

DYNAMIC PROGRAMMING

Q:5.A

AIM:

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

ALGORITHM :

Step 1: Start

Step 2: Input an integer n

Step 3: Initialize an array dp of size n+1

Step 4: Set dp[0] to 1

Step 5: For each index i from 1 to n, set dp[i] to 0

Step 6: For each index i from 1 to n, do Steps 7 and 8

Step 7: Add dp[i - 1] to dp[i]

Step 8: If i >= 3, add dp[i - 3] to dp[i]

Step 9: Print dp[n]

Step 10: Stop

PROGRAM :

```
1  #include <stdio.h>
2  int countWays(int n) {
3
4      int dp[n + 1];
5
6
7      dp[0] = 1;
8      dp[1] = 1;
9      dp[2] = 1;
10     dp[3] = 2;
11
12     for (int i = 4; i <= n; i++) {
13         dp[i] = dp[i - 1] + dp[i - 3];
14     }
15
16     return dp[n];
17 }
18
19 int main() {
20     int n;
21     scanf("%d", &n);
22
23     printf("%d\n", countWays(n));
24
25     return 0;
26 }
27
```

OUTPUT :

	Input	Expected	Got	
✓	6	6	6	✓
✓	25	8641	8641	✓
✗	100	24382819596721629	-1800446499	✗

RESULT:

The above program is executed successfully.

QUESTION 5.B

AIM:

Playing with Chessboard:

Ram is given with an $n \times n$ chessboard with each cell with a monetary value. Ram stands at the $(0,0)$, that is the position of the top left white rook. He is 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 $n \times n$ chessboard values

Output Format

Print Maximum monetary value of the path

ALGORITHM :

Step 1: Start

Step 2: Input an integer n

Step 3: Initialize a 2D array board of size $n \times n$

Step 4: For each row i from 0 to $n-1$, and each column j from 0 to $n-1$, input $\text{board}[i][j]$

Step 5: Call $\text{maxMonetaryPath}(n, \text{board})$ and store the result in result

Step 6: Print result

7: Stop

Function $\text{maxMonetaryPath}(n, \text{board})$:

Step 1: Initialize a 2D array dp of size $n \times n$

Step 2: Set $\text{dp}[0][0]$ to $\text{board}[0][0]$

Step 3: For each column j from 1 to $n-1$, set $\text{dp}[0][j] = \text{dp}[0][j-1] + \text{board}[0][j]$

Step 4: For each row i from 1 to $n-1$, set $\text{dp}[i][0] = \text{dp}[i-1][0] + \text{board}[i][0]$

Step 5: For each row i from 1 to $n-1$, and each column j from 1 to $n-1$, set $\text{dp}[i][j] = \text{board}[i][j] + \max(\text{dp}[i-1][j], \text{dp}[i][j-1])$

Step 6: Return $\text{dp}[n-1][n-1]$

```

1 #include <stdio.h>
2
3 int main() {
4     int n;
5
6     scanf("%d", &n);
7
8
9     int board[n][n];
10
11     for (int i = 0; i < n; i++) {
12         for (int j = 0; j < n; j++) {
13             scanf("%d", &board[i][j]);
14         }
15     }
16
17     int dp[n][n];
18
19     dp[0][0] = board[0][0];
20     for (int j = 1; j < n; j++) {
21         dp[0][j] = dp[0][j-1] + board[0][j];
22     }
23
24
25     for (int i = 1; i < n; i++) {
26         dp[i][0] = dp[i-1][0] + board[i][0];
27     }
28
29     for (int i = 1; i < n; i++) {
30         for (int j = 1; j < n; j++) {
31             dp[i][j] = board[i][j] + (dp[i-1][j] > dp[i][j-1] ? dp[i-1][j] : dp[i][j-1]);
32         }
33     }
34
35
36     printf("%d\n", dp[n-1][n-1]);
37
38     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! ✓

RESULT :

The above program is executed successfully.

Q:5.C

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	2
azb	

ALGORITHM :

Step 1: Start

Step 2: Input two strings s1 and s2

Step 3: Calculate the lengths len1 of s1 and len2 of s2

Step 4: Initialize a 2D array dp of size (len1 + 1) x (len2 + 1)

Step 5: For each index i from 0 to len1, and each index j from 0 to len2, do Steps 6-8

Step 6: If i == 0 or j == 0, set dp[i][j] = 0

Step 7: If s1[i-1] == s2[j-1], set dp[i][j] = dp[i-1][j-1] + 1

Step 8: Otherwise, set dp[i][j] to the maximum of dp[i][j-1] and dp[i-1][j]

Step 9: Print dp[len1][len2]

Step 10: Stop

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[10],s2[10];
    scanf("%s",s1);
    scanf("%s",s2);
    int len1=strlen(s1);
    int len2=strlen(s2);
    int dp[len1 + 1][len2 + 1];
    for(int i=0;i<=len1;i++)
    {
        for(int j=0;j<=len2;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][j-1]>dp[i-1][j])
                    dp[i][j]=dp[i][j-1];
                else
                    dp[i][j]=dp[i-1][j];
            }
        }
    }
    printf("%d",dp[len1][len2]);
}
```

output

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

Passed all tests! ✓

Q:5.D**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 :

Step 1: Start

Step 2: Input an integer n

Step 3: Initialize an array arr of size n

Step 4: For each index i from 0 to n-1, input arr[i]

Step 5: Call subsequence(arr, n) and store the result in result

Step 6: Print result

7: Stop

Function subsequence(arr, n):

Step 1: Initialize an array dp of size n

Step 2: Set each element in dp to 1

Step 3: Initialize maxlen to 1

Step 4: For each index i from 1 to n-1, do Steps 5-7

Step 5: For each index j from 0 to i-1, if arr[i] >= arr[j] and dp[i] < dp[j] + 1, set dp[i] = dp[j] + 1

Step 6: If maxlen < dp[i], set maxlen = dp[i]

7: Return maxlen


```

#include <stdio.h>
int subsequence(int arr[],int n){
    int dp[n];
    int maxlen=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(arr[i]>=arr[j] && dp[i]<dp[j]+1){
                dp[i]=dp[j]+1;
            }
        }
        if(maxlen<dp[i]){
            maxlen=dp[i];
        }
    }
    return maxlen;
}

int main(){
    int n;
    scanf("%d",&n);
    int arr[n];
    for (int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    int result=subsequence(arr,n);
    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	✓

Passed all tests! ✓

RESULT :

The above program is executed successfully.