1306：最长公共子上升序列

题目描述

给定两个整数序列，写一个程序求它们的最长上升公共子序列。

当以下条件满足的时候，我们将长度N的序列S1,S2,...,SN 称为长度为M的序列A1,A2,...,AM的上升子序列：

存在1≤i1<i2<...<iN≤M，使得对所有1≤j≤N，均有Sj=Aij，且对于所有的1≤j<N，均有Sj<Sj+1。

输入

每个序列用两行表示，第一行是长度M(1≤M≤500)，第二行是该序列的M个整数Ai(-231<=Ai<231)

输出

在第一行，输出两个序列的最长上升公共子序列的长度L。在第二行，输出该子序列。如果有不止一个符合条件的子序列，则输出任何一个即可。

输入样例

```
5
1 4 2 5 -12
4
-12 1 2 4
```

输出样例

```
2
1 4
```

解析

本题的核心问题有两个，分别是：

最长公共上升子序列(LCIS)

LCIS的输出

我们一一来进行讲解。

LCIS的四重循环解法

这种解法的思路非常的简单，易于理解，但是性能也最差，无法在较短的时间内通过。不过，我们很有必要通过这个解法，来彻底的理解LCIS的概念。

给出两个字符串，假设分别为"a b c d e"和"a b e d a"，我们将这个数据改成下面的表格，以便于观察。

		0	1	2	3	4	5
		""	1	2	3	4	5
0	""						
1	1						
2	2						
3	5						
4	4						
5	1						

和LCS一样，空字符与任意字符串的LCIS长度均为0，我们进行填入。

		0	1	2	3	4	5
		""	1	2	3	4	5
0	""	0	0	0	0	0	0
1	1	0					
2	2	0					
3	5	0					
4	4	0					
5	1	0					

下面开始正式搜索LCIS。

我们用dp[i][j]表示在a字符串的前[i]字符，b字符串的前[j]字符中，取出的LCIS的值为多少

很容易想得到，当两个字符串不相等的时候，它们一定不在LCIS中，即dp[i][j]=0;

当两个字符串相同的时候，它们一定在LCIS中，但是长度是多少呢？

此刻，我们只能分别沿着a字符串和b字符串向前搜索，直到找到分别小于a[i]和b[j]的字符串，然后用这个数值+1，就是当前位置的LCIS值了。

填表如下：

		0	1	2	3	4	5
		""	1	2	3	4	5
0	""	0	0	0	0	0	0
1	1	0	1	0	0	0	0
2	2	0	0	2	0	0	0
3	5	0	0	0	0	0	3
4	4	0	0	0	0	3	0
5	1	0	1	0	0	0	0

```
//依次遍历两个字符串
for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
        //当两个字符串相等的时候，需要从前面找到一个比当前b[j]小的数据
       if (a[i] != b[j]) {
            dp[i][j] = 0;
        } else {
            //分别向前寻找两个字符
          for (int k = 1; k < i; ++k) {
                for (int l = 1; l < j; ++l) {
                    //前面出现的数字均小于当前的两个数字
                 if (a[k] < a[i] && b[l] < b[j]) {
                        //更新长度
                    dp[i][j] = max(dp[i][j], dp[k][l]) + 1;
                    }
                }
            }
        }
    }
}
```

四重循环解法的效率非常的低，那么可不可以优化呢？我们继续往下看。

LCIS的三重循环解法

先来观察上面的循环查找

```
for (int k = 1; k < i; ++k) {
    for (int l = 1; l < j; ++l) {
        //前面出现的数字均小于当前的两个数字
       if (a[k] < a[i] && b[l] < b[j]) {
            //更新长度
          dp[i][j] = max(dp[i][j], dp[k][l]) + 1;
       }
    }
}
```

在这里，我们仔细思考一下，其实，我们并不真的需要分别从a,b串向前寻找，事实上，我们只要能找到一个a[k]<a[i]，或者b[l]<b[j]即可。

原因如下：

1、因为a[i] == b[j]

2、又因为dp[k][l]代表的是在a串中前k，b串中前l的公共上升子序列，也就是说a[k] == b[l]

3、因此，原本的a[k]<a[i]&&b[l]<b[j]完全等同于a[k]<a[i]||b[l]<b[j]，二者满足其一即可，因为它们是一模一样的！

那么，我们在这里就可以把原本的四重循环进行优化。

首先，我们需要变更dp[i][j]数组的含义，新的数组为表示在a字符串的前[i]字符，b字符串的前[j]字符中，且以b[j]为结尾时，取出的LCIS的最大值为多少。

此处，使用了一个“前缀和”类似的思想。如果还不理解，情参考下面的表格。

原本定义

		0	1	2	3	4	5
		""	1	2	3	4	5
0	""	0	0	0	0	0	0
1	1	0	1	0	0	0	0
2	2	0	0	2	0	0	0

新定义

		0	1	2	3	4	5
		""	1	2	3	4	5
0	""	0	0	0	0	0	0
1	1	0	1	1	1	1	1
2	2	0	0	2	2	2	2

dp[i][j]数组存储了以b[j]作为截止字符串时，前a[i]个字符串中最大的LCIS。

因此当a[i] != b[j]时，dp[i][j]就等于dp[i-1][j]。原理是虽然a[i]不等于b[j]，但是a[i]之前的数字可能等于b[j]啊。这样，我们就必然能够找到前一个LCIS的长度。

当a[i] == b[j]时，我们依然需要向前去寻找前一个LCIS的值，很明显，此刻的a[i]应该减1，因为其存储的是前缀信息。那么j该如何变化，减1吗？

减1肯定是不正确的，因为我们b[j-1]并不能保证其一定小于b[j]，因此，我们只能枚举整个b字符串，直到找到小于b[j]且最大的那一个。

代码如下：

```
//遍历a字符串
for (int i = 1; i <= n; ++i) {
    //遍历b字符串
   for (int j = 1; j <= m; ++j) {
        //两个字符串相等的时候开始向前查找
      if (a[i] == b[j]) {
          //当前的最大长度
        int maxn = 0;
          //再次遍历全部的b字符串，直到找到比当前小的字符
        for (int k = 1; k <= j - 1; ++k)
            if (dp[i - 1][k] > maxn && b[k] < b[j]) {
                maxn = dp[i - 1][k];
            }
          dp[i][j] = maxn + 1;
       }
        //不相等的时候，直接取前缀信息
      else {
          dp[i][j] = dp[i - 1][j];
       }
    }
}
```

到此，我们就对本题进行了一次优化。当然，还存在更多的优化方法，不过，我们不在这里展开了，有兴趣的同学可以自行搜索。

路径打印

现在，我们还剩下另外一个问题，如何打印路径？

其实方法很简单，开一个pre数组记录答案。

每当状态变化时，我们均已当前的b串索引作为键值，存储前一个字符的索引。这样，我们就能从最后一个字符找到整个的路径链。

例如： a串 1 2 3 4 5

b串 1 2 5 4 1

最大长度在b[4]处产生，那么我们就存储pre[4] = 2，因为前一个LCIS是在b[2]处产生，那么pre[2]又会指向1，这样，我们就能够打印整个路径链。

编码

```
#include <bits/stdc++.h>

using namespace std;
int n, m, a[501], b[501], dp[501][501], pre[501];

void print(int x) {
```

```
    if (x == 0) {
        return;
    }
    //继续向前查找
    print(pre[x]);
    //打印当前的数据
    printf("%dp ", b[x]);
}

int main() {
    //获取两个字符串的输入
    scanf("%dp", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%dp", a + i);
    }
    scanf("%dp", &m);
    for (int i = 1; i <= m; ++i) {
        scanf("%dp", b + i);
    }
    //开始遍历a,b串
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            //当两个字符相等时，进行转移，向前搜索前一个LCIS
            if (a[i] == b[j]) {
                //以b[j]为结尾的最大LCIS长度
                int maxn = 0;
                //出现最大长度时的前序索引
                int ind = 0;
                //从头开始遍历b串，直到找到满足条件的前序数值
                for (int k = 1; k <= j - 1; ++k) {
                    if (dp[i - 1][k] > maxn && b[k] < b[j]) {
                        //记录以以b[j]为结尾的最大LCIS长度
                        maxn = dp[i - 1][k];
                        //记录前序索引
                        ind = k;
                    }
                }
                //更新最大长度值
                dp[i][j] = maxn + 1;
                //记录路径
                pre[j] = ind;
            }
                //字符串不相等时，向前转移，读取前缀数据
            else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }
```

```
    //全局最大值
    int ans = 0;
    //出现最大值时的索引
    int ind = 0;
    //遍历找到最大值
    for (int j = 1; j <= m; ++j) {
        if (ans < dp[n][j]) {
            ans = dp[n][j];
            ind = j;
        }
    }
    //输出最大值
    cout << ans << endl;
    //以链表的方式打印前序路径
    print(ind);
    cout << endl;
    return 0;
}
```