CS 577 # Assignment 8 – Dynamic Programming 2 Fall 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: Yongsang Park Wisc id: 908 022 3169

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).*

   String x' is a *repetition* of x if it is a prefix of x^k (k copies of x concatenated together) for some integer k. So $x' = 10110110110$ is a repetition of $x = 101$. 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 x' and y', so that x' is a repetition of x and y' is a repetition of y. For example, if $x = 101$ and $y = 00$, then $s = 100010010$ is an interleaving of x and y, since characters $1, 2, 5, 8, 9$ form 10110—a repetition of x—and the remaining characters $3, 4, 6, 7$ form 0000—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.

Solution:

```
Repetition
input : S, x, y
lenS = length of S
lenX = length of x
lenY = length of y
xIndex = 0
yIndex = 0
for i = 0 to (lenS-1)
  if (xIndex < lenX && s[i] == x[xIndex])
    xIndex = xIndex+1
  else if (yIndex < lenY && S[i] == y[yIndex])
    yIndex = yIndex+1
  end
end
If xIndex == lenX && yIndex == lenY
  S' = substring of s (0:lenS)
end
check s if each lenS alphabets is same as S'.
If they are all true, return true
else, false.
```

2. Consider the following problem: you are provided with a two dimensional matrix M (dimensions, say, $m \times n$). 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:

Matrices Ones

Input: M

Make a matrix S that is the same size as M.

Copy the first row and column of M and paste into S

1	1	0	1
1	1	0	0
1	0	0	1

1 2 0 1
2 2 0 0
2 0 0 1

```
for 1 < i < M.length
|  for 1 < j < M[0].length
|  |  if M[i][j] == 1
|  |  |  S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1
|  |  else // if M[i][j] == 0
|  |  |  S[i][j] = 0
|  |  end
|  end
end
```

find the # of cells that larger than 2.

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.

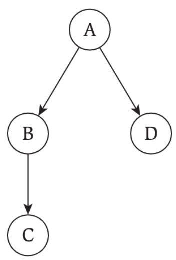
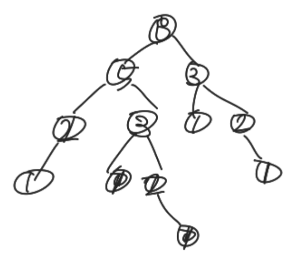


Figure 1: A hierarchy with four people. The fastest broadcast scheme is for A to call B in the first round. In the second round, A calls D and B calls C. If A were to call D first, then C could not learn the news until the third round.

   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:



We need to figure out the subordinates.
Each node in the tree, find the # of subordinates
by using post-order traverse.
Let list S store the # of subordinates.
Create a list T
for i in 0 to n (# nodes)
$T[i] = \max(j + T[S[j]])$ (for j in 0 to i)
end
It will yield the # rounds.

According to the tree which stores # subordinates, make the ranking officer call the large value node. This sequence will be the optimal sequence.

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:

Make a $(|V|-1) \times |V|$ matrix M. // This matrix will store the edge counts

For $n \in V$ with $(n, w) \in E$
  $M[1][n] = 1$
end

When running Bellman-Ford algorithm, check step for $n \in V$:
  let T be the edge matrix
  if $T[i-1][n]$ is a minimum, add $M[i-1][n]$ to $M[i][n]$
  $\forall m$ & $(m, n) \in E$, find the min of $T[i-1][m] + c_{mn} + M[i-1][m]$ to $T[i][n]$
end

$M[|V|-1][v]$ will print the shortest path of $v - w$.

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:

item #

4	0	0	4	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

Capacity

Max value: 19

Items: 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
```