将它们垂直摞成一

根柱子，并要求满足下面两个条件：

(1)除最顶上的一块积木外，任意一块积木的上表面同且仅同另一块积木的下表面接触，并且要求下面的积木的上表面能包含上面的积木的下表面，也就是说，要求下面的积木的上表面的两对边的长度分别大于等于上面的积木的两对边的长度。

(2)对于任意两块上下表面相接触的积木，下面的积木的编号要小于上面的积木的编号。

最后，根据每人所摞成的 M 根柱子的高度之和来决出胜负。

请你编一程序，寻找一种摞积木的方案，使得你所摞成的 M 根柱子的高度之和最大。

输入输出

输入文件是 INPUT.TXT。文件的第一行有两个正整数 N 和 M(1≤M≤N≤100)，分别表示积木总数和要求摞成的柱子数。这两个数之间用一个空格符隔开。接下来 N 行依次是编号从 1 到 N 的 N 个积木的尺寸，每行有三个 1 至 1000 之间的整数，分别表示该积木 a 边、b 边和 c 边的长度。同一行相邻两个数之间用一个空格符隔开。

输出文件是 OUTPUT.TXT。文件只有一行，为一个整数，表示 M 根柱子的高度之和。

样例

INPUT.TXT
4 2 **10 5 5** **8 7 7** **2 2 2** **6 6 6**

OUTPUT.TXT
24

3、多边形 (IOI'98)

多边形(Polygon)游戏是单人玩的游戏，开始的时候给定一个由 N 个顶点构成的多边形(图 1 所示的例子中，N=4)，每个顶点被赋予一个整数值，而每条边则被赋予一个符号：+(加法运算)或者*(乘法运算)，所有边依次用整数 1 到 N 标识。

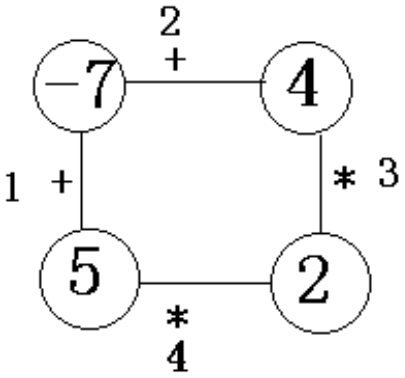


图 1 一个多边形的图形表示

首次移动(first move)，允许将某条边删除；

接下来的每次顺序移动(subsequent moves)，包括下面步骤：

1、 选出一条边 E，以及由 E 联接的顶点 V_1 和 V_2；

2、 用一个新的顶点，取代边 E 及其所联接的两个顶点 V_1 和 V_2。新顶点要赋予新的值，这个值是对 V_1 和 V_2，做由 E 所指定的运算，所得到的结果。

所有边都被删除后，只剩下一个顶点，游戏结束。游戏的得分就是该顶点的数值。

游戏实例：略。

任务：

编写一个程序，对于任意给定的多边形，计算可能的最高得分，并且列举出所有的可以导致最高得分的被首次移动的边。

输入数据：

文件 POLYGON.IN 给出的是，由 N 个顶点构成的多边形。文件包括 2 行：

第一行记录的是数值 N；

第二行包含所有边(1，…，N)分别被赋予的符号，以及嵌入到两条边之间的顶点的数值(第一个整数值对应于与 1 号、2 号边同时相连的顶点；第二个整数对应于与 2 号、3 号边同时相连的顶点；…；等等。最后一个数值对应于与 N 号、1 号边同时相连的顶点)，符号和数值之间由一个空格分隔。边的符号有 2 种：字母 t(对应于+)，字母 x(对应于*)。

输入实例：

4

t -7 t 4 x 2 x 5

这个输入文件对应于图 1 所示的多边形。第二行的第一个字符是 1 号边的符号。

输出数据：

在文件 POLYGON.OUT 的第一行，你的程序必须输出在输入文件指定条件下可能得到的最高得分。

有些边如果在首次移动中被删除，可以导致最高得分。在输出文件的第二行，要求列举出所有这样的边，而且按照升序输出，其间用一个空格分开。

输出实例：

33

1 2

4、正则表达式匹配 (ACM'97 亚洲赛区/上海赛题)

ACM International Collegiate Programming Contest

Asia Regional Contest Shanghai 1997

Problem A Pattern Matching Using Regular Expression

Input file: regular.in

A regular expression is a string which contains some normal characters and some meta characters. The meta characters include,

.
means any character

[c1-c2]
means any character between c1 and c2 (c1 and c2 two characters)

[^c1-c2]
means any character not between c1 and c2 (c1 and c2 are two characters)

*****
means the character before it can occur any times

+
means the character before it can occur any times but at least one time

\
means any character follow should be treated as normal character

You are to write a program to find the leftmost substring of a given string, so that the substring can match a given regular expression. If there are many substrings of the given string can match the regular expression, and the left positions of these substrings are same, we prefer the longest one.

Input

Every two lines of the input is a pattern-matching problem. The first line is a regular expression, and the second line is the string to be matched. Any line will be no more than 80 character. A line with only an "end" will terminate the input.

Output

For each matching problem, you should give an answer in one line. This line contains the string to be matched, but the leftmost substring that can match the regular expression should be bracketed. If no substring matches the regular expression, print the input string.

Sample Input

```
.*
asdf
f.*d.
sefdfsde
[0-9]+
asd345dsf
[^\*-\*]
**asdf**fasd
b[a-z]*r[s-u]*
abcdefghijklmnopqrstuvwxyz
[T-F]
dfkgjf
end
```

Output for the Sample Input

```
(asdf)
se(fdfsde)
asd(345)dsf
**(a)sdf**fasd
a(bcdefghijklmnopqrstu)vwxyz
dfkgjf
```

中文：

ACM 国际大学生程序设计竞赛 亚洲区竞赛 上海 1997

问题 A　正则表达式匹配

输入文件：regular.in

正则表达式是一个包含一般字符与广义字符的字符串。广义符包括：

- . 代表任何字符
- [c1-c2] 代表任何位于 c1 与 c2 之间的字符 (c1 和 c2 是两个字符)
- [^c1-c2] 代表任何不位于 c1 与 c2 之间的字符 (c1 和 c2 是两个字符)
- * 代表它之前的字符可以出现任意多次，也可以不出现
- + 代表它之前的字符可以出现任意多次但至少出现一次
- \ 代表它之后的任何一个字符应看作一般字符

你要写一个程序找出所给串中最左边的能够匹配所给正则表达式的子串。如果有多个子串能匹配该正则表达式而且它们左边开始位置相同，我们要最长的。

输入

输入文件中每两行是一个正则表达式匹配问题。前一行是一个正则表达式，后一行是待匹配的串。每行长度皆不超过 80 个字符。一个仅含“end”的串表示输入文件的结束。

输出

每一个匹配问题，你应当在一行中给出你的答案。这一行包含了待匹配的串，但最左边的能匹配该正则表达式的子串应当用括号括起来。如果没有子串匹配该正则表达式，原样输出待匹配的串即可。

样例输入

```
.*
asdf
f.*d.
sefdfsde
[0-9]+
asd345dsf
[^\*-\*]
**asdf**fasd
b[a-z]*r[s-u]*
abcdefghijklmnopqrstuvwxyz
[T-F]
dfkgjf
end
```

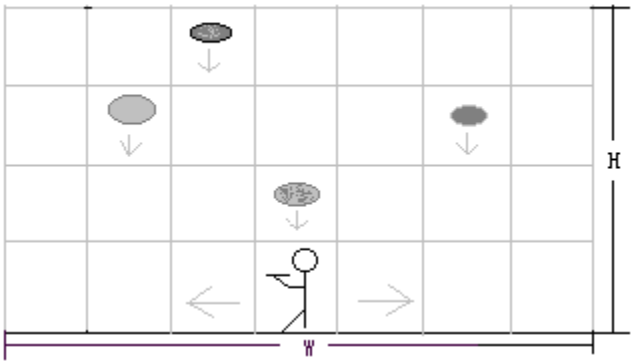
样例输出

(asdf)
se(fdfsde)
asd(345)dsf
(a)sdffasd
a(bcdefghijklmnopqrstu)vwxyz
dfkgjf

5、免费馅饼 (NOI'98)

SERKOI 最新推出了一种叫做“免费馅饼”的游戏。

游戏在一个舞台上进行。舞台的宽度为 W 格，天幕的高度为 H 格，游戏者占一格。开始时游戏者站在舞台的正中央，手里拿着一个托盘。下图为天幕的高度为 4 格时某一个时刻游戏者接馅饼的情景。



游戏开始后，从舞天幕顶端的格子中断出现馅饼并垂直落。游戏者左右移动接馅饼。游戏者每秒以向左或向右移动格或两格，也可以站原地不动。

馅饼有很多种，游戏者事先根据自己的口味，对各种馅饼依次打了分。同时，在 8-308 电脑的遥控下，各种馅饼下落的速度也是不一样的，下落速度以格/秒为单位。

当馅饼在某一秒末恰好到达游戏者所在的格子中，游戏者就收集到了这块馅饼。

写一个程序，帮助我们的游戏者收集馅饼，使得所收集馅饼的分数之和最大。

输入

输入文件的第一行是用空格隔开的两个正整数，分别给出了舞台的宽度 W（1 到 99 之间的奇数）和高度 H（1 到 100 之间的整数）。

接下来依馅饼的初始下落时间顺序给出了所有馅饼的信息。每一行给出了一块馅饼的信息。由四个正整数组成，分别表示了馅饼的初始下落时刻（0 到 1000 秒）、水平位置、下落速度（1 到 100）以及分值。游戏开始时刻为 0。从 1 开始自左向右依次对水平方向的每格编号。

输入文件中同一行相邻两项之间用一个或多个空格隔开。

输出

输出文件的第一行给出了一个正整数，表示你的程序所收集的最大分数之和。其后的每一行依时间顺序给出了游戏者每秒的决策。输出 0 表示原地不动、1 或 2 表示向右移动一步或两步、-1 或-2 表示向左移动一步或两步。输出应持续到游戏者收集完他要收集的最后一块馅饼为止。

样例输入

```
3 3
0 1 2 5
0 2 1 3
1 2 1 3
1 3 1 4
```

样例输出

```
12
-1
1
1
```