HOMEWORK 7

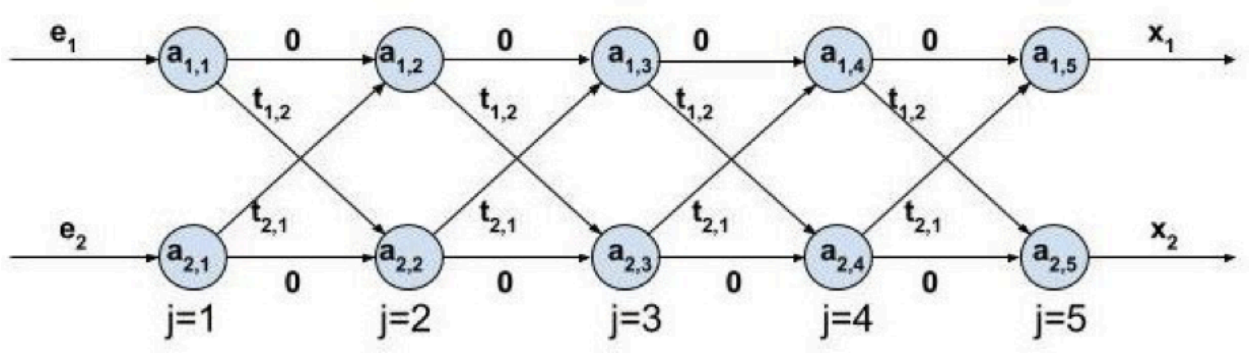
Q1. A car factory has k assembly lines, each with n stations. A station is denoted by $S_{i,j}$ where i indicates that the station is on the i-th assembly line (0<i<=k), and j indicates the station is the j-th station along the assembly line (0<j<=n). Each station is dedicated to some sort of work like 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 $s_{1,1}$ -> $s_{2,2}$ -> $s_{1,3}$ -> $s_{2,4}$ -> $s_{2,5}$, 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)

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

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

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

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.

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

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

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

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)

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

c) Write the recurrence relation: (6 Points)

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

e) Write the time complexity (2 Points)

UNGRADED QUESTIONS

Q4. You've started a hobby of retail investing into stocks using a mobile app, RogerGood. You magically gained the power to see N days into the future and you can see the prices of one particular stock. Given an array of prices of this particular stock, where prices[i] is the price of a given stock on the ith day, find the maximum profit you can achieve through various buy/sell actions. RogerGood also has a fixed fee per transaction. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction. (20 pt)
Solution:

a. Define (in plain English) subproblems to be solved. (4 pts)

b. Write a recurrence relation for the subproblems (6 pts)

c. Using the recurrence formula in part b, write pseudocode to solve the problem. (5 pts)

d. Make sure you specify :
    1. base cases and their values (2 pts)
    2. where the final answer can be found (1 pt)

e. What is the complexity of your solution? (2 pts)

Q5. Tommy and Bruiny are playing a turn-based game together. This game involves *N* marbles placed in a row. The marbles are numbered 1 to *N* from the left to the right. Marble *i* has a positive value m_i. On each player's turn, they can remove either the leftmost marble or the rightmost marble from the row and receive points equal to the sum of the remaining marbles' values in the row. The winner is the one with the higher score when there are no marbles left to remove.
Tommy always goes first in this game. Both players wish to maximize their score by the end of the game. Assuming that both players play optimally, devise a Dynamic Programming algorithm to return the difference in Tommy and Bruiny's score once the game has been played for any given input.

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

b) Write down the recurrence relation (8 Points)

c) Write down the the pseudo-code for this algorithm (8 Points)

d) Write down the time complexity (2 Points):

Q6. Consider a group of people standing in a line of size n. Everyone's heights are given in an array height = [h0, h1, ….. Hn-1]. Write an efficient algorithm to find the longest subsequence of people with strictly increasing heights and **return it**. Give the pseudo code and the recurrence relation.

For example: height = [6,1,2,3,4,5], ans = [1,2,3,4,5]

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

b) Write down the base cases (1 Points)

c) Write down the recurrence relation (3 Points)

d) Write down the pseudoCode: (4 Points)

e) Write down the time complexity and space complexity (2 Points)