CAS CS 330. Problem Set 8

Problem 2

Algorithm:

here x,y,z are three integers that represents the length of the three strings , string z is an interleaving of string x and string y.

List A[ ] is a x*y matrix that contains all the combinations for x and y to form z, and everything thing in list A is initialized to None

X is the string x, Y is the string y, Z is the string z

```
optimal (x,y,z):
    if x == 0 && y==0:
        if z == 0:
            return True
        else:
             return False
    else:
        if A is not None:
            return the element in A
        else:
            if X[x] == Z[z] and Y[y]==Z[z]:
                A[x][y] = optimal(x-1,y,z-1)
                    and optimal (x,y-1,z-1)
            elif X[x] == Z[z]:
                A[x,y] = optimal(x-1,y,z-1)
            elif Y[y] == Z[z]:
                A[x][y]= optimal(x,y-1,z-1)
            return A[x][y]
```

Analyze:

we know string Z is an interleaving of string X and string Y, otherwise, we will just return False. This algorithm starts at the end of the string Z and try to figure out whether the last element of string Z belongs to X or Y. If the last element of X and Y are different, we can easily tell where should the last element of Z go, because it has to one of the two, and if they are different, it's easy to distinguish. When the last element of X and Y are the same, we need to check both options and see from which way we get a True solution.

Because in this algorithm, we need to fill out a x by y matrix, and for every updating step, the things we do are in constant time, therefore, the running time for this algorithm is in Theta(xy), is x = y = n then the running time = Theta(n^2)

Problem 3

Algorithm:

assume F[ ] is an input which is a n by k matrix and contains all the factors for each company on each day, eg. F[1][1] = $f_{1,1}$

N is the number of days, K is the number of companies

A is a N by K matrix

optimal(n,k):

```
if n == 1:
    if k == 1:    #because we start at company 1 at day 1.
        return F[1][1]
    else:
        return F[1][k]/c
else:
    list = [0]*K  #create a list with length K
    for i  = 1: K and i doesn't equal to k:
        list[i] = (f[n][i] /c )* optimal(n-1,i)
    list[k] = F[n][k] * optimal(n-1,k)
    max = the max value in the list
    A[n,k] = max
    return A[n,k]
```

optimal_helper(n):

```
if n == 1:
    return the maximum in F[1]
for j = 1:K:
    A[n,j]   = F[n][j] * optimal(n-1,j)
```

```
Output = [ ]
Find_ Solution(n): #find the company in order and put them in to output list
        if n == 0:
                return 0
        else:
                company = 0
                max = -1
                for i in range(K):
                        if A[n][i] > max:
                                max = A[n][i]
                                company = i
                output.append(company)
                Find_Solution(n-1)
Return_ Solution()
        Find_Solution(n)
        Output.reverse()
        for element in Output:
                print(element /n)
```

Analyze:

In this algorithm i build a N by K matrix as memory, and the running time for this algorithm is the time we need to fill out this matrix. However, when i try to fill out the table, i use a for-loop to find the maximum of all the company- combinations. Therefore, the running time for this algorithm is Theta(k*k*n).