CS 577 **Assignment 7 – 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

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 one of two jobs for your team to undertake. The two jobs available to you each week are a low-stress job and a high-stress job.

   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 (positive) 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:

Algorithm does not consider first week appropriately.

Example: $l: \langle 10, 10, 10, 10 \rangle$
$h: \langle 50, 50, 5, 1 \rangle$

For week 1, based on algo, week 1: no job,
week 2: $h_2 = 50$

However, ~~this~~ this results in final prft of
$\langle 0, 50, 10, 10 \rangle = 70$ when optimal should be
$\langle 50, 10, 10, 10 \rangle = 80$.

Thus, the proposed algorithm does not take into account possibility of high-stress job on the first week.

(b) Give an efficient algorithm that takes in the sequences L and H and outputs the greatest possible profit.

Solution:

~~2D Matrix, i rows, 2 colums (for l & h) v, v[i, c] is total profit at that week for choice c~~

~~v[1, c] = c (ie, first week initialize to profit of low/high stress~~

~~v[0, c] = 0 (ie initialize 0th week to 0 profit).~~

2 arrays: low & high
n: total num weeks
L[i] & H[i], profit from scheduling low/high stress on that week.

Bellman Equation

low[i] & high[i]: max profit up till week i with low on week i & high on week i resp.

For i=0, low[0]=high[0]=0. For i=1, low[1]=L[1] & high[1]=H[1]

For i>1, low[i] = L[i] + max(low[i-1], high[i-1])

high[i] = H[i] + max(low[i-2], high[i-2])

Solution value = max(low[n], high[n]).

(c) Prove that your algorithm in part (c) is correct.

Solution:

I will use strong induction.

~~Assuming up~~

Base case: not possible to earn any profit on 0th week so for

n=0, low[0]=high[0]=0 holds.

n=1: Both low/high stress jobs can be picked, so respectively, low[1]=L[1] & high[1]=H[1] would have the right values. soln = max(low[1], high[1]) then returns max profit.

~~IH~~ 0≤n≤k

Assuming ~~IH~~ holds, ie up till kth week, low[k] and high[k] has the right values which ~~is~~ is the max profit for picking low stress on week k and high stress ~~of~~ on week k respectively, soln holds as well.

Then for the (k+1)st week, low[k+1] would be L[k+1] + max of prev week (kth week) and high[k+1] would be max of ~~prev~~ 2 weeks ago (k-1)

Given my IH, both low[k+1] & high[k+1] values would hold.

Therefore, my solution would return the max(low[k+1], high[k+1]), ie the max profit attainable on (k+1)st week.

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:

$N : \langle 1, 10, 1, 10 \rangle$

$S : \langle 10, 1, 10, 1 \rangle$

$M = 1$

start at N, so sequence n for optimal work schedule would be

N S N S

$n = \langle 1, 1+1, 1+1, 1+1 \rangle$, total operating cost $= 7$, moved 3 times, $7 < 10$ if stayed in any city for >1 month.

(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: $N : \langle 1, 10, 1, 10 \rangle$ $M = 20$. let accumulated cost be c

$S : \langle 10, 1, 10, 1 \rangle$

Based on algorithm, month 1 will be N, ~~test~~ $c = 1$, since $N_2 > S_2$, month 2 will be S, $c = 1 + 1 + 20 = 22$, since $S_3 > N_3$, month 3 will be N, $c = 22 + 1 + 20$ and month 4 will be $c = 43 + 1 + 20 = 64$.

But optimal is staying in 1 city (either), for sequence N_1, N_2, N_3, N_4,

$c = 1 + 10 + 1 + 10 = 22$ and $22 < 64$, thus algorithm does not correctly solve problem.

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

Solution: 2 arrays: ny, sf M: moving cost

n: total num of months.

N[i], S[i], operating cost at NY & SF on week i.

ny[1] = N[1] & sf[1] = S[1].

Bellman Equ: $ny[i] = \min(ny[i-1], sf[i-1]+M) + N[i]$

$sf[i] = \min(sf[i-1], ny[i-1]+M) + S[i]$.

Solution = $\min(ny[n], sf[n])$.

(d) Prove that your algorithm in part (c) is correct.

Solution: Proof by strong induction.

Base case: n = 1, can start any location without incurring moving cost

so ny[1] = N[1], sf[1] = S[1] ✓

and soln = min(ny[1], sf[1]) returns min value of starting at either office.

IH: Assume holds for $1 \le n \le k$, ie solution returns optimal solution (min. operating cost) for k months.

Inductive step:

For $(k+1)^{st}$ week, we refer to values ~~ny[i-1]~~ ny[k] & sf[k] which would be optimal values by IH.

ny[k+1] & sf[k+1] considers ny[k] & sf[k] & moving cost if necessary and chooses min value, adding the operating cost for $(k+1)^{st}$ month as well N[k+1] and S[k+1]. Solution is then minimum of ny[k+1] & sf[k+1] which is a standard operation, thus $(k+1)^{st}$ week's solution holds, returning optimal solution.

Therefore, my algorithm in part (c) is correct.

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.

(a) Give a recursive algorithm that takes in s, c, k, and the sequence $\{d_i\}$, and outputs the minimum cost. (The algorithm does not need to be efficient.)

Solution:

```
min Cost
Input: s, c, k, di, currMonth.
Output: min Cost
if currMonth >= length(di): return 0.
orderCost = k + min Cost (s, c, k, di, currMonth + 1)
for i ~~from~~ currMonth to length(di) - 1:
    storage cost = c × No. of trucksinStock
    remainingMonthsCost = min(s, c, k, di, i+1)
    total Cost = storage Cost + remainingMonthsCost
    if totalCost < order Cost:
        orderCost = total Cost.

return order Cost
```

(b) Give an algorithm in time that is polynomial in n and s for the same problem.

Solution:

Bellman eqn:

cost(i, j) represent min. cost to satisfy d[1], d[2], ..., d[i].
with inventory of j trucks at beginning of month i.

$$\text{cost}(i,j) = \min\{\text{cost}(i-1,j) + cj, \quad \text{\# Do not order trucks.}$$
$$\text{cost}(i-1,\cancel{k}\,l) + k + lc + \text{cost}(i,j) \quad \text{\# Order trucks}$$
$$\text{for } l \text{ from } 1 \text{ to } \min(d[i], s-l)\}$$

Initialize cost(0,0) to be k since need to place initial order for trucks.

Solution with minimum output is ~~Ott~~ min(cost(i,j)) for $0 \leq j \leq s$.

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

Solution:

Proof by Induction.

Base case: cost(0,0).

Since we are starting with 0 inventory, it is mandatory to place order to fulfil d[1]. Thus, cost(0,0) = k.

IH: Assume cost(i,j) is correct for i^{th} month, for j trucks where $0 \leq j \leq S$.

i.e we have min cost value in each cell for month i consider all possible inventory scenario.

Then for $(i+1)^{st}$ month, if we do not place order, for $0 \leq j \leq S$

cost(i+1, j) ~~= cost(i,j) + cj~~ can be 1 of 2 values.

1. Do not place order, but need storage of trucks

   so cost(i+1, j) = cost(i,j) + cj.

2. Place order, also need storage of trucks.

   so cost(i+1, j) = cost(i,j) + cj + k.

   If order, how many trucks to order: $1 \leq n \leq S-k$.

   consider all possibilities & how it affects cost of next month.

We then find minimum of 1. and 2. since goal is to output minimum cost.

4. Alice and Bob are playing another coin game. This time, there are three stacks of n coins: A, B, C. Starting with Alice, each player takes turns taking a coin from the top of a stack – they may choose any nonempty stack, but they must only take the top coin in that stack. The coins have different values. From bottom to top, the coins in stack A have values $a_1, \ldots, a_n$. Similarly, the coins in stack B have values $b_1, \ldots, b_n$, and the coins in stack C have values $c_1, \ldots, c_n$. Both players try to play optimally in order to maximize the total value of their coins.

   (a) Give an algorithm that takes the sequences $a_1, \ldots, a_n$, $b_1, \ldots, b_n$, $c_1, \ldots, c_n$, and outputs the maximum total value of coins that Alice can take. The runtime should be polynomial in n.

Solution:

2 players: Assume Alize & Bob play optimally.

$M[a, b, c]$ is max possible value for Alize when she chooses from $A[a], B[b], C[c]$, assuming Bob plays optimally.

Alize k^{th} turn: she will maximize her value

$$= \max\{A[a] + \text{BobOpt}(A[a-1], B[b], C[c]),\ B[b] + \text{BobOpt}(A[a], B[b-1], C[c]),\ C[c] + \text{BobOpt}(A[a], B[b], C[c-1])\}.$$

$$\text{BobOpt}(A[a], B[b], C[c]) = \min\{\text{AlizeOpt}(A[a-1], B[b], C[c]),\ \text{AlizeOpt}(A[a], B[b-1], C[c]),\ \text{AlizeOpt}(A[a], B[b], C[c-1])\}.$$

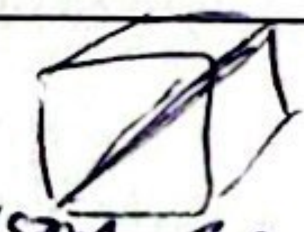
So Bellman Eqn is:

$$M[a,b,c] = \max\{A[a] + \min\{M[a-2,b,c], M[a-1,b-1,c], M[a-1,b,c-1]\},\ B[b] + \min\{M[a-1,b-1,c], M[a,b-2,c], M[a,b-1,c-1]\},\ C[c] + \min\{M[a-1,b,c-1], M[a,b-1,c-1], M[a,b,c-2]\}\}$$

Solution at $M[a_n, b_n, c_n]$.

   (b) Prove the correctness of your algorithm in part (a).

Solution:



Prove runtime, 3D array, we fill $\frac{1}{2}$ in each dimension, so $O(n^3)$.

Proof by induction, Base case, $n=0$ trivial

$n=1$, 3 coins, Alize picks max/least val coin.

& Bob takes middle value.

IH: every scenario played out where $n=k$, ie Alize picks moves that maximizes her value.

Inductive step: For every scenario played out up till k, we know that $k+1$st move relies on $k-1$ or $k-2$ steps back based on Bellman Eqn, which falls under the IH, therefore every $(k+1)$st step played out would maximize Alize's value as well, and we can then find the answer in $M[a_n, b_n, c_n]$.