COMP 3711 Final, Spring 2022, Wednesday May 25

Part 2: 6:00-7:15

P7, 5% Dynamic Progr. 1	P8, 15% Dynamic Progr. 2	P9, 10%, Spanning Trees	P10, 10%, Dijkstra	P11, 10%, Max Flows

Problem 7, Dynamic Programming 1, 5%
Recall the *rod-cutting problem* as discussed in the lecture slides. The recurrence describing the maximum revenue for a rod of length n is:

$$r[n]=\max(p[i] + r[n-i]), \forall\ i\ 1\le i\le n,$$

where $p[i]$ is the price for a rod of length i (given), and $r[n-i]$ is the maximum revenue for length $n-i$. The base cases are $p[0]=0$ and $r[0]=0$.
The original problem assumes that we can cut for free. Now assume that each cut has fixed cost c. Change the above recurrence accordingly, to represent the new maximum revenue $r[n]$ for length n.

Problem 8, Dynamic Programming 2, 15%
You are given an array $A[1..n]$, where all elements are distinct positive numbers. Describe a dynamic programming algorithm for finding a subsequence of elements in ascending order that maximizes the sum of elements in the subsequence. For example, if A=[7, 1, 4, 2, 8, 6], the *maximum sum subsequence* is [7,8].
Explain the recurrence, provide the pseudo-code and discuss the complexity of your algorithm. Your pseudo-code should print the elements of the maximum sum subsequence (in this example it should print 7 and 8 - any order is fine)
Hint: This problem is similar to the longest increasing subsequence in your tutorials. Define MS[i] to be the max sum of the increasing subsequence that ends at A[i], and use array keep[i] to store the previous element in the subsequence. For example, MS[5]=15 (i.e., 7+8) and keep[5]=7. Similarly, MS[6]=11 (i.e., 1+4+6) and keep[6]=3.

Problem 9, Minimum Product Spanning Tree, 10%
Consider a connected undirected graph G. A *minimum product spanning tree* is the spanning tree on G that minimizes the *product* of the weights of the edges. Describe how to find the minimum product spanning tree for graphs with the following constraints. You can use any minimum sum spanning tree algorithm (e.g., Prim, Kruskal) as a black box.

Question 1, 4%: All edge weights are > 0
Question 2, 6%: All edge weights are < 0

Problem 10, Dijkstra, 10%
Consider a computer network represented by a directed graph G, where nodes are computers and the edges represent direct connections between them. The edges do not drop packets, but each node v has a weight $w(v)$ that corresponds to the probability that an incoming packet is NOT dropped at v. Given a starting node s, compute the most reliable path to each other node v, i.e., the path along which a packet has the highest probability to reach v. Assume that the starting node s does not drop packets, i.e., $w(s)=1$.
Hint: Convert G to a new graph G' that has weights on the edges, instead of the nodes, and apply Dijkstra. Describe G' and write the pseudocode for the algorithm.

Problem 11, Maximum Flows, 10%

Consider the following exercise in you third assignment. You are given two arrays ROW[1..*n*] and COLUMN[1..*m*] of natural numbers. ROW[*i*] counts the number of 1's in the *i*-th row of a *n*×*m* binary matrix. Similarly, COLUMN[*j*] counts the number of 1's in the *j*-th row of the matrix.

For example, ROW[2 , 2] and COLUMN [2, 1, 1, 0] correspond to the 2x4 matrix:

1 1 0 0

1 0 1 0

Summing all the elements across row 1 we get 1+1+0+0 = 2. Similarly, across row 2, we get 1 + 0 + 1 + 0 = 2. Thus, ROW is [2, 2]. If we sum the elements across the columns, we get 1 + 1 = 2, 1 + 0 = 1, 0 + 1 = 1, and 0 + 0 = 0. Therefore, COLUMN is [2, 1, 1, 0]. Note that the sum of elements in ROW is the same as that of COLUMN (in this example 4).

Given ROW[1..*n*] and COLUMN[1..*m*], your goal is to reconstruct a *n*×*m* binary matrix, *by converting the problem to max flow*. Draw or describe the corresponding graph, including edge capacities, and explain how the max flow is used to give a solution to the original problem.