Ex:6 **Date**:20.10.24

Name: Veronica Regina Paul **Register number**:230701377

Dynamic Programming

AIM

To find the number of ways to represent a given integer n using only the numbers 1 and 3.

ALGORITHM

1. Define the Problem:
    - Given a positive integer n, we need to determine how many unique ways n can be represented as a sum of the integers 1 and 3.
    - Order matters, so each distinct arrangement of 1s and 3s is counted as a different way.
2. Dynamic Programming Approach:
    - Let dp[i] represent the number of ways to represent the number i using only 1 and 3.
    - Define the base cases:
        - dp[0] = 1 since there is one way to sum up to 0 (by choosing no numbers).
        - dp[1] = 1 (only one way: 1).
        - dp[2] = 1 (only one way: 1 + 1).
        - dp[3] = 2 (two ways: 1 + 1 + 1 and 3).
    - For each number i from 4 to n, we calculate dp[i] by the recurrence relation:
        - dp[i] = dp[i - 1] + dp[i - 3]
            - dp[i - 1] accounts for adding a 1 to all combinations that sum to i - 1.
            - dp[i - 3] accounts for adding a 3 to all combinations that sum to i - 3.
3. Output:
    - Print the value of dp[n], which gives the number of ways to represent n with 1 and 3.

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

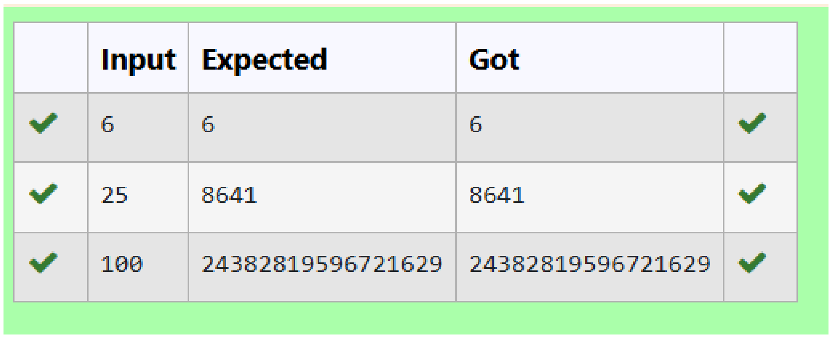
6

PROGRAM

```
#include <stdio.h>


long long countWays(int n) {

   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);
    printf("%lld\n", countWays(n));
    return 0;
}
```

OUTPUT



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

2-DP-Playing with chessboard

AIM

To find the maximum monetary path value on an n*n chessboard from the top-left corner (0,0) to the bottom-right corner (n-1, n-1), moving only right or down.

ALGORITHM

1. Define the Problem:
   - Each cell (i, j) on the n*n chessboard has a monetary value.
   - We need to maximize the sum of monetary values collected while moving from the top-left corner to the bottom-right corner.
   - Moves are restricted to either one cell to the right or one cell downwards.
2. Dynamic Programming Approach:
   - Define a 2D array dp where dp[i][j] holds the maximum monetary path value to reach cell (i, j).
   - Initialize dp[0][0] to chessboard[0][0] as it is the starting point.
   - Fill the first row and first column of dp:
     - For the first row: dp[0][j] = dp[0][j-1] + chessboard[0][j] (only right moves are possible).
     - For the first column: dp[i][0] = dp[i-1][0] + chessboard[i][0] (only downward moves are possible).
   - For the rest of the cells (i, j):
     - dp[i][j] = chessboard[i][j] + max(dp[i-1][j], dp[i][j-1])
     - This formula takes the maximum path value from either the top or left cell to maximize the total path value.
3. Output:
   - The value dp[n-1][n-1] will hold the maximum monetary path value from the top-left to the bottom-right.

PROBLEM

Playing with Chessboard:

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

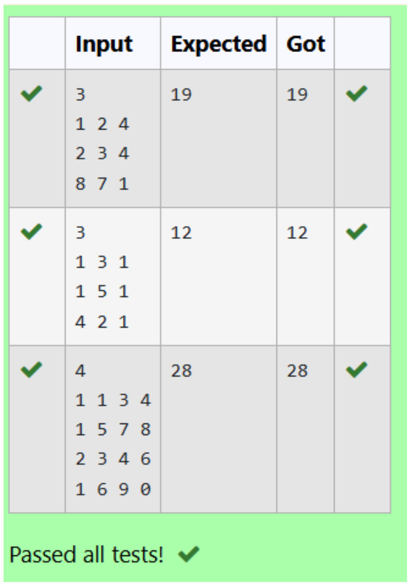
Print Maximum monetary value of the path

PROGRAM

```
#include <stdio.h>


int main() {

   int n;

   scanf("%d", &n);


   int chessboard[n][n];

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

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

         scanf("%d", &chessboard[i][j]);

      }

   }


   int dp[n][n];

   dp[0][0] = chessboard[0][0];
```

```
    for (int i = 1; i < n; i++) {
        dp[i][0] = dp[i-1][0] + chessboard[i][0];
        dp[0][i] = dp[0][i-1] + chessboard[0][i];
    }

    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            if (dp[i-1][j] > dp[i][j-1]) {
                dp[i][j] = chessboard[i][j] + dp[i-1][j];
            } else {
                dp[i][j] = chessboard[i][j] + dp[i][j-1];
            }
        }
    }
    printf("%d\n", dp[n-1][n-1]);
    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	✓

Passed all tests! ✓

3-DP-Longest Common Subsequence

AIM

To find the length of the longest common subsequence (LCS) between two strings s1 and s2 using dynamic programming.

ALGORITHM

1. **Define the Problem**:
   - Given two strings s1 and s2, find the length of their longest common subsequence.
   - A subsequence is a sequence derived by deleting some characters without changing the order of remaining characters.
2. **Dynamic Programming Approach**:
   - Define a 2D array dp where dp[i][j] holds the length of the LCS for the substrings s1[0...i-1] and s2[0...j-1].
   - Initialize dp[0][j] and dp[i][0] to 0 for all i and j since an empty substring has no common subsequence.
   - For each character pair (s1[i-1], s2[j-1]):
     - If s1[i-1] == s2[j-1], set dp[i][j] = dp[i-1][j-1] + 1.
     - Otherwise, dp[i][j] = max(dp[i-1][j], dp[i][j-1]).
   - The final answer, dp[m][n], will contain the length of the LCS for s1 and s2.
3. **Output**:
   - Print dp[m][n] as 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

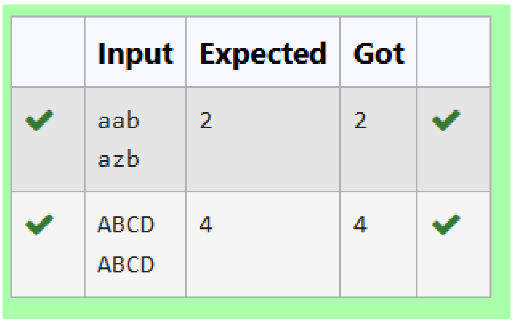
For example:

Input	**Result**
aab azb	2

PROGRAM

```
#include <stdio.h>
#include <string.h>

int lcsLength(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 {
                    if (dp[i-1][j] > dp[i][j-1]) {
                        dp[i][j] = dp[i-1][j];
                    } else {
                        dp[i][j] = dp[i][j-1];
                    }
                }
            }
        }
    }

    return dp[m][n];
}

int main() {
    char s1[100], s2[100];

    scanf("%s", s1);
    scanf("%s", s2);

    int result = lcsLength(s1, s2);
    printf("%d\n", result);

    return 0;
}
```

OUTPUT



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

4-DP-Longest non-decreasing Subsequence

AIM

To find the length of the longest non-decreasing subsequence in a given sequence of integers.

ALGORITHM

1. Initialize DP Array:
    - Create an array dp where dp[i] will store the length of the longest non-decreasing subsequence ending at index i.
    - Initialize all elements of dp to 1, since the minimum length of any subsequence is 1 (the element itself).
2. Populate DP Array:
    - Loop through the array with index i (from 1 to n-1).
    - For each i, loop through all indices j (from 0 to i-1).
        - If arr[i] >= arr[j], this means arr[i] can extend the subsequence ending at j.
        - Update dp[i] as dp[i] = max(dp[i], dp[j] + 1) to ensure it stores the longest subsequence ending at i.
3. Find Maximum Length:
    - Iterate through the dp array to find the maximum value. This maximum value represents the length of the longest non-decreasing subsequence.
4. Return Result:
    - Output the maximum length found in the dp array.

PROBLEM

Problem statement:

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]

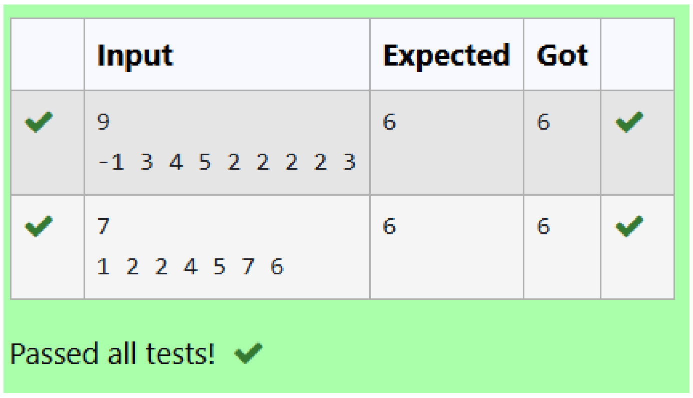
the subsequence is [-1,2,2,2,2,3]

Output:6

PROGRAM

```
#include <stdio.h>


int longestNonDecreasingSubsequence(int arr[], int n) {
    int dp[n];
    int i, j, max_len = 1;


    for (i = 0; i < n; i++) {
        dp[i] = 1;
    }


    for (i = 1; i < n; i++) {
        for (j = 0; j < i; j++) {
            if (arr[i] >= arr[j]) {
                if (dp[i] < dp[j] + 1) {
                    dp[i] = dp[j] + 1;
                }
            }
        }
    }


    for (i = 0; i < n; i++) {
        if (dp[i] > max_len) {
            max_len = dp[i];
        }
    }


    return max_len;
}
int main() {
    int arr[] = {-1, 3, 4, 5, 2, 2, 2, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);

    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	✓

Passed all tests! ✓