1.状态压缩dp

将状态内容较小(1/0),状态位数较小的状态通过用二进制来表示，压缩为一个整数，再通过位运算来找出状态。

(1)位运算

> 与运算 & :相同位则为1,不同位则为0;
>
> 或运算 | :当前位有1为1，都0则为0;
>
> 异或运算 ^ :不同则为1,相同则为0
>
> 取反运算~: 1变0,0变1
>
> 左移运算<< : 整体左移，最低位补零，可等价为乘2；
>
> 右移运算>>:整体右移，最高位补与符号位相同的位，可等价为除2。

```
for(int i = s;s;i = (i - 1) & s)//遍历子集
n&1;                 //取出最低位，可判断奇偶
(n >> k) & 1;        //取出k的第k位.
n & ((1 << k) - 1); //取前k位
n ^ (1 << k);        //第k为取反
n | (1 << k)         //第k位取1
n & ( ~ (1 << k))    //第k位取0
```

(2) 状态压缩

```
 洛谷P1879
农场主John新买了一块长方形的新牧场，这块牧场被划分成M行N列(1 ≤ M ≤ 12 ; 1 ≤ N ≤ 12)，每一格都是一块正方形的土地。John打算在牧场上的某几格里种上美味的草，供他的奶牛们享用。
遗憾的是，有些土地相当贫瘠，不能用来种草。并且，奶牛们喜欢独占一块草地的感觉，于是John不会选择两块相邻的土地，也就是说，没有哪两块草地有公共边。
John想知道，如果不考虑草地的总块数，那么，一共有多少种种植方案可供他选择？（当然，把新牧场完全荒废也是一种方案）
```

如果我们直接表示状态，那么我们需要12*12的数组表示状态。若暴力的表示状态，则需要开 n x n 的数组，每个层的状态枚举需要 2 ^ n，每次检查是否符合条件需需要O(n),总共有 n ^ 3 x (2 ^ n) x (2 ^ n) 的复杂度，大概是1e8左右，并且状态还不好表示。

我们通过观察可以发现，每个点的状态可以只表示为有或无，那么我们可以将长度为n的状态压缩至一个整数内，用整数的每一个二进制位来表示当前位是否有草坪。

```
 for(int l=0;l<(1<<n);l++){
        if((l&(l>>1)))  continue;
        if((mp[1]|l)!=mp[1])    continue;
        dp[1][l]=1;
  }//初始化第一行的情况
for(int i=2;i<=m;i++){//行数
    for(int j=0;j<(1<<n);j++){//当前行的所有状态
        int flag=1;
```

```
            if((j&(j>>1)))  continue;//判断当前行是否有相邻的情况
            if((mp[i]|j)!=mp[i]) continue;//判断是否在贫瘠的位置种草了
            for(int l=0;l<(1<<n);l++){
                if((l&(l>>1)))  continue;//判断上一行是否有相邻的情况
                if((mp[i-1]|l)!=mp[i-1]) continue;//判断前一层的那个状态是否在贫瘠位置种草
                if(l&j) continue;   //判断此行和上一行是否相邻
                dp[i][j]+=dp[i-1][l];//转移
                dp[i][j]%=mod;
            }
        }
}
```

这样枚举状态，就可以O(1)检查状态合法性，不需要开额外的空间，只需要使用简单的位运算即可完成状态的表示和转移。

2.数位dp

通过记忆化的方式，通过搜索每一个数位的状态，来得到结果

> 杭州人称那些傻乎乎粘嗒嗒的人为62（音：laoer）。
> 杭州交通管理局经常会扩充一些的士车牌照，新近出来一个好消息，以后上牌照，不再含有不吉利的数字了，这样一来，就可以消除个别的士司机和乘客的心理障碍，更安全地服务大众。
> 不吉利的数字为所有含有4或62的号码。例如：
> 62315 73418 88914
> 都属于不吉利号码。但是，61152虽然含有6和2，但不是62连号，所以不属于不吉利数字之列。
> 你的任务是，对于每次给出的一个牌照区间号，推断出交管局今次又要实际上给多少辆新的士车上牌照了。

```
int num[25];
int dp[25][10];
int dfs(int pos,int pre,int stu,bool lim){
    if(pos==-1)return 1;
    if(lim==0&&dp[pos][stu]!=-1){
        return dp[pos][stu];
    }
    int up=9;
    if(lim)up=num[pos];
    int tp=0;
    for(int i=0;i<=up;i++){
        if(pre==6&&i==2)continue;
        if(i==4)continue;
        tp+=dfs(pos-1,i,i==2,lim && i==num[pos]);
    }
    if(!lim)dp[pos][stu]=tp;
    return tp;
}
```

> 当一个数字，从左到右依次看过去数字没有出现先递增接着递减的"山峰"现象，就被称作 Valley Number。它可以递增，也可以递减，还可以先递减再递增。在递增或递减的过程中可以出现相等的情况。
>
> 比如，1，10，12，212，32122都是 Valley Number。
>
> 121，12331，21212则不是。
>
> 度度熊想知道不大于N的Valley Number数有多少。
>
> 注意，前导0是不合法的。

```
int dp[150][12][5];
const ll mod= 1e9+7;
//1 ↑    2 ↓
// pos 当前数位 ,pre前一位, statu 上升or下降状态 ,lead 是否前导0,limit 是否限制上界
ll dfs(int pos,int pre,int statu,bool lead,bool limit){
    if(pos==-1){//枚举到最后一位了。
        if(lead)return 0;//全是0, 算无解
        else return 1;//有非0, 算一个解
    }
    if(dp[pos][pre][statu]&&!limit&&!lead){//如果当前求的不是带限制的, 并且已出现非0并且这个状态求过
        return dp[pos][pre][statu];
    }
    ll up;
    ll res=0;
    //求出上界, 无限制则为9
    if(limit)up=info[pos];
    else up=9;
    for(int i=0;i<=up;i++){
        if(lead){//如果还有前导0
            if(i==0)res+=dfs(pos-1,0,0,1,0);//传递前导0
            else {
                if(limit&&i==up)res+=dfs(pos-1,i,0,0,1);//传递非前导0
                else res+=dfs(pos-1,i,0,0,0);
            }
        }else{
            //与前一位相比, 若i小于则说明下降, 大于则说明上升, 等于的话状态不变。
            if(i<pre){
                if(statu==1)continue;
                if(limit&&i==up)res+=dfs(pos-1,i,2,0,1);
                else res+=dfs(pos-1,i,2,0,0);
            }else if(i==pre){
                if(limit&&i==up)res+=dfs(pos-1,i,statu,0,1);
                else res+=dfs(pos-1,i,statu,0,0);
            }else{
                if(limit&&i==up)res+=dfs(pos-1,i,1,0,1);
                else res+=dfs(pos-1,i,1,0,0);
            }
        }
        res%=mod;
    }
    if(!limit&&!lead){//如果当前结果可用, 那么记录下来。
        dp[pos][pre][statu]=res;
    }
    return res;//返回结果
}
```

3.计数dp

> 假设虚伪有一个h行w列的棋盘，棋盘上的格子有的是可以经过的，有的是不可以经过的。一开始在棋盘的左上角（第一行第一列）有一颗棋子，这颗棋子每次只能往右或者往下移动一格。那么虚伪要将其从起点移动到右下角(h，w)一共有多少种移法？

```
bool cmp(point a,point b){
    if(a.x==b.x)return a.y<b.y;
    return a.x<b.x;
}

sort(info,info+m+1,cmp);
for(int i=0;i<=m;i++){
     int x=info[i].x,y=info[i].y;
     dp[i]=C(x+y-2,x-1);
     for(int j=0;j<i;j++){//遍历所有右下的状态
         int xj=info[j].x , yj=info[j].y;
         if(xj<=x&&yj<=y){//如果是右下的状态
             dp[i]=dp[i]-dp[j]*C(x-xj+y-yj,x-xj);//减去答案.
         }
     }
 }
```

4.单调队列优化dp

> 洛谷P1725
>
> 某一天，琪露诺又在玩速冻青蛙，就是用冰把青蛙瞬间冻起来。但是这只青蛙比以往的要聪明许多，在琪露诺来之前就已经跑到了河的对岸。于是琪露诺决定到河岸去追青蛙。
>
> 小河可以看作一列格子依次编号为0到N，琪露诺只能从编号小的格子移动到编号大的格子。而且琪露诺按照一种特殊的方式进行移动，当她在格子i时，她只移动到区间[i+l,i+r]中的任意一格。你问为什么她这么移动，这还不简单，因为她是笨蛋啊。
>
> 每一个格子都有一个冰冻指数A[i]，编号为0的格子冰冻指数为0。当琪露诺停留在那一格时就可以得到那一格的冰冻指数A[i]。琪露诺希望能够在到达对岸时，获取最大的冰冻指数，这样她才能狠狠地教训那只青蛙。
>
> 但是由于她实在是太笨了，所以她决定拜托你帮它决定怎样前进。
>
> 开始时，琪露诺在编号0的格子上，只要她下一步的位置编号大于N就算到达对岸。

题意：在给定序列中找出一条路径使其经过的点之和最大，且每次可走的距离在给定区间 [l,r][l,r] 以内。

{ dp[i] = max{dp[j] + a[i]}(i−r⩽j⩽i−l)dp[i] = max{dp[j] + a[i]} (i−r⩽j⩽i−l) }

　而我们每次转移需要在 在 [i - r],[i - l] 中选择最大值转移过来。相当于求每次前面这段区间的最值。而我们可以使用单调队列来维护这个区间最值，从而达到优化决策，让决策过程变成O(1),的时间复杂度。

```
dp[0]=a[0];
    for(int i=1;i<=n;i++){//起点是1,因为最少跳1.
        while(h<=t&&dp[Q[t]]<=dp[i-1])--t;//维护单调递减
        Q[++t]=i-1;//入队
        while(h<=t&&Q[h]<i-r)++h;//保证决策点合法
        dp[i]=dp[Q[h]]+a[i];//取出最优决策点
        if(i>=n-r)ans=max(ans,dp[i]);//顺便直接维护答案。
    }
```

5.斜率优化dp

> P 教授要去看奥运，但是他舍不下他的玩具，于是他决定把所有的玩具运到北京。他使用自己的压缩器进行压缩，其可以将任意物品变成一堆，再放到一种特殊的一维容器中。
>
> P 教授有编号为 1 ~ n 的 n 件玩具，第 i件玩具经过压缩后的一维长度为 C_i。
>
> 为了方便整理，P教授要求：
>
> - 在一个一维容器中的玩具编号是连续的。
> - 同时如果一个一维容器中有多个玩具，那么两件玩具之间要加入一个单位长度的填充物。形式地说，如果将第 i 件玩具到第 j个玩具放到一个容器中，那么容器的长度将为
> - $$x = j - i + \sum_{k=i}^{j} C_k x$$
>
> 制作容器的费用与容器的长度有关，根据教授研究，如果容器长度为 xx，其制作费用为 (x-L)^2(*x*−*L*)2。其中 LL 是一个常量。P 教授不关心容器的数目，他可以制作出任意长度的容器，甚至超过 LL。但他希望所有容器的总费用最小。

设dp[i]为装好前i个玩具的最小花费,S[i]为 $S[n] = \sum_{i=1}^{n}(C[i]+1)$，则有转移方程为

$$dp[i] = min(dp[j] + (S[i] - S[j] - 1 - L)^2)$$

我们预先给L + 1，可得

$$dp[i] = dp[j] + (S[i] - S[j] - L)^2$$

拆开平方部分，可得最后的式子

$$dp[i] = S[i]^2 - 2S[i]L + dp[j] + (S[j] + L)^2 - 2S[i]S[j]$$

我们设 j_1, j_2 ($0 \le j_1 < j_2 < i$) 为 i的两个决策点，且满足决策点j_2 优于 j_1

那么有可列出不等式

$$S[i]^2 - 2S[i]L + dp[j_2] + (S[j_2] + L)^2 - 2S[i]S[j_2] \le S[i]^2 - 2S[i]L + dp[j_1] + (S[j_1] + L)^2 - 2S[i]S[j_1]$$

提出 $S[i]$ 到不等式一边，可得

$$2S[i] \ge \frac{(dp[j_2] + (S[j_2] + L)^2) - (dp[j_1] + (S[j_1] + L)^2)}{S[j_2] + L - S[j_1]}$$

我们设$Y(j) = dp[j] + (S[j] + L)^2, X(j) = S[j]$,则上式可以表示为

$$2S[i] \geq \frac{Y(j_2) - Y(j_1)}{X(j_2) - X(j_1)}$$

我们称其为斜率式。也就是是说，对于任意一个固定的i，对于可选 j_1, j_2 ,如果满足上式，那么j_2是优于j_1的。

此时，上式右部与i无关，只与j有关。所以我们可以用一个单调队列维护满足上面斜率式的最大值。

```
 db check(int i,int j){
    return ((d[i] + g[i]) - (d[j] + g[j])) / (f[i] - f[j]);
}
 for(int i=1;i<=n;i++){
     while(l < r && check(q[l],q[l+1]) < 2 * f[i])l++;
     int j = q[l];
     d[i] = d[j] + (f[i]-f[j]-1-L)*(f[i]-f[j]-1-L);
     while(l < r && check(q[r],q[r-1]) > check(i,q[r-1]))r--;//满足队尾

     q[++r] = i;//入队
 }
```

如果S[i]不单调的话怎么办？

分析下式

$$dp[i] = S[i]^2 - 2S[i]L + dp[j] + (S[j] + L)^2 - 2S[i]S[j]$$

变为

$$dp[j] = S[i]^2 + 2S[i]L + dp[i] - (S[j] + L)^2 + 2S[i]S[j]$$

$$dp[j] = S[i]^2 + dp[i] - (S[j] + L)^2 + 2S[i](S[j] + L)$$

左侧放所有可以独立存在的关于j的数，右边放一个i,j共同的数，和剩下的常数。

$$dp[j] + (S[j] + L)^2 = S[i]^2 + dp[i] + 2S[i](S[j] + L)$$

令 $x_i = S[j] + L, b_i = dp[i] - S[i]^2 + 2S[i]L, y_i = dp[j] + (S[j] + L)^2$可得$y = kx + b$。

注意：b中，S[i]是常数，不影响，而dp[i]是唯一改变，所以截距最大就为dp[i]最大，截距最小就为dp[i]最小。

这里的替换与上面斜率式$2S[i] \geq \frac{(dp[j_2]+(S[j_2]+L)^2)-(dp[j_1]+(S[j_1]+L)^2)}{S[j_2]-S[j_1]}$的替换相同，

为什么上面式子是$x_i = S[j]$,这里不同了？因为上面的L带入可以消掉。

最后得到$y = kx + b$ 的形式，再分别维护关于点(x_i, y_i)的凸包，即可。