**DYNAMIC PROGRAMMING**

**AIM:**

## Playing with Numbers

**ALGORITHM:**

1. Read integer n (the target number).
2. Define function ways(n):
   * Create array dp of size n + 1.
   * Set dp = 1 (base case).
   * Initialize all other elements in dp to 0.
   * For each index i from 1 to n:Update
   * Return dp[n].
3. In main():
   * Read integer n.
   * Call ways(n) and store the result.
   * Print the result.

**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 ways(int n) {

long long dp[n + 1];

dp[0] = 1;

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

dp[i] = 0;

}

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

dp[i] += dp[i - 1];

if (i >= 3) {

dp[i] += dp[i - 3];

}

}

return dp[n];

}

int main() {

int n;

scanf("%d", &n);

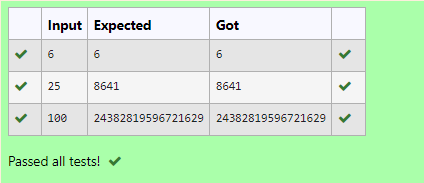
long long result = ways(n);

printf("%lld\n", result);

return 0;

}

**OUTPUT:**

****

**AIM:**

Playing with Chessboard

**ALGORITHM:**

1. Read integer n (size of the chessboard).
2. Create a 2D array board[n][n].
3. Fill board with user input.
4. Define function path(n, board):
   * Create a 2D array dp[n][n].
   * Set dp = board.
   * For each column j from 1 to n-1:Set
   * For each row i from 1 to n-1:Set
   * For each cell (i, j) from (1, 1) to (n-1, n-1):Set
   * Return dp[n-1][n-1].
5. In main():
   * Call path(n, board) and store the result.
   * Print the result.

**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 path(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] + (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 board[n][n];

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

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

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

}

}

int result = path(n, board);

printf("%d\n", result);

return 0;

}

**OUTPUT:**



**AIM:**

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

## **ALGORITHM:**

1. Read two strings s1 and s2.
2. Define function common(s1, s2):
   * Get lengths m and n of s1 and s2.
   * Create a 2D array dp[m + 1][n + 1].
   * For each index i from 0 to m:For each index
   * Return the value of dp[m][n].
3. In main():
   * Call common(s1, s2) and store the result.
   * Print the result.

**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 common(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);

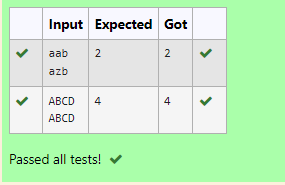
int result = common(s1, s2);

printf("%d\n", result);

return 0;

}

**OUTPUT:**



**AIM:**

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

**ALGORITHM:**

1. Read integer n (size of the array).
2. Create array arr of size n.
3. Fill arr with user input.
4. Define function dec(arr, n):
   * Create array dp of size n.
   * Initialize all elements of dp to 1.
   * For each index i from 1 to n-1:For each index
   * Initialize variable maxLength = 0.
   * For each index i from 0 to n-1:If
   * Return maxLength.
5. In main():
   * Call dec(arr, n) and store the result.
   * Print the result.

**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 dec(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 = dec(arr, n);

printf("%d\n", result);

return 0;

}

**OUTPUT:**

