21级实验室暑假第九讲

目录

线性dp

线性 dp： 具有线性 “阶段” 划分的动态规划算法。

无论线性 dp 的状态表示上是一维的还是多维的，都是线性上的递推。dp 的阶段沿着各个维度线性增长，从一个或多个边界点开始，有方向地向整个状态空间转移、拓展。最后每个状态上都保留了以自身为 “目标” 的子问题的最优解。

由于之前讲动态规划涉及一些线性dp(建议结合之前的第十一讲看看)，故此部分大致回顾及扩展一下。

线性dp

LIS问题

例题1：最长上升子序列（模板）

- ✓ 题目描述: 给定n个数a_i,求数值单调递增的子序列的最长长度为多少
- ✓ 状态表示: f[i]表示以a_i为结尾的最长上升子序列的长度
- ✓ 转移方程: $当j \in [1,i)且a_j < a_i时, f[i] = \max(f[i], f[j]+1)$
- ✓ 初始值: $f[i] = 1(1 \le i \le n)$

线性dp

LIS问题

例题1：最长上升子序列（模板）

- ✓ 题目描述： 给定n个数a_i,求数值单调递增的子序列的最长长度为多少
- ✓ 方法一：$O(n^2)$,两重循环

```
//当a[j]<a[i]且1<=j<i  f[i]=max(f[i],f[j]+1)
for(int i=1;i<=n;i++){
    f[i]=1;//初始值
    for(int j=1;j<i;j++)
        if(a[j]<a[i])
            f[i]=max(f[i],f[j]+1);
    res=max(res,f[i]);
}
```

那有没有办法再优化其时间复杂度？

当然是可以的，可用**树状数组维护其前缀和的最大值**

线性dp

LIS问题

例题1：最长上升子序列（模板）

- ✓ 题目描述： 给定n个数a_i,求数值单调递增的子序列的最长长度为多少
- ✓ 方法二：O(nlogn),**树状数组维护前缀和最大值**(数据范围若较大或存在负数,此时需进行**离散化**操作)

```
void update(int x,int c){
    for(int i=x;i<=n;i+=lowbit(i))
        d[i]=max(d[i],c);
}

int query(int x){
    int res=0;
    for(int i=x;i>=1;i-=lowbit(i))
        res=max(res,d[i]);
    return res;
}
```

```
for(int i=1;i<=n;i++){
    f[i]=query(a[i]-1)+1;
    ans=max(ans,f[i]);
    update(a[i],f[i]);
}
```

线性dp

LIS问题

例题2：最长不上升/不下降子序列

- ✓ 题目描述：登山的过程中，有 n 个景点，每个景点的高度为 ai(存在高度相同的点)。你希望尽可能浏览多的景点，但是你有个习惯，就是一旦开始下山，就不能再往上走了。n<=1000
- ✓ 先非严格单调递增往上走，到达一个点，然后非严格单调递减往下走。要是我们能求出到达每个点的最长不下降子序列的长度，和每个点往后可以达到的最长不上升子序列长度就好了。
- ✓ 前者可以直接正向最长上升子序列模板 $O(n^2)$ 变形求，后者可以转化为**逆向的最长不下降子序列长度**，然后也最长上升子序列模板 $O(n^2)$ 变形求

线性dp

LIS问题

例题3：思维转化

- ✓ 题目描述: 一个东西朝向的河，有 n 个城市在北岸，n个城市在南岸。有 n 对友好城市关系，即北岸的城市 i与南岸的城市a_i是友好城市关系。每对友好城市之间可建一条航道，但是**航道之间不能交叉**，求最多能建多少条航道。n<=1000

- ✓ 对于北岸的城市 i,j，当且仅当 i<j 并且$a_i<a_j$时不交叉，可发现就是最长上升子序列问题。

线性dp

LIS问题

例题4：最长上升子序列的方案数

- ✓ 题目描述:　给定 n 个数a_i，求最长上升子序列的长度 len ， 以及有多少个长度为 len 的上升子序列。

- ✓ 状态表示:
- ✓ f[i] ： 以 a_i为结尾的最长上升子序列的长度
- ✓ g[i]:　以a_i结尾的上升子序列长度为f[i]的子序列个数

- ✓ 初始状态:
- ✓ f[i]=g[i]=1

- ✓ 转移方程:

$$当j\in[1,i),且a_j<a_i时,可分为如下三种情况:$$

$$当f[j]+1<f[i]时,不作处理;$$

$$当f[j]+1=f[i]时,g[i]+=g[j]$$

$$当f[j]+1>f[i]时,g[i]=g[j],f[i]=f[j]+1$$

线性dp

LIS问题

例题5：思维+不下降子序列的最大和

- ✓ 题目描述： 给定 n 个数a_i，要求通过将数字重新摆放位置的规则，使得这 n 个数从小到大递增。若将数字 a_i 重新摆放了一次，则消耗为a_i 。求出这个消耗值的最小值。例如：7 1 2 3，
- ✓ 方法1：移动7到末尾，消耗值为 7 ；
- ✓ 方法2：分别移动 1 2 3 到开头，消耗值为 1+2+3 = 6 。$n <= 1000$, $a_i \in [-1e7, 1e7]$
- ✓ 一个数最多只会被重新摆放一次。假设存在一个不下降子序列，则**只需要将除了该不下降子序列的元素重排即可，消耗值为他们的和**。问题转化为：求得一个不下降子序列，使得他们的和最大

线性dp

i
int
integer
intern
internet

LIS问题

例题6：广义+偏序求最长子序列

- ✓ 题目描述：给定 n 个字符串，求得一个最长子序列，使得这个子序列中，前一个均是后一个的前缀。n<=2e3

- ✓ 前缀关系其实是满足一个二维偏序关系的。可以用 $O(n^2 \times len(s))$ 的时间复杂度完成。

```
int check(int x,int y){
    if(s[x].size()>s[y].size())
        return 0;
    for(int i=0;i<s[x].size();i++)
        if(s[x][i]!=s[y][i])
            return 0;
    return 1;
}
```

```
int res=0;
for(int i=1;i<=n;i++){
    f[i]=1;
    for(int j=1;j<i;j++)
        if(check(j,i))
            f[i]=max(f[i],f[j]+1);
    res=max(res,f[i]);
}
```

线性dp

LCS问题

例题： 最长公共子序列（模板）

- ✓ 题目描述： 给定两个长度分别为 n,m 的字符串 A,B，求最长公共子序列的长度。n,m<=1000
- ✓ 状态表示： f[i][j] ： 表示A 的前 i 个字符 B 的前 j 个字符的最长公共子序列的长度
- ✓ 转移方程： $当a[i]=b[j]时, f[i][j]=f[i-1][j-1]+1$

  $当a[i]\neq b[j]时, f[i][j]=\max(f[i-1][j], f[i][j-1])$

  详见过程见第十一讲的讲义
- ✓ 初始值: $f[i][0]=f[0][j]=0$
- ✓ 时间复杂度:O(nm)

线性dp

矩阵路径问题（只能往右或往下）

例题1： 矩阵最大取值

- ✓ 题目描述： 一个大小为 n × m的矩阵，其第 i 行第 j 列的权值为 a_{ij}，**从左上角到右下角走**，每次只能向右走或向下走，经过一个点就会将该点的权值取走，求能取到的和的最大值为多少。n,m<=1000
- ✓ 状态表示：f[i][j]：表示到达第 i 行第 j 列时，能取到的最大值之和。
- ✓ 转移方程： $f[i][j]=\max(f[i-1][j],f[i][j-1])+a[i][j]$
- ✓ 初始值： f[1][1]=a[1][1]
- ✓ 时间复杂度： O(nm)

线性dp

矩阵路径问题（只能往右或往下）

例题2：矩阵路径计数

- ✓ 题目描述： 一个大小为 n×m 的矩阵，有 q 个点a_{ij}不能被到达。从左上角到右下角走，每次只能向右走或向下走，求一共有多少条路径。n,m<=1000,q<=nm，保证起点和终点可到达。
- ✓ 状态表示：f[i][j]：一共有多少条路径到达第 i 行第 j 列。
- ✓ 转移方程：$当a_{ij}=0时, f[i][j]=0;$

  $当a_{ij}=1时, f[i][j]+=f[i-1][j]+f[i][j-1]$
- ✓ 初始值： **f[1][1]=1**
- ✓ 时间复杂度： O(nm)

线性dp

矩阵路径问题（只能往右或往下）

例题3：多维矩阵取数

- ✓ 题目描述： 一个大小为 n×m 的矩阵，其第 i 行第 j 列的权值为 a_{ij}，从左上角到右下角走，每次只能向右走或向下走，经过一个点就会将该点的权值取走（取走后该值变为0），求走两次，能取到的和的最大值为多少。n,m<=100

- ✓ 状态表示：考虑两次一起走，f[i][x][y][u][v]：走了 i 步，第一次走到 (x,y)，第二次走到 (u,v)，能取到的最大值

- ✓ 转移方程：

$$
\begin{aligned}
&f[i][x][y][u][v] = \max \\
&\{f[i-1][x-1][y][u][v], f[i-1][x][y-1][u][v], \\
&f[i-1][x][y][u-1][v], f[i-1][x][y][u][v-1]\} \\
&+[(x,y) \neq (u,v)] \times \{a_{xy} + a_{uv}\} + [(x,y) = (u,v)] \times a_{xy}
\end{aligned}
$$

线性dp

矩阵路径问题（只能往右或往下）

例题3：多维矩阵取数

- ✓ 题目描述： 一个大小为 n×m 的矩阵，其第 i 行第 j 列的权值为 a_{ij}，从左上角到右下角走，每次只能向右走或向下走，经过一个点就会将该点的权值取走（取走后该值变为0），求走两次，能取到的和的最大值为多少。n,m<=100

- ✓ 优化：

- ✓ 状态表示：i+2 = x+y = u+v，我们只需要知道 i,x,u 就可以知道 y = i+2-x ,v = i+2-u 了

- ✓ 转移方程：

$$f[i][x][u]=\max\{f[i-1][x-1][u-1],f[i-1][x-1][u],f[i-1][x][u-1],f[i-1][x][u]\}+(x\neq u)\times(a_{x(i+2-x)}+a_{u(i+2-u)})+(x=u)\times a_{x(i+2-x)}$$

- ✓ 初始化: f[0][1][1]=a_{11}

线性dp

括号匹配问题

例题：只有 ()[] 的括号匹配

- ✓ 题目描述： 给定一个只包含 ()[] 的字符串，求出一个最长合法括号匹配子串，并输出，字符串长度<=1000000
- ✓ 状态表示：f[i]表示以s[i]为结尾的字符串的最长括号匹配
- ✓ 转移方程：当**s[i]为)或者]**时，应该怎样转移呢?
- ✓ f[i−1]表示什么？不正是以s[i−1]结尾的最长括号匹配吗，那么如果s[i−1−f[i−1]]与s[i]匹配的话，f[i]一定等于**f[i−1]+2+f[i−f[i−1]−2]**

f[i−1]代表s[i−1−f[i−1]]与s[i]中间的一段即s[i−1]的最长括号匹配
2即s[i−1−f[i−1]]与s[i]匹配，增加长度为2
s[i−f[i−1]−2]即s[i−f[i−1]−1]的前一个字符，这里**不要漏掉它还可以构成的最长括号匹配的长度**

区间dp

枚举区间断点，合并两区间

- 区间动态规划就是先计算出小区间的权值，大区间由两个已计算好的小区间合并而成，我们通过枚举大区间被两个小区间拆分的断点来求出大区间的最优值。其很明显的至少由两层循环组成，且大部分都是三层循环，通常习惯将：

- 左右端点设为l,r，k设为端点，len设为区间长度
- 通过枚举k为断点计算区间最优值
- 环：通常将其倍长处理，然后求所有长度为 n 的区间即可

通用模板

```
for(int i=1;i<=n;i++)
    f[i][i]=init();//初始化

for(int len=2;len<=n;len++){
    for(int l=1;l+len-1<=n;l++){
        int r=l+len-1;
        for(int k=l;k<r;k++)
            f[l][r]=do(l,k,k+1,r);//进行某种操作
    }
}
```

区间dp

1 3 5 2

最小值：先合并1 3得到4 5 2,再合并5 2得到4 7,最后合并4 7得到11,结果为4+7+11=22

最大值：先合并3 5得到1 8 2,再合并8 2得到1 10,最后合并1 10得到11,结果为8+10+11=29

枚举区间断点，合并两区间
例题1：合并石子

- ✓ 问题描述：已知 n 个石子，每个石子i的权值是 a_i，每次操作可将两堆相邻石子合并，并得到与合并后的石子数量一样的得分，求**最小/最大**得分。
- ✓ 考虑 [l,r] 这堆石子是由 [l,k],[k+1,r] 这两堆石子合并而成

其中**f数组是求最小值,g数组是求最大值**

```
for(int i=1;i<=n;i++)
{
    for(int j=1;j<=n;j++)
        f[i][j]=inf,g[i][j]=-inf;
    f[i][i]=g[i][i]=0;
}
```

```
for(int len=2;len<=n;len++)
    for(int l=1;l+len-1<=n;l++){
        int r=l+len-1;
        for(int k=l;k<r;k++){
            f[l][r]=min(f[l][r],f[l][k]+f[k+1][r]+sum[r]-sum[l-1]);
            g[l][r]=max(g[l][r],g[l][k]+g[k+1][r]+sum[r]-sum[l-1]);
        }
    }
```

区间dp

枚举区间断点，合并两区间
例题2：环形合并石子

- ✓ 问题描述：已知 **n 个石子围成一圈**，每个石子i的权值是 a_i，每次操作可将两堆相邻石子合并，并得到与合并后的石子数量一样的得分，求**最小/最大**得分。

- ✓ 考虑 [l,r] 这堆石子是由 [l,k],[k+1,r] 这两堆石子合并而成，此时采用倍长的思想解决

其中**f数组是求最小值,g数组是求最大值**

区间dp

枚举区间断点，合并两区间

例题3：多边形

- ✓ 问题描述：给定 n 个数 ai 在一个环上。给定 n 个符号 c_i（×/+）在每条边上，要求删除一条边，使得剩下的 n 个数的链通过任意的先后运算，可以变得最大的值。求这个最大的值。n<=50,$a_i \in [-32768,32767]$

- ✓ 问题分析：枚举断边再计算是显然的。但是计算有个明显的贪心错误，就是直接用 f[l][r] 来表示区间[l,r]能算成的最大值，因为**有负数和乘法的存在**，无法通过这种局部最优推出全局最优了。不妨设置两个状态，区间[l,r] 能算成的最大/最小值，这样貌似就 ok 了。环的话，倍长一下就 ok 了

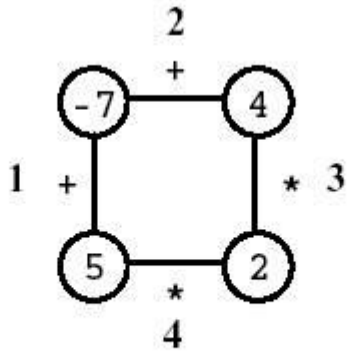

Figure 1. Graphical representation of a polygon

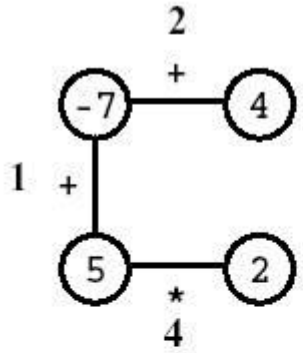

Figure 2. Removing edge **3**

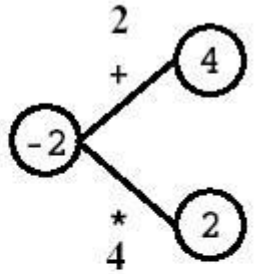

Figure 3. Picking edge **1**

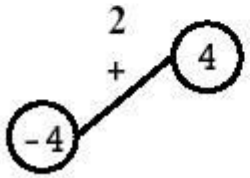

Figure 4. Picking edge **4**

Figure 5. Picking edge **2**

区间dp

枚举区间断点，合并两区间

例题3：多边形

- ✓ 问题描述：给定 n 个数 ai 在一个环上。给定 n 个符号 c_i（×/+）在每条边上，要求删除一条边，使得剩下的 n 个数的链通过任意的先后运算，可以变得最大的值。求这个最大的值。
  n<=50,a_i∈[−32768,32767]

- ✓ 状态表示： f[l][r]：区间[l,r] 能算出的最小值； g[l][r]：区间 [l,r] 能算出的最大值

- ✓ 转移方程： 枚举断点 k ， 然后分类判断整数负数之类就行了，转移方程略

- ✓ 初始化过程略

```
for(int len=2;len<=n;len++)
    for(int l=1;l+len-1<=2*n;l++){
    int r=l+len-1;
    for(int k=l;k<r;k++){
        char op=c[k+1];
        int maxl=g[l][k],maxr=g[k+1][r];
        int minl=f[l][k],minr=f[k+1][r];


 if(op=='t'){//加法
     f[l][r]=min(f[l][r],minl+minr);
     g[l][r]=max(g[l][r],maxl+maxr);
 }
 else if(op=='x'){//乘法
     int x1=minl*minr,x2=minl*maxr,x3=maxl*minr,x4=maxl*maxr;
     f[l][r]=min({f[l][r],x1,x2,x3,x4});
    g[l][r]=max({g[l][r],x1,x2,x3,x4});
 }

    }
}
```

区间dp

枚举区间断点，合并两区间
例题4：涂色

问题描述：给定一排 n 个方块的初始颜色和目标颜色，一共有三种。每次可以将连续的多个方块涂上某颜色，求涂成目标颜色的最少次数。

状态表示： f[i][j]：将区间[i,j] 涂成目标颜色需要的最少次数

当dp[i][j] 满足i=j 时，有：**dp[i][j]=1**　表示i到i只需要1次染色次数

当 dp[i][j] 满足 i≠j&str[i]≠str[j] 时，有：

dp[i][j]=min(dp[i][k]+dp[k+1][j])

k 为 [i,j) 之间的一个断点，就是区间DP的常见套路。i 为左端点， j 为右端点。

当dp[i][j] 满足 i≠j&str[i]=str[j]时，有：　**dp[i][j]=min(dp[i+1][j],dp[i][j−1])**

当 i 和 j 颜色一样，即可以认为 i 是涂 [i+1,j] 的时候 顺便 涂上的，或者 j 是涂 [i,j−1] 的时候顺便涂上的。 又因为求最小值，所以取min 。

区间dp

字符串折叠问题，枚举折点 NEERCYESYESYESNEERCYESYESYES=2(NEERC3(YES))=14

例题:

问题描述：字符串折叠问题：重复的多个字符串S(三个)可被折叠省略为3(S)，求一个长字符串可被折叠成的最短折叠长度。例如：AAAAAAAAAABABABCCD=9(A)3(AB)CCD=12

为撒CC不折叠,因为折叠后反而变长了，变成了 2(C)

状态表示： f[l][r] 表示区间 [l,r] 内的字符串可被折叠成的最短长度

不考虑折叠：f[l][r]=min(f[l][r],f[l][k]+f[k+1][r])

考虑折叠：让 [l,r] 以[l,k] 折叠，若可以折叠，则

$$f[l][r]=\min(f[l][r],f[l][k]+2+num[len/(k-l+1)])$$

其中**2代表多折叠形成的括号()**,而num[len/(k-l+1)]表示折叠块数的位数

区间dp

字符串折叠问题，枚举折点

例题：

问题描述：字符串折叠问题：重复的多个字符串S(三个)可被折叠省略为3(S)，求一个长字符串可被折叠成的最短折叠长度。例如：AAAAAAAAABABABCCD=9(A)3(AB)CCD=12

```
int check(int l,int r,int len){
    for(int i=l;i<=r;i++)
        if(s[i]!=s[(i-l)%len+l])
        //判断[l,r]之间的每一位i与第一段对应的字符是否相同
            return 0;
    return 1;
}
```

```
for(int len=2;len<=n;len++)
    for(int l=1;l+len-1<=n;l++){
        int r=l+len-1;
        for(int k=l;k<r;k++)//不折叠
            f[l][r]=min(f[l][r],f[l][k]+f[k+1][r]);
            for(int k=l;k<r;k++){//折叠
                int lenn=k-l+1;
                if(len%lenn!=0)
                    continue;
                if(check(l,r,lenn))
                    f[l][r]=min(f[l][r],f[l][k]+2+num[len/lenn]);
                //2是多的(),num[len/lenn]折叠块数的位数
            }
    }
```

区间dp

只能从左边界或右边界添加元素的区间DP

区间 [l,r] 并不能被拆分为[l,k],[k+1,r]，只能被拆分为f[l,r-1][l+1,r]，因为最后加入一个点只能从最左边或最右边加入。

例题：从左边界或右边界添加元素

问题描述：假设有 n 个人，每个人身高为 a_i，排成一列。从前往后依次重新进入新的队列中。如果**当前加入的人比前一个人高，则其只能加入到新的队列的最后面**；否则**其只能加入到新的队列的最前面**。给定一个最终的排列，请问有多少种原来的排列满足重新排队之后变成最终的样子。n<=1000

有 6 个人站成一个初始队形，身高依次为 1850,1900,1700,1650,1800,1750

按以下步骤获得最终排出的队形：

- ✓ 1850。
- ✓ 1850, 1900，因为 1900 > 1850。
- ✓ 1700,1850,1900，因为 1700<1900。
- ✓ 1650,1700,1850,1900，因为 1650<1700。
- ✓ 1650,1700,1850,1900,1800，因为 1800>1650。
- ✓ 1750,1650,1700,1850,1900,1800，因为 1750<1800。

因此，最终排出的队形是 1750,1650,1700,1850,1900,1800。

区间dp

只能从左边界或右边界添加元素的区间DP

区间 [l,r] 并不能被拆分为[l,k],[k+1,r]，只能被拆分为f[l,r-1]或f[l+1,r]，因为最后加入一个点只能从最左边或最右边加入。

例题：从左边界或右边界添加元素

问题描述：假设有 n 个人，每个人身高为 a_i，排成一列。从前往后依次重新进入新的队列中。如果**当前加入的人比前一个人高，则其只能加入到新的队列的最后面**；否则**其只能加入到新的队列的最前面**。给定一个最终的排列，请问有多少种原来的排列满足重新排队之后变成最终的样子。n<=1000

状态表示： f[l][r][k=0/1]：最终排列为 [l,r] 且最后加入的人是**最前面(左端)/最后面(右端)**的人的方案数

区间dp

只能从左边界或右边界添加元素的区间DP

区间 [l,r] 并不能被拆分为[l,k],[k+1,r]，只能被拆分为f[l,r-1][l+1,r]，因为最后加入一个点只能从最左边或最右边加入。

例题：从左边界或右边界添加元素

问题描述：假设有 n 个人，每个人身高为 a_i，排成一列。从前往后依次重新进入新的队列中。如果**当前加入的人比前一个人高，则其只能加入到新的队列的最后面**；否则**其只能加入到新的队列的最前面**。给定一个最终的排列，请问有多少种原来的排列满足重新排队之后变成最终的样子。n<=1000

转移方程： 每一次的插入位置只与上一个数有关，那么我们只要分出 4 种情况进行讨论即可

当右端点的数大于上一个且上一个插入在右端时：f[l][r][1] += f[l][r - 1][1];

当右端点的数大于左端点且上一个插入在左端时：f[l][r][1] += f[l][r - 1][0];

当左端点的数小于后一个且上一个插入在左端时：f[l][r][0] += f[l + 1][r][0];

当左端点的数小于右端点且上一个插入在右端时：f[l][r][0] += f[l + 1][r][1];

区间dp

只能从左边界或右边界添加元素的区间DP

区间 [l,r] 并不能被拆分为[l,k],[k+1,r]，只能被拆分为f[l,r-1][l+1,r]，因为最后加入一个点只能从最左边或最右边加入。

例题：从左边界或右边界添加元素

问题描述：假设有 n 个人，每个人身高为 a_i，排成一列。从前往后依次重新进入新的队列中。如果**当前加入的人比前一个人高，则其只能加入到新的队列的最后面**；否则**其只能加入到新的队列的最前面**。给定一个最终的排列，请问有多少种原来的排列满足重新排队之后变成最终的样子。n<=1000

目标值：f[1][n][0]+f[1][n][1]

初始化： 按理解是f[i][i][0]=f[i][i][1]=1，可是如果只有一个人，则方案数为2，错了。我们默认一个人的时候，是从左边进去的。即：f[i][i][0]=1

区间dp

只能从左边界或右边界添加元素的区间DP

例题：从左边界或右边界添加元素

问题描述：假设有 n 个人，每个人身高为 a_i，排成一列。从前往后依次重新进入新的队列中。如果**当前加入的人比前一个人高，则其只能加入到新的队列的最后面**；否则**其只能加入到新的队列的最前面**。给定一个最终的排列，请问有多少种原来的排列满足重新排队之后变成最终的样子。n<=1000

```
for(int len=2;len<=n;len++){
        for(int l=1;l+len-1<=n;l++){
                int r=l+len-1;
                if(a[r]>a[r-1])
                        f[l][r][1]+=f[l][r-1][1];
                if(a[r]>a[l])
                        f[l][r][1]+=f[l][r-1][0];
                if(a[l]<a[l+1])
                        f[l][r][0]+=f[l+1][r][0];
                if(a[l]<a[r])
                        f[l][r][0]+=f[l+1][r][1];
                f[l][r][0]%=mod;
                f[l][r][1]%=mod;
        }
}
```