CS 577 **Assignment 8 – More Dynamic Programming** Fall 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: Ethan Yan Wisc id: 9084649137

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).*

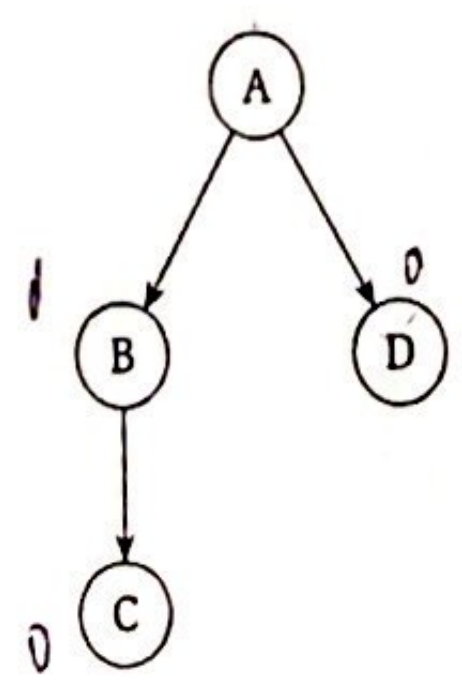
   In a hierarchical organization, each person (except the ranking officer) reports to a unique superior officer. The reporting hierarchy can be described by a tree T, rooted at the ranking officer, in which each other node v has a parent node u equal to his or her superior officer. Conversely, we will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

   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.

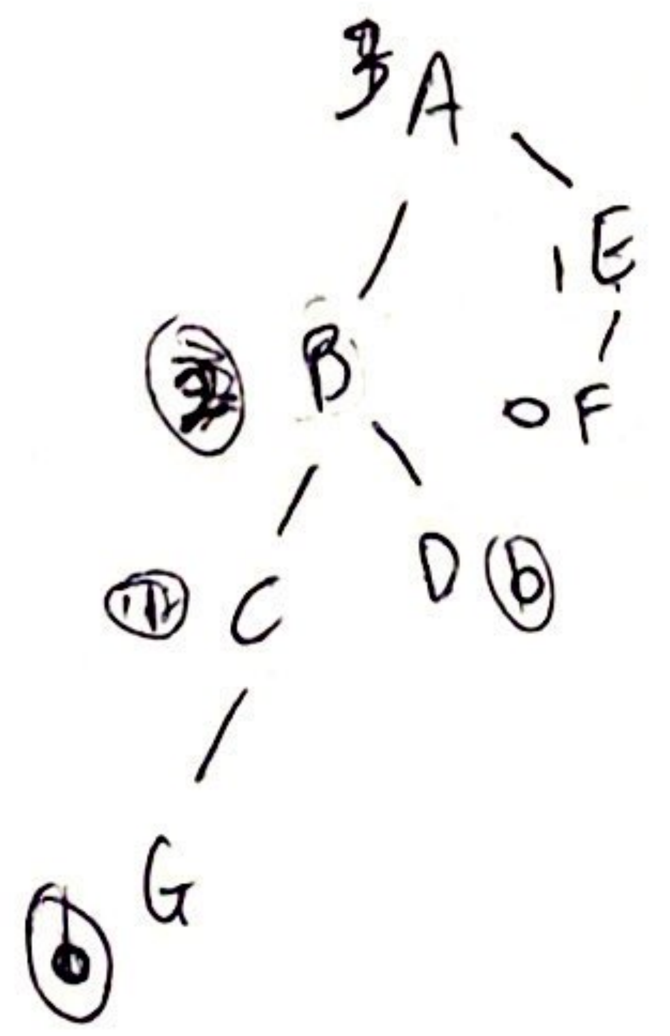
   Note that each person in this process can only call *direct* subordinates on the phone.

   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.



2 [B, D] 1+0 2+1 [D, B]

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.



R(v)

[c, D]

(R(C)+1, R(D)+2)

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)

Use BFS starting at root, can start ~~after~~ at multiple points after, as long as visited.

sequence = [ ]

~~BFS(visited):~~
~~n = len(visited)~~
~~for i = 1 to n:~~

BFS(root, visited):
    root.visited = True
    sequence.append(root)
    for ~~i~~ root has next that is not in visited:
        BFS(root.next, visited)
    ~~for vi~~ while visited:
        BFS(visited[i]).

return sequence.

Do BFS & recursively start BFS at all visited nodes.

(b) Give an efficient dynamic programming algorithm.

start from longest length?

~~Array M of length of n~~   ~~M[i] represents min. rounds needed to notify all subordinates of node i~~

$R(v)$: minimum number of rounds needed to contact all nodes of subtree with root v.

$S_v[1 \dots n]$ is the list of subordinates of v ordered in descending order of $R(v)$.

Then the Bellman Equation for v would be:

$$R(v) = \max_{i=1,2,\dots,n} \left(i + R(S_v[i]) \right)$$

Solution is at R(ranking officer).

Use a stack to keep track of nodes not explore, initialize w ranking officer.

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

Use strong induction.

~~There is~~

For a root v with subordinates $(u_1, u_2, \ldots, u_k)$.

suppose not sorted in descending order.

Then there is a u_i and u_j such that

$$i < j \quad \text{and} \quad R(u_i) < R(u_j)$$

Then we can swap u_i & u_j

without an increase in $R(v)$.

since $R(v) = \max_{i=1,2,\ldots,n} (i + R[S_v[i]])$

so by swapping, $i + R(u_j) \leq j + R(u_i)$

which does not increase $R(v)$.

Therefore, we can continue implementing these inversions until we get $S_v[1, \ldots, n]$ ordered in descending order of $R(v)$.

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)

countSquares (matrix): Input m×n matrix

~~if matrix = [1]:~~

~~return 1~~

~~else if matrix = [0]:~~

~~return 0.~~

~~else~~

if m=0; return 0

N ← countsquares (Matrix[2...m][1...n])

for i ← 1,...,n:

k ← number of squares with M[1,i] as upper left corner

N ← N+k

return N.

(b) Give an efficient dynamic programming algorithm.

dp array of m×n.

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

dp[i][j] is # of sub squares with M[i][j] as bottom right corner.

Solution is sum of whole dp array.

Runtime is O(nm) since we populate every entry in dp array (m×n) once.

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

In the dp solution we are considering every square sub matrizes. as we populate the dp array with each index i, j indicating the square sub matrizes with it as the bottom right corner.

Suppose a largest length l square with bottom right corner at M[i][j], this also includes 3 length l-1 squares with bottom right corners at M[i-1][j], M[i][j-1], M[i-1][j-1].

Therefore dp[i][j] ≤ min (dp[i-1][j], dp[i][j-1], dp[i-1][j-1]).

This can be reiterated until length 1 squares, concluding that we considered all square sub matrizes.

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

Invert M by changing all 1's to 0's and 0's to 1's and re-run dp algorithm.

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)

```
Interleave (s, X, Y)        set X = x^k, Y = y^k.

  if len(s) = 0:
        return True
  if s[0] ≠ X[0] & s[0] ≠ Y[0], return False.
  if s[0] = X[0]
        b_x = Interleave (s, X[2...n], Y)
  if s[0] = Y[0]
        b_y = Interleave (s, X, Y[2...n])

  return (b_x or b_y).
```

   (b) Give an efficient dynamic programming algorithm.

Let x^* & y^* be infinite repetition of x & y resp. Let $S(i,j)$ be if substring $s_1 s_2 \dots s_{i+j}$ is an interleaving of $x_1^* \dots x_i^*$ and $y_1^* \dots y_j^*$

$$S(i,j) = [S(i-1,j) \wedge (s_{i+j} == x_i^*)] \vee [S(i,j-1) \wedge (s_{i+j} == y_j^*)]$$

Run time is $O(n^2)$ since i and j both at most length n (length of s)

$S(0,0) = \text{True}$ (True if there is some $i+j=n$ with $S(i,j) = \text{True}$)

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

Base case clearly true.
By strong induction assume $S(i-1, j)$ and $S(i, j-1)$ returns correct values. Then substring $s_1 s_2 \cdots s_{i+j}$ can be interleaved as long as s_{i+j} matches x_i^* or y_j^*, otherwise returning false.
By recurrence & strong induction it holds, all repetitions of x & y checked as well.

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.)

2D Matrix M of n(# of edges in path) x vertices
↳ M[n][i] is shortest path from i to w using ≤ i edges

Initialize $M[1][i]$ = cost of c_{iw}.

2D Matrix N where $N[n][i]$ is # of paths from i to w. $N[1][i] = 1$

if no edge from i to w, then $M[1][i] = \infty$, $N[1][i] = 0$.

So $M[n][i] = \min_{j \in V} \{c_{ij} + M[n-1][j]\}$

$N[n][i] = \sum_j N[n-1][j]$ for $j \in V$ with $c_{ij} + M[n-1][j] = M[n][i]$.

To find solution, first find cost of shortest path from v to w,

$c = \min_{1 \le n \le |V|-1} M[n][v]$ then using each j with $M[j][v] = c$,
return $m = \sum_j N[j][v]$.

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.

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

Capacity: 6

capacity = 6.

$$\text{items}\quad \begin{array}{c} 4 \\ 3 \\ 2 \\ 1 \\ \\ \end{array} \begin{array}{c} \begin{bmatrix} 12 & 12 & 16 & 16 & 17 & 19 \\ 12 & 12 & 12 & 15 & 17 & 17 \\ 0 & 0 & 3 & 5 & 5 & 5 \\ 0 & 0 & 0 & 5 & 5 & 5 \end{bmatrix} \\ \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \end{array} \quad M$$

capacity.

Max value : 19

Itemes used : 2, 3, 4

$$\cancel{\max\{ M[2][5-1]+12, \}}$$

$$\max\{ \underbrace{M[5-2][3]+4}_{16}, 17 \}$$