数位动态规划

信智学院　陈琰宏

什么是数位DP

数位dp是一种计数用的dp，一般就是要统计一个区间[le,ri]内满足一些条件数的个数。所谓数位dp，字面意思就是在数位上进行dp。数位还算是比较好听的名字.

数位：把一个数字按照个、十、百、千等等一位一位地拆开，关注它每一位上的数字。

如果拆的是 十进制数，那么每一位数字都是 0~ 9，其他进制可 类比 十进制。

所以要引入数位的概念完全就是为了dp。数位dp的实质就是换一种暴力枚举的方式，使得新的枚举方式满足dp的性质，然后记性记忆化搜索。

数位 DP 的基本原理

数位 DP：用来解决一类特定问题，这种问题比较好辨认，一般具有这几个特征：

- 要求统计某区间满足一定条件的数的数量（即，最终目的为计数）
- 这些条件经过转化后可以使用「数位」的思想去理解和判断
- 输入会提供一个数字区间（有时也只提供上界）来作为统计的限制
- 上界很大（比如 10^9），暴力枚举验证会超时

数位 DP 的基本原理

考虑人类计数的方式，最朴素的计数就是从小到大开始依次加一。但我们发现对于位数比较多的数，这样的过程中有许多重复的部分。例如，从 7000 数到 7999、从 8000 数到 8999、和从 9000 数到 9999 的过程非常相似，它们都是后三位从 000 变到 999，不一样的地方只有千位这一位，所以我们可以把这些过程归并起来，将这些过程中产生的计数答案也都存在一个通用的数组里。此数组根据题目具体要求设置状态，用递推或 DP 的方式进行状态转移。

数位 DP 中通常会利用常规计数问题技巧，比如把一个区间内的答案拆成两部分相减

$$ans_{[l,r]} = ans_{[0,r]} - ans_{[0,l-1]}$$

[lq2021]9348二进制问题

小蓝最近在学习二进制。他想知道 1 到 N 中有多少个数满足其二进制表示中恰好有 K 个 1。你能帮助他吗?

输入一行包含两个整数 N 和 K。$1 \leqslant N \leqslant 10^{18}$, $1 \leqslant K \leqslant 50$ 。

输出一个整数表示答案。

输入样例

7 2

输出样例

3

分析：

因为$1 \leqslant N \leqslant 10^{18}$，$1 \leqslant K \leqslant 50$，显然枚举肯定不行，考虑其他方法。非常符合数位dp的模型。

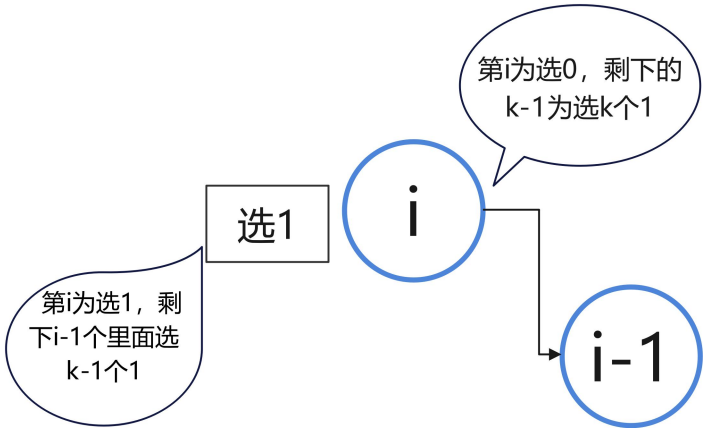
1. 求什么：满足要求（其二进制表示中恰好有 K 个 1）的方案数。

dp[i][j]:从前i位数中选择，其中有j位是1的方案数。

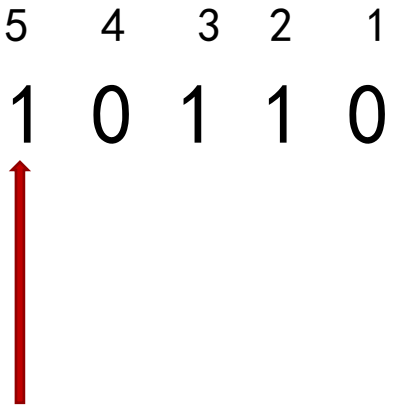
2. 怎么求：

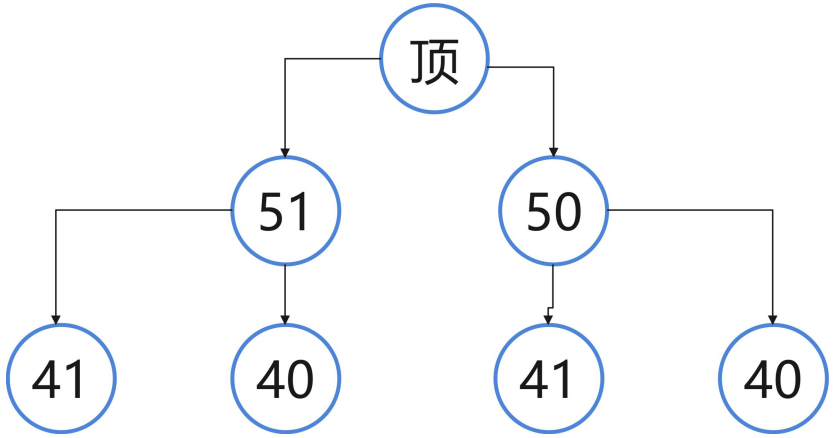
dp[i][j]=dp[i-1][j-1]+dp[i-1][j];

等价于 组合数的求解 $C_n^m = C_{n-1}^{m-1} + C_{n-1}^m$



分析：（假设 N的二进制为 10110，k取3）





因为N=10110,如果第五位、第四位都选1，则当前的数将大于N,不符合要求。
而dp[i][j] 中不能确定1的位置，故 将状态重新描述为描述dp[i][j][t].

dp[i][j][t]:从前i位数中选择k位是1,

并且最高位第i为的值为t(t=1或0)的方案数。

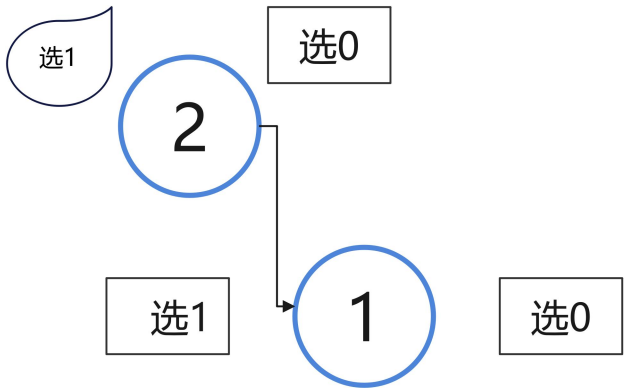
分析：

状态转移方程:

$$dp[i][j][1] = dp[i-1][j-1][0] + dp[i-1][j-1][1],$$

$$dp[i][j][0] = dp[i-1][j][0] + dp[i-1][j][1].$$

初始化：$dp[1][1][1] = 1, dp[1][0][0] = 1$



$dp[1][1][1] = 1$　　$dp[1][0][0] = 1$

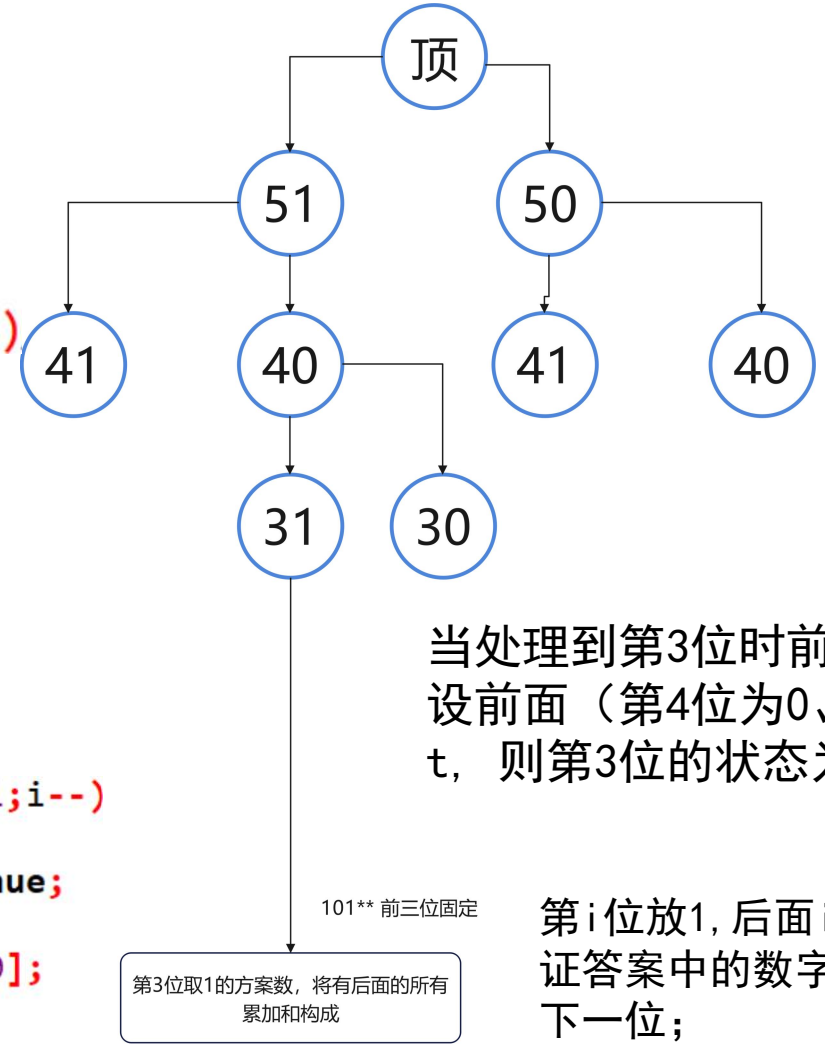
枚举组合数：

```
4  #define int long long
5  int n,k,len;
6  int num[75];//存二进制下的n
7  int dp[75][75][5];//默认初始值为0
8  //1e18,64位 数组大于64，就可以
9  void Init()//预处理
10 {//
11     dp[1][1][1]=1;
12     dp[1][0][0]=1;//初始化
13     for(int i=2;i<=64;i++)//状态转移(从第2位开始)
14     {
15         for(int j=0;j<=i;j++)
16         {
17             if(j>0)dp[i][j][1]=dp[i-1][j-1][0]+dp[i-1][j-1][1];
18             dp[i][j][0]=dp[i-1][j][0]+dp[i-1][j][1];
19         }
20
21     }
22 }
```

主函数：

```
45  signed main()
46  {
47      cin>>n>>k;
48      Init();
49      while(n)//将n转为2进制
50      {
51          num[++len]=n%2;//位数从1开始 1~len
52          n/=2;
53      }
54      if(k>len) cout<<"0\n";//此情况显然不可能有数字满足要求
55      else cout<<solve(len)<<"\n";
56      return 0;
57  }
```

代码：（假设 N的二进制为 10110，k取3）




```
for(int i=len-1;i>=1;i--)
```

因为N的第4位是0，所以不处理，因为取1的话将大于N.

```
if(!num[i])continue;
```

```
for(int i=len-1;i>=1;i--)
{
    if(!num[i])continue;
    if(t>k)break;
    sum+=dp[i][k-t][0];
    t++;
}
```

```
for(int i=k;i<len;i++)
 sum+=dp[i][k][1]; //
```

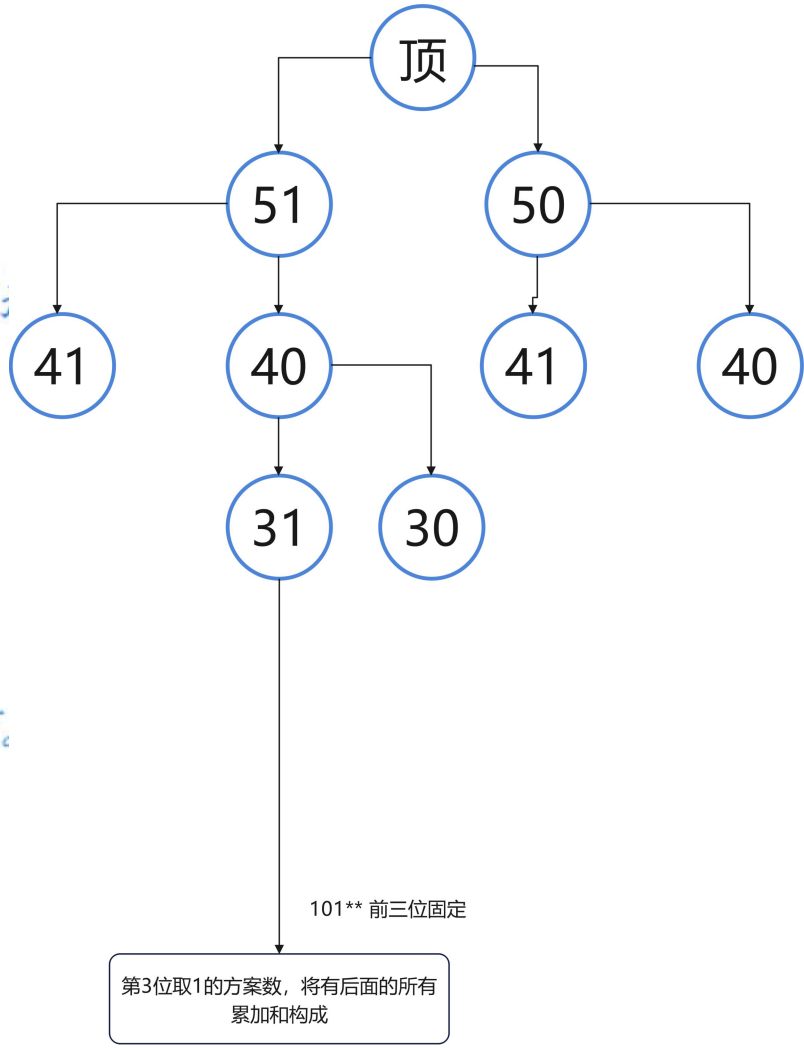
dp[i][k][1]表示第i位取1 且 前i位中取k个1的方案数，累加后即为所有满足要求的方案数。如果第i位取0，则满足条件的方案将在dp[i-1][k][1]中体现，故累加和即为第5位为0的所有方案数。

当处理到第3位时前面的各位的值都是确定的，假设前面（第4位为0、第5位为1）已取的1的个数是t，则第3位的状态为 dp[i][k-t][0]

第i位放1，后面i-1位上的情况不能用组合数计算，因为要保证答案中的数字比原数字小，所以固定第i位为1，继续枚举下一位；

代码：（假设 N的二进制为 10110，k取3）

```
24  int solve(int len)
25  {
26      int sum=0;
27      //第len位是0的情况
28          for(int i=k;i<len;i++)//统计1～len-1位所有满足的数
29           sum+=dp[i][k][1]; //长度为1~k-1是没有意义，因为无法选
30
31      //第len位取1的情况
32          int t=1;//t表示：从高位开始 1出现的个数
33          //因为len位取1，所以t 初始为1
34          for(int i=len-1;i>=1;i--)//统计第len位中满足的数
35          {
36              if(!num[i])continue;//当前位是0,不处理
37              if(t>k)break;//剩下数字含1个数必定大于k,故退出循环
38              sum+=dp[i][k-t][0];
39                  //dp[i][k-t][1];
40              t++;
41          }
42      return sum;
43  }
```



[6423]度的数量

求给定区间[X,Y] 中满足下列条件的整数个数：这个数恰好等于 K 个互不相等的 B 的整数次幂之和。

例如，设 X=15,Y=20,K=2,B=2，则有且仅有下列三个数满足题意：

$17=2^4+2^0$

$18=2^4+2^1$

$20=2^4+2^2$

第一行包含两个整数 X 和 Y，接下来两行包含整数 K 和 B。输出只包含一个整数，表示满足条件的数的个数。

输入：15 20 2 2

输出：3

15：1111
16：10000
17：10001
18：10010
19：10011
20：10100

分析：

题目就是在一个区间[x,y]内，有多少个符合题意的数，这里的符合题意是指：这个数恰好等于 K 个互不相等的 B 的整数次幂之和。这句话的另一个意思，就是“这个数的B进制表示中，其中有K位上是1、其他位上全是0”。假设B=10,B 的整数次幂为1,10,100,1000,...， 加起来必然是形如 1001010 的数，即这个数在B进制表示的各位数字只能是0和1，且1的个数为k个。

比如样例中B=2，K=2，这个数是18.就是把18化为二进制，18化为二进制数为：10010，其中有2位1，其他位都是0。

这是一道典型的数位dp题。

分析：le=60 ri=98 k=2 B=4

把数字n转化成B进制数，从高位向低位枚举，假设枚举到第i位，第i位上的数字是x,分以下几种情况讨论：（last记录第i位之前放置1的个数）

1. x==0:直接跳过，继续枚举下一位；
2. x==1,第i位分成两种情况：

(1)第i位放0,后面i-1位上可以放k-last个1,res +=f[i-1][k-last];

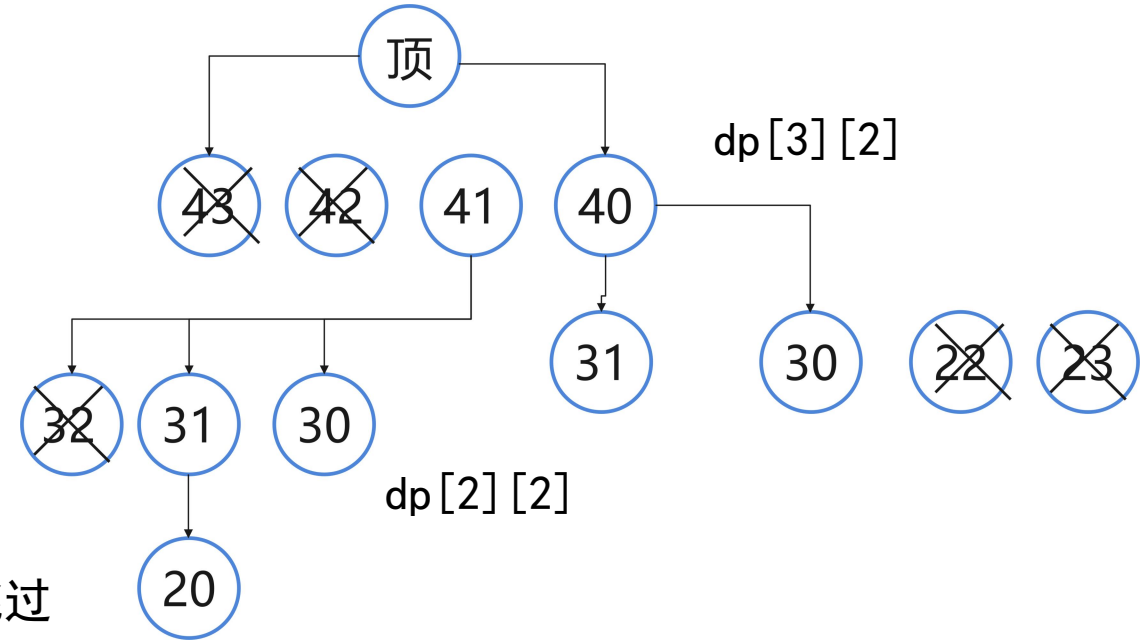
(2)第i位放1,后面i-1位上的情况不能用组合数计算，因为要保证答案中的数字比原数字小，所以固定第i位为1,继续枚举下一位；

98:1202, $98=4^3+2*4^2+2*4^0$

80:1100, $96=4^3+4^2$

68:1010, $68=4^3+4^1$

... ...



把数字n转化成B进制数，从高位向低位枚举，假设枚举到第i位，第i位上的数字是x,分以下几种情况讨论：
(last记录第i位之前放置1的个数)

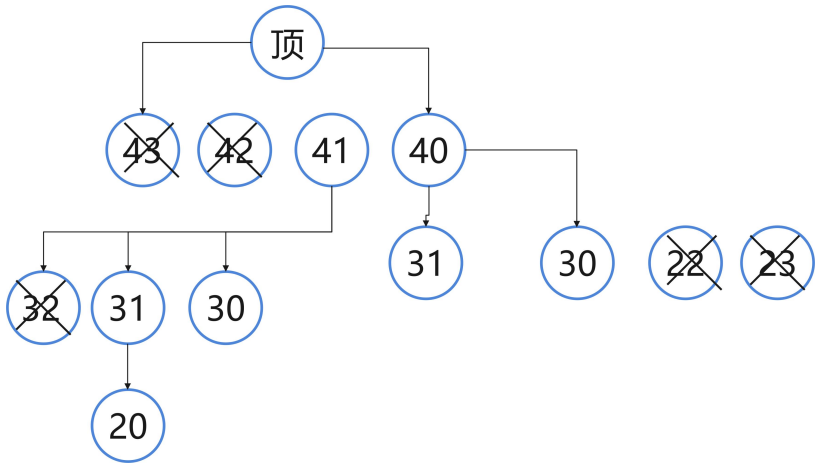
1. x==0:直接跳过，继续枚举下一位；

2. x==1,第i位分成两种情况：

(1)第i位放0,后面i-1位上可以放k-last个1,

```
res +=f[i-1][k-last];
```

(2)第i位放1,后面i-1位上的情况不能用组合数计算，因为要保证答案中的数字比原数字小，所以固定第i位为1,继续枚举下一位；



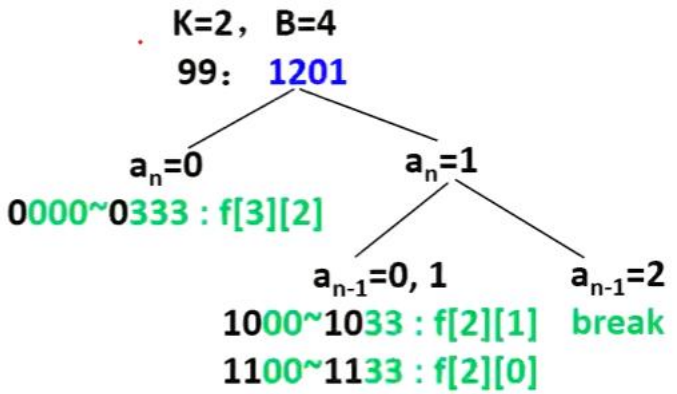
3. x>1,第i位分成三种情况：

(1)第i位放0,后面i-1位上可以放k-last个1,res +=f[i-1][k-last];

(2)第i位放1,后面i-1位上可以放k-last-1个1,res +=f[i-1][k-last-1];

(3)第i位放大于1的数，已经不合要求，没必要继续枚举，直接break。

（如果放了大一1的数，后面的一定不符合要求，因此就不用枚举）



预处理组合数

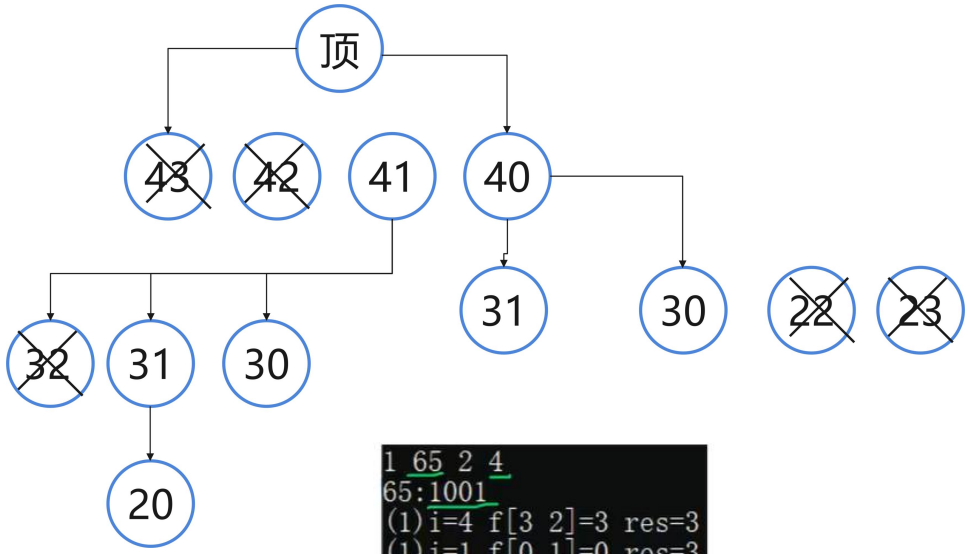
```
 8  const int N = 35;
 9  int K, B;
10  int f[N][N];
11  void init(){//求组合数
12      for(int i=0;i<=N;i++){
13          for(int j=0;j<=i;j++){
14              if(!j)f[i][j]=1;
15              else f[i][j]=f[i-1][j-1]+f[i-1][j];
16          }
17      }
18  }
19  int dp(int n){
45
46  int main()
47  {
48      init();
49      int l, r;
50      cin >> l >> r >> K >> B;
51      cout << dp(r) - dp(l - 1) << endl;
52      return 0;
53  }
```

等价于 组合数的求解

$$C_n^m = C_{n-1}^{m-1} + C_{n-1}^m$$

le=4 ri=22 k=2 B=4

```cpp
int dp(int n){
    if(n == 0) return 0; //如果上界n是0，直接就是0种
    vector<int> nums;
    while(n) nums.push_back( n% B) , n/= B;

    int res = 0;
    int last = 0; //Last表示第i位之前放置1的个数
    for(int i = nums.size()-1; i>= 0; i--){//从最高位开始遍历每一位

        int x = nums[i]; //取当前位上的数
        if(x>0){ //只有x>0的时候才可以讨论左右分支,=0直接跳过

            res += f[i][ K -last];//第i为放0

            if(x > 1){//第i位的x值大于1
                //当前位填1，从剩下的所有位（共有i位）中选K-last-1个数。
               if(K - last -1 >= 0) res += f[i][K -last -1];
                break;//第i位放大于1的数，不合要求，则break
            }

            else {//第i位==1时，不能用组合数计算，继续枚举下一位
                last ++;
                //如果已经填的个数last > 需要填的个数K，不合法break
                if(last > K) break;
            }

        }
        if(i==0 && last == K) res++;////特判，走到末位的情况
    }
    return res;
}
```




```
1 65 2 4
65:1001
(1)i=4 f[3 2]=3 res=3
(1)i=1 f[0 1]=0 res=3
(3)i=1 res=4
4
```

```
1 97 2 4
97:1201
(1)i=4 f[3 2]=3 res=3
(1)i=3 f[2 1]=2 res=5
(2)i=3 f[2 0]=1 res=6
6
```

```
1 85 2 4
85:1111
(1)i=4 f[3 2]=3 res=3
(1)i=3 f[2 1]=2 res=5
(1)i=2 f[1 0]=1 res=6
6
```

[3724] windy数

windy定义了一种windy数。

不含前导零且相邻两个数字之差至少为2的正整数被称为windy数。

windy想知道，在A和B之间，包括A和B，总共有多少个windy数。

100%的数据，满足1<=A<=B<=2000000000

输入	输出
1 10	9
25 50	20
10 20	8

[3724] windy数

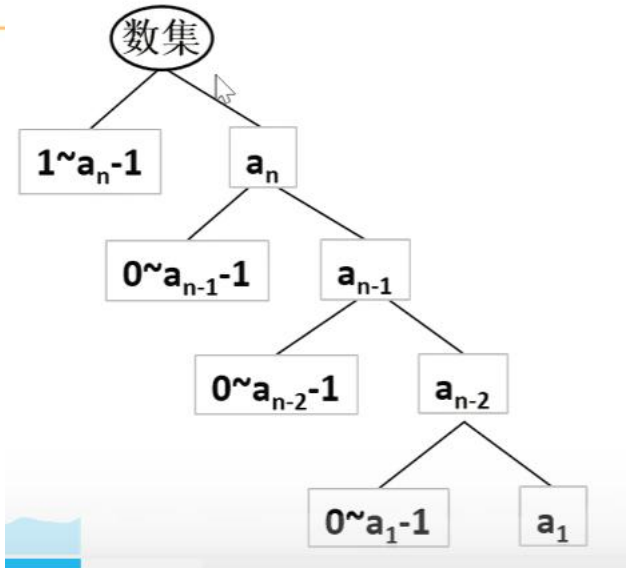
Windy数 数位DP

区间转化：

欲求[a, b]内Windy数的个数，先求[0, x]的Windy数的个数dp(x)，
答案即dp(b) – dp(a-1)。

分类填数：

设整数x一共n位，x表示为 $a_n\ a_{n-1}\ a_{n-2}\ \dots\ a_1$ ，从高位到低位枚举填数。
因为不含前导零，所以最高位只能填$1\sim a_n$，其他位可以填$0\sim a_i$。
每个位上填数时，分为两类：$0\sim a_i-1$和a_i，这样填数可以保证不超过x。



例: x = 2572

1000 ~ 1999

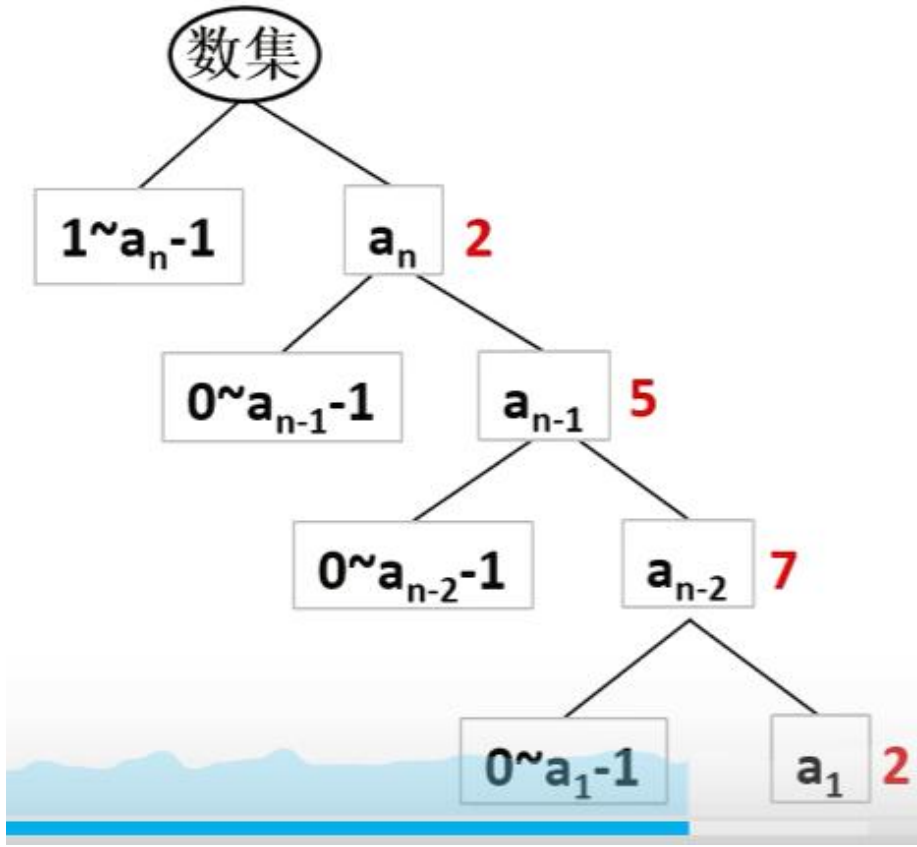
2000 ~ 2499

2500 ~ 2569

2570 ~ 2571

2572

[3724] windy数



例: x = 2572

1000 ~ 1999

2000 ~ 2499

2500 ~ 2569

2570 ~ 2571

2572

f[i][j] 表示一共有i位，且最高位数字为j的Windy数的个数。

分段统计：我们用last记录上一位数字，然后枚举当前位j，如果abs(j-last) >= 2，就累加答案，res += f[i][j]。

前面的答案都是n位的，对于位数低于n位的，再分段统计，累加到答案中就行了。
例，100~999，10~99，1~9。

```
const int N = 12;
int a[N]; //把整数的每一位数字抠出来，存入数组
int f[N][10];//f[i][j]表示一共有i位，且最高位数字为j的Windy数的个数

void init(){ //预处理Windy数的个数
  for(int i=0; i<=9; i++) f[1][i]=1;//一位数
  for(int i=2; i<N; i++) //阶段：枚举位数
    for(int j=0; j<=9; j++) //状态：枚举第i位
      for(int k=0; k<=9; k++) //决策：枚举第i-1位
        if(abs(k-j)>=2) f[i][j] += f[i-1][k];
}
```

3101：2^k进制数