CS 577 # Assignment 8 – Dynamic Programming 2 Spring 2023

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: Kayley Seow Wisc id: kseow

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.

   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.

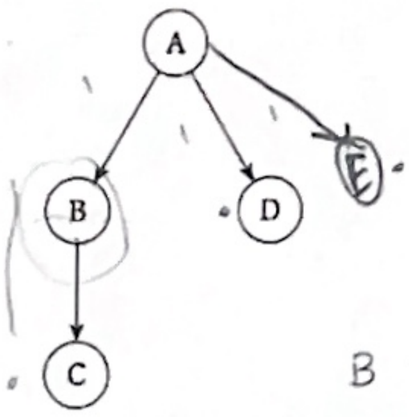


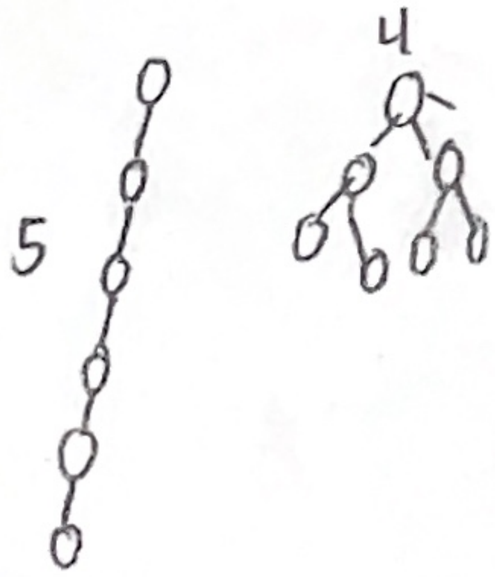
Figure 1: A hierarchy with four people. The fastest broadcast scheme is for A to call B in the first round. In the second round, A calls D and B calls C. If A were to call D first, then C could not learn the news until the third round.

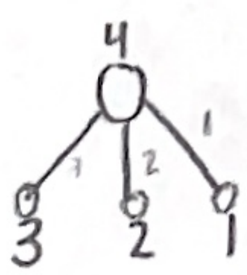
B D E C

- notify person with more subordinates first
- add 1

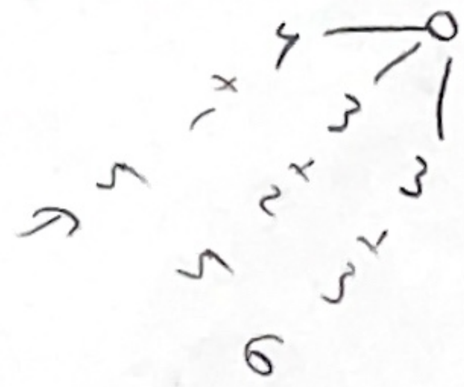
The questions are on the next page.

R(node n)





max (3,2,1)
+1 +2 +3



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)

Solution:

Given the node, our first focus will to be to get how long it takes for each children to complete their subordinate contact. With these values, we then will sort them from longest time to shortest time, and then we will recurse or for loop through the values, decreasing, of the nodes, where we start i=1, and increment it with every loop until we reach the last time. At the first sorted value, we get the longest runtime, and we can return the answer. ~~Take the max after we take the indices addition.~~

(b) Give an efficient dynamic programming algorithm.

Solution:

$$A[n] = \begin{cases} A[\text{leaf nodes}] = 0 \\ \max(A[c-i]+1 \quad \text{for } i = 1 \ldots \text{len}(c)) \end{cases}$$

where c is the # of children of the node

Namix: One dimension, one df because of the node passed in, at A[100]

Runtime: We have an n dimension 1D namix, and we have to loop through them for the number of children leaf nodes, which gives us $O(n^2)$

Fill up the array from the leaf nodes.

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

Solution:

Base case: A[n], n = leaf node = 0. With this modified DP problem, at any leaf we return 0 as there is no one it needs to report to.

Inductive Hypothesis

→ IH: At current node, the fastest broadcast scheme depends on longest time first.

→ for the k+1th node, max(A[c-i]+1 for i=1, len(c)) where c # of children of k+1

→ By strong induction, we know that A(k) will return best because A[c-i] returns max

→ Algo chooses max to go first

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:

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

S M

$\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \rightarrow \begin{bmatrix}1&1&1\\1&2&2\\1&2&3\end{bmatrix}$

max sum

Solution:

The algorithm uses a 0 if it is at a 0, but if it is a 1, it will count. It looks at the one above, left, and diagonal in order to determine the current square matrices at the current index.

$\begin{bmatrix}1&0&0&0\\1&1&1&0\\0&1&1&0\end{bmatrix} \rightarrow \begin{bmatrix}1&0&0&0\\1&0&1&0\\0&1&2&0\end{bmatrix}$

   (b) Give an efficient dynamic programming algorithm.

Solution:

$S[m,n] = \begin{cases} M[m,n]=0 \\ M[m,n]=1,\ \min\{S[i-1,j], S[i,j-1], S[i-1,j-1]\}+1 \end{cases}$

Matrix is 2d, two dfs for either 1 or 0

Runtime, we would have n^2, ds we have to build the n x n matrix

Fill up from the upper left corner of matrix

Solution at [n,n] sum of all the values

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

Solution:

Base case: 0x0 matrix, sum is 0

Induction Step:

→ IH: Algorithm has the correct sum of the total square

→ for the k+1th and l+1th node, $\min\{S[k,l+2], S[k+1,l], S[k,l]\}+1$ gives us the sum for value.

→ By IH, we know S[k,l+1], S[k+1,l], and S[k,l] gives us respective sum

→ Strong induction means S[m,n] gives us the max optimal sum solution

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

Solution:

change the if statement to count 0's instead of 1's, counting would still be the same.

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 10110a repetition of $x$and the remaining characters $3, 4, 6, 7$ form 0000a 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)

Solution:

```
opt(i, lx, ly):
  if i == len(s): return true
  boolean success = false
  if S[i+1] == x[(lx+1) % len(x)]
     success = opt(i+1, (lx+1)%|x|, ly)
  if S[i+1] == y[(ly+1) % len(y)]
     success = opt(i+1, lx, (ly+1)%len(y))
  return success
```

for the string, we want to iterate through both the x and y to try the interleaving

(b) Give an efficient dynamic programming algorithm.

Solution:

M[i][j] = M[i][j-1] + M[i+1][j]

- Matrix is two dimensions, to account for x and y
- Represents substring from i to j on interleaving of x and y
  - j-i < len(x) or j-1 < len(y) and fill with true if true x or y

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

Solution:

Base case: if true is string smaller than |x|+|y|, doesn't match, will not be interleaving.

We remove substrings that match and then compare with our interleaveS.

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:

There are no negative cycles, we can use a modified shortest path algorithm to find the shortest path, to keep track of each node entering w. We start with the full graph, and fill in the 2D matrix M, where M[i][j] is the shortest path from j to w using the i edges. We have the Bellman equation of:

$M[i][v] = \min\{M[i-1][v], \min_{u \in V}\{M[i-1][u] + C_{vu}\}\}$, where V is the set of vertices i edges away from our v.

We find the shortest path of M[n-1][w], finding the vertex we pass through, furthermore if from our graph and repeat the process until we have a shortest path value greater than the first one we stored in the matrix.

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

if over capacity:

~~capacity~~

0 1 2 3

[(4,5), (3, 3) (1, 12) (2, 4)]

Solution:

items (i)	0	1	2	3	4	5	6
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

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
```