CS 577 # Assignment 7 – Dynamic Programming Fall 2022

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: Zinnia Nie Wisc id: 908 319 4044

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:

This solution does not take into account the fact that a high stress job can be scheduled on week 1. Thus, the optimal schedule will not be output for sequences $L = \langle 1, 2, 3 \rangle$ $H = \langle 10, 4, 4 \rangle$.

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

Solution: Dichotomy: high stress or low stress job

Matrix: M[i] is a 1D array with length n+1
- i is the week number

Bellman: $M[i] = \max\{M[i-1] + l_i, M[i-2] + h_i\}$

Populate: zeroth week $M[0] = 0$
for the first week $M[1] = \max\{l_1, h_1\}$
Then fill from lower indices to higher.

Solution: at M[n]

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

Solution: Proof by Induction

Base Case: $M[0] = 0$, 0 profit is earned by working for 0 weeks

$M[1] = \max\{l_1, h_1\}$, in the first week there are no restraints so we simply want to maximize profit

Inductive Step: For all $k < i$ where $i \geq 2$, assume the matrix is correctly filled

In week i, if we want to pick a low stress job, then we are allowed to work the week before, so we need the optimal for weeks 1 to i-1, which will be at M[i-1] in the matrix.
If a high stress is chosen, then we can't work week i-1. The value at M[i-2] will be the optimal profit without week i-1.
These are the only choices, so we just need the max between the two.

Termination: Each entry M[i] needs the two entries before it and there is n+1 entries to fill. Thus the program will finish once M[n] is reached.
Finally M[n] will be where max profit for a series of jobs from week 1 to n will be and that is the solution.

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: $M = 2$

$N = \langle 1, 4, 1, 4 \rangle$

$S = \langle 4, 1, 4, 1 \rangle$

In this instance, you start in NY with cost 1. In month 2, moving to SF with cost 1+2 is less than staying in NY with cost 4. In month 3 and 4, it is the same situation where move cost + cost in other city is cheaper than the cost of staying.

(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:

This algorithm does not take into account the moving cost.

Thus a sequence like $N = \langle 4, 2, 3 \rangle$

$S = \langle 2, 2, 5 \rangle$

with $M = 10$ would not get the optimal solution with the above algorithm.

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

Solution: Dichotomy: operate in N or S

Matrix: $X[i,2]$, where i is the month and the 2 is either N or S depending on start

Bellman: $X[i,N] = N_i + \min\{X[i-1,N], X[i-1,S]+M\}$

$X[i,S] = S_i + \min\{X[i-1,S], X[i-1,N]+M\}$

Populate: $X[1,N]=N_1$, $X[1,S]=S_1$

Then populate from lower to higher indices

Solution: $\min\{X[n,N], X[n,S]\}$ is the schedule of lowest cost.

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

Solution: Proof by Induction:

Base case: $X[1,N]$, the first month for the schedule starting at NY can only have cost of N_1,

$X[1,S]$, similarly for schedule starting at SF

Inductive step: For $k<i$ where $i>1$, the matrix indices $X[i,N]$ and $X[i,S]$ have the optimal costs for starting in each city.

At each month, the only options are staying an incuring cost N_i or S_i or moving and incuring opposite cost + M moving cost. Since the previous month $i-1$ has the correct schedule cost ending in either N or S, the optimal cost will be the lower cost between either staying or moving + penalty. Our Bellman correctly calculates this value.

Termination: Bellman uses the previous value by 1 month to calculate next month so if there is a limited number of months the matrix and program will be finite,

Solution, we want to be minimized, so since we now have a schedule that differs by starting city, the optimal will be the lower cost.

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:

```
Truck Demand:
    if len(d)=1: return k+c
    if d1<s:
        return k + TruckDemand(s, (s-d1) c, k, {di+1})
    else if d1>s:
        return k + c + TruckDemand(s, 0, k, {di+1})
```

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

Solution:

Matrix: M[i,j] is a nxs matrix where n is the month and s is the # of trucks leftover

Bellman: if $j+d_i > s$, $M[i,j] = M[i-1,0] + k$ ←

else, $M[i,j] = \min(M[i-1, j+d_i] + c(j+d_i), M[i-1,0] + k)$

Populate: Fill M[0,0] with the value 0 and then fill by rows.

Solution: M[n,0]

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

Solution:

Base Case: when there are 0 cars in storage, in the 0th month there is no cost

Recursive: For $k<i, h<j$ and $i,j>0$, the Matrix $M[k,h]$ has the correct optimal value

when the trucks stored + trucks demanded is greater than the storage limit, no trucks are left behind and a new order has to be made. So we incure a cost of k and also the cost of the previous month with no stored cars, since we would have to reorder anyways, so no stored cars is the minimum cost. Thus the previous value is $M[i-1,0]$, with no stored cars and the constant order placement fee k.

When trucks stored + demand is less than storage limit, there is a chance storing trucks would actually cost less than reordering, so we look at the cost of the previous month with enough stored trucks and compare with the cost of reordering and choose the minimum. Previous month with enough is $M[i-1, j+d_i]$ which has to be added with storage cost for all cars $c(j+d_i)$, and this is compared with the above calculation for no stored cars and placing order to determine minimum.

Termination:

The optimal solution results when all months pass and we have no trucks leftover because leftover trucks would add storage cost. This is at $M[n,0]$

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:

Matrix: M[i,j,k] is a 3D nxnxn matrix
Best score with i coins picked from A, j from B, k from C

Bellman: $M[i,j,k] = \max\{ a[i] + \min\{M[i+2,j,k], M[i+1,j+1,k], M[i+1,j,k+1]\},$
$b[j] + \min\{M[i+1,j+1,k], M[i,j+2,k], M[i,j+1,k+1]\},$
$c[k] + \min\{M[i+1,j,k+1], M[i,j+1,k+1], M[i,j,k+2]\}\}$

Populate: M[n,n,n] = 0
Then populate diagonal and continue from there until M[0,0,0]

Solution: M[0,0,0]

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

Solution:

Base Case: when n,n,n coins are picked from each stack, then all coins are gone and the best value is 0.

Inductive Step: For k>i the Matrix has the optimal value

Bob wants to minimize Alice's value, so he will pick a coin from the stack that does that. Alice thus wants to maximize her value while Bob picks a coin to minimize her value. If Bob picks from same stack, then the stored value is +2 for that stack, if he picks a different stack then it is +1 for the stack Alice chose and the stack Bob chooses.

Terminate: matrix population stops when reaching M[0,0,0] since all input is gone.
The solution is at M[0,0,0] where no coins have yet been picked.

5. 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, Python, or Rust. 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, an end time of 4, and a weight of 5. The second instance has 3 jobs.

   The objective of the problem is to determine a schedule of non-overlapping intervals with maximum weight and to return this maximum weight. 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 exactly one newline. The correct output to the sample input would be:

```
5
5
```

   or, written with more explicit whitespace,

```
"5\n5\n"
```

   Notes:

   - Endpoints are exclusive, so it is okay to include a job ending at time t and a job starting at time t in the same schedule.
   - In the third set of tests, some outputs will cause overflow on 32-bit signed integers.