动态规划

DP

333lfy

ZUCC ACM Group

2022.01.18

目录

1. 相关概念
2. 数塔
3. 象棋马
4. 背包问题
5. 线性 dp 问题

目录

1. 相关概念
2. 数塔
3. 象棋马
4. 背包问题
5. 线性 dp 问题

相关概念

- 动态规划是运筹学的一个分支，是求解决策过程最优化的过程。

相关概念

- 动态规划是运筹学的一个分支，是求解决策过程最优化的过程。
- 将所给问题的过程，恰当的分为若干相互联系的阶段，以便能按一定的次序求解问题。

相关概念

- 动态规划是运筹学的一个分支，是求解决策过程最优化的过程。
- 将所给问题的过程，恰当的分为若干相互联系的阶段，以便能按一定的次序求解问题。
- 阶段的划分一般是根据时间和空间的特征进行的，但是要能够把问题的过程转化为多阶段决策问题。

相关概念

- 动态规划是运筹学的一个分支，是求解决策过程最优化的过程。
- 将所给问题的过程，恰当的分为若干相互联系的阶段，以便能按一定的次序求解问题。
- 阶段的划分一般是根据时间和空间的特征进行的，但是要能够把问题的过程转化为多阶段决策问题。
- 主要内容是建立问题中的状态，以及通过状态之间的联系相互转移需求最优决策。

解题思路

基本思想：给定一个复杂的问题，我们可以将其 转化 为较简单的 子问题 ，根据子问题的解得到原问题的解。

定义问题状态

定义问题的状态，以及目标值。

解题思路

基本思想：给定一个复杂的问题，我们可以将其 转化 为较简单的 子问题 ，根据子问题的解得到原问题的解。

定义问题状态

定义问题的状态，以及目标值。

考虑状态转移

原问题的目标值（最大、最小、计数），一定是由子问题的目标值转移而来。

解题思路

基本思想：给定一个复杂的问题，我们可以将其 **转化** 为较简单的 **子问题** ，根据子问题的解得到原问题的解。

定义问题状态

定义问题的状态，以及目标值。

考虑状态转移

原问题的目标值（最大、最小、计数），一定是由子问题的目标值转移而来。

解决子问题

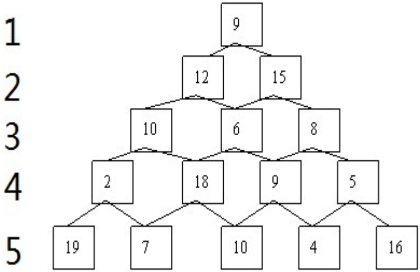
子问题的目标值求解仅局限于子问题的规模，与由哪个母状态转移而来无关。

目录

1. 相关概念
2. **数塔**
3. 象棋马
4. 背包问题
5. 线性 dp 问题

数塔

数塔问题是动态规划中的一个经典模型。如图是一个数塔，从塔的顶端开始，每次只能往左或往右向下走，找到一条路径使得路径上的数字和最大。



数塔

- 对于此问题，贪心一定是不适用的，无法找到最大和。

数塔

- 对于此问题，贪心一定是不适用的，无法找到最大和。
- 考虑枚举所有的路线和，显然是指数级别的时间复杂度，当层数稍大就已无法解决。

数塔

- 对于此问题，贪心一定是不适用的，无法找到最大和。
- 考虑枚举所有的路线和，显然是指数级别的时间复杂度，当层数稍大就已无法解决。
- 那么考虑获取点数的过程，因为是要路径和最大，因此从上往下和从下往上情况是一样的。

数塔

- 对于此问题，贪心一定是不适用的，无法找到最大和。
- 考虑枚举所有的路线和，显然是指数级别的时间复杂度，当层数稍大就已无法解决。
- 那么考虑获取点数的过程，因为是要路径和最大，因此从上往下和从下往上情况是一样的。
- 考虑从下往上，每一层只能从下一层的俩个块中选取大权值的加入自身权值。

数塔

- 对于此问题，贪心一定是不适用的，无法找到最大和。
- 考虑枚举所有的路线和，显然是指数级别的时间复杂度，当层数稍大就已无法解决。
- 那么考虑获取点数的过程，因为是要路径和最大，因此从上往下和从下往上情况是一样的。
- 考虑从下往上，每一层只能从下一层的俩个块中选取大权值的加入自身权值。
- 那么像这样考虑完一层之后，树塔的阶数就减少了，也维护了权值最大的情况。

数塔

- 对于此问题，贪心一定是不适用的，无法找到最大和。
- 考虑枚举所有的路线和，显然是指数级别的时间复杂度，当层数稍大就已无法解决。
- 那么考虑获取点数的过程，因为是要路径和最大，因此从上往下和从下往上情况是一样的。
- 考虑从下往上，每一层只能从下一层的俩个块中选取大权值的加入自身权值。
- 那么像这样考虑完一层之后，树塔的阶数就减少了，也维护了权值最大的情况。
- 这样一来一层一层转移，由此就能传递出最后的答案。

数塔

用一个二维数组 a 来存储数塔的原始数据，然后用一个 dp 数组存储过程中最大路径和（包括自身数值）。

9				
12	15			
10	6	8		
2	18	9	5	
19	7	10	4	16

Figure: 数组 a 的数据如上，设 a[1][1] 作为起点数据下标

数塔

- 初始化数组 dp 拷贝数组 a 数据，从倒数第二层开始动态规划，我们有 $dp[i][j] = max(dp[i+1][j], dp[i+1][j+1]) + dp[i][j]$。同时数组 path 存储当前可选的最优路径（左/右）。

```
for 倒数第二行到首行 i do
    for 每一行的所有元素块 j do
        if dp[i+1][j]>dp[i+1][j+1] then
            path[i][j]=j,dp[i][j]+=dp[i][j]+dp[i+1][j]
        else
            path[i][j]=j+1,dp[i][j]+=dp[i][j]+dp[i+1][j+1]
```

数塔

- 最后数组 dp 存储情况如下

59				
50	49			
38	34	29		
21	28	19	21	
19	7	10	4	16

数塔

- 时间复杂度为 $O(n^2)$ 的级别。

数塔

- 时间复杂度为 $O(n^2)$ 的级别。
- 如果是要求解有多少条路径满足最大路径和，这时应该如何转移状态？

目录

1 相关概念

2 数塔

3 象棋马

4 背包问题

5 线性 dp 问题

象棋马

问题描述

有一只中国象棋中的“马”，在半张棋盘的左上角出发，向右下角跳去。规定只许向右跳（可上，可下，但不允许向左跳）。求从起点 A(1,1) 到终点 B(m,n) 共有多少种不同跳法。

象棋马

问题描述

有一只中国象棋中的“马”，在半张棋盘的左上角出发，向右下角跳去。规定只许向右跳（可上，可下，但不允许向左跳）。求从起点 A(1,1) 到终点 B(m,n) 共有多少种不同跳法。

输入格式

输入文件只有一行，两个整数 m 和 n （1 m,n 20），两个数之间有一个空格。

象棋马

问题描述

有一只中国象棋中的“马”，在半张棋盘的左上角出发，向右下角跳去。规定只许向右跳（可上，可下，但不允许向左跳）。求从起点 A(1,1) 到终点 B(m,n) 共有多少种不同跳法。

输入格式

输入文件只有一行，两个整数 m 和 n （1 m,n 20），两个数之间有一个空格。

输出格式

输出文件只有一个整数，即从 A 到 B 全部的走法。

象棋马

- 首先考虑限制条件，棋子只能向由跳，且不能跳出棋盘边界。

象棋马

- 首先考虑限制条件，棋子只能向由跳，且不能跳出棋盘边界。
- 那么靠右的位置只有可能从左侧的位置转移。

象棋马

- 首先考虑限制条件，棋子只能向由跳，且不能跳出棋盘边界。
- 那么靠右的位置只有可能从左侧的位置转移。
- 按从左到右的次序（按列优先）将状态转移出去。

象棋马

- 首先考虑限制条件，棋子只能向由跳，且不能跳出棋盘边界。
- 那么靠右的位置只有可能从左侧的位置转移。
- 按从左到右的次序（按列优先）将状态转移出去。
- 即可统计出所有的方案数。

象棋马

- 令 dp[1][1]=1，其余为 0，从第二列开始动态规划，我们有 $dp[i][j] = dp[i-1][j-2] + dp[i-2][j-1] + dp[i-1][j+2] + dp[i-2][j+1]$。

```
int dp[1010][1010];
dp[1][1]=1;
for 遍历列号 i 从左到右 do
    for 遍历行号 j 从上到下 do
        dp[i][j]=dp[i-1][j-2]+dp[i-2][j-1]+dp[i-1][j+2]+dp[i-2][j+1]
输出 dp[n][m]
```

象棋马

回溯法

- 时间复杂度为 $O(nm)$，另外此题还可以用回溯法解决。

```
function F(int i,int j)//返回值类型为 int
    if i<1||j<1||i>m||j>n then return 0;
    if i==1j==1 then return 1;
    if dp[i][j]>0 then return dp[i][j];//剪枝
    dp[i][j]=f(i-2,j-1)+f(i-2,j+1)+f(i-1,j-2)+f(i-1,j+2);
    return dp[i][j];
输出 f(n,m)
```

- 如果棋盘中添加了一些固定的点，考虑蹩马脚的情况。

- 如果棋盘中添加了一些固定的点，考虑蹩马脚的情况。
- 设定几个固定点作为马从 A 到 B 路径的必经点，又该如何计算。

- 如果棋盘中添加了一些固定的点，考虑蹩马脚的情况。
- 设定几个固定点作为马从 A 到 B 路径的必经点，又该如何计算。
- 求走到终点的最小步数。

象棋马

一些拓展

- 如果棋盘中添加了一些固定的点，考虑蹩马脚的情况。
- 设定几个固定点作为马从 A 到 B 路径的必经点，又该如何计算。
- 求走到终点的最小步数。
- 有无可能经过棋盘上所有的点，无需最后到达终点。

目录

1 相关概念

2 数塔

3 象棋马

4 背包问题

5 线性 dp 问题

背包问题

- 背包问题是一种组合优化的 NP 完全问题。

背包问题

- 背包问题是一种组合优化的 NP 完全问题。
- 问题可以描述为：给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。

背包问题

- 背包问题是一种组合优化的 NP 完全问题。
- 问题可以描述为：给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。
- 背包问题也是非常经典的动态规划问题，利用动态规划，我们可以在伪多项式时间复杂度求解。

背包问题

- 背包问题是一种组合优化的 NP 完全问题。
- 问题可以描述为：给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。
- 背包问题也是非常经典的动态规划问题，利用动态规划，我们可以在伪多项式时间复杂度求解。
- 这里介绍三类比较基础的背包问题。

- 最基本的背包问题就是 01 背包问题。

- 最基本的背包问题就是 01 背包问题。

问题描述

一共有 N 件物品，第 i（i 从 1 开始）件物品的重量为 w[i]，价值为 v[i]。在总重量不超过背包承载上限 W 的情况下，能够装入背包的最大价值是多少？

- 如果使用暴力枚举的方法，每个物品有取和不取两种状态，总的时间复杂度为 $O(2^n)$。

- 如果使用暴力枚举的方法，每个物品有取和不取两种状态，总的时间复杂度为 $O(2^n)$。
- 显然这是不可接受的，我们可以利用动态规划将时间复杂度降至 $O(NW)$。

- 如果使用暴力枚举的方法，每个物品有取和不取两种状态，总的时间复杂度为 $O(2^n)$。
- 显然这是不可接受的，我们可以利用动态规划将时间复杂度降至 $O(NW)$。
- 发现变量只有物品总重量和总价值。

- 如果使用暴力枚举的方法，每个物品有取和不取两种状态，总的时间复杂度为 $O(2^n)$。
- 显然这是不可接受的，我们可以利用动态规划将时间复杂度降至 $O(NW)$。
- 发现变量只有物品总重量和总价值。
- 考虑以上两变量之间的状态联系进行转移。

背包问题

01 背包

- 定义 dp[i][j] 表示将前 i 件物品装进限重为 j 的背包可获得的最大价值，其中 $dp[i][j]=0$。开始对每件物品进行遍历，有如下两种转移方法。

```
int dp[N+5][W+5];
for 遍历每件物品 do
    for W 到 w[i] 枚举限重值 do//注意必须逆向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+v[i]);
输出最大价值 dp[N][W]
```

背包问题

01 背包

- 定义 dp[i][j] 表示将前 i 件物品装进限重为 j 的背包可获得的最大价值，其中 $dp[i][j] = 0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入第 i 件物品，$dp[i][j] = dp[i-1][j]$。
- 装入第 i 件物品，$dp[i][j] = dp[i-1][j-w[i]] + v[i]$。

```
int dp[N+5][W+5];
for 遍历每件物品 do
    for W 到 w[i] 枚举限重值 do//注意必须逆向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+v[i]);
输出最大价值 dp[N][W]
```

- 定义 dp[i][j] 表示将前 i 件物品装进限重为 j 的背包可获得的最大价值，其中 $dp[i][j] = 0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入第 i 件物品，$dp[i][j] = dp[i-1][j]$。
- 装入第 i 件物品，$dp[i][j] = dp[i-1][j-w[i]] + v[i]$。
- 转移状态方程：$dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i]] + v[i])$。

```
int dp[N+5][W+5];
for 遍历每件物品 do
    for W 到 w[i] 枚举限重值 do//注意必须逆向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+v[i]);
输出最大价值 dp[N][W]
```

背包问题

01 背包

- 定义 dp[i][j] 表示将前 i 件物品装进限重为 j 的背包可获得的最大价值，其中 $dp[i][j] = 0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入第 i 件物品，$dp[i][j] = dp[i-1][j]$。
- 装入第 i 件物品，$dp[i][j] = dp[i-1][j-w[i]] + v[i]$。
- 转移状态方程：$dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i]] + v[i])$。

```
int dp[N+5][W+5];
for 遍历每件物品 do
    for W 到 w[i] 枚举限重值 do//注意必须逆向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+v[i]);
输出最大价值 dp[N][W]
```

- 这玩意其实能写成一维的，dp[j]=max(dp[j],dp[j-w[i]]+v[i])。

- 完全背包也叫无穷背包，与 01 背包类似，唯一的区别是每种物品可取的数量不限。

- 完全背包也叫无穷背包，与 01 背包类似，唯一的区别是每种物品可取的数量不限。

问题描述

在 N 种物品中选取若干件（同一种物品可多次选取）放在空间为 V 的背包里，每种物品的体积为 c1，c2，⋯，cn，与之相对应的价值为 w1,w2，⋯，wn. 求解怎么装物品可使背包里物品总价值最大。

背包问题
完全背包

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j]=0$。开始对每件物品进行遍历，有如下两种转移方法。

```
int dp[N+5][V+5];
for 遍历每件物品 do
    for W[i] 到 V 枚举限重值 do//注意必须正向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-c[i]]+w[i]);
输出最大价值 dp[N][V]
```

背包问题
完全背包

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j]=0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入第 i 件物品，$dp[i][j]=dp[i-1][j]$。
- 装入第 i 件物品，$dp[i][j]=dp[i-1][j-c[i]]+w[i]$。

```
int dp[N+5][V+5];
for 遍历每件物品 do
    for W[i] 到 V 枚举限重值 do//注意必须正向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-c[i]]+w[i]);
输出最大价值 dp[N][V]
```

背包问题

完全背包

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j]=0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入第 i 件物品，$dp[i][j]=dp[i-1][j]$。
- 装入第 i 件物品，$dp[i][j]=dp[i-1][j-c[i]]+w[i]$。
- 转移状态方程：$dp[i][j]=max(dp[i-1][j],dp[i-1][j-c[i]]+w[i])$。

```
int dp[N+5][V+5];
for 遍历每件物品 do
    for W[i] 到 V 枚举限重值 do//注意必须正向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-c[i]]+w[i]);
输出最大价值 dp[N][V]
```

背包问题
完全背包

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j]=0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入第 i 件物品，$dp[i][j]=dp[i-1][j]$。
- 装入第 i 件物品，$dp[i][j]=dp[i-1][j-c[i]]+w[i]$。
- 转移状态方程：$dp[i][j]=max(dp[i-1][j],dp[i-1][j-c[i]]+w[i])$。

```
int dp[N+5][V+5];
for 遍历每件物品 do
    for W[i] 到 V 枚举限重值 do//注意必须正向枚举
        dp[i][j]=max(dp[i-1][j],dp[i-1][j-c[i]]+w[i]);
输出最大价值 dp[N][V]
```

- 同样可以只开一维的空间，dp[j]=max(dp[j],dp[j-c[i]]+w[i])。

- 多重背包和完全背包类似，转移方程略加修改即可。

- 多重背包和完全背包类似，转移方程略加修改即可。

问题描述

有 N 种物品和一个容量为 V 的背包。第 i 种物品最多有 n[i] 件可用，每件物品容量是 c[i]，价值是 w[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

背包问题
完全背包

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j] = 0$。开始对每件物品进行遍历，有如下两种转移方法。

```
int dp[N+5][V+5];
for 遍历每件物品 i do
    for 枚举背包容量 j do//从小到大
        for 枚举选取物品数量 k do 从小到大
            if j<k*c[i] then dp[i][j]=dp[i-1][j];
            elsedp[i][j]=max(dp[i-1][j],dp[i-1][j-k*c[i]]+k*w[i]);
输出最大价值 dp[N][V]
```

背包问题
完全背包

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j] = 0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入 k 件第 i 件物品，$dp[i][j] = dp[i-1][j]$。
- 装入 k 件第 i 件物品，$dp[i][j] = dp[i-1][j-k*c[i]] + k*w[i]$。

```
int dp[N+5][V+5];
for 遍历每件物品 i do
    for 枚举背包容量 j do//从小到大
        for 枚举选取物品数量 k do 从小到大
            if j<k*c[i] then dp[i][j]=dp[i-1][j];
            elsedp[i][j]=max(dp[i-1][j],dp[i-1][j-k*c[i]]+k*w[i]);
输出最大价值 dp[N][V]
```

背包问题

完全背包

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j]=0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入 k 件第 i 件物品，$dp[i][j]=dp[i-1][j]$。
- 装入 k 件第 i 件物品，$dp[i][j]=dp[i-1][j-k*c[i]]+k*w[i]$。
- 转移状态方程：
  $dp[i][j]=max(dp[i-1][j],dp[i-1][j-k*c[i]]+k*w[i])$。

```
int dp[N+5][V+5];
for 遍历每件物品 i do
    for 枚举背包容量 j do//从小到大
        for 枚举选取物品数量 k do 从小到大
            if j<k*c[i] then dp[i][j]=dp[i-1][j];
            elsedp[i][j]=max(dp[i-1][j],dp[i-1][j-k*c[i]]+k*w[i]);
输出最大价值 dp[N][V]
```

- 定义 dp[i][j] 表示将前 i 件物品装进空间限制为 j 的背包可获得的最大价值，其中 $dp[i][j]=0$。开始对每件物品进行遍历，有如下两种转移方法。
- 不装入 k 件第 i 件物品，$dp[i][j]=dp[i-1][j]$。
- 装入 k 件第 i 件物品，$dp[i][j]=dp[i-1][j-k*c[i]]+k*w[i]$。
- 转移状态方程：
  $dp[i][j]=max(dp[i-1][j],dp[i-1][j-k*c[i]]+k*w[i])$。

```
int dp[N+5][V+5];
for 遍历每件物品 i do
    for 枚举背包容量 j do//从小到大
        for 枚举选取物品数量 k do 从小到大
            if j<k*c[i] then dp[i][j]=dp[i-1][j];
            elsedp[i][j]=max(dp[i-1][j],dp[i-1][j-k*c[i]]+k*w[i]);
输出最大价值 dp[N][V]
```

- 同样可以只开一维的空间，dp[j]=max(dp[j],dp[j-k*c[i]]+k*w[i])。

- 前面几种背包类型如果要求必须装满如何求解。

- 前面几种背包类型如果要求必须装满如何求解。
- 如果某些物品不能同时取；或者取这个物品前必须保证取其他物品，该怎么处理。

- 前面几种背包类型如果要求必须装满如何求解。
- 如果某些物品不能同时取；或者取这个物品前必须保证取其他物品，该怎么处理。
- 杂七杂八的背包类型：二维费用背包问题、分组背包问题等等。

目录

1 相关概念

2 数塔

3 象棋马

4 背包问题

5 线性 dp 问题

线性 dp 问题

- 线性动态规划，是较常见的一类动态规划问题，其是在线性结构上进行状态转移，这类问题不像背包问题、区间 DP 等有固定的模板。

线性 dp 问题

- 线性动态规划，是较常见的一类动态规划问题，其是在线性结构上进行状态转移，这类问题不像背包问题、区间 DP 等有固定的模板。
- 以上几类问题有些是线性 dp 问题，但其实很多时候线性没那么容易体现出来。

线性 dp 问题

- 线性动态规划，是较常见的一类动态规划问题，其是在线性结构上进行状态转移，这类问题不像背包问题、区间 DP 等有固定的模板。
- 以上几类问题有些是线性 dp 问题，但其实很多时候线性没那么容易体现出来。
- 线性动态规划的目标函数为特定变量的线性函数，约束是这些变量的线性不等式或等式，目的是求目标函数的最大值或最小值。

线性 dp 问题

- 线性动态规划，是较常见的一类动态规划问题，其是在线性结构上进行状态转移，这类问题不像背包问题、区间 DP 等有固定的模板。
- 以上几类问题有些是线性 dp 问题，但其实很多时候线性没那么容易体现出来。
- 线性动态规划的目标函数为特定变量的线性函数，约束是这些变量的线性不等式或等式，目的是求目标函数的最大值或最小值。
- 大部分题目需要根据实际问题推导转移过程得到答案。

- 来看几个典型题目。

给定一个长度为 N 的数列，求数值严格单调递增的子序列的长度最长是多少。

输入格式

第一行包含整数 N。 第二行包含 N 个整数，表示完整序列。

输出格式

输出一个整数，表示最大长度。

数据范围

$1 \le N \le 1000$，

$-10^9 \le$ 数列中的数 $\le 10^9$

输入样例:

7

3 1 2 1 8 5 6

输出样例:

4

线性 dp 问题

- 来看几个典型题目。

给定两个长度分别为 N 和 M 的字符串 A 和 B，求既是 A 的子序列又是 B 的子序列的字符串长度最长是多少。

输入格式

第一行包含两个整数 N 和 M。

第二行包含一个长度为 N 的字符串，表示字符串 A。

第三行包含一个长度为 M 的字符串，表示字符串 B。

字符串均由小写字母构成。

输出格式

输出一个整数，表示最大长度。

数据范围

$1 \le N, M \le 1000$

输入样例:

4 5

acbd

abedc

输出样例:

3

- 来看几个典型题目。

一个长度为N的数字串,要求用K个乘号将它分成 $K+1$个部分，求这$K+1$个数的乘积最大值。

例：312，N=3,K=1

两种分法：

1）3*12=36

2）31*2=62

所以得到答案62

输入样例

```
4 2
1231
```

输出样例

```
62
```

动态规划

- 动态规划是算法竞赛中常见的内容。
- 动态规划一般都会结合许多算法出现，所以有时（离谱题目）状态转移方程非常难写。
- 动态规划的相关题目难度差异很大，所以建议碰到这类题目先尝试建立状态模型和相关转移（前提得看出来这是动态规划），还是非常灵活的一个知识点。

END