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: Carson Miller Wisc id: 9081473028

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
```

L = {5, 5, 5, 5}
H = {60, 9, 11, 9}

The maximum profit for this case is 76, scheduling the high-stress jobs worth 60 and 11 and the last low-stress job worth 5. JobSequence doesn't consider that you can schedule a high-stress job for Week 1, throwing off the schedule from the start. JobSequence would schedule the low-stress job for Week 1, the high-stress job for Week 3, and the low-stress job for Week 4, giving a total profit of 21, which is less than 76.

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

The DP solution yields a 1D matrix M, where M[j] holds the maximum profit from week 1 to week j. Initialize M[0] = 0 (starting profit of zero before any jobs can be scheduled) and M[1] = max(l_1, h_1).

Dichotomy: Choose low-stress job or choose high-stress job and have no job previous week

Bellman Equation: M[j] = max(M[j-1] + l_j, M[j-2] + h_j)

Populate the array M from index 1 to n and the solution will be stored at M[n].

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

For a given cell, the value being stored either adds the value of a low-stress job to the value from the previous week or adds the value of the high-stress job to the value from 2 weeks ago, effectively scheduling no job the week prior to the high-stress job. For weeks 1 to j, M[j] holds the maximum profit.

It takes constant time to populate each cell and there are n cells, so the runtime of this algorithm is O(n)

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.

M = 1

N = {2, 7, 2, 7}
S = {4, 1, 4, 1}

For this example, you would start in New York in month 1, move to SF in month 2, back to NY in month 3, and back to SF for month 4. This yields an operating cost of 9, which is optimal for this example. Staying in a location rather than moving at any month would incur greater operating cost than 9.

(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
```

This algorithm does not consider the fixed moving cost at all. Here's a counterexample:
M = 100
N = {5, 8, 4, 3}
S = {2, 9, 1, 4}

WorkLocSeq would suggest to move every month, as the lower operating cost alternates between the two locations each month. It would give a minimum operating cost of 314, while the optimal solution would suggest staying in SF the whole time at an operating cost of 16.

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

The DP solution yields a 2D matrix T, where T[i, j] holds the minimum operating cost for being in location i (0 for SF and 1 for NY) in month j. Initialize T[0][1] to S_1 and T[1][1] to N_1.

Dichotomy: moving or not moving

Bellman Equations: T[0, j] = min(T[0][j-1] + S_j, T[1][j-1] + S_j + T)
T[1, j] = min(T[1][j-1] + N_j, T[0][j-1] + N_j + T)

Solution is the min(T[0][j], T[1][j]), populate from 1 to j in both columns at the same time.

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

Base Case:
T[0][1] = S_1 and T[1][1] = N_1,
This is true because we can start in either city without paying the moving cost M.

Inductive Step:
If we move from one location to the other between months, we must pay the moving cost M, as well as the operating cost of the location we are moving to for that month. This follows that we would populate the array as the months increase, taking into account the total cost to be in either location for a given month with respect to where we were the previous month.

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.)

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

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

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.

3D array M, where M[i, j, k) is the maximum possible value for Alice when choosing from $a_1,\ldots,a_j$, $b_1,\ldots,b_j$, $c_1,\ldots,c_k$.

Trichotomy: pick from A, B, or C

Bellman Equation:

$$M[i, j, k) = \max\{a_i + \min\{M[i-2,j,k], M[i-1,j-1,k], M[i-1,j,k-1]\},$$
$$b_j + \min\{M[i-1,j-1,k], M[i,j-2,k], M[i-1,j,k-1]\},$$
$$c_k + \min\{M[i-1,j,k-1], M[i,j-1,k-1], M[i,j,k-2]\}\}$$

M[1,1,1] = max(a1,b1,c1)
Assume any value that "doesn't exit" (index out of bounds) is 0. This represents an empty stack.
Solution: M[n,n,n], populate from 1 to I, from 1 to j, and from 1 to k

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

To prove correctness, we will do strong induction on cell population order.

Base Case:

i=1, j=1, k=1. If there is one coin in each stack, take the coin with the maximum value.

Inductive Step:

Assume the recursive call to M within the Bellman equation correctly returns the maximum possible value at that position, this algorithm considers all possible choices of the trichotomy. Each line of the max function considers the possibilities of Alice choosing between stacks A, B, and C and then the corresponding stacks if Bob were to choose from stacks A, B, and C. In this algorithm, we assume that Bob plays optimally.

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.