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

Register No.: 230701346 **Name: Subasri V**

Dynamic Programming

5.a. Playing with Numbers

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:

1) Initialize an array ways of size n+1 to store the number of ways to represent each number from 0 to n.

2) Set ways[0] to 1

3) Iterate from 1 to n and for each number i, calculate the number of ways to represent i using the numbers 1 and 3.

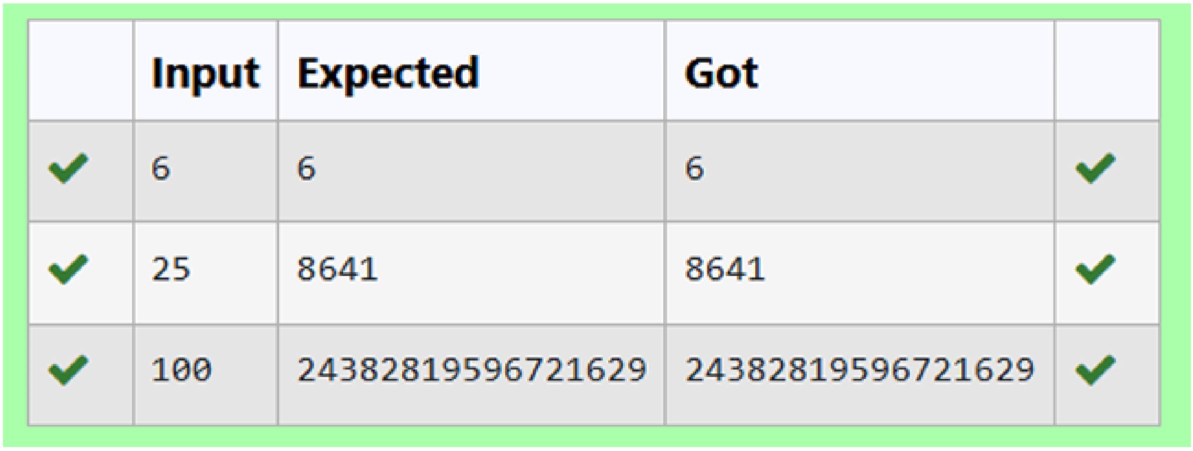
4) For each i, add the number of ways to represent i-1 and i-3 to ways[i].

5) Return ways[n] as the result.

Program:

```
 #include <stdio.h>

 int countWays(int n) {

    int ways[n + 1];

   ways[0] = 1;  // Base case: 1 way to represent 0


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

      ways[i] = 0;

      if (i >= 1) ways[i] += ways[i - 1];

      if (i >= 3) ways[i] += ways[i - 3];

   }

   return ways[n];

}


int main() {

   int n;

   printf("Enter a positive integer: ");

   scanf("%d", &n);

    int result = countWays(n);

   printf("%d", result);

}
```

Output:



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

5.b. 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:

1) Initialize a 2D array dp of size n*n to store the maximum monetary value that can be collected up to each cell.

2) Set dp[0][0] to the monetary value of the starting cell (0,0).

3) Iterate through each cell (i, j) in the chessboard:

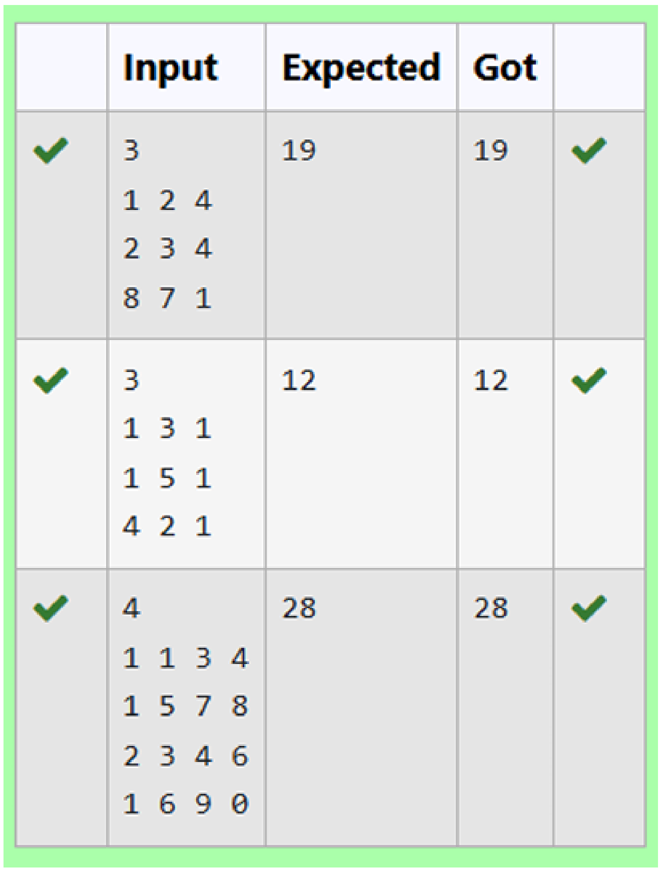
- If i > 0, update dp[i][j] to be the maximum of dp[i][j] and dp[i-1][j] + value[i][j].
- If j > 0, update dp[i][j] to be the maximum of dp[i][j] and dp[i][j-1] + value[i][j].

5) The value at dp[n-1][n-1] will be the maximum monetary value that can be collected.

Program:

```
#include <stdio.h>

#define MAX 100

int max(int a, int b) {

  return (a > b) ? a : b;

}

int maxMonetaryPath(int n, int value[MAX][MAX]) {

  int dp[MAX][MAX];

  dp[0][0] = value[0][0];


  for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i > 0) {
                dp[i][j] = max(dp[i][j], dp[i-1][j] + value[i][j]);
            }
            if (j > 0) {
                dp[i][j] = max(dp[i][j], dp[i][j-1] + value[i][j]);
            }
        }
    }
   return dp[n-1][n-1];
}
int main() {
    int n;
    int value[MAX][MAX];
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &value[i][j]);
         }
    }
    int result = maxMonetaryPath(n, value);
    printf("%d", 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	✓

3. DP-Longest Common Subsequence

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

Algorithm:

```
1. Read strings s1 and s2.
2. Let m = length of s1 and n = length of s2.
3. Initialize 2D array dp of size (m+1) x (n+1).
4. For i = 0 to m:
   a. For j = 0 to n:
      i. If i == 0 or j == 0:
         A. Set dp[i][j] = 0 (base case: LCS with empty string).
      ii. Else if s1[i-1] == s2[j-1]:
         A. Set dp[i][j] = dp[i-1][j-1] + 1 (LCS increases by 1).
      iii. Else:
         A. Set dp[i][j] = max(dp[i-1][j], dp[i][j-1]) (LCS doesn't increase).
5. Return dp[m][n] as the length of the Longest Common Subsequence.
```

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);


}
```

Algorithm

4.DP -Longest non-decreasing Subsequence

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

```
1. Read the integer n (size of array).
2. Initialize array sequence[] of size n.
3. Read the elements into sequence[].
4. Initialize dp[] array of size n with all values set to 1.
5. Initialize maxLength = 1.
6. For i = 1 to n-1:
   a. For j = 0 to i-1:
      i. If sequence[j] <= sequence[i]:
         A. Set dp[i] = max(dp[i], dp[j] + 1).
   b. Update maxLength = max(maxLength, dp[i]).
7. Return maxLength as the length of the Longest Non-Decreasing Subsequence.
```

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);

}
```