数学模型的建立、比较和应用

苏州中学 邵铮

关键字: 数学模型 算法 母函数

【摘要】

数学模型是解决实际问题的一种基本工具。将实际问题抽象成一个数学模型，运用数学工具进行求解，并将结果应用于具有相同特征的一类问题中，是解决问题的一种基本的途径。本文首先介绍了数学模型的一些性质，然后建立了三种不同的数学模型来求解一个问题，将三种数学模型相互比较，得出数学模型抽象性与高效性之间的关系，再将数学模型推广应用于另两个问题的求解，得出数学模型抽象性与可推广性之间的关系，最后总结全文，揭示出有关数学模型的一些普遍规律。

一、引论

实际问题往往是纷繁而复杂的，而其中的规律也是隐藏着的，要想直接用计算机来求解实际问题往往有一定的困难。计算机擅长的是解决数学问题。因此我们有必要将实际问题抽象成数学模型，然后再用计算机来对数学模型进行求解。

与实际问题相比，数学模型有以下几个性质：

1. 抽象性：数学模型是实际问题的一种抽象，它去除了实际问题中与问题的求解无关的部分，简明地体现了问题的本质。这一点是下面两个性质的基础。

2. 高效性：数学模型中各个量之间的关系更为清晰，容易从中找到规律，从而提高求解的效率。由于这一点是由数学模型的抽象性决定的，因此数学模型的抽象化程度对数学模型效率的高低有重要的影响，这一点将在第二部分中详细阐述。

3. 可推广性：数学模型可以推广到具有相同性质的一类问题中。换句话说，解决了一个数学模型就解决了一类实际问题。这里的"相同性质"是指相同的本质表面看似毫不相干的问题可能有着相同的本质。由于这一点也是由数学模型的抽象性决定的，因此数学模型的抽象化程度对数学模型的推广范围也有重要的影响，这一点将在第三部分中详细阐述。

二、数学模型的建立和比较

由于考虑问题的角度不同，面对同一个实际问题，可能建立起各种各样的数学模型。在各种数学模型中，我们要寻找的是效率高的模型。模型的效率同模型的抽象化程度有关，下面从一个实例中来分析它们之间的具体关系。

【多边形分割问题】将一个凸 n 边形用 n-3 条互不相交的对角线分割为 n-2 个三角形,求分割方案的总数.

如:n=5 时,有以下几种分割方案:

这道题可用以下几种方法来求解:

<1>.搜索法:

这种方法的思路是将各种分割方案全都列举出来。

显然，一组 n-3 条互不相交的对角线对应于一种分割方案，因此可把问题看作是求不同的对角线组的数目。

将 n 边形的 n 个顶点按顺时针方向编号为 1、2、3……n，则一条对角线可表示为一个数对(a1,a2)，a1、a2 分别表示对角线两端顶点的序号，a1<a2，a1 为对角线的始端，a2 为末端。

对角线在对角线组中的顺序是无关紧要的，因此，一个对角线组是一个集合，它的元素是对角线。

判断两条对角线是否相交是一个必须解决的问题。设两条对角线分别为(a1,a2)与(b1,b2)，若把表示对角线的数对看作开区间，那么两条对角线不相交的充要条件是两个区间有包含关系或他们的交集为空集。

于是，我们建立起解决本问题的第一个数学模型：

已知：n 的值,

一个集合由(n-3)个不同的开区间(i,j)组成,

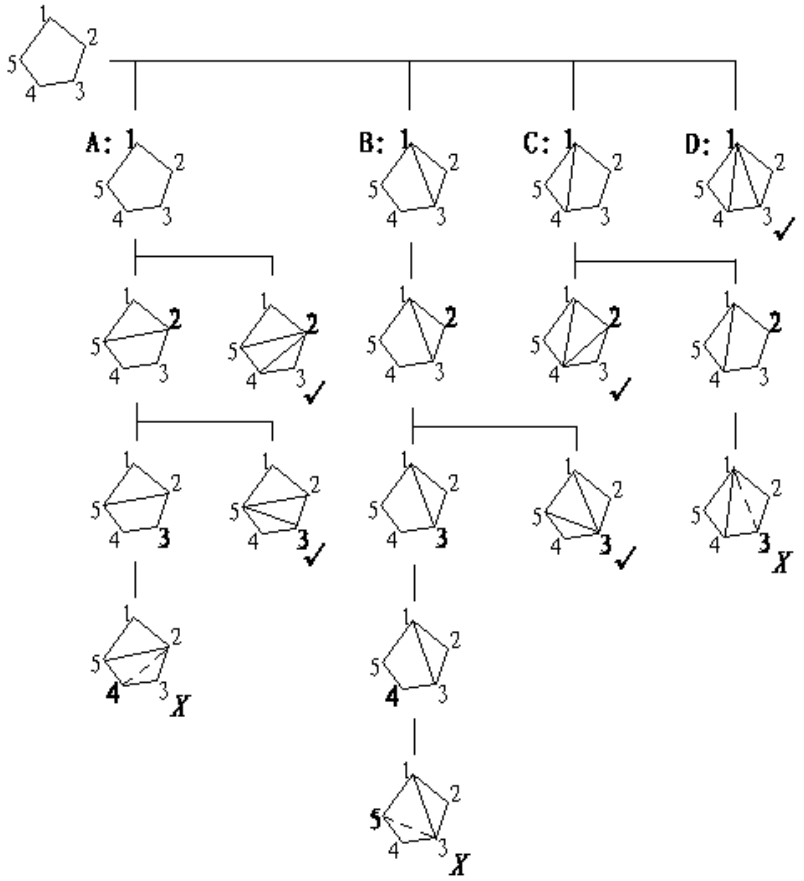
i∈{1..n-2}, j∈{i+2..n},(i≠1)或(j≠n)

同一个集合中任两个不同的开区间(i1,j1),(i2,j2)满足：

((i1,j1)∩(i2,j2)=空集)或((i1,j1)包含(i2,j2))或((i2,j2)包含(i1,j1))

求：不同的集合的个数

搜索时，先考虑以顶点 1 为始端的对角线，可以不连任何对角线(图一中 A)，也可以连(1,3)(图一中 B)，或连(1,4)(图一中 C)，或同时连(1,3)(1,4) (图一中 D)。对于每一种情况，再考虑以顶点 2 为始端的对角线，依此类推。当得到 n-3 条互不相交的对角线时，便找到了一种方案(参见图一)。



图一

在考虑以顶点 i 为始端的对角线时，有以下几条规则必须遵循：

1.与原有对角线相交的对角线不得选取。

2.当 i>=3 时，若顶点 i-1 为始端的对角线一条都未连，则对角线(i-2,i)必须是已经连的。

3.对角线的末端顶点序号必须大于 i。否则，顶点 i 将成为对角线的末端，另一个顶点 j(j<i)成为对角线的始端，这条对角线已在考虑以顶点 j 为始端的对角线时考虑过了，再考虑将引起重复。

按照以上三条规则，即可得到如图一的搜索树(图中打√的叶结点为不同的分割方案)。

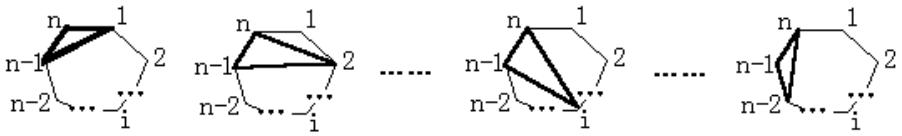
搜索法的数学模型较为复杂，用它可以求出具体方案，但它的抽象化程度不高，导致了求解时的低效率。为了使用上面的规则 2 来提高效率，求解过程还是从多边形及其对角线本身来考虑的，数学模型的作用仅体现在判断对角线是否相交上。用该方法编制的程序在 n 稍大时速度就很慢。(n=12 时已需运行时间 16.2 秒(486DX2/80)，测试结果见附表一。)

<2>.递推法:

上一种方法的数学模型中有很多与问题的要求无关的内容(如对角线的表示、对角线组的表示、每种具体方案)。在递推法建立的数学模型中，我们只考虑凸 n 边形的分割方案总数。

设 k 边形的分割方案总数为 A_k，于是得到 A 数列：$A_3,A_4,A_5...$

考虑 n 边形的分割方案总数 A_n。任取 n 边形的一条边，不妨取边(n-1,n), 若在某一种分割方案中，边(n-1,n)属于三角形(i,n-1,n),那么就将分割方案归入第 i 类，如图二所示。



图二

设第 i 类方案总数为 Bi，则

$$A_n = \sum_{i=1}^{n-2} B_i \quad ①$$

计算 Bi 可用如下方法：

对于第 i 类的方案，原 n 边形已被分割为一个 i+1 边形与一个(n-i)边形，下面的工作分为两步，第一步是将 i+1 边形分割为三角形，有 A_{i+1} 种方案，第二步是将(n-i)边形分割为三角形，有 A_{n-i} 种方案。为了表达的方便，令 $A_2=1$，于是有

$$B_i = A_{i+1} * A_{n-i} \quad ②$$

将②代入①得：

$$A_n = \sum_{i=1}^{n-2} (A_{i+1} * A_{n-i}) = \sum_{i=2}^{n-1} (A_i * A_{n+1-i}) \quad ③$$

于是，问题的数学模型即为：

已知：n 的值及数列 A_2,A_3,A_4……，

该数列满足：

$$A_2=1$$

$$A_j = \sum_{i=2}^{j-1} (A_i * A_{j+1-i}),\ j>2$$

求：A_n

利用这个模型，我们即可很方便地依次求出 $A_3,A_4 \ldots A_n$。

递推法的数学模型比搜索法的简明，抽象化程度更高，效率也高得多。用递推法编制的程序已能应付中等数据，在 n<100 时不超过一秒。但当 n 很大时仍然很慢，n=250 时需 18.7 秒(486DX2/80)，测试结果见附表一。

<3>.母函数法：

上一种方法的数学模型中已去除了很多与问题的要求无关的内容，但同时，问题只要求 An,而上述方法却将 $A_3 \sim A_n$ 都求出了。能否不求 $A_3 \sim A_{n-1}$ 而直接由 n 求出 An 呢？下面用母函数这种数学模型来解决这个问题。

将 A_2,A_3,A_4……作为幂级数的系数，令

$$Y(x) = \sum_{i=2}^{+\infty} A_i * x^i = A_2 * x^2 + A_3 * x^3 + \ldots\ldots \quad ④$$

如果能解出 Y(x),那么也就求出了 An.

为了求 Y(x),我们来看一下 Y(x)^2 的值:

$$Y^2(x) = \sum_{i=4}^{+\infty} [\sum_{j=2}^{i-2} (A_j * A_{i-j}) * x^i]$$

而 $\sum_{j=2}^{i-2}(A_j * A_{i-j}) = A_{i-1}$,因此有:

$Y^2(x) = A_3 * x^4 + A_4 * x^5 + \ldots\ldots$ ⑤

⑤-④*x 得:

$Y^2(x) - x * Y(x) + A_2 * x^3 = 0$

将 $A_2 = 1$ 代入，解出 $Y(x)$：

$$Y(x) = \frac{x \pm \sqrt{x^2 - 4x^3}}{2} = x * \frac{1 \pm \sqrt{1 - 4x}}{2}$$

$\sqrt{1-4x} = 1 - 2x - 2x^2 - 4x^3 \ldots\ldots$ 各项系数均为负数，而 Ai>0,故：

$$Y(x) = x * \frac{1 - \sqrt{1 - 4x}}{2}$$ ⑥，

其中 $\sqrt{1-4x} = \sum_{k=0}^{+\infty}\left[\frac{1}{2}\right]_k * (-4x)^k$, $\left[\frac{1}{2}\right]_k = \frac{\frac{1}{2} * (\frac{1}{2} - 1) * (\frac{1}{2} - 2) * \ldots * (\frac{1}{2} - k + 1)}{k!}$

于是，$$Y(x) = x * \frac{1 - \sum_{k=0}^{+\infty}\left[\frac{1}{2}\right]_k (-4x)^k}{2}$$

$$= x * \frac{1 - (1 + \left[\frac{1}{2}\right]_1 (-4x) + \left[\frac{1}{2}\right]_k (-4x)^2 + \ldots + \left[\frac{1}{2}\right]_k (-4x)^k + \ldots)}{2}$$

$$= -\frac{(-4)\left[\frac{1}{2}\right]_1 x^2}{2} - \frac{(-4)^2\left[\frac{1}{2}\right]_2 x^3}{2} - \ldots - \frac{(-4)^{k-1}\left[\frac{1}{2}\right]_{k-1} x^k}{2} - \ldots$$

所以，$$A_k = -\frac{(-4)^{k-1}\left[\frac{1}{2}\right]_{k-1}}{2}$$

$$= \frac{\frac{1}{2}(\frac{1}{2} - 1)(\frac{1}{2} - 2)\ldots(\frac{1}{2} + 2 - k)}{(k-1)!} * (-1)^k * 2^{2k-3}$$

$$= \frac{1 * (-1) * (-3) * \ldots * (5 - 2k)}{(k-1)!} * (-1)^k * 2^{k-2}$$

$$= \frac{1 * 3 * 5 * \ldots * (2k - 5)}{(k-1)!} * 2^{k-2}$$

$$= \frac{1 * 3 * 5 * \ldots * (2k-5) * 2 * 4 * 6 * \ldots * (2k-4)}{(k-1)!(k-2)!}$$

$$= \frac{(2k-4)!}{(k-1)!(k-2)!}$$

$$= \frac{1}{k-1} * C_{2k-4}^{k-2}$$

于是得出了由 n 直接求出 An 的数学模型：

已知：n 的值，

$$\mathbf{A_n = \frac{C_{2n-4}^{n-2}}{n-1}}$$ ⑦

求：A_n

求解时用公式⑦可直接计算 A_n。

在三个数学模型中，这一个表达最为简洁，抽象化程度最高。用它来求解的效率也最高。n=1000 时不超过 1 秒,n=5000 时也仅需 14.7 秒(486DX2/80)，测试结果见附表一与附表二。

搜索法作为一种最基本的方法，建立在一个较为复杂的数学模型之上，它的特点是可以求出每一种分割方法，但用这种方法来求方案总数显然针对性不强，因此效率很低。递推法是建立在数列这个数学模型之上的，由于去除了很多不必要的因素，效率大为提高，对于 n<=300 时有较好的效果。利用母函数这种数学模型求解，是对递推法的一种数学优化，得出了更为简洁的结论，当 n>300 时充分显示出其优势。

从以上的分析中可以得出这样的结论：数学模型的抽象化程度越高，它的效率越高。这个结论很容易理解，因为抽象化程度越高，数学模型中与问题无关的成分就越少，于是效率也就越高。相反的，若抽象化程度不够高，则数学模型中含有较多的与问题无关的成分，那么，效率也就要低一些。

三、数学模型的推广和应用

数学模型具有可推广性。

数学模型是建立在问题本质的基础上的，而不是建立在问题的表面现象上的。因此，虽然两个问题表面毫无关系，只要它们有相同的本质，就可以用相同的数学模型求解。然而，要看到两个问题有相同的本质并不是一件容易的事。这需要我们抛开问题的表面现象，仔细地比较分析，在问题之间建立对应关系。

数学模型的可推广性与数学模型的抽象化程度有着密切的关系。为解决同一个问题而建立起的不同的数学模型可能具有不同的可推广性。

下面将由母函数建立起的数学模型应用于另一些问题的求解。

【树的计数问题】求具有 n 个结点的二叉树的数目。

设具有 k 个结点的的二叉树的数目为 Dk，则

- 当 k=0 时，是一棵空树，只有一种。
- 当 k>0 时，二叉树可分为根结点、具有 i 个结点的左子树与具有 k-1-i 个结点的右子树。于是具有 k 个结点的二叉树的数目等于具有 i 个结点的二叉树的数目与具有 k-1-i 个结点的二叉树的数目的乘积。

将以上的分析写成公式，就是：

$$D_o = 1$$

$$D_k = \sum_{i=0}^{k-1} (D_i * D_{k-1-i})$$

比较上文中 A 数列与这里的 D 数列可知 $D_n = A_{n+2}$,于是将上文中的数学模

型(⑦式)稍加变换，即得：

$$D_n = \frac{C_{2n}^n}{n+1} \quad ⑧$$

至此，我们已将这个问题用上面的数学模型解决了。这个问题与[多边形分割问题]具有相同的本质，即它们计数的规律是一致的，因此，它们可用相同的数学模型求解。

为了将这种数学模型进一步推广，我们再将上一个问题分析一下：将每棵二叉树的 n 个结点一一编号，使每棵二叉树的前序序列都是 1,2,3…,n。由于前序序列与中序序列可唯一确定一棵二叉树，所以每棵二叉树的中序序列与其它的二叉树都是不同的。(一旦相同，那么这两棵二叉树就是同一棵二叉树了。)

另外，对于一棵前序序列确定的二叉树，它的中序序列可以由前序序列进栈与出栈生成。于是该数学模型又可直接用于下面问题的求解。

【火车进出栈问题】一列火车 n 节车厢，依次编号为 1,2,3,…,n。每节车厢有两种运动方式，进栈与出栈，问 n 节车厢出栈的可能排列方式有多少种。

将这个问题与上一个问题比较一下：列车原始的排列状态(1,2,3,…,n)正是二叉树的前序序列；列车车厢的进栈与出栈对应于二叉树结点的进栈与出栈；列车出栈后的排列状态正是二叉树的中序序列。

将两道题对应起来看，不难发现，列车出栈后的可能排列方式的数目就是二叉树的中序序列的数目，也就是二叉树的数目。

设 n 节车厢的排列方式有 En 种，则

$$E_n = \frac{C_{2n}^n}{n+1} \quad ⑨$$

于是，我们又用相同的数学模型解决了这个问题。

将数学模型推广到[树的计数问题]时，我们只是比较了相似的递推公式，可以说是一种简单的推广。而推广到[火车进出栈问题]时，则是从[树的计数问题]出发，将两个问题对应起来看，进行了很多逻辑分析，相比之下要复杂一些。事实上，很多数学模型的推广应用都需要进行仔细的分析。

从一个问题[多边形分割问题]出发建立起的数学模型 $X_n = \frac{C_{2n}^n}{n+1}$，公式中已完全略去了分割的具体内容，只留下了问题的本质：计数。由于公式表达了计数方法的实质内涵（ $X_0 = 0; X_k = \sum_{i=0}^{k-1} (X_i * X_{k-1-i})$ ）,于是就给了它进一步广泛应用于一类问题求解（外延）的可能。

再考虑一下[多边形分割问题]中搜索法的数学模型。在这两个问题中，搜索法的数学模型显然是不适用的。它包含着多边形的每一种具体的分割方案，没有

很好的体现问题计数的本质，因此影响了这种数学模型的可推广性。在这两个问题中，没有相应的概念对应于多边形的分割方案，于是，搜索法的数学模型便对这两个问题无能为力了。而数列与母函数两种方法的数学模型仍能应用于这两个问题。这正是由于后两种方法的数学模型更加抽象，所以更有利于它们的推广

四、总结

以上三个实例充分说明了数学模型的高效性、可推广性以及它们与抽象性之间的关系。

数学模型具有高效性。从实际问题中建立起来的数学模型可以去除无关的内容，关系清晰，有利于效率的提高。

数学模型具有可推广性。从实际问题中建立求解的数学模型，一个数学模型建立后，往往能将其应用于一类实际问题中。乍一看[分割多边形]与[火车进出栈]没有什么联系，但通过对模型的理解可以发现两个问题有着密切的内在联系：它们是可以用相同的数学模型来求解的。

数学模型的高效性与其抽象性是紧密联系的。数学模型越是抽象，它的效率也就越高。数学模型的可推广性与其抽象性也是紧密联系的。数学模型越是抽象它也就越容易被广泛应用。

【附件】

附表一:按以上三种数学模型设计的程序的运行时间的比较

n	运行时间(秒)(486/80)			结果
	搜索法	递推法	母函数法	
5	0.0	0.0	0.0	5
8	0.1	0.0	0.0	132
10	0.6	0.0	0.0	1430
11	3.2	0.0	0.0	4862
12	16.8	0.0	0.0	16796
20		0.0	0.0	477638700
30		0.1	0.0	263747951750360
40		0.1	0.0	176733862787006701400
50		0.1	0.0	131327898242169365477991900
70		0.3	0.0	862189239989602857261856406637011085 00
100		0.8	0.0	5774335806960135778218770060804285633402073162475661100 0
150		3.1	0.0	3959313147057001992888490018878757680451363792611793474 902551970920541958964206938 7800
200		8.3	0.1	3249701714469247204061030419891100129328703540371004596 972565531458474030562929950769133018913041197185787156 7302000
250		18.7	0.1	2942109465114274900932013291224718543203864499126811120 331716878369694940021100392829583154627202225799961741 962546561436757757673959435471617 2000
300		37.1	0.1	2833615951128645491952141298699350894649246764901164418 208859862469151903255965070803736549992753202965439344 706962132218771245433367832310452689722580702922416256 3399190436400

附表二:母函数法在大数据下的运行时间与结果

n	运行时间 (秒,486/80)	结果
500	0.2	3392161481258547533436328676042834151806671371051948224646322289022038948 4961666016394087658935561710828507304323072584219401165213694125111808676 7433831117255251093952216676752181549754039254079009518747834466367785846 2596953622134102520200986176150660638780745851952786342361786271048679068 0000
1000	0.6	1282659123120329389992418907864563882089032052318971850610071333658659691 7650636508971392269636384330131169530183206423519818758891241594708333432 8015864565893550987852108368101709312590228295938009433241620526247192140 9995611627511580863150740180266996754602824885598197887476283926138637334 4807651604630045903721337776816153681064661207184425277845724281859993280 9910085516200019191596126573215746216258241717105949596945086725752677939 9630744193300422624617858972825189717428936822555053619495392164363039745 4492990445740264272985056960596626000194076108609551836149060341111411420 3867955760000
5000	14.7	1990533983292846325166735064542780033892729043055532801808728231002712118 7676340856613205258009823911061689822819966058800117396543598135427586955 9792778429614014690786606171171528320083045044671489924972047501523334246 8138724573840895739739033811420784581508456866514843728085684023096771538 1553578655967556776543982717552905537280360991836287529653206624959870630 8121895633339730969725161266875649077021947002631530501156605401466085759 7536352915369221824992631352492185831978570219673534486780746350434887270 3806682765573357532768717863477686122111338669153041998736992669734521138 4136884623921590225108813107833859413755075491924954296242363596594466667 1446673358771831246640688142040703856338776629977766659654255268159378975 1044519117369900869457690210352595228237709489430574007459436019028016411 0038827696035585915387351199097375301331556293580890493918098622380755233 3892680521693312997068069144010511282780445912971074861274051992834639631 0561216451805165334432417634618203226858919012500057666732553341830111147 0819533812120988010445061003160557018172554817337108394059108173915553582 1737987646875652182998581745666283678405193386287633644237910478216042546 3668803378393982395552054129650108630734033416898205271951432275542407693 1819721455040492976792101947671393514768860471507412891459327520118017269 6786877749972031468886291116599954859230193707272332890105705243614350673 5132885611033291722431843274758902459024223795911169335414024073046732892 7395645459224329052247990854644972024029972827331377313107196086221062765 0188202833681955696454854968868373243539755357093842908311125187443209114 0686683008816625997915385029705586103119596051190592857213107922653357349 7461860596111378869832344698856089798444316515490995417460293855758521378 0659674972152322112734313586718289148701156895171132685679870431003809667 4186956424692388366951410846587759001196659372267179038623797279196316562 3988309844128732444707420528465376060417840530884098880455141863912466417 6032592842849445923263161630959760172685607996743155249441458740184540341 6372467040622320018510892266507167328210681172963043341422470770335588144 8203281065661300736427314789935878982943581444410274031000848491803374826 2914234177331202318998592244539101401954666570026383034960883253619564875 1981624029939353157525782759785276380074202722356998363216687878275696045 9799214540286045928686781434155385008768086213261980678561187332233776048 5664946553533366401151012142801380925362116746164301057451528028965946453 9972118062862177079566215629850386783557684491180752622175613789741648825 9143313542109632709368336593742684914689724272085005469649013886496227350 8257492244688170498625618918459291917844876832932013180312437483376087705 8196194803111496823543186375259372960899823055880506318903220600017289601 4818245773092045746539907057236677380540322691698615959422148103261747037 1581495719930597851695669555189979332882416754183716357001209090944294741 4739729976104880382195071324370634650548082582843292378198930180928100582 81353800000

【程序】

1.多边形分割问题 搜索法 (sousuo.pas)

```
{$A+,B-,D-,E-,F-,G+,I-,L-,N-,O-,P-,Q-,R-,S-,T-,V+,X+,Y-}
{$M 16384,0,655360}
Program SouSuo;
Const
  Max=30;
Type
  TPara=record l,r:integer;end; {区间类型}
Var
   method:Longint; {方案总数}
   Para:array[1..Max]of TPara; {区间组}
   n:integer;
   time:Longint;

Function M(a,b:integer):integer;{高精度整数类型}
begin if a<b then m:=b else m:=a;end;

Procedure Make; {搜索多边形的所有分割方案}
Var i,j:integer;
    sp,lp1,lp2:integer;
  Function Connect:boolean; {判断新加入区间组的区间是否与原有的区间有冲突}
  var i,j,k:integer;b1,b2:boolean;
  begin
    j:=para[sp].l;k:=para[sp].r;
    Connect:=true;
    for i:=1 to sp-1 do
      begin
        if (j=para[i].l)or(j=para[i].r) then continue;
        if (k=para[i].l)or(k=para[i].r) then continue;
        if ((j>para[i].l)and(j<para[i].r))xor
           ((k>para[i].l)and(k<para[i].r))
          then exit;
      end;
    Connect:=false;
  end;

  Function PreFalse:boolean; {检验是否有其它的冲突}
  var i:integer;j,k:integer;
  begin
```

```
  prefalse:=false;j:=para[sp].l;
  if j<=2 then exit;
  for i:=1 to sp do
    if (para[i].l=j-1)or(para[i].r=j-1) then exit;
  k:=j;j:=k-2;
  for i:=1 to sp do
    if (para[i].l=j)and(para[i].r=k) then exit;
  PreFalse:=true;
 end;

 Function Pop:boolean;forward;

 Function Push:boolean; {入栈}
 begin
  inc(sp);
  Para[sp].l:=lp1;Para[sp].r:=lp2;
  Push:=((lp1=1)and(lp2=n))or(lp1>n)or(lp2>n)or connect;
  if prefalse then
    begin Push:=true;pop;exit;end;
  inc(lp2);
  if lp2>n then
    begin inc(lp1);lp2:=lp1+2;end;
 end;

 Function Pop:boolean; {出栈}
 begin
  if sp=0 then
    begin dec(sp);pop:=false;exit;end;
  lp1:=Para[sp].l;lp2:=para[sp].r;dec(sp);
  inc(lp2);
  if lp2>n then
    begin inc(lp1);lp2:=lp1+2;end;
  Pop:=(lp1>n)or(lp2>n);
 end;
begin
 sp:=0; {栈顶指针置 0}
 lp1:=1;lp2:=3;
 method:=0;
```

```
  while (sp>=0) do
  begin
    if Push then while pop do;
    if sp=n-3 then {获得了一种方案}
      begin
        method:=method+1;
        while pop do;
      end;
  end;
  writeln('Total: ',method);
end;

var i:integer;s:string;
BEGIN
  write('Input N: ');
  readln(n);{输入多边形边数}
  time:=MemL[$40:$6c];
  if n<3 then writeln('Total: 0')
    else if n=3 then writeln('Total: 1')
    else MAKE; {搜索多边形的所有分割方案}
  writeln('Time: ',(Meml[$40:$6c]-time)/18.2:5:1);
    {输出所用的时间}
END.
```

2.多边形分割问题 递推法 (ditui.pas)

```
{$A+,B-,D-,E-,F-,G+,I-,L-,N+,O-,P-,Q-,R-,S-,T-,V+,X+,Y-}
{$M 16384,0,655360}
Program DiTui;
Const
     Len=100;Max=300;
Type
    Th=array[-1..Len+1]of integer;{高精度整数类型}
Var
   method:array[1..Max]of Th; {i 边形分割方案总数为 method[i]}
   n:integer; {要求的多边形的边数}
   time:Longint;

Function M(a,b:integer):integer; {取最大值}
```

```
begin if a<b then m:=b else m:=a;end;

Procedure Add(var a:Th;b:Th); {a:=a+b;a,b 为高精度整数类型}
var i,j:integer;
begin
  j:=0;
  a[-1]:=m(a[-1],b[-1]);
  for i:=0 to a[-1] do
    begin inc(j,a[i]+b[i]);
          a[i]:=j mod 10000; {每位 integer 存 4 位十进制数}
          j:=j div 10000;
    end;
  if j<>0 then
    begin inc(a[-1]);a[a[-1]]:=j;end;
end;

Procedure Mul(a,b:Th;var c:Th); {c:=a*b;a,b,c 为高精度整数类型}
var i,j:integer;k:Longint;
begin
  fillchar(c,sizeof(Th),0);
  for i:=0 to a[-1] do
    begin
      k:=0;
      for j:=0 to b[-1] do
        if i+j<=Len then
        begin
          inc(k,longint(a[i])*b[j]+c[i+j]);
          c[i+j]:=k mod 10000;
          k:=k div 10000;
        end;
      inc(c[i+b[-1]+1],k);
    end;
  c[-1]:=a[-1]+b[-1];
  if c[c[-1]+1]<>0 then inc(c[-1]);
end;

Procedure OutHigh(a:Th); {输出高精度整数}
var s:string[4];i,j:integer;
```

```
begin
  write('Total: ');
  j:=a[-1];write(a[j]);
  for i:=j-1 downto 0 do
    begin
      str(a[i],s);while s[0]<#4 do s:='0'+s;
      write(s);
    end;
  writeln;
end;
Procedure Make; {递推计算多边形分割总数}
var i,j:integer;a:Th;
begin
  fillchar(method,sizeof(method),0);
  method[2,0]:=1;
  method[3,0]:=1;
  fillchar(a,sizeof(a),0);
  for i:=4 to N do
    for j:=1 to i-2 do
    begin
      mul(method[j+1],method[i-j],a);
      Add(method[i],a);
    end;
  OutHigh(method[n]);
end;

var i:integer;s:string;
BEGIN
  write('Input N: ');
  readln(n); {输入多边形边数}
  time:=MemL[$40:$6c];
  if n<3 then writeln('Total: 0')
    else MAKE;{递推计算多边形分割总数}
  writeln('Time: ',(Meml[$40:$6c]-time)/18.2:5:1);
    {输出所用的时间}
END.
```

3.多边形分割问题 母函数法 见 muhanshu.Pas

```
{$A+,B-,D-,E-,F-,G+,I-,L-,N+,O-,P-,Q-,R-,S-,T-,V+,X+,Y-}
{$M 16384,0,655360}
Program MuHanShu;
Const
    Len=1400;Max=6000;
Type
   Th=array[0..Len+1]of integer;{高精度整数类型 1,按位存储}
   Ty=array[0..Max]of integer;{高精度整数类型 2,按因数存储}
Var
  fi,fo:text;fin,fon:string;
  n:integer;
  time:Longint;

Procedure Mul(var a:Th;b:integer); {a:=a*b;a 为高精度整数类型 1}
Var i:integer;k:Longint;
begin
 k:=0;
 for i:=1 to a[0] do
   begin
     k:=k+a[i]*longint(b);
     a[i]:=k mod 10000;
     k:=k div 10000;
   end;
 if k<>0 then
   begin inc(a[0]);a[a[0]]:=k;end;
end;

Function MaxPublic(a,b:integer):integer; {a,b 的最大公因数}
var i:integer;
begin
 repeat
   a:=a mod b;
   if a=0 then break;
   b:=b mod a;
 until b=0;
 MaxPublic:=a+b;
end;
```

```
Procedure Divide(var k:Ty;h:integer);{k:=k div h;k 为高精度整数类型 2}
Var i,j:integer;
begin
  for i:=1 to k[0] do
    if k[i] mod h =0 then
      begin k[i]:=k[i] div h;
           if k[i]=1 then begin k[i]:=k[k[0]];dec(k[0]);end;
           exit;
      end;
  for i:=k[0] downto 1 do
    if MaxPublic(k[i],h)>1 then
     begin
      j:=MaxPublic(k[i],h);
      h:=h div j;k[i]:=k[i] div j;
      if k[i]=1 then begin k[i]:=k[k[0]];dec(k[0]);end;
      if h=1 then exit;
     end;
end;

Procedure translate(k:Ty;var a:Th);{a:=k;a 为高精度整数类型 1,k 为高精度整数类
型 2}
Var i:integer;
begin
  a[1]:=1;a[0]:=1;
  for i:=1 to k[0] do  mul(a,k[i]);
end;

Procedure Make;{按公式计算多边形分割总数}
Var i,j:integer;k:Ty;a:Th;s:string[4];
begin
  k[0]:=n-2;
  for i:=1 to n-2 do k[i]:=(2*n-3-i);
  for i:=n-2 downto 2 do divide(k,i);
  divide(k,n-1);

  translate(k,a);

  write('Total: ');
```

```
  j:=a[0];write(a[j]);
  for i:=j-1 downto 1 do
    begin
      str(a[i],s);while s[0]<#4 do s:='0'+s;
      write(s);
    end;
  writeln;
end;

var i:integer;s:string;
BEGIN
  write('Input N(<=',Max,'): ');
  readln(n); {输入多边形边数}
  time:=MemL[$40:$6c];
  if n<3 then writeln('Total: 0')
    else MAKE; {按公式计算多边形分割总数}
  writeln('Time: ',(Meml[$40:$6c]-time)/18.2:5:1);
    {输出所用的时间}
END.
```

4.树的计数问题、火车进出栈问题 (tuiguang.pas)

```
{$A+,B-,D-,E-,F-,G+,I-,L-,N+,O-,P-,Q-,R-,S-,T-,V+,X+,Y-}
{$M 16384,0,655360}
Program TuiGuang;
Const
     Len=1400;Max=5002;
Type
    Th=array[0..Len+1]of integer;{高精度整数类型 1,按位存储}
    Ty=array[0..Max]of integer;{高精度整数类型 2,按因数存储}
Var
   fi,fo:text;fin,fon:string;
   n:integer;
   time:Longint;

Procedure Mul(var a:Th;b:integer); {a:=a*b;a 为高精度整数类型 1}
Var i:integer;k:Longint;
begin
  k:=0;
```

```
  for i:=1 to a[0] do
    begin
      k:=k+a[i]*longint(b);
      a[i]:=k mod 10000;
      k:=k div 10000;
    end;
  if k<>0 then
    begin inc(a[0]);a[a[0]]:=k;end;
end;

Function MaxPublic(a,b:integer):integer; {a,b 的最大公因数}
var i:integer;
begin
  repeat
    a:=a mod b;
    if a=0 then break;
    b:=b mod a;
  until b=0;
  MaxPublic:=a+b;
end;

Procedure Divide(var k:Ty;h:integer);{k:=k div h;k 为高精度整数类型 2}
Var i,j:integer;
begin
  for i:=1 to k[0] do
    if k[i] mod h =0 then
       begin k[i]:=k[i] div h;
             if k[i]=1 then begin k[i]:=k[k[0]];dec(k[0]);end;
             exit;
       end;
  for i:=k[0] downto 1 do
    if MaxPublic(k[i],h)>1 then
     begin
       j:=MaxPublic(k[i],h);
       h:=h div j;k[i]:=k[i] div j;
       if k[i]=1 then begin k[i]:=k[k[0]];dec(k[0]);end;
       if h=1 then exit;
     end;
```

```
end;

Procedure translate(k:Ty;var a:Th);{a:=k;a 为高精度整数类型 1,k 为高精度整数类
型 2}
Var i:integer;
begin
  a[1]:=1;a[0]:=1;
  for i:=1 to k[0] do  mul(a,k[i]);
end;

Procedure Make;{按公式计算}
Var i,j:integer;k:Ty;a:Th;s:string[4];
begin
  k[0]:=n-2;
  for i:=1 to n-2 do k[i]:=(2*n-3-i);
  for i:=n-2 downto 2 do divide(k,i);
  divide(k,n-1);

  translate(k,a);

  write('Total: ');
  j:=a[0];write(a[j]);
  for i:=j-1 downto 1 do
    begin
      str(a[i],s);while s[0]<#4 do s:='0'+s;
      write(s);
    end;
  writeln;
end;

var i:integer;s:string;
BEGIN
  write('Input N(<=',Max-2,'): ');
  readln(n); {输入 N}
  inc(n,2);
  time:=MemL[$40:$6c];
  MAKE; {按公式计算}
  writeln('Time: ',(Meml[$40:$6c]-time)/18.2:5:1);
```

```
  {输出所用的时间}
END.
```

【参考书目】

1. 《信息学奥林匹克》1998.1-2 中国计算机学会普及工作委员会 、TSINGHUA UNIVERSITY ACM STUDENT CHAPTER 主办，第 87 页、第 93-94 页；

2. 《数据结构》（第二版），严蔚敏、吴伟民编著，清华大学出版社 1992 年 6 月，第 150-154 页；

3. 《中学生数学建模读本》，孔凡海编著，江苏教育出版社 1998 年 1 月。