Name - Varun Gupta
Roll No - 2023101108

AAD A-4

A1)

Observation :

→ The problem can be modelled as "bounded knapsack problem" with repitition allowed, aiming to find minimum number of Tasks.

→ This is similar to problem where we minimize the number of coins required to form a sum.

→ It could be easily solved using DP.

→ Edge Cases :
- If T = 0 then answer is 0
- If all Tasks are greater than T, answer = -1

Assumption : All the "ai" are non-negative.

Algorithm :- We define an array dp of size (T+1) where dp[j] stores the minimum time required to get j points.

→ dp[0] = 0

→ dd[j] = ∞ ∀ j ∈ [1, T]

→ For each possible score, try all tasks and update the minimum time required for each total score.

Steps :-

- Loop from 1 to T (inclusive)
- For each point j, try all tasks and update the minimum time needed to reach "j" using

$$dp[j] = \min(dp[j], dp[j - a_i] + 1) \text{ if } j \geq a_i$$

- if dp[T] is still "∞" after filling the array return -1 otherwise return dp[T].

Code :-

```
MinTime (T, tasks, n):
    for i from 0 to T:
        | dp[i] = ∞
    dp[0] = 0
    for j in [1, T]:
        for all task in tasks s.t (j - task) ≥ 0:
            | dp[j] = min (dp[i], dp[j - task] + 1)
    return dp[T] == ∞ ? -1 : dp[T]
```

Run Time Analysis :-

For each score j from 1 to T, we try all N Tasks (as upper bound).

$$\text{Time Complexity} = O(T \times N)$$

Dry Run :-

T = 7

Tasks = [2, 3, 5]

$$dp = [0, \infty, \infty, \infty, \infty, \infty, \infty, \infty]$$

Filling the dp Array :-

dp[1] = ∞

dp[2] = min(∞, dp[2-2]+1) = 1

~~dp[3] = min(∞, dp[3-3]+1) = 1~~

~~dp[4] = min~~

Similarly

dp[3] = min(∞, dp[3-2]+1) = ∞

= min(∞, dp[3-3]+1) = 1

dp[4] = min(∞, dp[4-2]+1) = ∞

min(∞, dp[4-3]+1) = 2

~~min(∞, dp[4-4]+1)~~

Similarly

dp[5] = 1

dp[6] = 2

dp[7] = 2

Final, dp = [0, ∞, 1, 1, 2, 1, 2, 2]

Answer = 2 //

Proof of Correctness :-

Using Strong Induction

Base Case

For T = 0, dp[0] = 0

Let us assume dp[K] correctly represents the minimum number of seconds needed to reach K points (∀K s.t 0 ≤ K ≤ n)

Let us proove this for K = n+1

We have
We know $\forall a_j$, dp $[n-a_j]$ returns the minimum number of seconds for $n-a_j$ due to our inductive hypothesis.

So to get n points we need atleast another task of a_j points.. Since dp $[n-a_j]$ is the minimum value of time for $n-a_j$ points. If we perform another task it will take dp$[n-a_j]+1$ as time.
We have to ~~least value~~ find a_j such that dp$[n-a_j]$ is least. ∴ we iterate through all a_j's such that $n-a_j \geq 0$ ~~min~~ doing min (dp[n], dp$[n-a_j+1]$)

Hence proved by strong induction.

A2) Observation :-

→ for each node, either we include it in the sum and exclude it. If we include it then node i-1 cannot be included (no adjacent node i-1 can't be included (no adjacent nodes condition)

→ If we exclude it, then the sum obtained at i-1 will also be sum obtained at i.

→ In the include case, we also add the current weight of node to the sum obtaind when i-2 were considered.

Finally we take maximum of the sum

Algorithm:-

Define an array dp where dp[i] stores the maximum weight of an independent set from the first node upto the i-th node.

Recursive Idea:-

For each node i, we have two choices

1. exclude: for i just dp[i-1]
2. include: for i we must skip i-1 i.e dp[i-2] + w[i]

$$dp[i] = \max(dp[i-1], dp[i-2] + w_i)$$

Base Case:-

- $dp[0] = w_0$
- $dp[1] = \max(w_0, w_1)$

Steps:-

1. Create a dp array of size n.
2. Intialize the base case
3. For each node i from 2 to n-1:
   dp[i] = max(dp[i-1], w[i]+dp[i-2])
4. Return max value as dp[n-1].
5. For finding set we can backtrack.

Time & Space Complexity:

Time - O(n) - we iterate over nodes once
Space - O(n) - we use dp array of size n.

Dry Run :-

weights = [4, 2, 7, 3, 8]

dp[0] = 4, dp[1] = max (4, 2)
= 4

Fill the dp array

dp[2] = max (dp[1], w[2] + dp[0])
= max (4, 11) = 11

dp[3] = max (dp[2], w[3] + dp[1])
= max (11, 7) = 11

dp[4] = max (dp[3], w[4] + dp[2])
= max (11, 4 8 + 11) = 19

Answer = 19

Resconstructing Solution :-

We can Backtrack through dp array

1. Start from last (n-1)
2. If dp[i] == dp[i-1] skip node i
3. If dp[i] = w[i] + dp[i-2] include node i in set & move to i-2
4. include base if Reached (i.e if i = 0 & 1)

Example Backtracking :-

weights = [4, 2, 7, 3, 8]

Start at i = 4, dp[4] = 19 → w[4] + dp[2]

Include 4

Move to i = 2, dp[2] = 11 → w[2] + dp[0]

Include 2

Move to i = 0 Include 0

∴ Selected Nodes are 0, 2, 4.

Proof of Correctness :-

Base Case:

dp[0] = w[0]

i.e, if there is only one node, we include it to independent set.

$dp[1] = \max(dp[0], w[1]) = \max(w[0], w[1])$

If there are 2 nodes we include the node whichever has the maximum weight as both can't be included.

Let us assume dp[K] correctly represent the maximum weighted sum of independent set $\forall\ 0 \leq K \leq n$.

Let us prove this for $K = n+1$.
That is it could be found through the relation,

$dp[n+1] = \max(dp[n], d[n-1] + w[n+1])$

Since Two adjacent nodes can't be included in the independent set and the nodes form a path.

There are two cases :

(i) Include the $n+1^{th}$ node :-

If we include the $n+1^{th}$ node, n^{th} node cannot be included so the maximum sum

Obtained if we include $n-1$ node & the current node

$dp[n-1] \longrightarrow$ max weighted sum of $n-1$ nodes

So $dp[n-1] + w[n+1]$ is the answer in this case

(ii) Exclude the $(n+1)^{th}$ node:

If we Exclude the $(n+1)^{th}$ node the maximum is upto n nodes

According to our inductive hypothesis, $dp[n]$ would be the answer.

So our final answer would be maximum obtained from both the cases.

$$dp[n+1] = \max(dp[n], w[n+1] + dp[n-1])$$

Hence by Induction our Proof is correct.

Hence Proved.

Q4) Observation:

1. Optimal Substructure:
The minimum interaction Score
If we know the best way to partition first j dishes into $k-1$ platters, we can compute

interaction score for adding next segment [j+1, i] as a new platter.

2 Prefix-Sum Optimisation:
We can calculate num of sweets & savory in a segment easily in O(1)

Firstly the sequence of dishes must remains same which suggests that we must evaluate all possible way of splitting the sequence while ensuring that each platter gets atleast one dish also the question ask for the minimum score which a we need to go through all possible split, but the splits are overlapping so to avoid un neccesary calculation, this lead to dp.

Interaction score of each platter is calculated by multiplication of number of savory 2 sweet dishes. This points that we need a function on array that stores the precomputed interaction score for all possible subproblem to avoid traversing each 2 every time.

Algorithm:- Firstly we will make a 2D where array [i][j] will store the interaction score of placing the, i to j dishes in one platter.

Now we can use dp.
dp[i][j] represents the minimum interaction for placing the first i dishes in the j platters.

$$dp[i][k] = \min_{j<i} (dp[j][k-1] + array[j+1][i])$$

We will traverse through all possible ways of arranging it in (K-1) platters + arranging the remaining in the K^{th} platters will give us the minimum value for arranging

Time Complexity :-

To calculate array + calculate dp array

$\Rightarrow O(n^2) + O(Kn^2)$

$\Rightarrow O(Kn^2)$

Dry Run :-

~~Dishes = {1, 2, 2, 1, 2}~~

Dishes = { 1, 2, 2, 1, 1, 2 }

K = 2

Savory Prefix : [0, 1, 1, 1, 2, 3, 3]

Sweet Prefix : [0, 0, 1, 2, 2, 2, 3]

~~dp~~ Fill table for i = 1 to 6 & K = 1 to 2

i = 1 : K = 1, dp[1][1] = 0

i = 2 : K = 1 dp[2][1] = 1

K = 2 dp[2][2] = 0

i = 3 : K = 1 dp[3][1] = 2

K = 2 dp[3][2] = 0

Optimal Partition → dp[6][2] = 4

1 2 2 | 1 1 2

Total Score = 2 + 2 = 4

Proof :-

Base case

dp[0][0] = 0

dp[i][0] = -1 & dp[0][i] = -1

↳ Trivial if their is 0 dish or 0 platter

Suppose dp[m][n] store minimum interaction score for m dishes in n platters where $m \leq i$ & $n \leq j$.

dp[m+1][n] => As their is atleast one dish in the platter, we traverse through all possible combinations of arranging 1 to m dishes in n-1 dishes as dp[i][n-1] ($1 \leq i \leq m$) correctly stores the min interaction score if we find the one with min of all this combination, then that comes out to be minimum as well.

Hence dp[m+1][n] will also get min value.

Similar we can proove for dp[m][n+1].

Hence Proved.

A3) Observation :-

The traveller can only move south or east which mean traveller can reach destination from the path in south or east. Hence we have to traverse all possible path for each grid. To avoid this we use dp from down to top, so that we can store the calculated min. energy path & we can use it direcly without calcuting it again & again.

Algorithm :-
We define a 2D grid dp of size M x N and we measure min energy required to reach point (M, N) from (i, j) zone if $i \in [1, M]$, $j \in [1, N]$ in dp[i][j].

∴ As the traveler must have at least 1 energy to survive.

⇒ Base Case :-

dp[M-1][N-1] = max (1, 1 - grid[M-1][N-1])

∴ If dest gives the energy, min 1 energy is required by the traveller.

∴ If dest drains energy, they need enough energy to avoid dropping to zero.

Recursive Relation :

We start building from [M-1], [N-1] so from (i,j) we have two possible paths, (i+1, j) & (i, j+1)

∴ we take which every of them is least from (i,j) we move to (i+1, j) or (i, j+1) based on min (dp[i+1][j] , dp[i][j+1]).

dp[i][j] = max (1, min (dp[i+1][j], dp[i][j+1]) - grid[i][j].)

Boundary Condition :

For row $i = M-1$, we can only move right
For column $j = N-1$, we can only move left.

Pseudocode :-

```
Energy-Calc (M, N, grid[][]):
    dp ← [[0] * N] * M,
    dp[M-1][N-1] = max(1, 1 - grid[M-1][N-1])
    j = N-2
    while (j ≥ 0):
        dp[M-1][j] = max(1, dp[M-1][j+1] - grid[M-1][j])
        j--
    i = M-2
    while (i ≥ 0):
        dp[i][N-1] = max(1, dp[i+1][N-1] - grid[i][N-1])
        i--

    i = M-2
    while (i ≥ 0):
        j = N-2
        while (j ≥ 0):
            dp[i][j] = max(1, min(dp[i+1][j], dp[i][j+1]) - grid[i][j])
            j--
        i--
    return dp[0][0]
```

Analysis :-

For each (i, j), we evaluate dp[i][j] using dp[i+1][j], dp[i][j+1] & grid[i][j] in O(1) time.

Hence, as each cell is processed once & there are M x N cells.

TC ⇒ O(M x N)

Dry Run :-

$$grid = \begin{bmatrix} -2 & -3 & 3 \\ -5 & -10 & 1 \\ 10 & 30 & -5 \end{bmatrix}$$

$$dp = \begin{bmatrix} \_ & \_ & \_ \\ \_ & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

dest cell ⇒ dp[2][2] = max(1, 1-(-5)) = 6

Fill last row

dp[2][1] = max(1, 6-30) = 1

dp[2,0] = max(1, 1-10) = 1

$$dp = \begin{bmatrix} \_ & \_ & \_ \\ \_ & \_ & \_ \\ 1 & 1 & 6 \end{bmatrix}$$

Fill last column

$dp[1][2] = \max(1, 6-1) = 5$

$dp[0][2] = \max(1, 5-3) = 2$

$$dp = \begin{bmatrix} - & - & 2 \\ - & - & 5 \\ 1 & 1 & 6 \end{bmatrix}$$

fill remaining cells.

$dp[1][1] = \max(1, \min(5,1) - (-10)) = 11$

$dp[1][0] = \max(1, \min(1,11) - (-5)) = 6$

$dp[0][1] = \max(1, \min(2,11) - (-3)) = 5$

$dp[0][0] = \max(1, \min(5,6) - (-2)) = 8$

$$dp = \begin{bmatrix} 8 & 5 & 2 \\ 6 & 11 & 5 \\ 1 & 1 & 6 \end{bmatrix}$$

Answer => $dp[0][0] = 8$ //

Proof of Correctness :-

Base Case

grid is 1 x 1

$dp[0][0] = \max(1, 1 - grid[0][0])$

① if $grid[0][0]$ is +ve

⇒ The traveller gains energy coming to (0,0). Hence he starts with the min energy to survive i.e. 1.

(ii) If grid [0][0] is -ve
The travellel loses energy coming to (0,0)

∴ Hence he will require atleast 1 after losing energy. Hence the traveller should start with 1 - grid[0][0] energy.

Inductive Hypothesis:-
Let us assume that dp[a][b] are correctly populated ∀ a ∈ [i, m-1], ∀ b ∈ [j, N-1]

∴ for min energy req from (i, j) to (m-1, N-1) (i,j) have two possible paths (i+1, j) & (i, j+1).

∴ We choose the path with min energy req
⇒ min (dp[i+1][j], dp[i][j+1])

∴ To take account grid[i][j] energy req
⇒ min (dp[i+1][j], dp[i][j+1]) - grid[i][j]

But to take into account that min 1 energy is req to live.

∴ Hence dp[i][j] = max (1, min (dp[i+1][j], dp[i][j+1]) - grid[i][j]

∴ Hence Proved.