Exp No: 5 **Date: 10.09.24**

Register NO : 230701366 **Name:- Y.Thrilok Chander**

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
Expla1n+at1io+n1:+ T1h+e1r+e1 a re 6 ways to 6 represent number with 1 and 3

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 countWays(n)
{
    initialize a of size n + 1 // Array to store the number of ways

    a[0] = 1 // Base case: 1 way to climb 0 stairs
    a[1] = 1 // Base case: 1 way to climb 1 stair

    if n >= 2
    {
        a[2] = 1 // Base case: 1 way to climb 2 stairs
    }

    if n >= 3
    {
        a[3] = 2 // Base case: 2 ways to climb 3 stairs
    }

    // Fill the array for all stairs from 4 to n
    for i from 4 to n
    {
        a[i] = a[i - 1] + a[i - 3] // Total ways to climb i stairs
    }

    return a[n] // Return the number of ways to climb n stairs
}

function main()
{
```

```
 initialize n // Number of stairs
    read n from user

    result = countWays(n) // Calculate the number of ways
    print result // Print the result

    return 0
}
```

Program:

```
#include <stdio.h>

long long int countWays(int n) {
    long long int a[n + 1];

    a[0]  =  1;
    a[1] = 1;



    if (n >= 2) {
        a[2] = 1;
    }

    if (n >= 3) {
        a[3] = 2;
    }

    for (int i = 4; i <= n; i++) {
```

```
        a[i] = a[i - 1] + a[i - 3];
    }

    return a[n];
}

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

    long long int result = countWays(n);
    printf("%lld",result);

    return 0;
}
```

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 max(a, b)
{
    return (a > b) ? a : b // Return the maximum of a and b
}

function maxMonetaryPath(n, board)
{
    initialize dp[n][n] // Array to store maximum monetary path sums
```

```
    dp[0][0] = board[0][0] // Starting point

    // Fill the first row
    for j from 1 to n - 1
    {
        dp[0][j] = dp[0][j - 1] + board[0][j]
    }

    // Fill the first column
    for i from 1 to n - 1
    {
        dp[i][0] = dp[i - 1][0] + board[i][0]
    }

    // Fill the rest of the dp table
    for i from 1 to n - 1
    {
        for j from 1 to n - 1
        {
            dp[i][j] = board[i][j] + max(dp[i - 1][j], dp[i][j - 1])
        }
    }

    return dp[n - 1][n - 1] // Return the maximum monetary path to the bottom-right corner
}

function main()
{
    initialize n // Size of the board
```

```
 read n from user

    initialize board[n][n] // Create the board array
    for i from 0 to n - 1
    {
        for j from 0 to n - 1
        {
            read board[i][j] from user
        }
    }

    result = maxMonetaryPath(n, board) // Calculate the maximum monetary path
    print result // Print the result
}
```

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. 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. 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: