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: Matej Popovski ____________________ 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. 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.

`figures/q3.png`

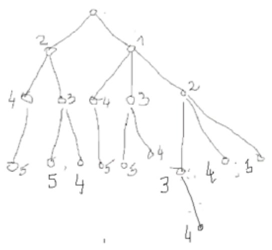
Figure 1: A hierarchy with four people. The fastest broadcast scheme is for A to call B in the first round. In the second round, A calls D and B calls C. If A were to call D first, then C could not learn the news until the third round.

The questions are on the next page.

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:



```
function f(v)
if v is leaf node
  return f(v)=0;
else
  S= Set of direct subtrees of node v sorted in descent
     ordered  by value f($s_j$) ,  f($s_1$>=f($s_2$>=...>=f($s_k$)  j=1,...k
  for each node $s_j$ in S in
        f(v) = max_j {j+ f(sj)};
  end;
```

(b) Give an efficient dynamic programming algorithm.

Solution:

For each node v in the tree, the table value x(v) is the number of rounds it take every node in its subtree to be notified. Let P1, P2,..Pk be children subtrees of v in descending order of their subtrees number of rounds x(P1)>=x(P2)....>=x(Pk).

Bellman equation: x(v)=max (j+x(Pj)) j=1,..k x(leaf)=0

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

Solution:

Base case x(leaf)=0. Staring from leafs, algorithm builds recurrently up for each node the value showing how many calls is needed to inform all its subordinated nodes. The root of the tree therefore will show the number of calls needed for the entire tree. There is finite number of nodes, and we do sorting which has time complexity of O(nlogn).

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)

Solution:

```
def countSubMatrices(M, i, j, count):
 if i == 0 or j == 0: # base case
    return count
 if M(i-1, j-1) == 1:
    count += 1 # increment the count by 1
    # compute the count of square sub-matrices that end at (i,j)
       count = min(countSubMatrices(M, i-1, j-1, count),
              countSubMatrices(M, i-1, j, count),
              countSubMatrices(M, i, j-1, count))
 else # if the current cell contains 0, then the count of square sub-matrices that end at (i,j) is 0
countSubMatrices(M, i-1, j-1, count)
return count
```

   (b) Give an efficient dynamic programming algorithm.

Solution:

Bel'lman equation

s(i, j)   min {s(i−1, j), s(i−1, j−1), s(i, j−1)} + 1   for M(i,j)=1
s(i,j) = 0   for M(i,j)=0
s(0,0) = M(0,0)

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

Solution:

The dynamic programming table s has the same dimensions as the matrix M. The value of s(i,j) represents the number of square sub-matrices with all 1s that end at position (i,j) of the matrix M.

The base case is when M(i,j)==0, in which case s(i,j)=0. If M(i,j)== 1, we compute s(i,j) as the minimum of the three adjacent cells to the left, above, and diagonally to the left and above, plus 1. This represents the number of square sub-matrices with all 1s that end at position (i,j) and whose sides are formed by the adjacent cells to the left, above, and diagonally to the left and above. . The time complexity of this algorithm is O(mn).

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

Solution:

The value of s(i,j) represents the size of the largest square submatrix with all 1s that has its bottom-right corner at (i, j). To count the total number of square submatrices with all 1s, we can sum up all the values in the matrix s. Similar to count the total number of square submatrices with 0s we can exchange the role of 1s and 0s in previous solutions

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)

Solution:

```
def isInterleaving(x, y, s, i, j, k):
  if k == len(s):
    # Base case: if we have processed all characters in s, return True if i and j have reached the end of x
   and y respectively, else return False
    return i == len(x) and j == len(y)
  if i < len(x) and s[k] == x[i]:
     # Check if the current character in s matches the current character in x
     if isInterleaving(x, y, s, i+1, j, k+1):
        # Recursive call for the case where the current character in s matches the current character in x
        return True
  if j < len(y) and s[k] == y[j]:
    # Check if the current character in s matches the current character in y
    if isInterleaving(x, y, s, i, j+1, k+1):
       # Recursive call for the case where the current character in s matches the current character in y
       return True
   # If the current character in s does not match either the current character in x or y, return False
  return False
```

The function takes in three strings x, y, and s, and three indices i, j, and k. The indices i and j represent the current positions in x and y, respectively, and the index k represents the current position in s. In each recursive call, we check if the current character in s matches the current character in x or y, and recursively call the function with the next positions in x or y, and in s. If the function returns True for either of these recursive calls, we return True. If no match is found, we return False. The time complexity of this algorithm is O(2^(m+n)).

(b) Give an efficient dynamic programming algorithm.

Solution:

dp(i,j) = (dp(i-1,j) and s(i+j-1) == x(i-1)) or (dp(i,j) and s(i+j-1) == y(j-1))

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

Solution:

To check if a string s is an interleaving of strings x and y using dynamic programming, we can use a 2D boolean array dp, where dp(i,j) represents whether the substring s(0...i+j-1) is an interleaving of the substrings x(0...i-1) and y(0...j-1). The base cases are dp(0,0) = True, because an empty string is an interleaving of two empty strings, and dp(0,j) = (y(0...j-1) == s(0...j-1)) and dp(i,0) = (x(0...i-1) == s(0...i-1)) for i > 0 and j > 0, because the only possible interleavings in these cases are x and y themselves.

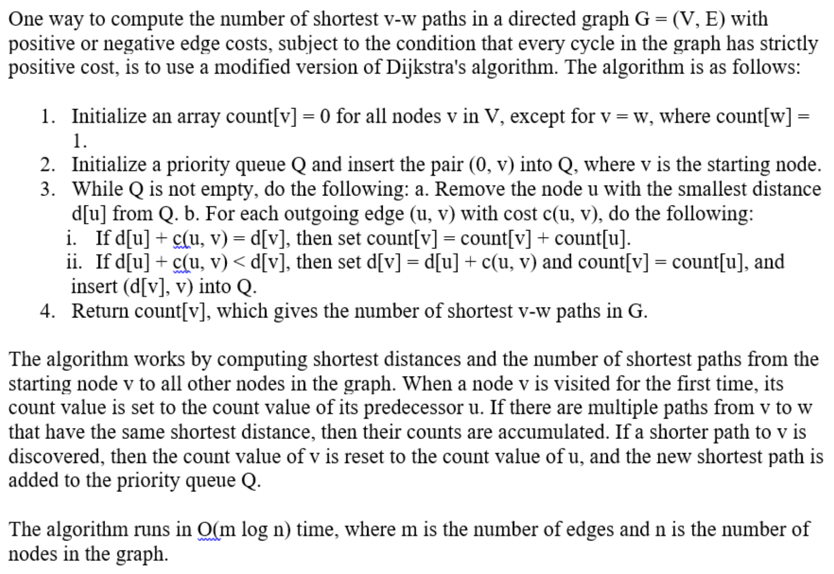
Equation checks if the current character in s matches the current character in either x or y. If it matches x, then we check if the substring s(0...i+j-2) is an interleaving of x(0...i-2) and y(0...j-1). If it matches y, then we check if the substring s(0...i+j-2) is an interleaving of x(0...i-1) and y(0...j-2). If either of these conditions is true, then dp(i,j) is set to True. The final answer is given by dp(m,n), where m and n are the lengths of x and y, respectively.

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.)



One way to compute the number of shortest v-w paths in a directed graph G = (V, E) with positive or negative edge costs, subject to the condition that every cycle in the graph has strictly positive cost, is to use a modified version of Dijkstra's algorithm. The algorithm is as follows:

1. Initialize an array count[v] = 0 for all nodes v in V, except for v = w, where count[w] = 1.
2. Initialize a priority queue Q and insert the pair (0, v) into Q, where v is the starting node.
3. While Q is not empty, do the following: a. Remove the node u with the smallest distance d[u] from Q. b. For each outgoing edge (u, v) with cost c(u, v), do the following:
   i. If d[u] + c(u, v) = d[v], then set count[v] = count[v] + count[u].
   ii. If d[u] + c(u, v) < d[v], then set d[v] = d[u] + c(u, v) and count[v] = count[u], and insert (d[v], v) into Q.
4. Return count[v], which gives the number of shortest v-w paths in G.

The algorithm works by computing shortest distances and the number of shortest paths from the starting node v to all other nodes in the graph. When a node v is visited for the first time, its count value is set to the count value of its predecessor u. If there are multiple paths from v to w that have the same shortest distance, then their counts are accumulated. If a shorter path to v is discovered, then the count value of v is reset to the count value of u, and the new shortest path is added to the priority queue Q.

The algorithm runs in O(m log n) time, where m is the number of edges and n is the number of nodes in the graph.

Solution:

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

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

Solution is made by items 2, 3, 4 with value 19 and capacity 6

Solution:

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
```