5分：我不会。

$n, m \leq 5$：爆搜一下。

$m = 2$：暴力枚举第一次的区间，然后枚举第二个区间的过程中处理一下合法位置个数即可。

$m = 3$：暴力枚举前两个区间，那么你可以知道每个位置有如下三种情况：

（1）一定要被第三个区间包含。

（2）一定不能被第三个区间包含。

（3）无所谓。

我们把每个位置按情况写成$1, 2, 3$，那么：我的第三个区间必须得包含所有的1，一定不能包含所有的2。扫一遍就知道区间左/右端点的合法区间了。

$n, m \leq 50$：

我们考虑把这个问题转化一下模型，变成：你需要指定m对括号，使得每个位置被包含的次数恰好在l_i, r_i范围内。

那么：我们用$dp_{i,j,k}$表示当前已经考虑到了第i个位置，目前一共出现过j对括号，其中k对括号目前只出现了`(`，$j - k$对括号目前已经出现了`()`，的方案数。

那么，我们就可以枚举在第$i + 1$个位置一共出现了几个`(`，几个`)`，来算方案数了。

这样时间复杂度是$O(nm^4)$的。

正解：

我们考虑把`(`和`)`的流程分开：在每个位置，先处理`)`，再处理`(`，即可。

也就是：$dp1_{i,j,k}$表示我该考虑在i处添加右括号时的方案，$dp2_{i,j,k}$表示我该考虑在i处添加左括号时候的方案。

具体转移不表，复杂度$O(nm^3)$，但实际上，这个 dp 在满数据下只需要 3e8 次循环，所以开了$3s$：

```
#include<bits/stdc++.h>
#define maxn 305
#define ll long long
#define mod 998244353
using namespace std;
ll C[maxn][maxn];
void init()
{
  C[0][0]=1;
  for (int i=1;i<maxn;i++)
    for (int j=0;j<=i;j++)
    {
      if (i==1 || j==0) C[i][j]=1;
      else C[i][j]=(C[i-1][j]+C[i-1][j-1])%mod;
    }
}
int n,m;
int l[maxn],r[maxn];
ll dp1[105][maxn][maxn],dp2[105][maxn][maxn];
void add(ll &x,ll y) {x=(x+y)%mod; return;}
int main()
{
  init();
  cin>>n>>m;
  for (int i=1;i<=n;i++) cin>>l[i];
  for (int i=1;i<=n;i++) cin>>r[i];
  dp2[0][0][0]=1; //dp2表示考虑完 () 的结果
  for (int i=0;i<=n;i++)
  {
    //首先转移dp1到dp2，再转移dp2到下一个dp1
    for (int j=0;j<=m;j++)
    for (int k=l[i];k<=r[i];k++) //左括号个数
```

```
      if (dp1[i][j][k])
      {
        for (int t=0;t<=k;t++) //在此处添加t个右括号
        {
          add(dp2[i][j][k-t],dp1[i][j][k]*C[k][t]);
        }
      }
      for (int j=0;j<=m;j++)
      for (int k=0;k<=j;k++) //左括号个数
      if (dp2[i][j][k])
      {
            //在i+1处添加t个左括号,
            //需要满足0<=t+k<=j以及l[i+1]<=t+k<=r[i+1]
        for (int t=max(0,l[i+1]-k);t<=min(m-j,r[i+1]-k);t++)
        {
          add(dp1[i+1][j+t][k+t],dp2[i][j][k]*C[m-j][t]);
        }
      }
    }
    cout<<dp2[n][m][0]<<endl;
  }
```