H.W - 6

Name: SRI MANVITH VADDEBOYINA
USC: 1231409457

1. 0-1 Knapsack problem. Design a dynamic programming algorithm to compute the optimal value you can get from a knapsack with capacity W.

Given infinitely many number of items of each type, the problem is similar to knapsack problem and optimal solution is given by,

$$OPT(i, w) = \max\begin{cases} OPT(i-1, w), \\ OPT(i, w - w_i) + v_i \end{cases}$$

and $OPT(0,0) = 0$

As there are infinitely many number of items of each type, we choose another item of type i and proceed with subproblem.

We have two options whether to choose the item (or) not. Based on this we have obtained the above recurrence relation.

This would give us the optimal solution you can get from a knapsack of capacity W.

2. $$OPT(K) = \begin{cases} \max(OPT(i)), & 0 \le i < k \text{ and } s_{i+1,k} \text{ is a word in dictionary} \\ 0, & \text{otherwise.} \end{cases}$$

& $OPT(0) = 0$

Let $s_{i,k}$ denote substring $s_i s_{i+1} \ldots s_k$.

$OPT(K) = 1 \rightarrow$ segmentation is possible
$0 \rightarrow$ otherwise.

Segmentation is possible if only the last word is in the dictionary and the remaining substring can be segmented.

Computing the value of OPT(0), ---OPT(n) using the above relation would give us the OPT(n) which is our solution.

This can be computed in $\Theta(n^2)$ time.

3. 'n' balloons indexed $0 \to n-1$

nums [-1] = nums [n] = 1

Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze run time complexity

Solution:

For this problem we have reccurence relation as,

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

dp[N+1][N+1]

```
BURST_BALLOONS(l, r):
    IF L>R:          MAX_COST = -INT_MAX
        RETURN 0  ->   IF (dp[i][j] != -1)
                          return dp[i][j]
    FOR EACH K: L -> R:
        COST = nums[l-1] * nums[k] * nums[r+1]
                  +
               BURST_BALLOONS(l, k-1)
                  +
               BURST_BALLOONS(k+1, r)
        MAX_COST = MAX (MAX_COST, COST)
    ENDFOR
    ~~RETURN MAX_COST~~
    dp[l][r] = MAX_COST
return dp[1][n]
```

Time Complexity:

$$T.C = N * N * N = N^3$$

2 states loop

$$T.C = O(N^3)$$

4. 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.

Solution:

Given a rod can be cut into pieces. Let the lengths of pieces be 0 -- N i.e length [0, 1, -- N]

Cost of each rod of length i is p_i dollars.
Cost[p_0, -- p_N] = cost of each piece of length i.

Let the optimal solution be OPT

$$OPT(i, k) = \begin{cases} cost[i-1] + OPT(i, k - length[i-1]) & \text{if } i \in \text{solution} \\ \text{else} \\ OPT(i-1, k) \end{cases}$$

This problem is similar to unbounded knapsack problem with

weights = lengths
values = costs
W = N

Using memoization,

dp[N+1][N+1], where dp[0][0] = 0,
others = -1.

ALGORITHM:

```
ROD_DP(length, cost, n, N):
    IF n=0 OR N=0:  return 0
    IF length[n-1] <= N:
        dp[n][N] = max (cost[n-1] + ROD-DP(
                   length, cost, n, N -
                   length[n-1])
```

ALGORITHM

```
ROD_DP (length, cost, n, N):
    IF n=0 OR N=0: RETURN 0
    IF length[n-1] <= N:
        dp[n][N] = max(cost[n-1] +
                        ROD_DP (length, cost, n,
                            N - length[n-1]),
                        ROD_DP (length, cost, n-1, N));

    ELSE:
        dp[n][N] = ROD_DP (length, cost, n-1, N)
return dp[n][N]
```

This algorithm gives us the maximum amount.

5. For this problem we have the recurrence relation as

$$\boxed{OPT(n) = \min_{1 \le j \le n} S^2_{j,n} + OPT(j-1)}$$

The last line of words $w_j, \ldots w_n$ is used in an optimum solution if and only if minimum is obtained using index j.

<u>ALGORITHM:</u>

$S_{ij} = \infty$ if words exceed total length L.

```
OPT [0]=0
FOR  k= 1,--n
        OPT[i] = min   (S²j,k + OPT (j-1))
               1≤j≤k
END FOR
Return OPT[n]
```

<u>Time Complexity:</u>

Each iteration : $O(n)$, n iterations $\Rightarrow O(n) * n$.

$T.C = \underline{O(n^2)}$

6. a) The below example will result in incorrect answer

	Minute 1	Minute 2	Minute 3
A	2	1	1
B	1	10	1

The given algorithm will follow the sequence of a_1, a_2, a_3 i.e AAA which gives value of $2+1+1=4$

But optimal solution would be BBB, which would give total of $1+10+1=12$.

b) Suppose we have optimal plans for sequences upto minute $(i-2)$ and ending with A in the last minute.

Let it be denoted by Max[A][i-2]. Similarly we have the optimal plans ending with B, Max[B][i-1] and Max[B][i-2] respectively.

Based on this we can find max[A][i] and ~~max[B]~~ Max[B][i].

$$Max[A][i] = \max(Max[A][i-1] + a_i, Max[B][i-2])$$

$$Max[B][i] = \max(Max[B][i-1] + b_i, Max[A][i-2])$$

$$\text{The optimal value} = \max(max[A][i], max[B][i])$$

ALGORITHM :

```
Max[A][0] := 0  and  Max[B][0] := 0
Max[A][1] := a1 ,  Max[B][1] := b1

Set i := 2

WHILE i <= n
IF MAX[A][i-1] + ai < Max[B][i-2]
    Max[A][i] := Max[B][i-2]

ELSE
    Max[A][i] := Max[A][i-1] + ai
ENDIF
Replace ai by bi
increment i
END WHILE
RETURN max(MAX[A][n], MAX[B][n])
END
```

$\boxed{T.C = O(n)}$