**卡特兰数**

卡特兰数有四个公式

建议直接学公式4

##公式一

递归公式

h(0)=h(1)=1

h(n)= h(0)\*h(n-1)+h(1)\*h(n-2) + ... + h(n-1)\*h(0) (n>=2)

如果我们用这个公式显然我们要使用递归算法，那么数据一大就在时空上很麻烦

##公式二

递推公式

h(n)=h(n-1)\*(4\*n-2)/(n+1)

这个公式应用递推，看上起十分和善

但对大数据呢？

我们注意到大数据的时候h(n)会很大，这时候题目一般会让你对某素数取模（当然你可以打高精度（划掉））

但你在取模过程中难保一个h(n)%mod=0

那么根据公式下面所有的数都会等于0，于是你就愉快的WA了

##公式三

组合数公式1

h(n)=C(2n,n)/(n+1) (n=0,1,2,...)

卡特兰数可以与组合数联系起来，得到上面的公式

而组合数就是一个杨辉三角，可以递推得到（这个不属于这道题的讨论范围我假装你们都会（逃））

但我们发现对于大数据你要取模，而对于除法你是没办法用膜的性质的（当然你可以应用逆元（划掉）），所以造成了麻烦

**##公式四**

组合数公式2

h(n)=c(2n,n)-c(2n,n-1) (n=0,1,2,...)

与组合数公式1不同这个是两个组合数的减法

减法是可以用膜的性质的，于是你可以愉快的AC了。

所以我写了这么多就是想说，对于一个特定的任务，可能会有很多方法求解，但其实只要稍稍分析一下就会发现有一种方法是通用而优美的，我在没认真思考前都是记的四个公式，但**是有一天我真的认真想过后才发现其实我就记住公式四就好了。**

所以学习啊，还是要学会认真思（tou）考（lan）

卡特兰数证明:

假设要到达(n,n)且不越过y=x可站在线上有C(2n,n)-C(2n,n-1)种走法

将越过y=x的路线越过y=x+1的部分关于y=x+1对称必最终到达（n-1，n+1）

而从原点出发到（n-1，n+1）有C(2n,n-1)种走法

要到（n-1，n+1）必触及y=x+1，所以将越过y=x+1的部分关于y=x+1对称终于点（n，n）

所以终于点（n，n）触及y=x+1的路线关于y=x+1对称和终于点（n-1，n+1）的路线一一对应

附录：

前18位卡特兰数：

如果n=1 1

如果n=2 2

如果n=3 5

如果n=4 14

如果n=5 42

如果n=6 132

如果n=7 429

如果n=8 1430

如果n=9 4862

如果n=10 16796

如果n=11 58786

如果n=12 208012

如果n=13 742900

如果n=14 2674440

如果n=15 9694845

如果n=16 35357670

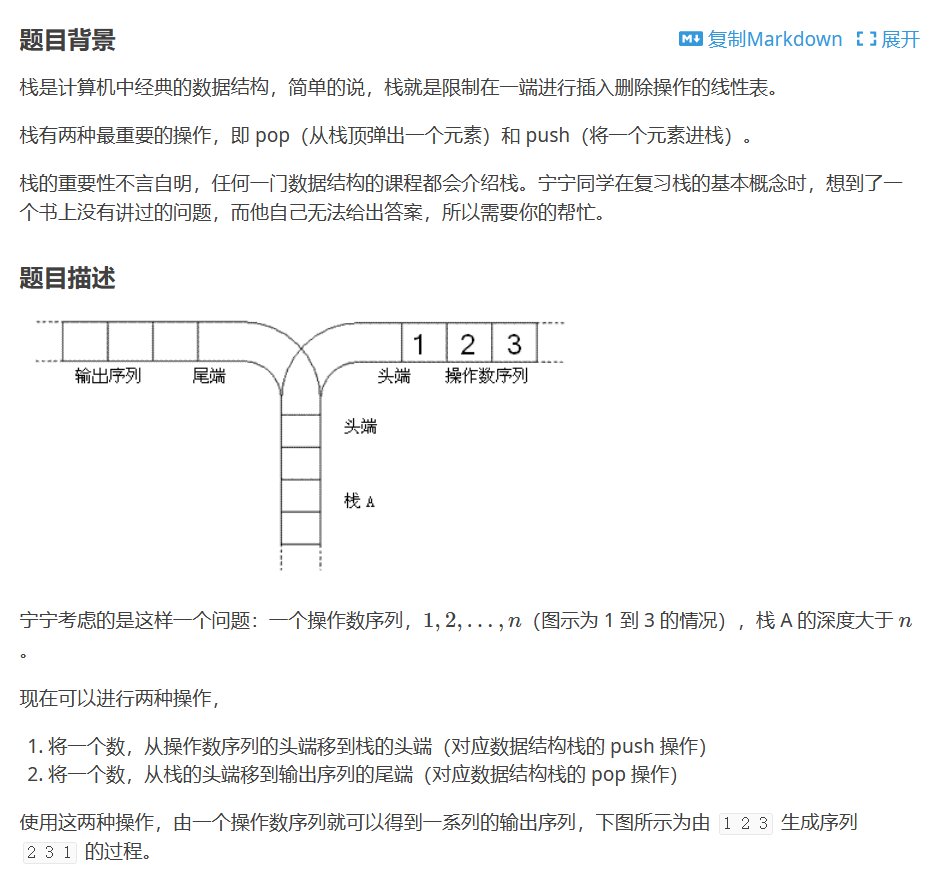
如果n=17 129644790

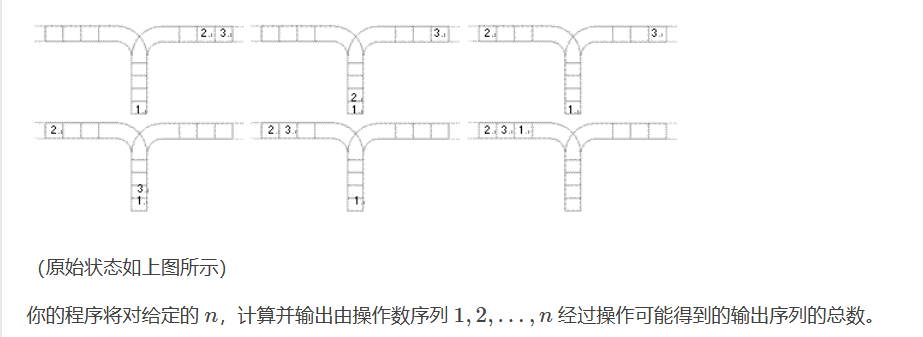
如果n=18 477638700

卡特兰数递归求法：

结合题目： 其中f[0][n]即为n时的卡特兰数

题目：







下面谈谈搜索(递归)思路：

递归：

既然记忆化搜索了，定义一个二维数组f[i,j]f[i,j]f[i,j]，用下标 iii 表示队列里还有几个待排的数，jjj 表示栈里有 jjj 个数，f[i,j]f[i,j]f[i,j]表示此时的情况数

那么，更加自然的，只要f[i,j]f[i,j]f[i,j]有值就直接返回；

然后递归如何实现呢？首先，可以想到，要是数全在栈里了，就只剩1种情况了，所以：i=0i=0i=0时，返回111；

然后，有两种情况：一种栈空，一种栈不空：在栈空时，我们不可以弹出栈里的元素，只能进入，所以队列里的数−1-1−1，栈里的数+1+1+1，即加上 f[i−1,j+1]f[i-1,j+1]f[i−1,j+1] ；另一种是栈不空，那么此时有出栈111个或者进111个再出111个 222种情况，分别加上 f[i−1,j+1]f[i-1,j+1]f[i−1,j+1] 和 f[i,j−1]f[i,j-1]f[i,j−1] ，便是此时的情况了，于是递归就愉快的结束了；

dp递推：

**f[i][j]=f[i-1][j]+f[i+1][j-1] (i>=1)**

**f[i][j]=f[i+1][j-1] (i=0)**

**因为栈里的数字只有两种选择，出去和不出去。**

**如果出去栈里的数字个数-1；如果不出去就要压栈，未进站的数字进来一个，未进栈的个数-1，栈内个数+1，所以f[i][j]=f[i-1][j]+f[i+1][j-1]**

**当栈内没有数字时，只能进栈，且此操作后的出栈情况就要取决于f[i+1][j-1]**

**（f[i][j]，i表示栈内数字的个数，j表示未进栈数字的个数，f计当前状态下有几种情况）**

**这样来看的话，边界也就很清楚了（当栈外没有数字时，只能出栈）**

**f[i][0]=1(0<=j<=n)**

卡特兰数递推：

解释一下原理：

建立数组f。f[i]表示i个数的全部可能性。

f[0] = 1, f[1] = 1; //当然只有一个

设 x 为当前出栈序列的最后一个，则x有n种取值

由于x是最后一个出栈的，所以可以将已经出栈的数分成两部分

1. 比x小
2. 比x大

比x小的数有x-1个，所以这些数的全部出栈可能为f[x-1]

比x大的数有n-x个，所以这些数的全部出栈可能为f[n-x]

这两部分互相影响，所以一个x的取值能够得到的所有可能性为f[x-1] \* f[n-x]

另外，由于x有n个取值，所以

ans = f[0]\*f[n-1] + f[1]\*f[n-2] + ... + f[n-1]\*f[0];

这，就是传说中的卡特兰数

卡特兰数直接求（推荐）：

因为可以gcd的原因推荐直接求C(2n,n)-C(2n,n-1)

卡特兰数证明:

假设要到达(n,n)且不越过y=x可站在线上有C(2n,n)-C(2n,n-1)种走法

将越过y=x的路线越过y=x+1的部分关于y=x+1对称必最终到达（n-1，n+1）

而从原点出发到（n-1，n+1）有C(2n,n-1)种走法

要到（n-1，n+1）必触及y=x+1，所以将越过y=x+1的部分关于y=x+1对称终于点（n，n）

所以终于点（n，n）触及y=x+1的路线关于y=x+1对称和终于点（n-1，n+1）的路线一一对应

附赠：卡特兰数：

case 1:cout<<"1"<<endl;break;

case 2:cout<<"2"<<endl;break;

case 3:cout<<"5"<<endl;break;

case 4:cout<<"14"<<endl;break;

case 5:cout<<"42"<<endl;break;

case 6:cout<<"132"<<endl;break;

case 7:cout<<"429"<<endl;break;

case 8:cout<<"1430"<<endl;break;

case 9:cout<<"4862"<<endl;break;

case 10:cout<<"16796"<<endl;break;

case 11:cout<<"58786"<<endl;break;

case 12:cout<<"208012"<<endl;break;

case 13:cout<<"742900"<<endl;break;

case 14:cout<<"2674440"<<endl;break;

case 15:cout<<"9694845"<<endl;break;

case 16:cout<<"35357670"<<endl;break;

case 17:cout<<"129644790"<<endl;break;

case 18:cout<<"477638700"<<endl;break;

case 19:cout<<"1767263190"<<endl;break;

case 20:cout<<"6564120420"<<endl;break;

case 21:cout<<"24466267020"<<endl;break;

case 22:cout<<"91482563640"<<endl;break;

case 23:cout<<"343059613650"<<endl;break;

case 24:cout<<"1289904147324"<<endl;break;

case 25:cout<<"4861946401452"<<endl;break;

case 26:cout<<"18367353072152"<<endl;break;

case 27:cout<<"69533550916004"<<endl;break;

case 28:cout<<"263747951750360"<<endl;break;

case 29:cout<<"1002242216651368"<<endl;break;

case 30:cout<<"3814986502092304"<<endl;break;

case 31:cout<<"14544636039226909"<<endl;break;

case 32:cout<<"55534064877048198"<<endl;break;

case 33:cout<<"212336130412243110"<<endl;break;

case 34:cout<<"812944042149730764"<<endl;break;

case 35:cout<<"3116285494907301262"<<endl;break;

case 36:cout<<"11959798385860453492"<<endl;break;

case 37:cout<<"45950804324621742364"<<endl;break;

case 38:cout<<"176733862787006701400"<<endl;break;

case 39:cout<<"680425371729975800390"<<endl;break;

case 40:cout<<"2622127042276492108820"<<endl;break;

case 41:cout<<"10113918591637898134020"<<endl;break;

case 42:cout<<"39044429911904443959240"<<endl;break;

case 43:cout<<"150853479205085351660700"<<endl;break;

case 44:cout<<"583300119592996693088040"<<endl;break;

case 45:cout<<"2257117854077248073253720"<<endl;break;

case 46:cout<<"8740328711533173390046320"<<endl;break;

case 47:cout<<"33868773757191046886429490"<<endl;break;

case 48:cout<<"131327898242169365477991900"<<endl;break;

case 49:cout<<"509552245179617138054608572"<<endl;break;

case 50:cout<<"1978261657756160653623774456"<<endl;break;

case 51:cout<<"7684785670514316385230816156"<<endl;break;

case 52:cout<<"29869166945772625950142417512"<<endl;break;

case 53:cout<<"116157871455782434250553845880"<<endl;break;

case 54:cout<<"451959718027953471447609509424"<<endl;break;

case 55:cout<<"1759414616608818870992479875972"<<endl;break;

case 56:cout<<"6852456927844873497549658464312"<<endl;break;

case 57:cout<<"26700952856774851904245220912664"<<endl;break;

case 58:cout<<"104088460289122304033498318812080"<<endl;break;

case 59:cout<<"405944995127576985730643443367112"<<endl;break;

case 60:cout<<"1583850964596120042686772779038896"<<endl;break;

case 61:cout<<"6182127958584855650487080847216336"<<endl;break;

case 62:cout<<"24139737743045626825711458546273312"<<endl;break;

case 63:cout<<"94295850558771979787935384946380125"<<endl;break;

case 64:cout<<"368479169875816659479009042713546950"<<endl;break;

case 65:cout<<"1440418573150919668872489894243865350"<<endl;break;

case 66:cout<<"5632681584560312734993915705849145100"<<endl;break;

case 67:cout<<"22033725021956517463358552614056949950"<<endl;break;

case 68:cout<<"86218923998960285726185640663701108500"<<endl;break;

case 69:cout<<"337485502510215975556783793455058624700"<<endl;break;

case 70:cout<<"1321422108420282270489942177190229544600"<<endl;break;

case 71:cout<<"5175569924646105559418940193995065716350"<<endl;break;

case 72:cout<<"20276890389709399862928998568254641025700"<<endl;break;

case 73:cout<<"79463489365077377841208237632349268884500"<<endl;break;

case 74:cout<<"311496878311103321137536291518809134027240"<<endl;break;

case 75:cout<<"1221395654430378811828760722007962130791020"<<endl;break;

case 76:cout<<"4790408930363303911328386208394864461024520"<<endl;break;

case 77:cout<<"18793142726809884575211361279087545193250040"<<endl;break;

case 78:cout<<"73745243611532458459690151854647329239335600"<<endl;break;

case 79:cout<<"289450081175264899454283846029490767264392230"<<endl;break;

case 80:cout<<"1136359577947336271931632877004667456667613940"<<endl;break;

case 81:cout<<"4462290049988320482463241297506133183499654740"<<endl;break;

case 82:cout<<"17526585015616776834735140517915655636396234280"<<endl;break;

case 83:cout<<"68854441132780194707888052034668647142985206100"<<endl;break;

case 84:cout<<"270557451039395118028642463289168566420671280440"<<endl;break;

case 85:cout<<"1063353702922273835973036658043476458723103404520"<<endl;break;

case 86:cout<<"4180080073556524734514695828170907458428751314320"<<endl;break;

case 87:cout<<"16435314834665426797069144960762886143367590394940"<<endl;break;

case 88:cout<<"64633260585762914370496637486146181462681535261000"<<endl;break;

case 89:cout<<"254224158304000796523953440778841647086547372026600"<<endl;break;

case 90:cout<<"1000134600800354781929399250536541864362461089950800"<<endl;break;

case 91:cout<<"3935312233584004685417853572763349509774031680023800"<<endl;break;

case 92:cout<<"15487357822491889407128326963778343232013931127835600"<<endl;break;

case 93:cout<<"60960876535340415751462563580829648891969728907438000"<<endl;break;

case 94:cout<<"239993345518077005168915776623476723006280827488229600"<<endl;break;

case 95:cout<<"944973797977428207852605870454939596837230758234904050"<<endl;break;

case 96:cout<<"3721443204405954385563870541379246659709506697378694300"<<endl;break;

case 97:cout<<"14657929356129575437016877846657032761712954950899755100"<<endl;break;

case 98:cout<<"57743358069601357782187700608042856334020731624756611000"<<endl;break;

case 99:cout<<"227508830794229349661819540395688853956041682601541047340"<<endl;break;

case 100:cout<<"896519947090131496687170070074100632420837521538745909320"<<endl;break;