尺子刻度

【问题】

一根 N 厘米长的尺子，只允许在上面刻 K 个刻度，要能用它量出 1~N 厘米的各种长度。请求出满足要求的本质不同的刻法数量。

两种刻法被视为本质不同的，当且仅当他们选择的刻度不同，且翻转后仍然不同。

比如当 N=10,K=7 时：
1 2 3 4 5 6 7 与 3 4 5 6 7 8 9 本质相同；
1 2 3 4 5 6 7 与 1 2 3 4 5 6 8 本质不同。

输入 N 和 K 的值，输出方法总数。

【分析】

首先要考虑刻度的表示方法。我们可以选取尺子的左端点作为零刻度，从左到右以正整数表示出刻度，到最右端就为 N。此时选取刻度的问题便转化为在 1~N-1 中选取 K 个互不相同的正整数的问题：

当 N=10,K=7 时存在：
1_|_|_|_|_|_|_|_ _ _1
0 1 2 3 4 5 6 7 10

这种问题可以用递归解决：

```
//使用 m 作为一个计数器，初始化为 K，每调用一层减少 1，直到 0 则选取完成
void ruler(int m){
if(m!=0){  //如果还有刻度需要选取
     //在范围内选取一个刻度
     ruler(m-1);  //继续调用这个函数
}
else{  //否则 K 个刻度已经全部选出
     //判断选取出的情况是否符合条件
}
return;
}
```

在储存刻度时，可以选择利用一个全局变量数组 a[]来储存刻度；而选取刻度时，规定只能在上一个刻度（设为 t）的右边（即 t+1~n+1

的范围内）选取剩余的刻度，这样选取刻度时就是有序的，并能够枚举出所有情况：

```
//定义全局变量 a[]，选出的刻度储存在 a 中；x 为上次选取的刻度的下一个刻度
void ruler(int x,int m){
if(m!=0){
     int t;
     for(t=x;t<n;t++){  //从 x~n-1 中任意选取一个刻度
         a[i++]=t;  //i 也为全局变量，初始化为 1，储存目前是第几个刻度
         ruler(t+1,m-1);  //递归调用
         a[- -i]=0;  //将储存的刻度归零
     }
}
else{
     //判断选取出的情况是否符合条件
}
return;
}
```

此时解决了选取刻度的问题，接下来就要解决判断的问题。由于储存的数据格式是距离左端点的距离，所以任意两个刻度能够量出的长度即为两个刻度值差的绝对值（左端点可视为 0，右端点可视为 N）。此时要解决的是计算出所有能够测量出的距离。那么就枚举任意两个刻度并记录其能够测量出的距离：

```
//在 ruler 函数中处理 a[]数组时，枚举使用了 1~k 的空间，那么可以用 a[0]=0
表示左端点，a[k+1]=n 表示右端点
bool check(void){
int x,y;
bool flag=true;
bool num[40]={0};  //num 数组用于储存能够测量出的距离
for(x=0;x<=k;x++)  //起点范围为 a[0]~a[k]
for(y=x+1;y<=k+1;y++)  //起点范围为 a[x+1]~a[k+1]
    num[a[y]-a[x]]=1;
    //储存时 a[]中的数一定是由小到大，不用考 a[y]-a[x]为负数
x=n-1;
while(x>0){
    if(num[x]!=1) {flag=false; break;}  //验证能否测量出 1~n-1 的任意长度
    x- -;
}
```

```
return flag;
}
```

ruler 函数中相应改为：

```
//tot 作为全局变量，初始化为 0，记录方案总数
void ruler(int x,int m){
if(m!=0){
    int t;
    for(t=x;t<n;t++) {a[i++]=t; ruler(t+1,m-1); a[- -i]=0;}
}
else{
    a[i]=n;  //为了方便 check 函数，将 a[i]设定为 n
    if(check()) tot++;
}
return;
}
```

此时解决了该问题的第一个部分：找出所有情况并验证其是否满足情况；而题目要求还有判断重复的情况。题目规定翻转后相同的情况为相同情况，那么翻转后相同是什么情况呢？输出满足条件的所有情况，可以看到：

当 N=9,K=7 时，有：
1 2 3 4 5 6 7……1
1 2 3 4 5 6 8……2
1 2 3 4 5 7 8……3
1 2 3 4 6 7 8……4
1 2 3 5 6 7 8……4
1 2 4 5 6 7 8……3
1 3 4 5 6 7 8……2
2 3 4 5 6 7 8……1
对应方案数为 4

当 N=8,K=7 时，有：
1 2 3 4 5 6 7……1
对应方案数为 1

观察 N=9,K=7 的情况，可发现每种方案都会重复，且重复的情况在枚举时都出现了两次；而在 N=8,K=7 时却没有出现重复的情况。这说明对于重复的情况需要具体分析：

当 N=9,K=7 时，第 1 种情况为：

1_|_|_|_|_|_|_|_ _1

0 1 2 3 4 5 6 7 9

对应重复的情况为：

1_ _|_|_|_|_|_|_|_1

0 2 3 4 5 6 7 8 9

当 N=8,K=7 时，第 1 种情况为：

1_|_|_|_|_|_|_|_1

0 1 2 3 4 5 6 7 8

不存在重复的情况。

观察可发现，不存在重复的情况自身是对称的，而存在重复的情况自身则是不对称的。还可以相应举出更多例子：

当 N=10,K=7 时，存在：

1_|_|_|_|_|_|_|_ _ _1

0 1 2 3 4 5 6 7 10

对应重复的情况为：

1_ _ _|_|_|_|_|_|_|_1

0 3 4 5 6 7 8 9 10

当 N=10,K=7 时，存在：

1_ _|_|_|_|_|_|_|_ _1

0 2 3 4 5 6 7 8 10

不存在重复的情况。

那么可以得出一个规律：自身对称的情况不存在重复，而自身不对称的情况必存在且仅存在 1 组与之重复的解。对于重复的情况，输出时可直接除以 2；对于不重复的情况，为了统一标准，就将总数加 1，这样输出时也可以直接除以 2。这样要解决的问题就是判定这种情况是否存在重复的情况，若不存在就将总数加 1。

而再观察自身对称的情况，可以发现：

当 N=10,K=7 时，存在：

1_ _|_|_|_|_|_|_|_ _1

0 2 3 4 5 6 7 8 10

此时 2+8=10,3+7=10,4+6=10,5+5=10

而自身不对称的情况则不存在这种规律。需要注意，（若存在）这种规律必须包含位于中间的刻度的判定，否则会误判这种情况：

当 N=10,K=7 时，存在：
1_|_|_|_|_ _ _|_|_|_1
0 1 2 3 4　　7 8 9 10
此时 1+9=10,2+8=10,3+7=10
而存在重复的情况：
1_|_|_|_ _ _|_|_|_|_1
0 1 2 3　　6 7 8 9 10

利用这种规律，可以编写出验证函数：

```
bool check2(void){
int x=1,y=k;
while(x<=y){  //当 x==y 时检测中间的刻度
    if(a[x]+a[y]!=n) return false;
    x++;y- -;
}
return true;
}
```

ruler 函数相应改为：

```
void ruler(int x,int m){
if(m!=0){
    int t;
    for(t=x;t<n;t++) {a[i++]=t; ruler(t+1,m-1); a[- -i]=0;}
}
else{
    a[i]=n;
    if(check()){
        tot++;
        if(check2()) tot++;
    }
}
return;
}
```

这样问题得以解决。需要注意，这种方法在 N,K 的值很大的时候会超时，因此不是最优解法。