CS 577 # Assignment 7 – Dynamic Programming Fall 2021

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: _____

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.313 q.2).*

   Suppose you are managing a consulting team and each week you have to choose a job for them to undertake. The set of possible jobs is divided into 2 categories of jobs: low-stress jobs and high-stress jobs.

   For week i, if you choose the low-stress job, you get paid ℓ_i dollars and, if you choose the high-stress job, you get paid h_i dollars. The difference with a high-stress job is that you can only schedule a high-stress job in week i if you have no job scheduled in week $i-1$.

   Given a sequence of n weeks, determine the schedule of maximum profit. The input is two sequences: $L := \langle \ell_1, \ell_2, \ldots, \ell_n \rangle$ and $H := \langle h_1, h_2, \ldots, h_n \rangle$ containing the value of the low and high jobs for each week. For Week 1, assume that you are able to schedule a high-stress job.

   (a) Show that the following algorithm does not correctly solve this problem.

Algorithm: JobSequence

```
Input  : The low (L) and high (H) stress jobs.
Output: The jobs to schedule for the n weeks
for Each week i do
    if h_{i+1} > ℓ_i + ℓ_{i+1} then
        Output "Week i: no job"
        Output "Week i+1: high-stress job"
        Continue with week i+2
    else
        Output "Week i: low-stress job"
        Continue with week i+1
    end
end
```

Solution:
Counter-example:
$L := \langle 1, 1, 1 \rangle$
$H := \langle 1, 10, 100 \rangle$

For this instance, the algorithm JobSequence produces a schedule of: $\langle -, H, L \rangle$ with a value of 11, whereas the optimal schedule is $\langle L, -, H \rangle$ for a value of 101.

(b) Give an efficient algorithm that takes in the sequences L and H and outputs the value of the optimal solution. Prove that your algorithm is correct.

Solution:

Dynamic programming approach:

- A $(n+2)$-element array s, where $s[i]$ contains the optimal value over the first i weeks and the indices run from -1 to n.
- Bellman Equation:
$$s[i] = \max\{s[i-1] + \ell_i, s[i-2] + h_i\} \ ,$$
where $s[-1] = s[0] = 0$.
- The max valued schedule for the n weeks is found at $s[n]$.

Optimality Proof: By strong induction over the weeks i.

Base case 1: $i = -1$ or 0. Nothing to schedule so optimal value is 0.

Base case 2: $i = 1$. Optimal schedule is the max value of h_1 and l_1 which corresponds to the Bellman equation.

Inductive Step: When scheduling week i, we can either schedule (1) a low stress job and combine it with the best schedule for the $i-1$ weeks or (2) a high stress job combined with the $i-2$ weeks. The Bellman equation takes the max of those two options, and, hence, $s[i]$ is the optimal overall value given that $s[i-1]$ and $s[i-2]$ are optimal for the smaller problems which they are from the induction hypothesis.

2. *Kleinberg, Jon. Algorithm Design (p.315 q.4).*

Suppose you're running a small consulting company. You have clients in New York and clients in San Francisco. Each month you can be physically located in either New York or San Francisco, and the overall operating costs depend on the demands of your clients in a given month.

Given a sequence of n months, determine the work schedule that minimizes the operating costs, knowing that moving between locations from month i to month $i-1$ incurs a fixed moving cost of M. The input consists of two sequences N and S consisting of the operating costs when based in New York and San Francisco, respectively. For month 1, you can start in either city without a moving cost.

(a) Give an example of an instance where it is optimal to move at least 3 times. Explain where and why the optimal must move.

Solution:
$N := \langle 1, 2M, 1, 2M \rangle$
$S := \langle 2M, 1, 2M, 1 \rangle$
Initially, it is optimal to start in NY. On the next month, the cost to stay in NY is $2M$ while the cost to move to SF would be $M + 1$. Hence it is optimal to move. This reverse of this costing makes it optimal to move in the next month as well. We can create a sequence for one location of alternating 1 and $2M$ costs with the other location being alternating $2M$ and 1. These sequences can be arbitrarily long, requiring an arbitrary number of moves for the optimal algorithm. To force at least 3 move, we require sequences of length 4.

(b) Show that the following algorithm does not correctly solve this problem.

Algorithm: WorkLocSeq

```
Input  : The NY (N) and SF (S) operating costs.
Output: The locations to work the n months
for Each month i do
    if N_i < S_i then
        Output "Month i: NY"
    else
        Output "Month i: SF"
    end
end
```

Solution:
Counter-example:
$NY := \langle 1, 2 \rangle$
$SF := \langle 2, 1 \rangle$
$M := 100$
The above algorithm will start in NY and then move to SF for month 2. The overall cost being 102, whereas the optimal algorithm will arbitrarily pick either NY or SF and stay for both moths for an overall cost of 3.

(c) Give an efficient algorithm that takes in the sequences N and S and outputs the value of the optimal solution. Prove that your algorithm is correct.

Solution:

Dynamic programming approach:

- A $2 \times n$-element matrix s, where $s[\{1,2\}][i]$ contains the optimal value over the first i months and being in NY for month i if the first coordinate is 1 and SF if the first coordinate is 2. The second index runs from 1 to n.
- $s[1][1] = N_1$ and $s[2][1] = S_1$.
- Bellman Equations for $i = 2$ to n,
  - For NY in month i:

$$s[1][i] = N_i + \min\{s[1][i-1], s[2][i-1] + M\}$$

  - For SF in month i:

$$s[2][i] = S_i + \min\{s[1][i-1] + M, s[2][i-1]\}$$

- The minimum valued schedule for the n months is $\min_{j \in \{1,2\}} s[j][n]$.

Optimality Proof: By induction over the months i.

Base case: $i = 1$**.** The minimum when starting in NY is N_1 and the minimum of starting in SF is S_1 which corresponds to the definition of $s[1][1]$ and $s[2][1]$.

Inductive Step: WLOG consider the situation that we will be in NY for month i. When scheduling month i in NY, we should take the minimum of NY for month $i-1$ and the minimum of SF for month $i-1$ plus the move cost. The Bellman equation takes the minimum of those two options (accounting for the move cost with M), and, hence, produces the optimal overall value for $s[1][i]$ given that $s[1][i-1]$ and $s[2][i-1]$ are optimal for the smaller problems which they are from the induction hypothesis. The analogous argument holds for SF.

3. *Kleinberg, Jon. Algorithm Design (p.333, q.26).*

Consider the following inventory problem. You are running a company that sells trucks and predictions tell you the quantity of sales to expect over the next n months. Let d_i denote the number of sales you expect in month i. We'll assume that all sales happen at the beginning of the month, and trucks that are not sold are stored until the beginning of the next month. You can store at most s trucks, and it costs c to store a single truck for a month. You receive shipments of trucks by placing orders for them, and there is a fixed ordering fee k each time you place an order (regardless of the number of trucks you order). You start out with no trucks. The problem is to design an algorithm that decides how to place orders so that you satisfy all the demands $\{d_i\}$, and minimize the costs. In summary:

- There are two parts to the cost: (1) storage cost of c for every truck on hand; and (2) ordering fees of k for every order placed.
- In each month, you need enough trucks to satisfy the demand d_i, but the number left over after satisfying the demand for the month should not exceed the inventory limit s.

Give an algorithm that solves this problem in time that is polynomial in n and s. Prove that the algorithm is optimal.

Solution:
Dynamic programming approach:

- We work backwards from the last month, at each step deciding how many trucks to keep in storage for the next month, and using the lowest costs already calculated for future months.
- A 2D matrix m containing $(n \times (s+1))$-elements, where $m[i][j]$ contains the minimal value over months i through n given that we stored j trucks from month $i-1$ to i, and the indices are 2 to n for i and 0 to s for j.
- Initialize $m[n][j] = k + c \cdot j$ for all $j < d_n$, $m[n][j] = c \cdot j$ for $j = d_n$, and, in order to guarantee that in month n we do not have any extra trucks, $m[n][j] = \infty$ for all $j > d_n$.
- Define $f(i, j, j') = 0$ if $j \geq d_i + j'$ and 1 otherwise. That is, f is an indicator function, where i is the current month, j is the storage from month $i-1$ to i, and j' is the storage from month i to $i+1$. f is 1 if we need to order more trucks (if our current storage j is not enough to cover both the demand for next month, and the number of trucks we plan to store after next month).
- Bellman Equations for $i = n-1$ to 2 and $j = 0$ to s,

$$m[i][j] = c \cdot j + \min_{j' :\ \max\{0, j-d_i\} \leq j' \leq s} (k \cdot f(i, j, j') + m[i+1][j'])$$

- The minimum value for the n months is at $m[1][0]$, i.e., month 1 with no trucks stored from month 0 to 1.

Optimality Proof: By reverse induction over the months i, $m[i][j]$ is optimal given that j trucks are stored from month $i-1$ to month i.
Base case 1: $i = n$**.** The equation as defined above calculates the minimum cost for each $0 \leq j \leq s$.
Inductive Step: For each j for $m[i][j]$, by definition of the problem, the storage cost is $c \cdot j$. The minimizer portion of the Bellman equation considers all valid possibilities for the number of trucks j' to store for month $i+1$ given j. Notice that for a j larger than d_i, j' cannot be less than $j - d_i$. If j' is large enough to require an order, the cost k is included as determined by $f(i, j, j')$. The last part considered in the minimizer is $m[i+1][j']$ which is the minimal value for the parameters $i+1$ and j' from the induction hypothesis.
Running time: The matrix consists of $(n \cdot (s+1))$ and, for each cell, we consider $O(s)$ cells from the previous month. Overall, we have $O(n \cdot s^2)$

4. Implement the optimal algorithm for Weighted Interval Scheduling (for a definition of the problem, see the slides on Canvas) in either C, C++, C#, Java, or Python. Be efficient and implement it in $O(n^2)$ time, where n is the number of jobs. We saw this problem previously in HW3 Q2a, where we saw that there was no optimal greedy heuristic.

   The input will start with an positive integer, giving the number of instances that follow. For each instance, there will be a positive integer, giving the number of jobs. For each job, there will be a trio of positive integers i, j and k, where $i < j$, and i is the start time, j is the end time, and k is the weight.

   A sample input is the following:

```
2
1
1 4 5
3
1 2 1
3 4 2
2 6 4
```

   The sample input has two instances. The first instance has one job to schedule with a start time of 1 and an end time of 4. The second instance has 3 jobs.

   For each instance, your program should output the total weight of the intervals scheduled on a separate line. Each output line should be terminated by a newline. The correct output to the sample input would be:

```
5
5
```

   Note: In the third set of tests, some outputs will cause overflow on 32-bit signed integers.