Ex. No: 5 **Date: 10.09.24**

Register No.: 230701368 **Name: AL UMA**

DYNAMIC PROGRAMMING

5.A Playing With Number

AIM:

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

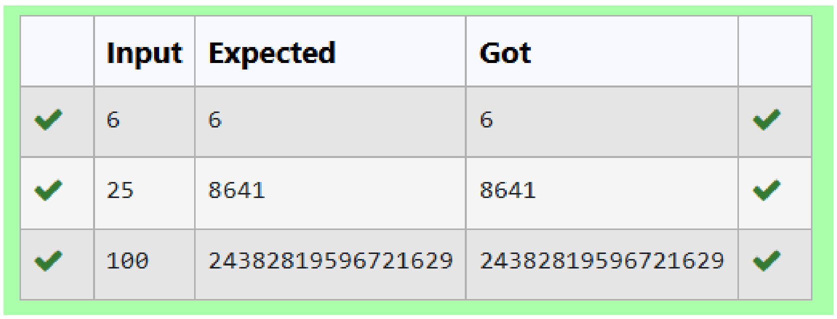
ALGORITHM:

```
Function main()
    // Step 1: Read the number of elements n
    Initialize n  // Number of elements
    Read n from user  // Input the value of n

    // Step 2: Initialize the array a of size n+1 and set initial values
    Initialize array a of size n+1  // Declare an array a of size n+1
    Set a[0] = 1  // Base case: a[0] is set to 1 (first condition)

    For i from 1 to n  // Loop from 1 to n
        Set a[i] = 0  // Initialize a[i] to 0 for all i from 1 to n
    End For

    // Step 3: Update the array based on the recurrence relation
    For i from 1 to n  // Loop from 1 to n
        Set a[i] = a[i] + a[i-1]  // Add a[i-1] to a[i]

        // Step 3.1: If i >= 3, add a[i-3] to a[i]
        If i >= 3
            Set a[i] = a[i] + a[i-3]  // Add a[i-3] to a[i]
        End If
    End For

    // Step 4: Output the result a[n]
    Print a[n]  // Output the final value of a[n]
End Function
```

PROGRAM:

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

OUTPUT:



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

5.B 2-DP-Playing With ChessBoard

AIM:

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

ALGORITHM:

```
Function max(a, b)
   // Step 1: Return the maximum of a and b
   If a > b
      Return a
   Else
```

```
        Return b
End Function

Function maxMonetaryPath(n, board)
    // Step 1: Initialize a 2D array dp to store the maximum monetary path
    Initialize dp as a 2D array of size n x n

    // Step 2: Set the starting point dp[0][0] as the value of board[0][0]
    Set dp[0][0] = board[0][0]

    // Step 3: Fill the first row (i = 0) of dp
    For j from 1 to n-1
        Set dp[0][j] = dp[0][j-1] + board[0][j]
    End For

    // Step 4: Fill the first column (j = 0) of dp
    For i from 1 to n-1
        Set dp[i][0] = dp[i-1][0] + board[i][0]
    End For

    // Step 5: Fill the remaining cells of dp using the recurrence relation
    For i from 1 to n-1
        For j from 1 to n-1
            Set dp[i][j] = board[i][j] + max(dp[i-1][j], dp[i][j-1])
        End For
    End For

    // Step 6: Return the value at dp[n-1][n-1] (the bottom-right corner of dp)
    Return dp[n-1][n-1]
End Function
```

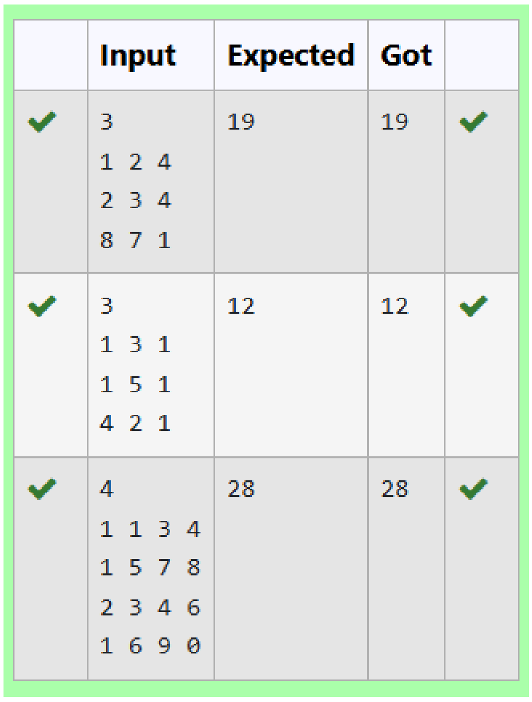
```
Function main()
    // Step 1: Read the size of the board (n)
    Initialize n
    Read n from user  // Input the value of n (size of the board)

    // Step 2: Initialize the board as a 2D array of size n x n
    Initialize board as a 2D array of size n x n

    // Step 3: Read the board elements
    For i from 0 to n-1
        For j from 0 to n-1
            Read board[i][j] from user  // Input each element of the board
        End For
    End For

    // Step 4: Call maxMonetaryPath to compute the maximum monetary path
    Initialize result
    Set result = Call maxMonetaryPath(n, board)  // Compute the result

    // Step 5: Output the result
    Print result  // Output the maximum monetary path value
End Function
```

PROGRAM:

```
#include <stdio.h>


int max(int a, int b) {
    return (a > b) ? a : b;
}
```

```
int maxMonetaryPath(int n, int board[n][n]) {
    int dp[n][n];

    dp[0][0] = board[0][0];

    for (int j = 1; j < n; j++) {
        dp[0][j] = dp[0][j - 1] + board[0][j];
    }

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

    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = board[i][j] + max(dp[i - 1][j], dp[i][j - 1]);
        }
    }

    return dp[n - 1][n - 1];
}

int main() {
    int n;
    scanf("%d", &n);
    int board[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &board[i][j]);
        }
    }

    int result = maxMonetaryPath(n, board);
    printf("%d\n", result);
}
```

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	✓

5.C 3-DP-Longest Common Subsequence

AIM:

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

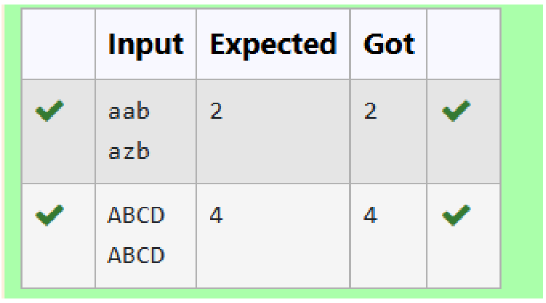
ALGORITHM

```
int longestCommonSubsequence(s1, s2)
{
    m = length of s1  // Length of first string
    n = length of s2  // Length of second string
    initialize dp[m + 1][n + 1]  // DP table

    // Initialize the DP table with base cases
    for i from 0 to m
    {
        for j from 0 to n
        {
            if i == 0 or j == 0
            {
                dp[i][j] = 0  // Base case: LCS of an empty string
            }
            else if s1[i - 1] == s2[j - 1]
            {
                dp[i][j] = dp[i - 1][j - 1] + 1  // Characters match
            }
            else
            {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])  // Characters do not match
            }
        }
    }

    return dp[m][n]  // Return length of LCS
}

function main()
{
    initialize s1[100], s2[100]  // Arrays to hold the strings

    read s1 from user
    read s2 from user

    result = longestCommonSubsequence(s1, s2)  // Calculate LCS
    print result  // Print the result
}
```

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];

    // Initialize the DP table with base cases
    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", result);

}
```

OUTPUT



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

5.D 4-Longest Non-Decreasing Subsequence

AIM:

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

.

ALGORITHM:

```
int longestNonDecreasingSubsequence(n, sequence)
{
   initialize dp[n]  // Array to hold the lengths of subsequences
   maxLength = 1  // Initialize the maximum length

   // Initialize dp array where each element is 1
   for i from 0 to n - 1
   {
      dp[i] = 1
   }

   // Calculate the length of the longest non-decreasing subsequence
   for i from 1 to n - 1
   {
      for j from 0 to i - 1
```

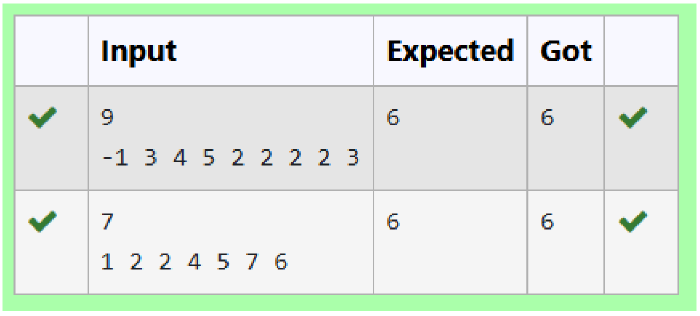
```
        {
            if sequence[j] <= sequence[i]
            {
                dp[i] = max(dp[i], dp[j] + 1)  // Update dp[i] if a longer subsequence is found
            }
        }

        maxLength = max(maxLength, dp[i])  // Update the maximum length found
    }

    return maxLength  // Return the length of the longest non-decreasing subsequence
}

function main()
{
    initialize n  // Number of elements in the sequence
    read n from user

    initialize sequence[n]  // Array to hold the sequence

    // Read values into the sequence
    for i from 0 to n - 1
    {
        read sequence[i] from user
    }

    result = longestNonDecreasingSubsequence(n, sequence)  // Calculate result
    print result  // Print the result
```

```
}
```

PROGRAM

```
#include <stdio.h>

int longestNonDecreasingSubsequence(int n, int sequence[]) {
    int dp[n];
    int maxLength = 1;

    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 (sequence[j] <= sequence[i]) {
                dp[i] = (dp[i] > dp[j] + 1) ? dp[i] : dp[j] + 1;
            }
        }

        maxLength = (maxLength > dp[i]) ? maxLength : dp[i];
    }

    return maxLength;
}

int main() {
    int n;
    scanf("%d", &n);

    int sequence[n];

    for (int i = 0; i < n; i++) {
        scanf("%d", &sequence[i]);
    }

    int result = longestNonDecreasingSubsequence(n, sequence);
    printf("%d", result);

}
```

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	✓