综合动态规划

Robert_JYH
洛谷网校

动态规划(DP)

- 预先解决小规模的问题，用这些小问题的解去推出大问题的。
- 这种解决问题的思维称为动态规划，简称 DP。

- 三要素：状态、阶段、决策
- 三个基本条件：子问题重叠性、无后效性、最优子结构性质

一、线性DP的基本模型

序列问题：最长上升子序列

- 在一个给定的序列中，找到一个最长的子序列，使得这个子序列元素的数值依次递增。
- 子序列：子序列就是在原来序列中找出一部分组成的序列（子序列不一定连续）

- 例如，长度为 7 的序列：3 1 2 1 8 5 6
- 最长上升子序列为 1 2 5 6

序列问题：最长上升子序列

- 在一个给定的序列中，找到一个最长的子序列，使得这个子序列元素的数值依次递增。

- 设计状态（描述当前遇到的问题，涉及什么状态就放什么）：

  $f[x]$ 表示以 $a[x]$ 结尾的最长上升子序列的长度

- 决策：转移的条件（思考能从哪里转移来）：

  $1 \le j < i$ 且 $a[i] > a[j]$

- 转移（转移时会有什么影响）：

  最长上升子序列的长度相较之前加 1

- $f[i] = \max_{1 \le j < i \text{ and } a[i] > a[j]}(f[j] + 1)$

扩展：$O(n \log n)$求最长上升子序列

- 在一个给定的序列中，找到一个最长的子序列，使得这个子序列元素的数值依次递增。

- 维护一个辅助序列 num，$num[i]$ 表示在所有长度为 i 的上升子序列中，末尾元素的最小值。设 num 序列当前有len个元素。
- 从前往后考虑每个数，如果当前数比 $num[len]$ (序列尾部)更大，直接将其插入序列尾部，序列长度 $len++$。
- 否则用它替换掉序列中，第一个大于等于它的数（保持序列单调性）。
- 最后得到的序列长度即为最长上升子序列长度。

扩展：$O(n \log n)$求最长上升子序列

```
int len=1;
num[1]=a[1];
for(int i=2;i<=n;i++) {//依次遍历每个位置

    if(a[i]>num[len]) { //如果大于序列尾部，直接插入到序列尾部
        ++len;
        num[len]=a[i]; //直接插入到序列尾部
    }
    else{ //否则，通过二分查找，替换掉序列中第一个大于等于它的数
        int pos=lower_bound(num+1,num+len+1,a[i])-num;
        num[pos]=a[i];
    }
}
```

序列问题：最长公共子序列

- 求出两个序列最长的公共子序列的长度。

- 设计状态（描述当前遇到的问题，涉及什么状态就放什么）：
- $f[i][j]$ 表示两个序列到位置i，j为止的最长公共子序列的长度
- 决策：转移的条件（思考能从哪里转移来）：
- 三种情形，$(i,j-1)(i-1,j)(i-1,j-1)$最后一种满足$a[i]=b[j]$
- 转移（转移时会有什么影响）：最后一种的长度相较之前加1，其它不变
- $f[i][j]=\max(f[i-1][j],f[i][j-1],f[i-1][j-1]+1)$

例题-P1004 [NOIP2000 提高组] 方格取数

- 设有 $N \times N$ 的方格图 $(N \leq 200)$，每个方格上有一个自然数。某人从图的左上角的 A 点出发，可以向下行走，也可以向右走，直到到达右下角的 B 点。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字 0）。

- 此人从 A 点到 B 点共走两次，试找出 2 条这样的路径，使得取得的数之和最大。
- (1)两路径可有重叠部分
- (2)两路径不可有重叠部分（除AB点外）

A 0	0	0	0	0	0	0	0
0	0	13	0	0	6	0	0
0	0	0	0	7	0	0	0
0	0	0	14	0	0	0	0
0	21	0	0	0	4	0	0
0	0	15	0	0	0	0	0
0	14	0	0	0	0	0	0
0	0	0	0	0	0	0	0 B

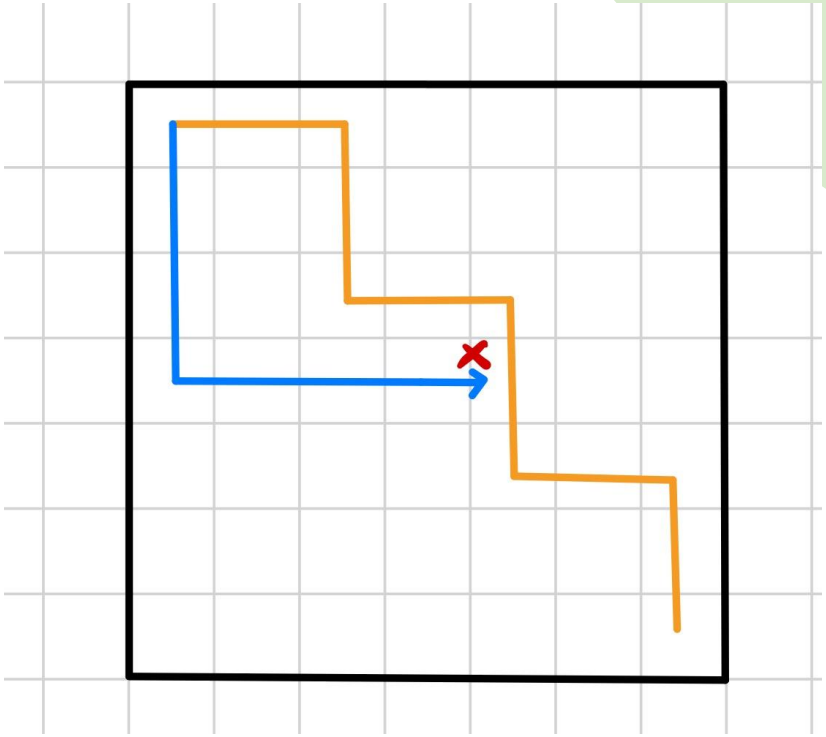
坐标类问题

- 首先分析第一问：两路径可有重叠部分 。
- 考虑两条路径同时走，很自然地想到用$f[x_1][y_1][x_2][y_2]$表示两点位置及此时最大值。
- 由于同一时刻$x_1 + y_1 = x_2 + y_2 = T + 1$，可新设一个变量表示时刻，得到 $f[T][x_1][x_2]$。
- 在很多题目中我们都可以利用变量间的相互制约关系来改进状态，以优化时间、空间。

- 在本题中可以利用滚动数组、倒序等方式进一步优化空间。

坐标类问题

- 然后分析第二问：两路径不可有重叠部分 。
- 对于不可重叠的条件可以$x_1 < x_2$，然后在转移时做简单判断即可。



例题-P1004 [NOIP2000 提高组] 方格取数

```
b[2][1][1] = 0;
for (int i = 3; i < n + m; i++) { //枚举时刻
    for (int x = 1; x < n; x++) { //枚举一个位置
        for (int y = x + 1; y <= n; y++) { //枚举另一个位置
            //枚举所有可能的转移方式
            int t = b[i][x][y];
            t = max(t, b[i - 1][x][y]);
            t = max(t, b[i - 1][x - 1][y]);
            t = max(t, b[i - 1][x][y - 1]);
            t = max(t, b[i - 1][x - 1][y - 1]);
            if (t == -1) {
                continue;
             }
            b[i][x][y] = a[x][i - x] + a[y][i - y] + t;
        }
    }
}
cout << b[n + m - 1][n - 1][n] << endl;
```

例题-P2513 [HAOI2009] 逆序对数列

- 对于一个数列 a_i，如果有 $i < j$ 且 $a_i > a_j$，那么我们称它们为一对逆序对数。
- 对于一个由 $1 \sim n$ 组成的排列，可以很容易求出有多少个逆序对数。
- 求逆序对数为 m 的排列有多少个？
- 答案对 10000 取模。
- $n \leq 1000, m \leq 1000$

例题-P2513 [HAOI2009] 逆序对数列

- 设$f[i][j]$表示由 1 到 i 组成的所有排列中，逆序对数量为 j 的排列个数。
- 考虑把数字 i 插入到之前的排列中。将数字 i 插入到位置 k 后，会产生 $i-1-k$ 个逆序对数。

$$f[i][j]=\sum_{k=0}^{i-1} f[i-1][j-(i-1-k)]$$

$$=\sum_{k=j-i+1}^{j} f[i-1][k]$$

- 时间复杂度 $O(n^2k)$，无法通过本题。

例题-P2513 [HAOI2009] 逆序对数列

- 设$f[i][j]$表示由 1 到 i 组成的所有排列中，逆序对数量为 j 的排列个数。
- 考虑把数字 i 插入到之前的排列中。将数字 i 插入到位置 k 后，会产生 $i-1-k$ 个逆序对数。

$$f[i][j]=\sum_{k=\max(0,j-i+1)}^{j} f[i-1][k]$$

- 我们需要转移的同时记录前缀和，设

$$sum[i][j]=\sum_{k=0}^{j} f[i][k]$$

-

$$f[i][j]=\begin{cases} sum[i-1][j]-sum[i-1][j-i], & j-i\geq 0 \\ sum[i-1][j], & j-i<0 \end{cases}$$

例题-P2513 [HAOI2009] 逆序对数列

```
f[0][0] = 1;
for (int i = 0; i <= m; i++) sum[0][i] = 1;
for (int i = 1; i <= n; i++) {
    for (int j = 0; j <= m; j++) {
        f[i][j] = sum[i - 1][j];
        if (j - i >= 0) f[i][j] -= sum[i - 1][j - i];
        sum[i][j] = f[i][j];
        if (j > 0) sum[i][j] += sum[i][j - 1];
        f[i][j] %= mod, sum[i][j] %= mod;
    }
}
if (f[n][m] < 0) f[n][m] += mod;
```

二、记忆化搜索

例题-P1044 [NOIP2003 普及组] 栈

- 一个$1\sim n$的序列，现在可以进行两种操作：

（1）将一个数，从操作数序列的头端移到栈的头端（对应数据结构栈的 push 操作）

（2）将一个数，从栈的头端移到输出序列的尾端（对应数据结构栈的 pop 操作）

- 输出经过n次操作可能得到的输出序列的总数。
- $n \leq 18$

例题-P1044 [NOIP2003 普及组] 栈

- 首先考虑暴力的搜索写法。

```
void dfs(int x,int y){
    if(x==0){ans++;return;}
    if(y>0)dfs(x,y-1);//出栈
    dfs(x-1,y+1); //入栈
}//t约为10^9
```

- 我们可以保存dfs的结果，一并返回。

```
int dfs(int x,int y){
    if(x==0){return 1;}
    int ans=0;
    if(y>0)
       ans+=dfs(x,y-1);
    ans+=dfs(x-1,y+1);
    return ans;
}//t约为10^9
```

例题-P1044 [NOIP2003 普及组] 栈

- 我们发现可以进一步用数组保存所有搜过的状态，避免重复搜索。

```
int dfs(int x,int y){
    if(f[x][y]) return f[x][y];
    if(x==0){return 1;}
    if(y>0) f[x][y]+=dfs(x,y-1);
    f[x][y]+=dfs(x-1,y+1);
    return f[x][y];
}//t=325
```

- 这与DP的递推写法是对应的。

```
for(int x=0;x<=n;x++)
    for(int y=0;y<=n;y++)
        if(x==0)f[x][y]=1;
        else if(y==0)f[x][y]=f[x-1][y+1];
        else f[x][y]=f[x][y-1]+f[x-1][y+1];
```

记忆化搜索

- 不依赖外部变量，答案直接通过转移记录在数组中，无后效性
- 和递推同样为 DP 的表现形式，便于我们从暴力的搜索解法直接转化而来
- 思考过程：（无返回值暴搜）-有返回值暴搜-记忆化-（DP）

  或：DP式子-记忆化搜索

三、背包问题

01背包

- 有 N 件物品和一个容量为 V 的背包。每件物品只有一件。放入第 i 件物品耗费的费用是 C_i，得到的价值是 W_i。求解将哪些物品装入背包可使价值总和最大。

- $F[i,v]=\max\{F[i-1,v],F[i-1,v-C_i]+W_i\}$
- 空间优化
  1) 滚动数组
  2) 倒序循环 $F[v]=\max\{F[v],F[v-C_i]+W_i\}$
- 初始化:

(1)恰好装满$F[]=\{0,-\infty,\dots\}$

(2)价值最大$F[]=\{0\}$

完全背包与多重背包

- 完全背包：有 N 件物品和一个容量为 V 的背包。每件物品有无数件。放入第 i 件物品耗费的费用是 C_i，得到的价值是 W_i。求解将哪些物品装入背包可使价值总和最大。
- 正序循环

$$F[v] = \max\{F[v], F[v - C_i] + W_i\}$$

- 多重背包：有 N 件物品和一个容量为 V 的背包。每件物品有 M_i 件。放入第 i 件物品耗费的费用是 C_i，得到的价值是 W_i。求解将哪些物品装入背包可使价值总和最大。
- 当成01背包即可，可使用二进制拆分。
- 单调队列优化将在提高组阶段学习。

例题-P5662 [CSP-J2019] 纪念品

- 小伟突然获得一种超能力，他知道未来T天N种纪念品每天的价格
- 每天，小伟可以进行以下两种交易无限次：

  1) 任选一个纪念品，若手上有足够金币，以当日价格购买该纪念品；

  2) 卖出持有的任意一个纪念品，以当日价格换回金币。
- 每天卖出纪念品换回的金币可以立即用于购买纪念品，当日购买的纪念品也可以当日卖出换回金币。
- 小伟现在有M枚金币，他想要在T天后拥有尽可能多的金币。
- $T \leq 100, N \leq 100, M \leq 1000$，价格 $\leq 10^4$

例题-P5662 [CSP-J2019] 纪念品

- 对于当天的某件商品，我们可以选择：不买/买了第二天卖/买了过几天卖。
- 第三种情况等效于我们每天先卖掉它，然后当天立刻买回来。于是我们可以只考虑相邻两天之间的交易。

- 对于相邻两天，我们把今天手里的钱当做背包的容量，把商品今天的价格当成它的消耗，把商品明天的价格当做它的价值。
- 于是，我们进行 $t-1$ 轮完全背包。每一天结束后把总钱数加上今天赚的钱即可。

例题-P5662 [CSP-J2019] 纪念品

```
for(int k=1;k<T;k++){ //依次考虑每一天
    memset(f,0,sizeof f);
    for(int i=1;i<=n;i++) //对于每天执行完全背包
        for(int j=a[k][i];j<=m;j++)
            f[j]=max(f[j],f[j-a[k][i]]+a[k+1][i]-a[k][i]);
    m+=f[m];
}
```

其它背包类型

- 混合背包：每件物品有一/多/无限件。为了时间效率，需要判断物品种类后将三种物品分开写（特别是完全+多重）。

- 二维费用：如果有 N 件物品和一个容量为 V 、U的背包。放入第 i 件物品耗费的费用是 C_i、D_i，得到的价值是 W_i。求解将哪些物品装入背包可使价值总和最大。
- DP式子加一维即可。注意隐藏的二维费用：共可取U件物品。

- 分组背包：物品被划分为 K 组，每组中的物品互相冲突，最多选一件。求解将哪些物品装入背包可使价值总和最大。
- 循环改为组号、倒序重量、组内物品（注意循环顺序）。

例题-P1064 [NOIP2006 提高组] 金明的预算方案

- 共有m件物品，金明把想买的物品分为两类：主件与附件，附件是从属于某个主件的，如果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有 0 个、1个或 2 个附件。每个附件对应一个主件，附件不再有从属于自己的附件。
- 他把每件物品规定了一个重要度，用整数 $1 \sim 5$ 表示。他还查到了每件物品的价格。他希望在不超过 n 元的前提下，使每件物品的价格与重要度的乘积的总和最大。
- $n \leq 32000, m \leq 60$

例题-P1064 [NOIP2006 提高组] 金明的预算方案

- 对于每个主件以及依附于它的附件，构成一个物品组，对于这个物品组，我们有五种购买方式：
- 不购买
- 买且只买主件
- 买主件，并且买附件1
- 买主件，并且买附件2
- 买这个主件，并且买附件1和附件2

- 最外层遍历物品组编号，第二层遍历钱数，转移时取以上五种情况中的最大值即可。

例题-P1064 [NOIP2006 提高组] 金明的预算方案

```
for(int i=1;i<=m;i++){ //遍历每个物品组
    for(int j=N;j>=v[i];j--){ //遍历钱数
        //依次考虑五种购买方式
        dp[j]=max(dp[j],dp[j-v[i]]+v[i]*p[i]);
        if(j>=v[i]+v1[i])
            dp[j]=max(dp[j],dp[j-v[i]-v1[i]]+v[i]*p[i]+v1[i]*p1[i]);
        if(j>=v[i]+v2[i])
            dp[j]=max(dp[j],dp[j-v[i]-v2[i]]+v[i]*p[i]+v2[i]*p2[i]);
        if(j>=v[i]+v1[i]+v2[i])
            dp[j]=max(dp[j],
             dp[j-v[i]-v1[i]-v2[i]]+v[i]*p[i]+v1[i]*p1[i]+v2[i]*p2[i]);
        mx=max(mx,dp[j]);
    }
}
```

依赖背包

- 简化的依赖背包：背包问题的物品间存在某种“依赖”的关系。物品 i 依赖于物品 j，表示若选物品 i，则必须选物品 j。为了简化起见，我们假设没有某个物品既依赖于别的物品，又被别的物品所依赖；另外，没有某件物品同时依赖多件物品。

- 对于构成依赖关系的物品，对其做出的各个决策可看成单独物品，全部决策构成一个互斥物品组。分物品组编号、容量、组内决策三层考虑。

- 依赖背包：如果不满足上述假设，则需引入树形DP解决，我们将在之后的课程中学习。

例题-P5020 [NOIP2018 提高组] 货币系统

- 在网友的国度中共有 n 种不同面额的货币，第 i 种货币的面额为 $a[i]$，你可以假设每一种货币都有无穷多张。为了方便，我们把货币种数为 n、面额数组为 $a[1..n]$ 的货币系统记作 (n,a)。
- 在一个完善的货币系统中，每一个非负整数的金额 x 都应该可以被表示出，即对每一个非负整数 x，都存在 n 个非负整数 $t[i]$ 满足 $a[i]\times t[i]$ 的和为 x。然而，在网友的国度中，货币系统可能是不完善的，即可能存在金额 x不能被该货币系统表示出。例如在货币系统 $n=3,a=[2,5,9]$ 中，金额 1,3就无法被表示出来。
- 两个货币系统 (n,a) 和 (m,b) 是等价的，当且仅当对于任意非负整数 x，它要么均可以被两个货币系统表出，要么不能被其中任何一个表出。
- 现在网友们打算简化一下货币系统。他们希望找到一个货币系统 (m,b)，满足 (m,b) 与原来的货币系统 (n,a) 等价，且 m 尽可能的小。他们希望你来协助完成这个艰巨的任务：找到最小的 m。
- $n\le 100, a_i\le 25000$

例题-P5020 [NOIP2018 提高组] 货币系统

- 可以证明，在与货币系统 (n,a) 等价且 m 最小的货币系统 (m,b) 中，必然有一种货币系统面值都在 a中出现过。
- 若 a 中一种面额 a_i 能被比这个面额更小的面额表示，则 a_i 不用保留。
- 那么只需要用完全背包的方法检查这个面额是否能被更小的面额表示出来即可。

例题-P5020 [NOIP2018 提高组] 货币系统

```
sort(a+1,a+n+1);
mx=a[n];
for(int i=1;i<=mx;++i)f[i]=0; //初始化
f[0]=1;
for(int i=1;i<=n;++i){
    int w=a[i];
    if(f[a[i]]){--ans;continue;} //若能被表示出，则不需要
    for(int j=w;j<=mx;++j)if(f[j-w])f[j]=1; //完全背包转移
}
```

其它设问

- 求方案数：

1.求方案总数：初值$f[]=\{1,0,0,\dots\}$，将max/min 改为求和

2.求最优方案数：两个数组，一个记录最大价值，一个记录最大价值的最优方案数

- 输出方案：

1.最优方案：记录下每个状态的最优值是由状态转移方程的哪一项推出来的，换句话说，记录下它是由哪一个策略推出来的。便可根据这条策略找到上一个状态，从上一个状态接着向前推即可。

2.第k优方案：对于每个状态维护一个大小为k的解的队列，每次转移将两个队列合并后的前k优解记录。

小结

- 背包问题的关键是能够准备地识别、转化模型。
- 遇到看上去像背包，但又不够明显的问题，可以从泛化背包的角度考虑。

- 背包问题向我们揭示了动态规划的一些基本内容：

（1）设问角度：最优解/第k优解/可行解及方案数、具体方案

（2）状态设置：最基本的状态设置为，题目中出现一个全局变量就设置一维状态，之后再视图简化状态

（3）转移时要通过正逆序等方式使DP无后效性，同时注意循环顺序

（4）空间优化：滚动数组/减去不再使用的维度

四、区间DP

例题-P1775 石子合并（直线）

设有$N(N \leq 300)$堆石子排成一排，每堆石子有一定的质量m_i $(m_i \leq 1000)$。

现在要将这N堆石子合并成为一堆。每次只能合并相邻的两堆，合并的代价为这两堆石子的质量之和。

试找出一种合理的方法，使总的代价最小，并输出最小代价。

例题-P1775 石子合并（直线）

- 状态：设$f[i][j]$表示从区间i到j合并产生的最小值
- 决策：枚举分界点 k，即新区间是由哪两个区间合并而来的
- 转移：$f[L][R]=\{f[L][R],f[L][k]+f[k+1][R]+\text{代价}\}$
- 最终答案为$f[1][N]$

例题-P1775 石子合并（直线）

```
for (int L = 2; L <= n; L++)
{ // 枚举长度
    for (int i = 1; i <= n - L + 1; i++)
    { // 枚举左端点
        int j = i + L - 1;
        f[i][j] = inf;
        for (int k = i; k < j; k++)
        { // 枚举分界点
            f[i][j] = min(f[i][j], f[i][k] + f[k + 1][j] +
sum[j] - sum[i - 1]);
        }
    }
}
cout << f[1][n] << endl;
```

区间DP

- 区间动态规划问题的基本思想是，对于每个区间，他们的最优值都是由几段更小区间的最优值得到，于是可以分而治之。
- 状态：一般设定$f[i][j]$表示$[i,j]$这个区间的最优值。
- 决策与转移：大区间是由小区间推导出来的，于是需要枚举分界点转移，并附加代价。

例题-P1005 [NOIP2007 提高组] 矩阵取数游戏

- 对于一个给定的$n \times m$的矩阵，矩阵中的每个元素为非负整数。游戏规则如下：

1. 每次取数时须从每行各取走一个元素，共n个。m次后取完矩阵所有的元素；

2. 每次取走的各个元素只能是该元素所在行的行首或行尾；

3. 每次取数都有一个得分值，为每行取数的得分之和；每行取数的得分 = 被取走的元素值$\times 2^i$，其中i表示第i次取数。

- 游戏结束总得分为 m 次取数得分之和。出取数后的最大得分。
- $n, m \leq 80$

例题-P1005 [NOIP2007 提高组] 矩阵取数游戏

- 各行之间独立不受影响，$f[i][j]$ 表示当前行还剩下区间$[i,j]$中的数没取时，本行得分的最大值。
- $f[i][j]$ 必然从 $f[i-1][j]$ 或 $f[i][j+1]$ 中的一个转移过来，对于每行两层循环枚举即可。
- $f[i][j]=\begin{cases} f[i-1][j]+\text{取}a[i-1]\text{的得分} \\ f[i][j+1]+\text{取}a[j+1]\text{的得分} \end{cases}$
- 第 x 行的答案为 $ans_x=\max_{1\le i\le n}\{f[i][i]+2^m\times a[i]\}$

例题-P1005 [NOIP2007 提高组] 矩阵取数游戏

```
pre[0] = 1;
for (int i = 1; i <= m; i++) pre[i] = pre[i - 1] * 2; //预处理幂次
for (int i = 1; i <= n; i++) { //按行依次处理
    memset(f, 0, sizeof(f));
    for (int j = 1; j <= m; j++) scanf("%d", &a[j]);
    for (int j = 1; j <= m; j++) //枚举左右端点
        for (int k = m; k >= j; k--) { //考虑从左/右转移而来
            f[j][k] = max(f[j][k],
                          f[j - 1][k] + pre[m + j - k - 1] * a[j - 1]);
            f[j][k] = max(f[j][k],
                          f[j][k + 1] + pre[m + j - k - 1] * a[k + 1]);
        }
    maxx = 0;
    for (int j = 1; j <= m; j++) maxx = max(maxx, f[j][j] + pre[m] * a[j]);
        //枚举最后一个转移点
    ans += maxx;
}
//注：本题需要使用高精度
```

例题-P4342 [IOI1998] Polygon

- 多边形是一个玩家在一个有n个顶点的多边形上的游戏。
- 第一步，删除其中一条边。
- 随后每一步：选择一条边连接的两个顶点V1和V2，用边上的运算符计算V1和V2得到的结果来替换这两个顶点。
- 游戏结束时，只有一个顶点，没有多余的边。
- 计算最高可能的分数，并输出初始删掉哪条边得到最高分数。
- $n \leq 50$

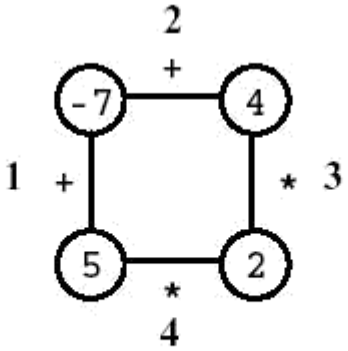


Figure 1. Graphical representation of a polygon

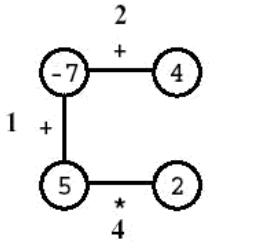


Figure 2. Removing edge **3**

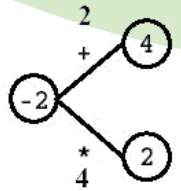


Figure 3. Picking edge **1**

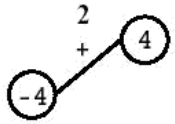


Figure 4. Picking edge **4**

Figure 5. Picking edge **2**

例题-P4342 [IOI1998] Polygon

- 第一次删掉哪条边可以枚举，之后便转变为正常的区间DP。
- 但要注意，如果只保存$[l,r]$的最大值，这样DP的结果不满足最优子结构与无后效性，因为对于乘法存在负负得正的情形。
- 为了解决这个问题，我们需要保存$[l,r]$的最大值与最小值。

	加法	乘法
最大值	最大加最大	最大乘最大 最小乘最小
最小值	最小加最小	最大乘最大 最小乘最小 最大乘最小

例题-P4342 [IOI1998] Polygon

```
for (int len = 2; len <= n; len++) { //枚举长度
    for (int l = 1; l + len - 1 <= 2 * n; l++) { //枚举左端点
        int r = l + len - 1;
        for (int k = l + 1; k <= r; k++) //枚举中间点
            if (ch[k] == 't') { //考虑符号
                f[l][r] = max(f[l][r], f[l][k - 1] + f[k][r]);
                g[l][r] = min(g[l][r], g[l][k - 1] + g[k][r]);
            } else {
                f[l][r] = max(f[l][r], max(f[l][k - 1] * f[k][r],
                                           g[l][k - 1] * g[k][r]));
                g[l][r] = min(g[l][r], min(g[l][k - 1] * g[k][r],
                                  min(f[l][k - 1] * f[k][r],
                                  min(f[l][k - 1] * g[k][r],
                                          g[l][k - 1] * f[k][r]))));
            }
    }
}
```

五、环形DP

例题-P1880 [NOI1995] 石子合并（环形）

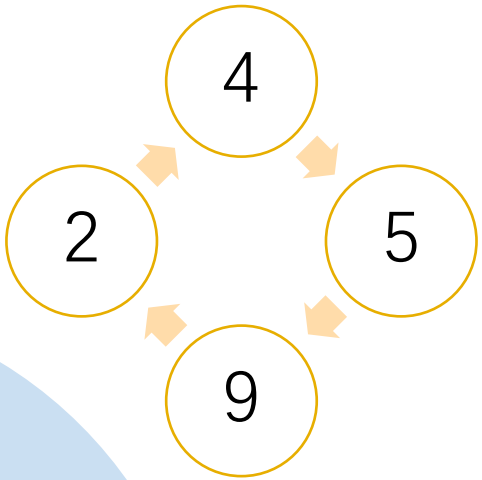
设有$N(N \leq 100)$堆石子排成一圈，每堆石子有一定的质量m_i $(m_i \leq 20)$。

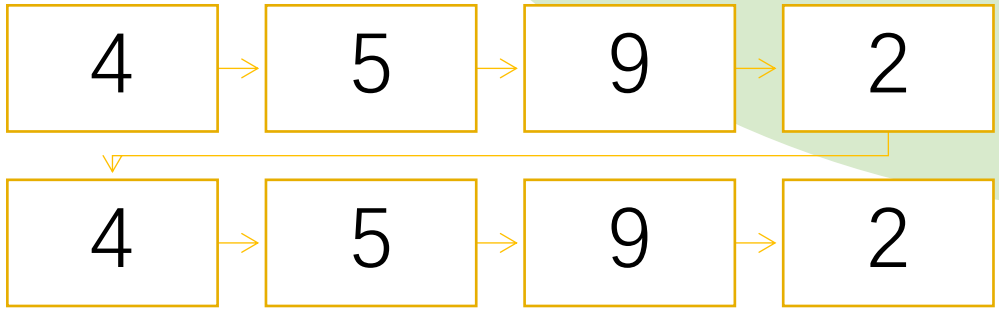
现在要将这N堆石子合并成为一堆。每次只能合并相邻的两堆，合并的代价为这两堆石子的质量之和。

试找出一种合理的方法，使总的代价最小/最大，并输出最小代价/最大代价。

环形的处理

- 对于环形问题，我们常常使用断环为链的技巧。
- 具体地，我们将原数组复制一份接在最后。
- 此时，每连续n个元素即对应断环为链的其中一种情况，依次考虑可能的n种情况即可。





例题-P1880 [NOI1995] 石子合并（环形）

```
for (int i = 1; i <= n; i++) // 复制原数组
    stone[i + n] = stone[i];
for (int L = 2; L <= n; L++)
{ // 枚举长度，注意上限仍为n
    for (int i = 1; i <= 2 * n - L + 1; i++)
    { // 枚举左端点
        int j = i + L - 1;
        Fmax[i][j] = 0;
        for (int k = i; k < j; k++)
            Fmax[i][j] = max(Fmax[i][j], Fmax[i][k] + Fmax[k + 1][j]);
        Fmax[i][j] += sum[j] - sum[i - 1];
    }
}
for (int i = 1; i <= n; i++) // 依次考虑可能的n种情况
    ans = max(ans, Fmax[i][i + n - 1]);
```

P1057 [NOIP2008 普及组] 传球游戏

- 老师带着同学们一起做传球游戏。
- 游戏规则是这样的：n 个同学站成一个圆圈，其中的一个同学手里拿着一个球，当老师吹哨子时开始传球，每个同学可以把球传给自己左右的两个同学中的一个。
- 聪明的小蛮提出一个有趣的问题：有多少种不同的传球方法可以使得从小蛮手里开始传的球，传了 m 次以后，又回到小蛮手里。
- 两种传球方法被视作不同的方法，当且仅当这两种方法中，接到球的同学按接球顺序组成的序列是不同的。
- $n, m \leq 30$

环形的处理

- 对于这类环形问题我们可以通过取模实现。（为方便，i,j从0到m/n）。
- 设$f[i][j]$表示已经传了 i 次球，且当前球在编号是 j 的小朋友手上的方案数。
- $f[i+1][(j-1+n)\%n] \mathrel{+}= f[i][j]$往左传
- $f[i+1][(j+1)\%n] \mathrel{+}= f[i][j]$ 往右传
- 另外，这种由当前状态到其它状态的DP方式叫刷表法，以前我们常用的由其它状态到当前状态的DP方式叫做填表法。

P1057 [NOIP2008 普及组] 传球游戏

```
f[0][0] = 1; //初始方案数为1
for(int i = 0; i < m; i++)
    for(int j = 0; j < n; j++)
        if(f[i][j]) {
            f[i + 1][(j - 1 + n) % n] += f[i][j]; //向左
            f[i + 1][(j + 1) % n] += f[i][j]; //向右
        }
printf("%d", f[m][0]);
```

动态规划(DP)总结

1.状态类型/结构:

序列、背包、区间、坐标、环形……

2.转移方式: 递推、记忆化搜索

3.优化:

(1)预处理、前缀和

(2)状态量: 跳过无用状态、改进状态表示、利用约束关系

(3)空间: 滚动数组