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

Register No.: 230701360 **Name: Tejushree sanjeevikumar**

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:

```
void function(int n){
    read n from input
    create array a of size n + 1

    set a[0] = 1
    for i = 1 to n:
        set a[i] = 0

    for i = 1 to n:
        increment a[i] by a[i - 1]
        if i >= 3:
            increment a[i] by a[i - 3]

    print a[n]
}
```

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. 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 maximum_monetary_value(n):
    read n from input
    create array A of size n * n
    create array dp of size n * n

    for i = 0 to n-1:
        for j = 0 to n-1:
```

```
        read A[i][j] from input

    set dp[0][0] = A[0][0]
    for i = 1 to n-1:
        set dp[i][0] = dp[i-1][0] + A[i][0]
    for j = 1 to n-1:
        set dp[0][j] = dp[0][j-1] + A[0][j]

    for i = 1 to n-1:
        for j = 1 to n-1:
            if dp[i-1][j] > dp[i][j-1]:
                set dp[i][j] = A[i][j] + dp[i-1][j]
            else:
                set dp[i][j] = A[i][j] + dp[i][j-1]

    print dp[n-1][n-1]
```

Program:

```
#include <stdio.h>

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

    int A[n][n];
    int dp[n][n];

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

```
    dp[0][0] = A[0][0];

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

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

    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] = A[i][j] + dp[i-1][j];
            } else {
                dp[i][j] = A[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	✔

5.c. 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:

```
function LCS(s1, s2):
    m = length of s1
    n = length of s2

    create 2D array dp with dimensions (m+1) x (n+1)

    for i = 0 to m:
        for j = 0 to n:
            if i == 0 or 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] = max(dp[i-1][j], dp[i][j-1])

    return dp[m][n]
```

Program:

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

int LCS(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[1000], s2[1000];
    scanf("%s", s1);
    scanf("%s", s2);
    printf("%d\n", LCS(s1, s2));
    return 0;
}
```

Output:

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

5.d. 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:

Function LongestNonDecreasingSubsequence(arr[], n):

1. Create an array dp[] of size n and initialize all elements to 1.

2. For i = 1 to n-1:

   a. For j = 0 to i-1:

      i. If arr[i] >= arr[j] (i.e., current element is greater than or equal to the previous element):

         - Set dp[i] = max(dp[i], dp[j] + 1).

3. Initialize maxLength = dp[0].

4. For i = 1 to n-1:

   a. If dp[i] > maxLength:

      i. Set maxLength = dp[i].

5. Return maxLength.

Program:

```
Function LongestNonDecreasingSubsequence(n):
    create array dp of size n
    for i = 0 to n-1:
        set dp[i] = 1


    for i = 1 to n-1:
```

```
    for j = 0 to i-1:
        if arr[i] >= arr[j]:
            set dp[i] = max(dp[i], dp[j] + 1)

set maxLength = dp[0]
for i = 1 to n-1:
    if dp[i] > maxLength:
        set maxLength = dp[i]

print maxLength
```

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	✔