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: _____ 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 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 $h_{i+1} > \ell_i + \ell_{i+1}$ **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: Algorithm enable taking the high -stressing job too early.

	w1	w2	w3	w4	
l	1	2	2	1	optimal solution is l1+h3+l4=12
h	2	4	10	4	while by this algorithm we will obtain solution h2+h4=8

example 1
i=1 h2>l1+l2 yes => week1 no job week2 h2 =4 new week i=1+2=3
i=3 h4>l3+l4 yes => week3 no job week4 h4 =4 i=3+2=5 **stop**

by algorithm h2+l3=6
optimal h1+l2+l3=25

example 2

	w1	w2	w3	
l	1	2	3	h2> l1+l2
h	20	5	4	week1 no job, week 2 h2=5 new week 1+2=3
				week 3 l2 +l3 > h4 => week3=l3 i=3+1=4
				week 4 no data

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

Solution:

We denote with OPT(i) maximal value achieved for weeks from 1 to i. We would like to have value in week i, either low or high–stress jobs. If it select low-stress job it will be optimal up to week i-1, followed by this job while if it select high-stress job at i it will be optimal up to i-2 followed by this job.

$OPT(i)=\max(l_i + OPT(i-1), h_i + OPT(i-2))$, and **$OPT(1)=\max(l_1, h_1)$** and **i=1,…,n,** for total value for n weeks O(n).
Actual values for each i can be traced through OPT(i).

Example

	w1	w2	w3	w4	by algorithm h2+h4=60
l	10	1	4	5	optimal h2+h4=60
h	20	50	10	10	

by alg OPT(1)=max(l1,h1)=h1=20
OPT(2)=max(1+20, 50+0)=h2 =>50
OPT(3)=max(4+50, 10+20)=l3 =>54
OPT(4)=max(5+54, 10+50)=h4=60

Backtrack if OPT(n) is achieved by h_n trace n-2 back position, if OPT(n) is achieved by l_n trace n-1 back position

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

Solution:

Strong induction

BC: n=1 . There is only 1 week, the greatest value will wins
OPT(1)=max(l_1, h_1)

Let OPT(k) by inductive hypothesis is max revenue for working weeks 1 to k.
We have to show that Bellman function choses optimal value for k+1.
If OPT(k+1) is calculated as max on l_k part of equation, than by inductive hypothesis it is calculated on the base of the optimal values in previous k weeks, OPT(k).
If OPT(k+1) is calculated as max on h_k part of equation, than by inductive hypothesis it is calculated on the base of optimal values in previous k-1 weeks, OPT(k-1).
Final value is maximal value taking into account both cases.
$OPT(k+1)=\max(l_{k+1} + OPT(k), h_{k+1} + OPT(k-1))$

Q2

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:

Moving costs=3

	month 1	month 2	month 3	month 4	month 5	all
NY	2	10	2	10	5	29
SF	10	2	10	2	5	29
optimal	2	2	2	2	5	13+3*3=22

Optimal : {NY,SF,NY,SF,SF} =22 staying at one location 29 (NY or SF)

(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:

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 Ni < Si then
    Output "Month i: NY"
  else
    Output "Month i: SF"
  end
end

Algorithm do not correctly calculate the cost. It chose smaller value for each month, neglecting the moving costs.

Moving costs M=6

	month 1	month 2	month 3	month 4	month 5	all
NY	2	10	2	10	5	29
SF	10	2	10	2	5	29
optimal	2	2	2	2	5	13+3*6=31

Optimal solution is staying at one town {NY,NY, NY,NY,NY} =29 , or {SF, SF, SF, SF, SF}=29 , comparing with moving at cheaper location together with moving costs (31). Algorithm do not include moving costs.

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

Solution:

We starts in the town with less maintaining cost $OPT_N(1)= OPT_S(1)=\min(N1,S1)$
(the same if we define $OPT_N(0)= OPT_S(0)=0$)

Bellman Equation for i=2,….,n

$OPT_N(i) = Ni+\min(OPT_N(i-1), M+OPT_S(i-1))$
$OPT_S(i) = Si+\min(OPT_S(i-1), M+OPT_N(i-1))$

Ni, Si are maintaining cost in NY and SF for i^{th} month. M is moving cost and $OPT_N(i)$ and $OPT_S(i)$ are related functions that calculate the minimal cost of i^{th} month that finish in NY or SF. Going backward we can determine locations and direction of moving. The cost of operating in month i is max ($OPT_N(i)$, $OPT_S(i)$).

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

Solution:

Strong induction.
BC. Optimal value in the first month is $OPT_N(1)= OPT_S(1)=\min(N_1,S_1)$ There is no moving costs, so we start with town with smallest maintaining costs.

Let OPT(k) by inductive hypothesis is minimal cost of maintaining the offices in NY and SF in months 1 to k. We have to show that algorithm with given Bellman function picks the location with minimal cost in $(k+1)^{th}$ month.
If OPT(k+1) is chosen to be calculated on the base of the cost of operating in the same city as in the previous step it is because the cost of moving and operating elsewhere is greater than the cost of staying in the same city.
If OPT(k+1) is chosen to be calculated on the base of the cost of operating in the other city as in the previous step it is because the cost of moving and operating elsewhere is less than the cost of staying in the same city.

Final value for the month k is minimal value taking into account both cases
$OPT(k)=\min(OPT_N(k), OPT_S(k))$

Example
Moving costs M=6

	month 1	month 2	month 3	month 4	month 5	all
NY	2	10	2	10	5	29
SF	10	2	10	2	5	29
optimal	2	2	2	2	5	13+3*6=31

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:

```
# {d1,....,dn} sales in months i=1,..,n, known given in array d
# si  stored tracks with cost  c x si  and si<=s
# order cost k  regardless of number of trucks ordered

  Algorithm: minCost(s, c, k, d):
  if n == 0;
    return 0
  else
# Case 1: Order just enough trucks to satisfy the demand for the current month
cost1 = minCost(s - di, c, k, d) + max(0, di - s) * c;
# Case 2: Order more trucks than needed to satisfy the demand for the current month
cost2 = float('inf');
for i in range(di);
    cost2 = min(cost2, k + minCost(s + i - di, c, k, d) + (i * c))
return min(cost1, cost2)
```

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

Solution:

OPT(i,s) 2D matrix, denote value of optimal solution for the end of month i with the inventory of s trucks left over that month. Dimension is nxs.
On the end of month n we want inventory to be empty, OPT(n,0). The number of sub-problems is n(S+1) having possible 0, 1....S left over tracks. S is capacity of inventory.

Bellman equation

if $s+d_i > S$ OPT(i,s) =OPT(i-1,0)+K //leftover plus demand greater than capacity, order is needed
if $s+d_i <= S$ OPT(i,s) =min (OPT(i-1,s+ d_i)+C(s+di), OPT(i-1,0)+K) // leftover plus demand smaller of capacity placing order or keeping s inventory

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

Solution:

BC: If i=1 month d1 trucks are ordered with cost K

Let OPT(i,s) denote optimal solution for the month i and amount of trucks (leftover) on inventory with capacity S. If s+di leftover plus demand for the month i are greater than inventory capacity and we want on the end of last month to have 0 leftover than order with cost K is placed and OPT(i,s) =OPT(i-1,0)+K

If leftover and demand for i^{th} month are smaller than inventory capacity but must cover demand, lower cost between paying inventory or placing and order for that month i is chosen.

Algorithm will stop in finite steps working for finite time of month.

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:

Let A[i], B[j] and C[k] be the stacks of coins. The top of each stack begins with index n so each draw from stack will count the possible drowns from i, j, k down from n, n-1, ...1. The stack A[i] will be empty after last draw i=1. In algorithm notation this means that if we come to position i in stack A[i] in one step we start from position i+1 in stack or position i+2 in 2 steps.

For Alice turn she can chose from either stack A[i], B[j], C[k] top coin. We assuming that Bob will play rational on the drowns so she will draw

Max { A[i]+ BobOPT(A[i+1], B[j],C[k]), B[j]+BobOPT(A[i], B[j+1],C[k]), BobOPT(A[i], B[j],C[k+1])

Bob will try to play so to minimize Alice next draw

BobOPY(A[i],B[j],C[k])= min {AliceOPT(A[i+1],B[j],C[k]), AliceOPT(A[i],B[j+1],C[k]), AliceOPT(A[i],B[j],C[k+1])}

Belman equation can be defined as matrix M[n,n,n] so algorithm is with complexity $O(n^3)$

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,j+2,k], M[i+1,j+1,k], M[i, j+1 k+1] },
C[k]+min{(M[i,j,k+2], M[i+1,j,k+1], M[i, j+1 k+1] }}

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

Solution:

. Strong induction

BC: If all the stacks are empty n=0, both players will collect 0 value coins.
If we have only one row n=1, Alice is in advantage, due odd number of coins.
(we can put restriction n even, to avoid this advantage.)

Let M[i,j,k] be maximal value, on all the stages up to i, j, k positions in the coin stacks. We have to prove that the next choice is optimal.
By Bellman equation Alice will maximise her choice taking into account the Bob's maximal possible choices, while Bob optimal choice is too minimise Alis next choice.
Algorithm will stop in finite steps having all over 3n possible choices

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.