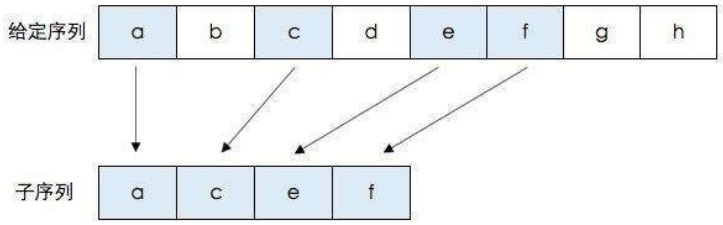
# 二维动规 2
## ——最长公共子序列

什么是子序列呢？即一个给定的序列的子序列，就是将给定序列中零个或多个元素去掉之后得到的结果。




如果在两个母串中出现相同的子序列，我们称为公共子序列。最长公共子序列（Longest Common Subsequence,LCS），顾名思义，是指在所有的子序列中最长的那一个。

# 最长公共子序列

【题目描述】

一个字符串 A 的子串被定义成从 A 中顺次选出若干个字符构成的串。如 A=“cdaad",顺次选 1，3，5 个字符就构成子串"cad",现给定两个字符串，求它们的最长共公子串。

【输入】

第一行两个字符串用空格分开。

【输出】

最长子串的长度。

【样例输入】

abccd aecd

【样例输出】

3

【提示】

字符串长度在 2000 以内

【分析】

1、暴力解法：假设母串 X 的长度为 n，母串 Y 的长度为 m， m<n， 对于母串 Y，我们可以暴力找出 2 的 m 次方个子序列，然后依次在母串 X 中匹配，算法的时间复杂度会达到指数级 O(n*2 的 m 次)。显然，暴力求解不太适用于此类问题。

oiClass

2、递归求解：我们要求 Xn 和 Ym 的最长公共子序列，考虑 X[n]和 Y[m]，如果 X[n]==Y[m]，则问题递归为 Xn-1 和 Yn-1 求最长公共子序列；如果 X[n]!=Y[m]，那么 Xn 和 Ym-1 去求最长公共子序列，或者 Xn-1 和 Ym 求最长公共子序列，两者求最长。上述的递归求解的办法中，重复的子问题多，效率低下。改进的办法——用空间换时间，用数组保存中间状态，方便后面的计算。这就是动态规划（DP)的核心思想了。

3、动态规划求解：定义状态变量 f[i][j]表示母串 X 的前 i 个字符与母串 Y 的前 j 个字符的最长公共子序列的长度，那么状态转移方程为：

$$f[i][j]=\begin{cases} f[i-1][j-1]+1, & X[i]==Y[j] \\ \max(f[i][j-1],f[i-1][j]), & X[i]!=Y[j] \end{cases}$$

边界是 f[0][0]=0。

【代码】

```
#include<iostream>
#include<cstring>
using namespace std;
int f[2020][2020];
char s1[2020],s2[2020];
int main(){
    cin>>s1>>s2;
    int len1=strlen(s1);
    int len2=strlen(s2);  //特别注意，字符串从 0 位置开始存储
    for(int i=1;i<=len1;i++){
        for(int j=1;j<=len2;j++){
            if(s1[i-1]==s2[j-1])f[i][j]=f[i-1][j-1]+1;
            else f[i][j]=max(f[i-1][j],f[i][j-1]);
        }
    }
    cout<<f[len1][len2];
}
```

# 回文词

【题目描述】

回文词是一种对称的字符串——也就是说，一个回文词，从左到右读和从右到左读得到的结果是一样的。任意给定一个字符串，通过插入若干字符，都可以变成一个回文词。你的任务是写一个程序，求出将给定字符串变成回文词所需插入的最少字符数。

比如字符串“Ab3bd”，在插入两个字符后可以变成一个回文词（“dAb3bAd”或“Adb3bdA”）。然而，插入两个以下的字符无法使它变成一个回文词。

【输入】

第一行包含一个整数 N，表示给定字符串的长度，3<=N<=5000

第二行是一个长度为 N 的字符串，字符串由大小写字母和数字构成。

【输出】

一个整数，表示需要插入的最少字符数。

【样例输入】

5

Ab3bd

【样例输出】

2

【分析】

将原串和反串求最长公共子序列，将长度 n 减去最长公共子序列长度，得到的即为不同的字母，每一个字母都可以通过插入一个字母使得原串和反串相同，所以答案为长度 n 减去最长公共子序列长度。

【代码】

```
#include<iostream>
using namespace std;
char a[5001],b[5001];
int n,f[5001][5001];
int main(){
    cin>>n;
    cin>>a;
    for(int i=0;i<n;i++){
        b[i]=a[n-i-1];
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(a[i-1]==b[j-1]){
                f[i][j]=f[i-1][j-1]+1;
            }else{
                f[i][j]=max(f[i][j-1],f[i-1][j]);
            }
        }
    }
    cout<<n-f[n][n];
}
```

# 编辑距离

【题目描述】

设有字符串 X，我们称在 X 的头尾及中间插入任意多个空格后构成的新字符串为 X 的扩展串，如字符串 X 为“abcbcd”，则字符串“abcb□cd”，“□a□bcbcd□”和“abcb□cd□”都是 X 的扩展串，这里“□”代表空格字符。

如果 A1 是字符串 A 的扩展串，B1 是字符串 B 的扩展串，A1 与 B1 具有相同的长度，那么我们定义字符串 A1 与 B1 的距离为相应位置上的字符的距离总和，而两个非空格字符的距离定义为它们的 ASCII 码的差的绝对值，而空格字符与其它任意字符之间的距离为已知的定值 K，空格字符与空格字符的距离为 0。在字符串 A、B 的所有扩展串中，必定存在两个等长的扩展串 A1、B1，使得 A1 与 B1 之间的距离达到最小，我们将这一距离定义为字符串 A、B 的距离。

请你写一个程序，求出字符串 A、B 的距离。

【输入】

输入文件第一行为字符串 A，第二行为字符串 B，A、B 均由小写字母组成且长度均不超过 2000，第三行为一个整数 K，1≤K≤100，表示空格与其它字符的距离。

【输出】

输出文件仅一行包含一个整数，表示要求的字符串 A、B 的距离。

【样例输入】

cmc

snmn

2

【样例输出】

10

【分析】

定义状态变量 f[i][j]表示字符串 s1 前 i 个字符和字符串 s2 前 j 个字符扩展到相同长度所需的最小距离，那么 f[i][j]=min(f[i-1][j]+k,f[i][j-1]+k,f[i-1][j-1]+abs(s1[i-1]-s2[j-1]))，其中 f[i-1][j]+k 表示 s1 的第 i 个字符与空格匹配，问题转化为 f[i-1][j]；f[i][j-1]+k 表示 s2 的第 j 个字符与空格匹配，问题转化啊为 f[i][j-1]；f[i-1][j-1]+abs(s1[i-1]-s2[j-1])表示 s1 的第 i 个字符和 s2 的第 j 个字符匹配，距离为 abs(s1[i-1]-s2[j-1])，问题转化为 f[i-1][j-1]。边界：f[0][0]=0，f[i][0]=f[i-1][0]+k，f[0][j]=f[0][j-1]+k。

【代码】

```
#include<iostream>
#include<cmath>
#include<cstring>
using namespace std;
char s1[2001],s2[2001];
int f[2001][2001],k;
int min(int a,int b,int c){
    return min(a,min(b,c));
}
int main(){
    cin>>s1>>s2>>k;
    int len1=strlen(s1);
    int len2=strlen(s2);
    for(int i=1;i<=len1;i++)f[i][0]=f[i-1][0]+k;
    for(int j=1;j<=len2;j++)f[0][j]=f[0][j-1]+k;
    for(int i=1;i<=len1;i++){
        for(int j=1;j<=len2;j++){
            f[i][j]=min(f[i-1][j]+k,f[i][j-1]+k,f[i-1][j-1]+abs(s1[i-1]-
s2[j-1]));
        }
    }
    cout<<f[len1][len2];
}
```

# 相似基因

【题目描述】

大家都知道，基因可以看作一个碱基对序列。它包含了 4 种核苷酸，简记作 A,C,G,T。生物学家正致力于寻找人类基因的功能，以利用于诊断疾病和发明药物。

在一个人类基因工作组的任务中，生物学家研究的是：两个基因的相似程度。因为这个研究对疾病的治疗有着非同寻常的作用。两个基因的相似度的计算方法如下：

对于两个已知基因，例如 AGTGATG 和 GTTAG，将它们的碱基互相对应。当然，中间可以加入一些空碱基-，例如：

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | G | T | G | A | T | - | G |
| - | G | T | - | - | T | A | G |

这样,两个基因之间的相似度就可以用碱基之间相似度的总和来描述，碱基之间的相似度如下表所示：

| | A | C | G | T | - |
|---|---|---|---|---|---|
| A | 5 | -1 | -2 | -1 | -3 |
| C | -1 | 5 | -3 | -2 | -4 |
| G | -2 | -3 | 5 | -2 | -2 |
| T | -1 | -2 | -2 | 5 | -1 |
| - | -3 | -4 | -2 | -1 | * |

那么相似度就是：(-3)+5+5+(-2)+(-3)+5+(-3)+5=9。因为两个基因的对应方法不唯一，例如又有：

| | | | | | | |
|---|---|---|---|---|---|---|
| A | G | T | G | A | T | G |
| - | G | T | T | A | - | G |

相似度为：(-3)+5+5+(-2)+5+(-1)+5=14。规定两个基因的相似度为所有对应方法中，相似度最大的那个。

【输入】

共两行。每行首先是一个整数，表示基因的长度；隔一个空格后是一个基因序列，序列中只含 A,C,G,T 四个字母。1<=序列的长度<=100。

【输出】

仅一行，即输入基因的相似度。

【样例输入】

7 AGTGATG

5 GTTAG

【样例输出】

14

【分析】

oiClass

本题与上题类似，注意权值的求解即可。

【代码】

```
#include<iostream>
#include<cstring>
using namespace std;
const int w[5][5]={{5,-1,-2,-1,-3},    //设定匹配的权值
                   {-1,5,-3,-2,-4},
                   {-2,-3,5,-2,-2},
                   {-1,-2,-2,5,-1},
                   {-3,-4,-2,-1,0}};
char s1[100],s2[100];
int f[101][101],len1,len2;
int ch(char a){
    if(a=='A')return 0;
    if(a=='C')return 1;
    if(a=='G')return 2;
    if(a=='T')return 3;
}
int max(int a,int b,int c){
    return max(a,max(b,c));
}
int main(){
    cin>>len1>>s1>>len2>>s2;
    for(int i=1;i<=len1;i++){
        f[i][0]=f[i-1][0]+w[ch(s1[i-1])][4];
    }
    for(int j=1;j<=len2;j++){
        f[0][j]=f[0][j-1]+w[ch(s2[j-1])][4];
    }
    for(int i=1;i<=len1;i++){
        for(int j=1;j<=len2;j++){
            int a=ch(s1[i-1]);
            int b=ch(s2[j-1]);
            f[i][j]=max(f[i-1][j]+w[a][4],f[i][j-1]+w[b][4],f[i-1][j-
1]+w[a][b]);
        }
    }
    cout<<f[len1][len2];
}
```

# 佳佳的魔杖

【题目描述】

配制成功了珍贵的 0 号药水，MM 的病治好了。轻松下来的佳佳意外的得到了一个好东西……那就是——一种非常珍贵的树枝。这些树枝可以用来做优质的魔杖！当然了，不能只做自己的，至少还要考虑到 MM 的对吧。选择怎样的切割方式来制作魔杖非常重要，关键问题是——一把魔杖既不能太长、又不能太短，且制作出来的魔杖不能有冲突……

佳佳得到的这些树枝在属性上完全相同。每一个树枝都有 n 段（用 1~n 编号），给定了每段的长度 L[i]和每段的魔力值 M[i]。单独的一段是不可以从中间切开的，你可以做的就是选择一段或连续的几段，把它们作为一个整体切下来，再用来制作魔杖。但是一根魔杖的长度不能太长——不能大于给定的值 hi；也不能太短——不能小于给定的值 lo。

魔杖有一个奇怪的要求：如果某一根魔杖的制作材料是另一根魔杖的一部分，则这两根魔杖之间将发生冲突。比如说树枝有三段，从左到右的长度分别为 4、 1、3，佳佳需要长度为 4 到 5 之间的魔杖。佳佳可以用一根树枝的前两段做出一个长度为 5 的魔杖，用一根树枝的后两段做出长度为 4 的魔杖；但他决不能用一根树枝的前两段做了魔杖后再单独使用另一根树枝的第一段做成魔杖，因为前者包含了后者的所有成分，这会导致冲突。

我们假设佳佳可以得到任意多这样的树枝。佳佳需要制作出若干个互不冲突的魔杖，使所有魔杖的魔力值之和最大。（魔杖的长度就是组成它的那些段的长度的总和，魔力值亦然）。

【输入】

第一行有三个用空格隔开的正整数，分别表示 n、lo、hi。

第二行的 n 个用空格隔开的正整数就是 L[1]、L[2]……L[n]。

第三行的 n 个用空格隔开的正整数就是 M[1]、M[2]……M[n]。

输入文件以一个回车/换行符结尾。

【输出】

只用输出一个整数，表示能够获得的魔力值的最大值。

【样例输入】

6 4 5

1 3 3 2 2 1

2 3 1 4 5 2

【样例输出】

21

【提示】

取[1 3] [3 2] [2 2 1]做成魔杖。

得到最大权值 2+3+1+4+4+5+2=21。

对于 30%的数据，n<=10；

对于 50%的数据，n<=100；

对于 100%的数据，n<=1000，lo<=hi<=2^31-1，L[i],M[i]<=100 000

【分析】

由于选出的区间不能相互包含，因此，这些区间当中，不可能有两个区间具有相同的左端点或右端点，即如果设左右端点分别是 x1,y1,x2,y2，那么这两段能共存的条件，就是 x1<x2&&y1<y2。

这就可以设计状态了，令 f[i][j]为 1..i 中的某些数作为左端点，且他们对应的右端点的最大值是 j 时的最优解。那么，如果区间 i，j 可以做成魔杖，问题转化为 f[i-1][j-1]，如果不能做成魔杖，问题转化为 max(f[i][j-1],f[i-1][j])。状态转移方程为

oiClass

$$f[i][j]=\max\begin{cases}f[i-1][j-1]+\text{区间 i，j 的魔力值}, & \text{区间 i，j 的长度在 lo 和 hi 之间}\\ \max(f[i-1][j],f[i][j-1]), & \text{区间 i，j 的长度和超过 lo 和 hi}\end{cases}$$

【代码】

```
#include<iostream>
using namespace std;
long long n,lo,hi,x,sl[1001],sm[1001],f[1001][1001];
int main(){
    cin>>n>>lo>>hi;
    for(int i=1;i<=n;i++){
        cin>>x;
        sl[i]=sl[i-1]+x;
    }
    for(int i=1;i<=n;i++){
        cin>>x;
        sm[i]=sm[i-1]+x;
    }
    for(int i=1;i<=n;i++){
        for(int j=i;j<=n;j++){
            f[i][j]=max(f[i-1][j],f[i][j-1]);
            if(sl[j]-sl[i-1]>=lo&&sl[j]-sl[i-1]<=hi){
                f[i][j]=max(f[i][j],f[i-1][j-1]+sm[j]-sm[i-1]);
            }
        }
    }
    cout<<f[n][n];
}
```

# 乐曲主题

【题目描述】

我们用 N(1 <= N <=5000)个音符的序列来表示一首乐曲，每个音符都是 1..88 范围内的整数，每个数表示钢琴上的一个键。很不幸这种表示旋律的方法忽略了音符的时值，但这项编程任务是关于音高的，与时值无关。

许多作曲家围绕一个重复出现的“主题”来构建乐曲。在我们的乐曲表示法中，“主题”是整个音符序列的一个子串，它需要满足如下条件：

1.长度至少为 5 个音符

2.在乐曲中重复出现(可能经过转调，见下)

3.重复出现的同一主题不能有公共部分。

“转调”的意思是主题序列中每个音符都被加上或减去了同一个整数值。给定一段乐曲，计算其中最长主题的长度(即音符数)。

【输入】

输出文件的第一行包含整数 N。下面的每一行(最后一行可能除外)包含 20 个整数，表示音符序列。最后一行可能少于 20 个音符。

【输出】

输出文件应只含一个整数，即最长主题的长度。如果乐曲中没有主题，那么输出 0。

【样例输入】

30

25 27 30 34 39 45 52 60 69 79 69 60 52 45 39 34 30 26 22 18

82 78 74 70 66 67 64 60 65 80

【样例输出】

5

【提示】

这个长度为 5 的主题是输入文件中第一行的最后 5 个音符和第二行开头 5 个音符

【分析】

首先，所谓转调问题，我们可以通过差分（a[i]-a[i-1]）求的序列如下：

2 3 4 5 6 7 8 9 10 -10 -9 -8 -7 -6 -5 -4 -4 -4 -4 64 -4 -4 -4 -4 1 -3 -4 5 15

题目问题转变为在差分序列中求最长的重复子串，而且子串不能重叠。

定义状态变量 f[i][j]表示子串以 i 结尾的序列和子串以 j 结尾的序列的重复长度，那么：

$$f[i][j] = \max\begin{cases} f[i-1][j-1]+1, & s[i] == s[j] \\ 0, & s[i] != s[j] \end{cases}$$

为了让 f[i][j]保证不重复，那么 f[i][j]最大值为 j-i-1。

题目求最大的 f[i][j]。

本题为求最长重复子串的模板题，值得好好揣摩。

【代码】

```
#include<iostream>
using namespace std;
int n,a[5001],f[5001][5001],maxn;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<=n-5;i++){  //求最长不重叠重复子串
        for(int j=i+5;j<=n;j++){
            if(a[i]-a[i-1]==a[j]-a[j-1]){  //差分，解决转调问题
                f[i][j]=min(j-i-1,f[i-1][j-1]+1);  //j-i-1 是避免子串重叠
                maxn=max(maxn,f[i][j]);
            }
        }
    }
    maxn++;  //因为差分的影响，第一数没有记录，在此加上
    if(maxn<5)cout<<0;
    else cout<<maxn;
}
```