# Code

## def get\_lcs\_tables(x, y):

    """return the lenth of the longest subsequence of sequence x and y

    Args:

        x ([list]): sequence x

        y (list)): sequence y

    return: the lenth of the longest subsequence,the mark of the last element of the get\_lcs\_tables

    before the latest one

    """

    m = len(x)

    n = len(y)

    ''' calculate the index from 0

    carefully to use the list resolution expression to create two dimension list'''

    b = [["" for i in range(0, n)] for j in range(0, m)]

    c = [[0 for i in range(n+1)] for j in range(m+1)]

    # for i in range(m+1):

    ''' the i,j index variable is to primary for traverse the table c and b

    the i=1,...,m;j=1,...n

    '''

    for i in range(1, m+1):

        for j in range(1, n+1):

            ''' debug :index out of range '''

            # print('x[i]=',x[i],'i=',i,"y[j]=",y[j],"j=",j)

            ''' traverse the x,y sequence

            index\_x=0,...,m-1,so make a offset:-1'''

            if x[i-1] == y[j-1]:

                c[i][j] = c[i-1][j-1]+1

                b[i-1][j-1] = "diagonal"

            # the compare the length of the lcs currently

            # if the up element >= the left element:

            elif c[i-1][j] >= c[i][j-1]:

                c[i][j] = c[i-1][j]

                b[i-1][j-1] = "up"

            else:

                c[i][j] = c[i][j-1]

                b[i-1][j-1] = "left"

    return c[m][n], b

## def get\_lcs(c,x,y,i,j,lcs\_sequences\_set,cs=""):

    # ''' without the table c,the sequence may not be a longest one ! '''

    # ''' initailly invoke is print\_lcs\_single(c,x,len(c),len(c[0])) '''

    #index variable i,j is primaryly to visit the table c:

    """get all largest common subsequnces without repeating

    Args:

        c (list): two dimension list(as a table)

        x (str): sequence x

        y (str): sequence y

        i (int): size of the table's rows

        j (int): size of the table's column

        lcs\_sequences\_set (str): contain all lcs

        cs (str, optional): accumulate the common subsequence before recursively invoke the function. Defaults to ""(empty string).

    """

    while(i>0 and j>0):

        # for j in range(1, j+1):

        while(i>0 and j>0):

            #     ''' debug :index out of range '''

            # # print('x[i]=',x[i],'i=',i,"y[j]=",y[j],"j=",j)

            # # ''' traverse the x,y sequence index\_x=0,...,m-1,so make a offset:-1'''

            if x[i-1]==y[j-1]:

                # c[i][j] = c[i-1][j-1]+1

                # print(x[i-1])

                cs+=x[i-1]

                # print(x[i-1],end=" ")

                i-=1

                j-=1

            else:

                # i-=1

                ''' there will hit one case following: '''

                if c[i-1][j] > c[i][j-1]:

                    i-=1

                elif c[i-1][j]==c[i][j]:

                    # pass

                    get\_lcs(c,x,y,i,j-1,lcs\_sequences\_set,cs)

                    get\_lcs(c,x,y,i-1,j,lcs\_sequences\_set,cs)

                    # i=i-1

                    # j=j-1

                    # return cs[::-1]

                    return

                else:

                    j-=1

    # print(cs[::-1])

lcs\_sequences\_set.add(cs[::-1])

''' invoke version 1.5 '''

## def version1\_5():

    ltl=length\_table\_lcs(x, y)

    print("lcs=",ltl[len(x)][len(y)],sep="")

    lcs\_sequences\_set=set()

    get\_lcs(ltl,x,y,len(x),len(y),lcs\_sequences\_set)

    print(lcs\_sequences\_set)

# version1\_0()

## version1\_5()

# 输入数据:

''' test data x,y: '''

x = "abcbdab"

y = "bdcaba"

# 运行结果

lcs=4

{'bdab', 'bcba', 'bcab'}