Homework 6 Solutions

1. Given a non-empty string s and a dictionary containing a list of unique words, design a dynamic programming algorithm to determine if s can be segmented into a space-separated sequence of one or more dictionary words. If *s* = "*algorithmdesign*" and your dictionary contains "*algorithm*" and "*design*". Your algorithm should answer Yes as *s* can be segmented as "*algorithm design*". (8 points)

   Solution: Let $s_{i,k}$ denote the substring $s_i s_{i+1} \ldots s_k$. Let *Opt*(*k*) denote whether the substring $s_{1,k}$ can be segmented using the words in the dictionary, namely $OPT(k) = 1$ if the segmentation is possible and 0 otherwise. A segmentation of this substring $s_{1,k}$ is possible if only the last word (say $s_i \ldots s_k$) is in the dictionary the remaining substring $s_{1,i}$ can be segmented. Therefore, we have equation:

   $$OPT(k) = OPT(i - 1)$$

   Rubrics:
   Correct recurrence relation: For every k, checking if OPT(i-1) is true and substring ($s_i \ldots s_k$) is part of dictionary for all 0<=i<=k - 3 points
   Correct implementation - 4 points
   Handling edge case where i-1<0 - 1 points

2. Given n balloons, indexed from 0 to $n - 1$. Each balloon is painted with a number on it represented by array *nums*. You are asked to burst all the balloons. If you burst balloon *i* you will get *nums[left]* $*$ *nums[i]* $*$ *nums[right]* coins. Here *left* and *right* are adjacent indices of *i*. After bursting the balloon, the left and right then become adjacent. You may assume *nums[−1]* = *nums[n]* = *1* and they are not real therefore you can not burst them. Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm. (8 + 4 points)
   Here is an example. If you have the *nums* arrays equals [3, 1, 5, 8]. The optimal solution would be 167, where you burst balloons in the order of 1, 5, 3 and 8. The left balloons after each step is:
   [3, 1, 5, 8] → [3, 5, 8] → [3, 8] → [8] → []
   And the coins you get equals:
   $(3 * 1 * 5) + (3 * 5 * 8) + (1 * 3 * 8) + (1 * 8 * 1) = 167$

   Solution: Let *OPT*(*l, r*) be the maximum number of coins you can obtain from balloons *l, l*+1, . . . , *r*− 1, *r*. The key observation is that to obtain the optimal number of coins for a balloon from *l* to *r*, we choose which balloon is the last one to burst. Assume that balloon *k* is the last one you burst, then you must first burst all balloons from *l* to $k - 1$ and all the balloons from $k + 1$ to *r* which are two sub problems. Therefore, we have the following recurrence relation:

   $$OPT(l, r) = \{OPT(l, k - 1) + OPT(k + 1, r) + nums[k] * nums[l - 1] * nums[r + 1]\}$$

We have the initial condition $OPT(l, r) = 0$ if $r < l$. For running time analysis, we in total have $O(n^2)$ and computation of each state takes $O(n)$ time. Therefore, the total time is $O(n^3)$.

Rubrics:
Correct recurrence relation:

- Correctly identified the subproblems - 2 points
- Correct expression to obtain coins after bursting last balloon k (nums[k]*nums[l-1]*nums[r+1]) - 2 points
- correct OPT(l,r) with correct ranges for l and r that is l<=r. - 4 points

Correct initial condition that is OPT(l,r)=0 when l>r - 2 points
Correct time complexity with explanation - 2points

3) You are in Downtown of a city where all the streets are one-way streets. At any point, you may go right one block, down one block, or diagonally down and right one block. However, at each city block (i, j) you have to pay the entrance fees fee(i, j). The fees are arranged on a grid as shown below:

	0	1	2	3	⋯	n
0	$fee_{(0,0)}$	$fee_{(0,1)}$	$fee_{(0,2)}$	$fee_{(0,3)}$	⋯	$fee_{(0,n)}$
1	$fee_{(1,0)}$	$fee_{(1,1)}$	$fee_{(1,2)}$	$fee_{(1,3)}$	⋯	$fee_{(1,n)}$
2	$fee_{(2,0)}$	$fee_{(2,1)}$	$fee_{(2,2)}$	$fee_{(2,3)}$	⋯	$fee_{(2,n)}$
3	$fee_{(3,0)}$	$fee_{(3,1)}$	$fee_{(3,2)}$	$fee_{(3,3)}$	⋯	$fee_{(3,n)}$
⋮	⋮	⋮	⋮	⋮	⋱	⋮
n	$fee_{(n,0)}$	$fee_{(n,1)}$	$fee_{(n,2)}$	$fee_{(n,3)}$	⋯	$fee_{(n,n)}$

Your objective is to travel from the starting point at the city's entrance, located at block (0,0), to a specific destination block (n,n). The city is laid out in a grid, and at each intersection or block (i, j), you might either incur a cost (pay an entrance fee) or receive a reward (get a payback) for passing through. These transactions are captured in a grid, with positive values representing fees and negative values representing paybacks.

You would like to get to your destination with the least possible cost.

Formulate the solution to this problem using dynamic programming.
a) Define (in plain English) subproblems to be solved. (4 points)
b) Write the recurrence relation for subproblems. (6 points)

Solution:
a) Let OPT(i, j) be the minimum cost to get from (0,0) to (i, j).
b)

OPT (i, j) = min (OPT (i − 1, j) + fee(i, j), OPT (i, j − 1) + fee(i, j),OPT (i − 1, j − 1) + fee(i, j))
OPT(-1, j) = OPT(i, -1) = BIG_CONSTANT
where $1 \leq i \leq n$ and $1 \leq j \leq n$
OPT(-1, -1) = OPT(0, -1) = OPT(-1, 0) = 0

Rubrics:
Correct subproblems definition using 2D dynamic programming - 4 points
Handling edge/base cases where i-1<0, j-1<0 - 2 points
Correct recurrence relation involving all 3 directions(down,diagonally down and right)- 2 points each.

4) Assume we have N workers. Each worker is assigned to work at one of M factories. For each of the M factories, they will produce a different profit depending on how many workers are assigned to that factory. We will denote the profits of factory i with j workers by P(i,j). Develop a dynamic programming solution to find the assignment of workers to factories that produces the maximum profit.(Mention the pseudocode). (8 points)

Solution:
B(m,j) denotes the maximum profit obtained after assigning j workers to first m factories.
Path(m, j) records the number of workers assigned to factory m when considering the best profit for j workers.

```
B(0,0) = 0
For m = 1, 2, ..., M
        For j = 1, 2, ..., N
                For k = 0 to j // Distribute workers among factories
                    If B(m-1, j-k) + P(m, k) > B(m, j):
                        B(m, j) = B(m-1, j-k) + P(m, k)
                         Path(m, j) = k // Store the number of workers assigned to factory m
                     EndIf
                EndFor

        Endfor
Endfor
//Initialize an empty list or array `Assignment` of size M to store the number of workers assigned
//to each factory.

j = N // Start with the total number of workers
For m = M to 1
   Assignment[m] = Path(m, j)
   j = j - Path(m, j) // Update the remaining number of workers for the next iteration
EndFor
```

Rubrics:
correct base case B(0,0)=0 - 1 points

correct recurrence relation - 2 points
correctly identifying max profit - 3 points
correctly identifying assignment of workers to factories - 2 points

5) You have two rooms to rent out. There are n customers interested in renting the rooms. The ith customer wishes to rent one room (either room you have) for d[i] days and is willing to pay bid[i] for the entire stay. Customer requests are non-negotiable in that they would not be willing to rent for a shorter or longer duration. Devise a dynamic programming algorithm to determine the maximum profit that you can make from the customers over a period of D days.

a) Define (in plain English) subproblems to be solved. (4 points)
b) Write the recurrence relation for subproblems. (6 points)

Solution:

a) Let OPT(d1, d2, i) be the maximum profit obtainable with d1 remaining days for room 1 and d2 remaining days for room 2 using the first i customers.

b) OPT(d1, d2, i) = max(bid[i] + OPT (d1 − d[i], d2, i − 1),
bid[i] + OPT (d1, d2 − d[i], i − 1),
OPT (d1, d2, i − 1))
Initial conditions
OPT(d1,d2,0) = 0
OPT(d1,d2,i) = -∞ if d1 < d[i] and d2 < d[i]

Rubrics:

a) correct subproblems - 4 pts
b) correct recurrence relations - 4 pts
c) correct base cases - 2 points

6) You are given a sequence of n numbers (positive or negative): $x_1, x_2, \ldots, x_n$ .Your job is to select a subset of these numbers of maximum total sum, subject to the constraint that you can't select two elements that are adjacent (that is, if you pick x_i then you cannot pick either x_{i-1} or x_i). On the boundaries, if x_1 is chosen, x_2 cannot be chosen; if x_n is chosen, then x_{n-1} cannot be chosen. Give a dynamic programming solution to find, in time polynomial in n, the subset of maximum total sum. (8 points)

Solution: OPT[-1, 0, 1, ..., n] := [0, ..., 0]
for i = 1, ..., n:
OPT[i] := max(OPT[i-2]+x_i, OPT[i-1])
S := [], j := n
while j > 0:

```
    if OPT[j] = OPT[j-2] + x_j:
        S.append(x_j)
        j := j-2
    else:
        j := j-1
return S
```

Rubrics:

Correctly handling edge cases when i-2<0 - 2 points

Correct recurrence relation - 3 points

Correct implementation - 3 points

7) Suppose you have a rod of length N, and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length i is worth p_i dollars. Devise a Dynamic Programming algorithm to determine the maximum amount of money you can get by cutting the rod strategically and selling the cut pieces. (8 points)

Solution: This problem is a variation of the unbounded knapsack problem.

Let length[0,1,..,N] be the array containing different cut sizes.

Let price[p0,p1,..,pN] be the array containing the prices of each cut size in length array.

Here,

OPT(i,a) = value of the optimal solution using a subset of rod lengths [1,...,i] with max allowed length of a.

If i belongs to the solution,

OPT(i,a) = price[i-1] + OPT(i,a – length[i-1])

Else,

OPT(i,a) = OPT(i-1,a)

Let memory[N+1][N+1] be the array used for memoization where 1st row and 1st column is initialized to 0 and all the other elements are initialized to -1.

CUT-ROD(length, price, n, N)

```
if memory[n][N] != -1
        return memory[n][N]
else if length[n-1] <= N
        return memory[n][N] = max(price[n-1] + CUT-ROD(length, price, n, N-
        length[n-1]), CUT-ROD(length, price, n-1, N))
else
        return memory[n][N] = CUT-ROD(length, price, n-1, N)
```

The maximum amount would be stored at memory[N+1][N+1]

Rubrics:
Correctly definition/interpretation of subproblems - 3 pts
Correct recurrence relation - 3 points
Correct base-cases - 2 points