CS 577 # Assignment 8 – Dynamic Programming 2 Spring 2022

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: ______________________ Wisc id: ______________________

Dynamic Programming

Do **NOT** write pseudocode when describing your dynamic programs. Rather give the Bellman Equation, describe the matrix, its axis and how to derive the desired solution from it.

1. *Kleinberg, Jon. Algorithm Design (p. 329, q. 19).*

   We say that a string s is an *interleaving* of x and y if its symbols can be partitioned into two (not necessarily contiguous) subsequences s' and s'', so that s' is a repetition of x and s'' is a repetition of y. Give an efficient algorithm that takes strings s, x, and y and decides if s is an interleaving of x and y.

   Note: We write x^k to denote k copies of x concatenated together. We say that a string x' is a repetition of x if it is a prefix of x^k for some number k. So $x' = 10110110110$ is a repetition of $x = 101$.

Solution:

Dynamic programming approach:

- Let ℓ represent the length of s. Then our matrix S will be a triangle, where the indices included are all i, j such that $i + j \leq \ell$. Additionally, let $x[i]$ and $y[j]$ refer to the ith and jth characters of x' and y', respectively.
- Bellman Equation:

$$S[i][j] = \max\{S[i-1][j] * a_{ijx}, S[i][j-1] * a_{ijy}\},$$

  where a_{ijx} is 1 if $S[i+j] = x[i]$, a_{ijy} is 1 if $S[i+j] = y[j]$, and otherwise are 0.
  Initial value $S[0][0] = 1$.
- S should be calculated in order of increasing i, j sums (so starting with $S[0][0]$, then $S[0][1]$ and $S[1][0]$, and so on).
- If $\exists i, j$ such that $i + j = \ell$ and $S[i][j] = 1$, then s is an interleaving of repetitions of x and y.

2. Consider the following problem: you are provided with a two dimensional matrix, M, (dimensions, say, m x n). Each entry of the matrix is either a **1** or a **0**. You are tasked with finding the total number of square sub-matrices of M with all **1**s. Give an $O(mn)$ algorithm to arrive at this total count.

   Furthermore, how would you count the total number of square sub-matrices of M with all **0**s?

Solution:

We can define the following bellman equation for our problem.

Consider the state $dp[i][j]$ = total number of square sub-matrices with all **1**s with cell (i,j) as the bottom right cell of each of these sub-matrices.

Then for each cell (i, j) with $M[i][j] = 1$, we have

$$dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]).$$

For each cell (i, j) with $M[i][j] = 0$, we have

$$dp[i][j] = 0.$$

To arrive at the final answer, we simply sum up all the entries of the dp matrix.

To calculate the total number of square sub-matrices with all **0**s, you can simply invert each entry in the matrix and use the same bellman equation as above.

3. *Kleinberg, Jon. Algorithm Design (p. 327, q. 16).*

   In a hierarchical organization, each person (except the ranking officer) reports to a unique superior officer. The reporting hierarchy can be described by a tree T, rooted at the ranking officer, in which each other node v has a parent node u equal to his or her superior officer. Conversely, we will call v a direct subordinate of u.

   Consider the following method of spreading news through the organization.

   - The ranking officer first calls each of her direct subordinates, one at a time.
   - As soon as each subordinate gets the phone call, he or she must notify each of his or her direct subordinates, one at a time.
   - The process continues this way until everyone has been notified.

   Note that each person in this process can only call *direct* subordinates on the phone.

   We can picture this process as being divided into rounds. In one round, each person who has already heard the news can call one of his or her direct subordinates on the phone. The number of rounds it takes for everyone to be notified depends on the sequence in which each person calls their direct subordinates.

   Give an efficient algorithm that determines the minimum number of rounds needed for everyone to be notified, and outputs a sequence of phone calls that achieves this minimum number of rounds.

Solution:

Dynamic programming approach:

- A 1-dimensional matrix T' whose indices correspond to the nodes in T. $T'[v]$'s value is the minimum number of rounds needed to contact all nodes in the subtree rooted at v in T. Additionally, let $S[1..n]$ represent the n direct subordinates of v, ordered (as we go) by decreasing value of T'.
- Bellman Equation:

$$T'[v] = \max_{i=1..n} \{i + T'[S[i]]\},$$

  where leaf nodes (people with no direct subordinates) have value 0 in T'.
- Since the value of a node u in T' depends on the children of u, we want to process T in a bottom-up manner, so use a post-order traversal to calculate T'.
- The minimum number of rounds needed will be found in the node in T' corresponding to the ranking officer.

The sequence of phone calls can be recovered from T' via the following greedy process: At the start of each round, every person who has been contacted will contact the direct subordinate of theirs who has the largest value in T' and has not yet been contacted.

4. *Kleinberg, Jon. Algorithm Design (p. 330, q. 22).*

   To assess how "well-connected" two nodes in a directed graph are, one can not only look at the length of the shortest path between them, but can also count the number of shortest paths.

   This turns out to be a problem that can be solved efficiently, subject to some restrictions on the edge costs. Suppose we are given a directed graph $G = (V, E)$, with costs on the edges; the costs may be positive or negative, but every cycle in the graph has strictly positive cost. We are also given two nodes $v, w \in V$.

   Give an efficient algorithm that computes the number of shortest $v - w$ paths in G. (The algorithm should not list all the paths; just the number suffices.)

Solution:

Run the Bellman-Ford shortest path algorithm on G, with the following additions:

- Make an additional $(|V| - 1) \times |V|$ matrix C to keep track of edge counts. In this matrix, $C[i][n]$ stores the number of simple shortest $n - w$ paths using i or fewer edges.
- For any $n \in V$ with $(n, w) \in E$, $C[1][n]$ should be initialized to 1. All other entries in C should be initialized to 0.
- When Bellman-Ford runs through the mininum checking step for $n \in V$:
  - If $M[i-1][n]$ is a minimum, add $C[i-1][n]$ to $C[i][n]$.
  - $\forall m$ s.t. $(m, n) \in E$, if $M[i-1][m] + c_{mn}$ that is a minimum, add $C[i-1][m]$ to $C[i][n]$.
- The number of shortest $v - w$ paths can be found in $C[|V| - 1][v]$.

5. The following is an instance of the Knapsack Problem. Before implementing the algorithm, run through the algorithm by hand on this instance. To answer this question, generate the table, indicate the maximum value, and recreate the subset of items.

item	weight	value
1	4	5
2	3	3
3	1	12
4	2	4

Capacity: 6

Solution:

4	0	12	12	16	16	17	19
3	0	12	12	12	15	17	17
2	0	0	0	3	5	5	5
1	0	0	0	0	5	5	5
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Max value: 19

Items used: 2, 3, 4

6. Implement the algorithm for the Knapsack Problem in either C, C++, C#, Java, or Python. Be efficient and implement it in $O(nW)$ time, where n is the number of items and W is the capacity.

The input will start with an positive integer, giving the number of instances that follow. For each instance, there will two positive integers, representing the number of items and the capacity, followed by a list describing the items. For each item, there will be two nonnegative integers, representing the weight and value, respectively.

A sample input is the following:

```
2
1 3
4 100
3 4
1 2
3 3
2 4
```

The sample input has two instances. The first instance has one item and a capacity of 3. The item has weight 4 and value 100. The second instance has three items and a capacity of 4.

For each instance, your program should output the maximum possible value. The correct output to the sample input would be:

```
0
6
```