DYNAMIC PROGRAMMING

PROBLEM-1

AIM: 1-DP-PLAYING WITH NUMBERS

ALGORITHM:

1. Read the integer `n` representing the number of steps.

2. Handle base cases directly for `n = 0, 1, 2, 3` with predefined results.

3. Use a dynamic programming array `dp` where `dp[i]` stores the number of ways to reach step `i`.

4. For `n ≥ 4`, calculate `dp[i] = dp[i - 1] + dp[i - 3]`.

5. Compute the values iteratively from step 4 to `n`.

6. Return and print `dp[n]`, the total number of ways to reach step `n`.

PROBLEM:

Playing with Numbers:

Ram and Sita are playing with numbers by giving puzzles to each other. Now it was Ram term, so he gave Sita a positive integer ‘n’ and two numbers 1 and 3. He asked her to find the possible ways by which the number n can be represented using 1 and 3.Write any efficient algorithm to find the possible ways.

Example 1:

Input: *6*
Output:*6*
Explanation: *There are 6 ways to 6 represent number with 1 and 3*

1+1+1+1+1+1
3+3
1+1+1+3
1+1+3+1
1+3+1+1
3+1+1+1

Input Format
First Line contains the number n

Output Format

Print: The number of possible ways ‘n’ can be represented using 1 and 3

Sample Input

6

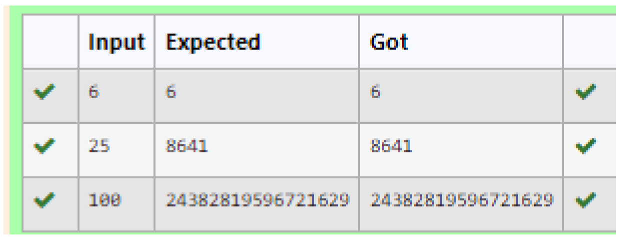
Sample Output

6

PROGRAM:

```
#include <stdio.h>
long long countWays(int n) {
    if (n < 0) return 0;
    if (n == 0) return 1;
    if (n == 1) return 1;
    if (n == 2) return 1;
    if (n == 3) return 2;
    long long dp[n + 1];
    dp[0] = 1;
    dp[1] = 1;
    dp[2] = 1;
    dp[3] = 2;
    for (int i = 4; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 3];
    }
    return dp[n];
}
int main() {
    int n;
    scanf("%d", &n);
    long long result = countWays(n);
    printf("%lld\n", result);
    return 0;
}
```

OUTPUT :



	Input	Expected	Got	
✓	6	6	6	✓
✓	25	8641	8641	✓
✓	100	24382819596721629	24382819596721629	✓

PROBLEM-2

AIM: 2-DP-PLAYING WITH CHESSBOARD

ALGORITHM:

1. Read the integer `n` and the `n x n` grid of monetary values.

2. Initialize the dynamic programming table `dp[0][0]` with the value from the grid's top-left corner.

3. Fill the first column and first row of `dp` by accumulating values from the grid.

4. For each cell, compute the maximum sum path by considering the maximum of the top (`dp[i-1][j]`) and left (`dp[i][j-1]`) cells.

5. Return the value at `dp[n-1][n-1]`, which contains the maximum monetary path sum.

PROBLEM:

Ram is given with an n*n chessboard with each cell with a monetary value. Ram stands at the (0,0), that the position of the top left white rook. He is been given a task to reach the bottom right black rook position (n-1, n-1) constrained that he needs to reach the position by traveling the maximum monetary path under the condition that he can only travel one step right or one step down the board. Help ram to achieve it by providing an efficient DP algorithm.

Example:
Input
3
1 2 4
2 3 4
8 7 1
Output:
19

Explanation:
Totally there will be 6 paths among that the optimal is
Optimal path value:1+2+8+7+1=19

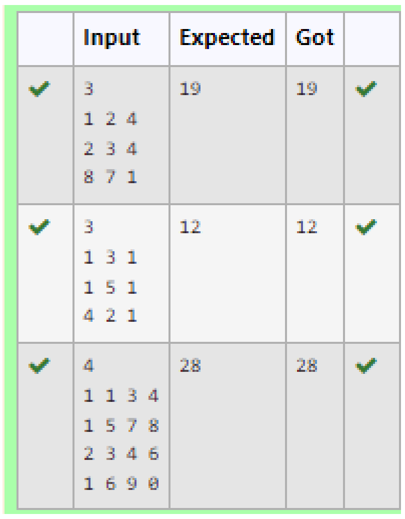
Input Format
First Line contains the integer n
The next n lines contain the n*n chessboard values
Output Format

Print Maximum monetary value of the path

PROGRAM:

```
#include <stdio.h>
int maxMonetaryPath(int n, int grid[n][n]) {
    int dp[n][n];
    dp[0][0] = grid[0][0];
    for (int i = 1; i < n; i++) {
        dp[i][0] = dp[i - 1][0] + grid[i][0];
    }
    for (int j = 1; j < n; j++) {
        dp[0][j] = dp[0][j - 1] + grid[0][j];
    }
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = grid[i][j] + (dp[i - 1][j] > dp[i][j - 1] ? dp[i - 1][j] : dp[i][j - 1]);
        }
    }
    return dp[n - 1][n - 1];
}
int main() {
    int n;
    scanf("%d", &n);
    int grid[n][n];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &grid[i][j]);
        }
    }
    int result = maxMonetaryPath(n, grid);
    printf("%d\n", result);
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✓	3 1 2 4 2 3 4 8 7 1	19	19	✓
✓	3 1 3 1 1 5 1 4 2 1	12	12	✓
✓	4 1 1 3 4 1 5 7 8 2 3 4 6 1 6 9 0	28	28	✓

PROBLEM-3

AIM: 3-DP-LONGEST COMMON SUBSEQUENCE

ALGORITHM:

1. Read two strings `s1` and `s2`.

2. Initialize a 2D DP array `dp` where `dp[i][j]` will store the length of the longest common subsequence of `s1[0..i-1]` and `s2[0..j-1]`.

3. Set the first row and column of the DP table to 0 (since the LCS with an empty string is 0).

4. Iterate through the strings and fill the DP table: if characters match (`s1[i-1] == s2[j-1]`), set `dp[i][j] = dp[i-1][j-1] + 1`, otherwise set `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`.

5. Return the value at `dp[m][n]`, which is the length of the longest common subsequence.

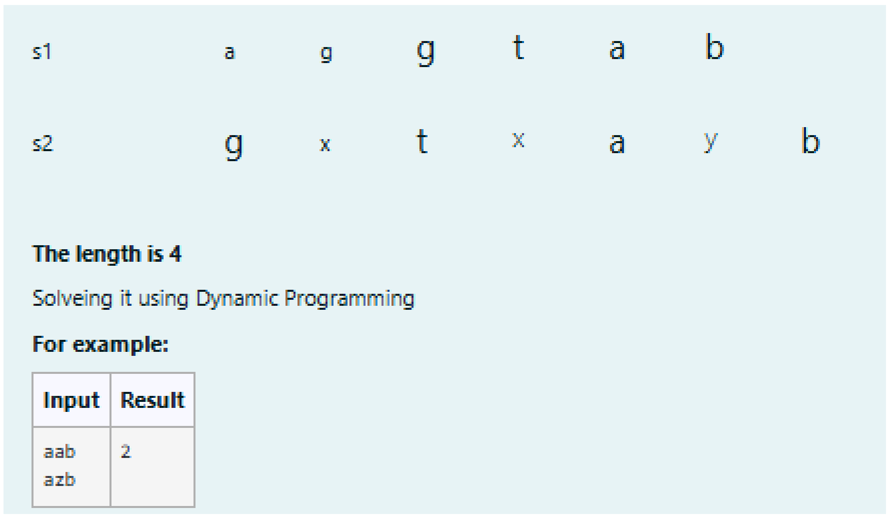
PROBLEM:

Given two strings find the length of the common longest subsequence(need not be contiguous) between the two.

Example:

s1: ggtabe

s2: tgatasb



s1	a	g	g	t	a	b	
s2	g	x	t	x	a	y	b

The length is 4

Solveing it using Dynamic Programming

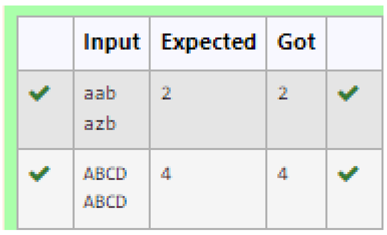
For example:

Input	Result
aab azb	2

PROGRAM:

```
#include <stdio.h>

#include <string.h>

int longestCommonSubsequence(char* s1, char* s2) {

    int m = strlen(s1);
    int n = strlen(s2);
    int dp[m + 1][n + 1];
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            } else if (s1[i - 1] == s2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];
            }
        }
    }
    return dp[m][n];
}

int main() {
    char s1[100], s2[100];
    scanf("%s", s1);
    scanf("%s", s2);
    int result = longestCommonSubsequence(s1, s2);
    printf("%d\n", result);
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✓	aab azb	2	2	✓
✓	ABCD ABCD	4	4	✓

PROBLEM-4

AIM: 4-DP-LONGEST NON DECREASING SUBSEQUENCE

ALGORITHM:

1. Read the integer `n` and the array `arr[]` of size `n`.

2. Initialize a `dp[]` array where each element is set to 1, as each element can be a subsequence of length 1.

3. Iterate through the array with two loops: for each element `arr[i]`, check all previous elements `arr[j]` (where `j < i`). If `arr[j] <= arr[i]`, update `dp[i]` to be the maximum of `dp[i]` and `dp[j] + 1`.

4. Find the maximum value in the `dp[]` array, which represents the length of the longest non-decreasing subsequence.

5. Return the maximum value found in `dp[]`.

PROBLEM:

Find the length of the Longest Non-decreasing Subsequence in a given Sequence.

Eg:

Input:9

Sequence:[-1,3,4,5,2,2,2,2,3]

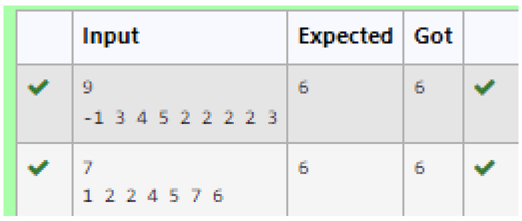
the subsequence is [-1,2,2,2,2,3]

Output:6

PROGRAM:

```
#include <stdio.h>

int longestNonDecreasingSubsequence(int arr[], int n) {

    int dp[n];

    for (int i = 0; i < n; i++) {

        dp[i] = 1;

    }

    for (int i = 1; i < n; i++) {

        for (int j = 0; j < i; j++) {

            if (arr[j] <= arr[i]) {

                dp[i] = (dp[i] > dp[j] + 1) ? dp[i] : (dp[j] + 1);
            }
        }
    }
    int maxLength = 0;
    for (int i = 0; i < n; i++) {
        if (dp[i] > maxLength) {
            maxLength = dp[i];
        }
    }
    return maxLength;
}

int main() {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    int result = longestNonDecreasingSubsequence(arr, n);
    printf("%d\n", result);
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✓	9 -1 3 4 5 2 2 2 2 3	6	6	✓
✓	7 1 2 2 4 5 7 6	6	6	✓