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: _____

More 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. 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.

   parent node u, direct subordinate v

   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.

   SO = subordinate office

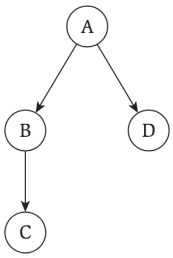
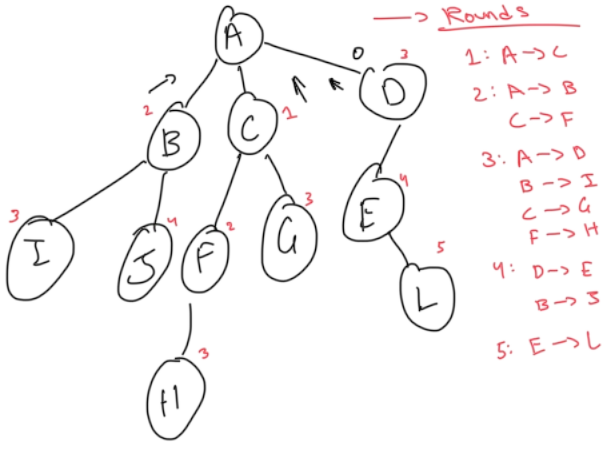


Figure 1: A hierarchy with four people. The fastest broadcast scheme is for A to call B in the first round. In the second round, A calls D and B calls C. If A were to call D first, then C could not learn the news until the third round.

* Call the subordinate with the largest amount of subordinates underneath them.
* This will ensure that the news is reached in the smallest # of rounds, as more simultaneous calls can be made.
→ i.e: B has 1 SO, D has 0, thus call B first.
→ If person has 0 subordinates, return 1 for the round.



* once a node is reached, they can start parallelly calling subordinates at the same time as other nodes.
* want to sort the list of nodes by how many subordinates it has to minimize the # of rounds / maximise # of nodes reached at a time.

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 by answering the following:

(a) Give a recursive algorithm. (The algorithm does not need to be efficient)

MinimizeRounds:
Input: A person p with subordinates $[s_1, s_2, \dots s_n]$, n number of subordinates
Output: Sequence of phone calls needed, and number of rounds required.
Base case: If $n = 0$: return $(0, [])$ (no rounds required, thus returning empty list)

```
order = []
for si in subordinates:
    order = order.append(MinimizeRounds(si))
M = [] //list to hold descending order of rounds
 M = min(order[si]) //get the min number of rounds from si
S = M[1...n] //schedule based on min number of rounds

return M, S
```

(b) Give an efficient dynamic programming algorithm.

Let M[n] denote the number of rounds needed to contact all nodes (direct subordinates) in the heirarchy, starting at the top of the heirarchy, person n
Let s_i be the direct subordinates of n, ordered in decreasing value of M(i).
↳ The heirarchy can be represented as a tree, thus every direct subordinate has its own set of subordinates, like a subtree. Thus we want to minimize the number of rounds in every subtree.
Base Case: $M[1] = 1$, only one subordinate, thus one round. $M[0] = 0$ → noone to call.
Bellman Equation: $M[i] = \min_{i=1\dots n} \{ i + M[s_i[i]] \}$ → Get the min. rounds of each subordinate

Solution:
Can be found at $M[n]$, where n = root of heirarchy.
To get the sequence of calls, we can reverse engineer the solution number by keeping a parallel array that adds each node that gets called at every table lookup.

Runtime: O(1) lookups
O(nlogn) sorting
n entries to look thru,
so O(n)
↳ $O(n^2 \log n)$?

(c) Prove that the algorithm in part (b) is correct.

We can prove the correctness through strong induction on the number of people in the heirarchy, n.

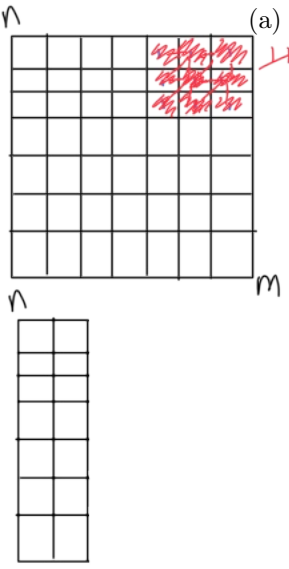
Base Case 1 : $n=0$: No subordinates to call, thus 0 is correct.

Base Case 2 : $n=1$: When there is only 1 person in the tree, only 1 call needs to be made, thus the base case holds.

Inductive step: Assume correctness in the table lookups performed, the algorithm would then find the minimum number of rounds out of all the calls to the sub-ordinates, thus finding the min. number of rounds required for that subtree, giving us the answer.

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 by answering the following:

4 or 13



(a) Give a recursive algorithm. (The algorithm does not need to be efficient)

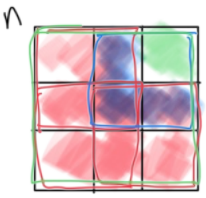
Square Ones: (M[m][n])

Input: A matrix M with dimensions m x n, each entry either 1 or 0

Output: Number of square sub-matrices of M with all 1s.

```
if m=0:
    return 0
total = SquareOnes(M[1...m-1][1...n])   // making m smaller, reaching base case
                                        we use n to go through all rows
for i in [1...n]:
    current_total = number of square submatrices with all 1s, with M[m][i]
                    as the top right cell.
    total += currentotal
return N
```

(b) Give an efficient dynamic programming algorithm.



check (i-1,j), (i,j-1), (i-1,j-1)

Let S(i,j) denote the number of square sub-matrices that contain all 1's in a matrix M that only contains 1's and 0's.

Base Case

Bellman Equation:

$$S[i,j] \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ 0 & \text{if } M[i][j]=0 \\ \min(S[i-1,j], S[i,j-1], S[i-1,j-1]) + 1) \end{cases}$$

Order of population: from 1 to m
from 1 to n

Solution can be found at S[m,n], as the whole matrix will be parsed through, thus the final answer will be returned at S[m,n], the top-right cell of the matrix.

Runtime: Population: O(mn) } total runtime is O(mn)
Work in each cell: O(1) }

(c) Prove that the algorithm in part (b) is correct.

We will prove correctness through strong induction on the size of the matrix.

Base Case 1: $m=0$ or $n=0$: In this case, no more square sub-matrices can occur thus the number of square matrices with 1s must be 0!

Base Case 2: If the current table lookup returns a zero, then there cannot be a sub-matrix with all 1's with this cell included, thus returning 0.

Inductive step: with $m=x$ and $n=y$, we assume that the table lookups return correct values. Thus, the lookups will correctly return how many submatrices of full 1s there are in the submatrix we are looking at (so $M[i-1,j]$, $M[i,j-1]$, and $M[i-1,j-1]$). We then take the minimum of that because if there is a 0 then that means the whole matrix (i.e.: at $M[i,j]$) cannot be a sub-matrix of 1s. Furthermore, the +1 is because a 1x1 1 is still a submatrix of 1, thus valid.

↳ Thus, from above, we have shown how our algorithm returns the correct answer.

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

Just convert checking for 1's to checking for 0's, as the base of the algorithm should work.

The "skip" case should also be changed to checking for 1's.

3. *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 by answering the following:

(a) Give a recursive algorithm. (The algorithm does not need to be efficient)

Interleaving:
Input: string s, x and y
Output: Answer if s is an interleaving x and y (Boolean)
if s is empty, return True
if x not in s AND y not in S:
  return False
is X True = False
is Y True = False
if s[0] = x[0] then:
  Interleaving(s, x[2...], y)
if s[0] = y[0] then:
  Interleaving(s, x, y[2...])
return isXTrue AND isYTrue

(b) Give an efficient dynamic programming algorithm.

Let x' and y' be infinite repetitions of x and y respectively. Let S[i,j] denote whether the substring $s_1 \dots s_{i+j}$ is an interleaving of $x'_1 \dots x'_i$ and $y'_1, \dots y'_j$

Base Case: $S[0,0] = \text{True}$, $S[0,j] = (s_j == y'_j)$, $S[i,0] = (s_i == x'_i)$

Bellman Equation: $S[i,j] = [S[i-1,j] \wedge s_{i+j} == x'_i] \wedge [S[i,j-1] \wedge s_{i+j} == y'_j]$

Sol: can be found at S[len(x), len(y)]
order of population: 1 to i, 1 to j
Runtime = $O(n^2)$

(c) Prove that the algorithm in part (b) is correct.

The base cases of the algorithm have to be true since they logically make sense to be true.

The general case states that the string s can be interleaved with x/y only if the last char (i/j) match with the char at x/y, and then this is then both checked. If both return true, then s must be interleaved with x and y.

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.)

Keep track of # of shortest paths and the cost of the shortest path in 2 2D parallel matrices, C[n][i] and S[n][i].
C[1][i] is the cost of the shortest path from i to w of length n.
S[1][i] = 1 if there is an edge. Otherwise, C[1][i] = ∞ and S[1][i] = 0

$$C[n][i] = \min\{c_{ij} + C[n-1][j]\}$$

$$M[n][i] = \sum_j M[n-1][j] \text{ for } j \in V$$

sol: C[n][v]
M[n][v]

5. The following is an instance of the Knapsack Problem. Before implementing the algorithm in code, 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.

Knapsack Problem DP:

$$V[i,w] = \max\left(V[i-1,w],\ x_{i,w}\cdot\left(V[i-1,W-w_i] + v_i\right)\right)$$

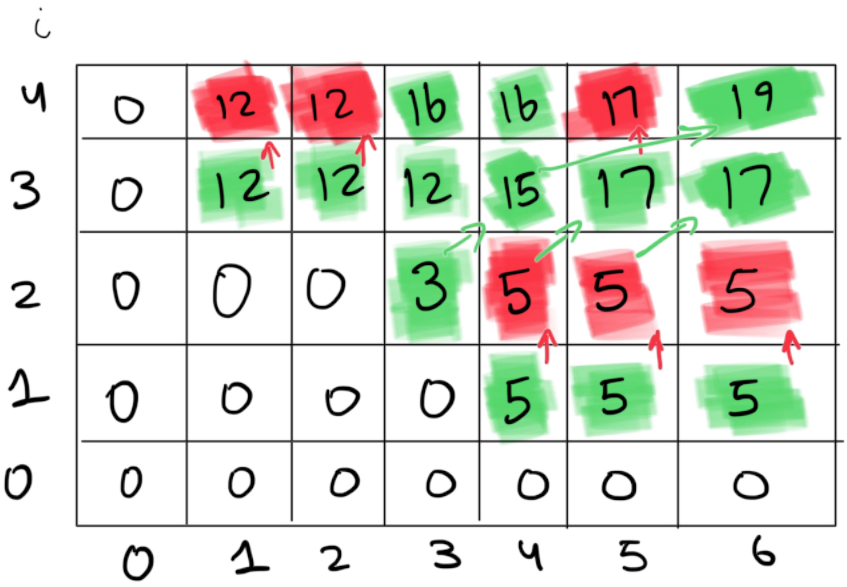
i = current item
W = Max weight available
w_i = current item weight
$x_{i,w}$ = indicator
v_i = value of item

BASE CASE
$V[i,0] = 0$
$V[0,w] = 0$

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

Capacity: 6

- take
- skip



Max Value = 19
↳ Acheived by using items 4, 3, 2

6. **Coding Question: Knapsack**

   Implement the algorithm for the Knapsack Problem in either C, C++, C#, Java, Python, or Rust. 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 nonnegative 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
```