HOMEWORK 7

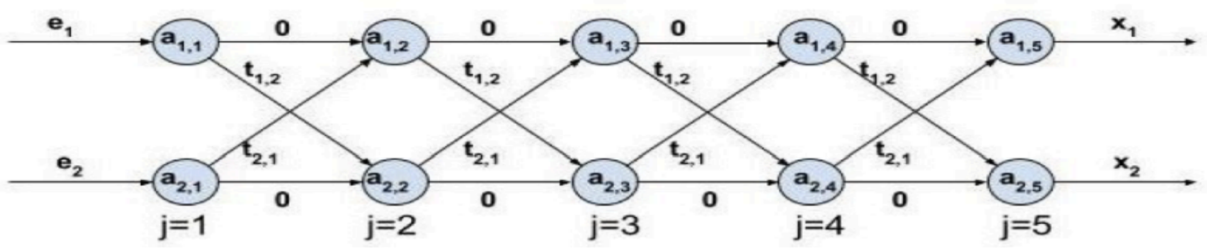
Q1. A car factory has k assembly lines, each with n stations. A station is denoted by $S_{i,j}$ where i indicates that the station is on the i-th assembly line (0<i<=k), and j indicates the station is the j-th station along the assembly line (0<j<=n). Each station is dedicated to some sort of work like engine fitting, body fitting, painting and so on. So, a car chassis must pass through each of the n stations in that order before exiting the factory. The time taken per station is denoted by $a_{i,j}$. Parallel stations of the k assembly lines perform the same task. After the car passes through station $S_{i,j}$, it will continue to station $S_{i,j+1}$ unless we decide to transfer it to another line. Continuing on the same line incurs no extra cost, but transferring from line x at station j-1 to line y at station j takes time $t_{x,y}$. Each assembly line takes an entry time e_i and exit time x_i which may be different for each of these k lines. Give an algorithm for computing the minimum time it will take to build a car chassis.

Below figure provides one example to explain the problem. The example shows k=2 assembly lines and 5 stations per line. To illustrate how to evaluate the cost, let's consider two cases: If we always use assembly line 1, the time cost will be:

$e_1 + a_{1,1} + a_{1,2} + a_{1,3} + a_{1,4} + a_{1,5} + x_1$

If we use stations $s_{1,1} \rightarrow s_{2,2} \rightarrow s_{1,3} \rightarrow s_{2,4} \rightarrow s_{2,5}$, the time cost will be :

$e_1 + a_{1,1} + t_{1,2} + a_{2,2} + t_{2,1} + a_{1,3} + t_{1,2} + a_{2,4} + a_{2,5} + x_2$



a. Define (in plain English) subproblems to be solved. (2 pts)

Ans:

The subproblem is to find the minimum time to reach that station by considering all possible paths and transfer options up to that point.

b. Recursively define the value of an optimal solution (recurrence formula) (8 pts)

Ans:

OPT(i, j) = min{OPT(i, j-1) + a(i , j), OPT(i, j-1) + a(i , j) + t(l,i)}

Where l iterates over all possible lies, a(i, j) is the time taken at station j on assembly line, and t(l, i) is the transfer time from the assembly line l to i.

c. Describe (using pseudocode) how the value of an optimal solution is obtained using iteration. Make sure you include any initialization required. (8 pts)

Ans:

```
def OPTFunc(k, n, e, x, a, t)
        opt = [[0] * n for _ in range(k)]
        for i in range(k):
                opt[i][0] = e[i] + a[i][0]
        for j in range(1, n):
                for i in range(k):
                        opt[i][j] = min(opt[i][j - 1] + a[i][j], min(opt[l][j - 1] + t[l][i] + a[i][j] for l in range(k)))
    result = min(x[i] + opt[i][n - 1] for i in range(k))
    return result
```

d. What is the complexity of your solution in part b? (4 pts)

Ans:

Since there are 2 for loops (one iterates k times while other iterates n times), the time complexity of this pseudocode is O(k.n).

Q2. There are n trading posts along a river numbered n, n – 1… 3, 2, 1. At any of the posts you can rent a canoe to be returned at any other post downstream. (It is impossible to paddle against the river since the water is moving too quickly). For each possible departure point i and each possible arrival point j(< i), the cost of a rental from i to j is known. It is C[i, j]. However, it can happen that the cost of renting from i to j is higher than the total costs of a series of shorter rentals. In this case you can return the first canoe at some post k between i and j and continue your journey in a second (and, maybe, third, fourth...) canoe. There is no extra charge for changing canoes in this way. Give a dynamic programming algorithm to determine the minimum cost of a trip by canoe from each possible departure point i to each possible arrival point j. For your dynamic programming solution, focus on computing the minimum cost of a trip from trading post n to trading post 1, using up to each intermediate trading post.

a. Define (in plain English) subproblems to be solved.

Ans:

Let OPT(i, j) denote the minimum cost of a canoe trip from trading post i to trading post j, considering all possible intermediate stops between I and j. We want to calculate OPT(n-1), representing the minimum cost of a trip from trading post n to trading post 1.

b. Recursively define the value of an optimal solution (recurrence formula) (5 pts)

Ans:

$OPT(i, j) = \min^{j-1}_{k = i+1} \{OPT(i,k) + OPT(k,j) + C[i,j]\}$

Where C[i, j] is the cost of renting a canoe from i to j, and the minimum is taken over all possible values of k between i and j.

c. What will be the iterative program for the above? (5 points)

Ans:

```
for i in range(n - 1, 0, -1):
        for j in range(i - 1, 0, -1):
                for k in range(i - 1, j, -1):
                        OPT[i][j] = min(OPT[i][j], OPT[i][k] + OPT[k][j] + C[i][j])
```

d. Analyze the running time of your algorithm in terms of n. (5 pts)

Ans:

Since there are three for loops iterating n times each, the time complexity is O(n^3).

Q3 Alice and Bob are playing an interesting game. They both each have a string, let them be a and b. They both decided to find the biggest string that is common between the strings they have. The letters of the resulting string should be in order as that in a and b but don't have to be consecutive. Discuss its time complexity. Write the subproblems in English, Recurrence Relation and Pseudo code as well (20 Points)

a. Define (in plain English) subproblems to be solved. (2 pts)

Ans:

Alice and Bob have two strings, a and b. They want to find the longest common subsequence (LCS) of characters present in both strings. The characters in the LCS must maintain their order in both strings, but they don't have to be consecutive.

b. Write down the base cases: (2 Points)

Ans:

Base cases:

1. Empty String Case
2. Comparing Characters
3. Handling Non-Matching Characters.

c. Write the recurrence relation: (6 Points)

Ans:

The recurrence relation for LCS length at a given position (i, j):

LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])

d. Write the pseudoCode for telling if such a string can be found and if so, find the resulting string. (8 Points)

Ans:

```
def LCSfunc(a, b):
    m = length(a)
    n = length(b)
    For i from 1 to m:
        For j from 1 to n:
            if a[i-1] == b[j-1]:
                LCS[i][j] = LCS[i-1][j-1] + 1
            else:
                LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
    return LCS[m][n]
result = LCSfunc(a, b)
print(result)
```

e. Write the time complexity (2 Points)

Ans:

Since there are 2 for loops (one iterates m times while other iterates n times), the time complexity of this pseudocode is O(m.n).