# 实验三

设序列 X=x1x2...xn和Y=y1y2...ym的最长公共子序列是Z=z1z2...zk,则：

若xn=ym,有xn=ym=zk,Zk-1是Xn-1和Ym-1的最长公共子序列。

若xn!=ym且xn!=zk,有Z是Xn-1和Y的最长公共子序列。

若xn!=ym且ym!=zk,有Z是X和Ym-1的最长公共子序列。

所以要找X和Y的最长公共子序列，先找Xn-1和Y的最长公共子序列与X和Ym-1的最长公共子序列。

设Xi和Yj的最长公共子序列长度为dp[i][j],按上述逻辑有动态方程：

dp[i][j]= 0 , i==0||j==0

dp[i-1][j-1] , i>0 && j>0 && X[i]==Y[j]

max(dp[i-1][j],dp[i][j-1]), i>0 && j>0 && X[i]!=Y[j]

## 代码

#include<bits/stdc++.h>

using namespace std;

const int maxn=105;//序列最长长度

char a[maxn],b[maxn];

int dp[maxn][maxn],f[maxn][maxn];

void dfs(int i,int j,string s){

if(i==0||j==0){

cout<<s<<"\n";

return ;

}

if(f[i][j]==1){

s=" "+s;//字符间用空格隔开

s=a[i]+s;//在最后输出需要倒序

dfs(i-1,j-1,s);

}

else if(f[i][j]==2){

dfs(i-1,j,s);//来自上方

}

else if(f[i][j]==3){

dfs(i,j-1,s);//来自左边

}

else{

//左上都有可能

dfs(i-1,j,s);

dfs(i,j-1,s);

}

}

int main(){

ios::sync\_with\_stdio(0);cin.tie(0),cout.tie(0);

int t = 1,Case=1;

for(cin>>t;Case<=t;Case++){

int n,m;

cin>>n>>m;

for(int i=1;i<=n;++i) cin>>a[i];

for(int i=1;i<=m;++i) cin>>b[i];

// 初始化dp

for(int i=0;i<=n;++i) dp[i][0]=0;

for(int j=0;j<=m;++j) dp[0][j]=0;

for(int i=1;i<=n;++i){

for(int j=1;j<=m;++j){

if(a[i]==b[j]){

//匹配上了

dp[i][j]=dp[i-1][j-1]+1;

f[i][j]=1;//来自左上

}

else if(dp[i-1][j]>dp[i][j-1]){

dp[i][j]=dp[i-1][j];

f[i][j]=2;//来自上方

}

else if(dp[i-1][j]<dp[i][j-1]){

dp[i][j]=dp[i][j-1];

f[i][j]=3;//来自左边

}

else {

dp[i][j]=dp[i][j-1];

f[i][j]=4;//左上都有可能

}

}

}

cout<<"Case "<<Case<<"\n";

cout<<"最长公共子序列的长度:"<<dp[n][m]<<"\n";//输出答案

cout<<"LCS(X,Y):\n";

dfs(n,m,"");

cout<<endl;

//c表

for(int i=0;i<=n;++i){

for(int j=0;j<=m;++j){

cout<<dp[i][j]<<" ";

}

cout<<"\n";

}

cout<<endl;

//b表

for(int i=0;i<=n;++i){

for(int j=0;j<=m;++j){

cout<<f[i][j]<<" ";

}

cout<<"\n";

}

cout<<endl;

}

return 0;

}

## 体会

样例中已经给出了方法，在b表中出现了4这个值，而书中只有1,2,3。很容易推断4表示来自两个方向都有可能。于是寻找所有最长子序列的递归程序就写出来了，因为有4这种转态可以分叉，所以可以搜索出所有的最长公共子序列。在调试过程中发现有个细节就是记录结果的字符串s必须倒序记录，因为程序是从尾部倒推递归下去的。