DYNAMIC PROGRAMMING

M.MOHANA

231901031

1.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

Solution:

```
#include <stdio.h>

int main() {

  int n;

  scanf("%d", &n);

  long long dp[1001];

  dp[0] = 1;

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

    dp[i] = 0;

    if (i - 1 >= 0)

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

```
        if (i - 3 >= 0)
            dp[i] += dp[i - 3];
    }
    printf("%lld\n", dp[n]);
    return 0;
}
```

2. 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.

Solution:

```
 #include <stdio.h>
int main() {
    int n;
    scanf("%d", &n);
    int board[100][100];
    int dp[100][100];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &board[i][j]);
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == 0 && j == 0)
```

```
                dp[i][j] = board[i][j];
            else if (i == 0)
                dp[i][j] = dp[i][j - 1] + board[i][j];
            else if (j == 0)
                dp[i][j] = dp[i - 1][j] + board[i][j];
            else
                dp[i][j] = (dp[i - 1][j] > dp[i][j - 1] ? dp[i - 1][j] : dp[i][j - 1]) + board[i][j];
        }
    }

    printf("%d\n", dp[n - 1][n - 1]);

    return 0;
}
```

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

Solution:

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

// Function to find max of two numbers
int max(int a, int b) {
    return (a > b) ? a : b;
}

int main() {
    char s1[100], s2[100];
    scanf("%s", s1);  // First string
    scanf("%s", s2);  // Second string

    int m = strlen(s1);
    int n = strlen(s2);

    int dp[100][100]; // dp[i][j] = LCS of first i chars of s1 and first j chars of s2

    // Build the dp table
    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] = 1 + dp[i - 1][j - 1];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }

    printf("%d\n", dp[m][n]);  // Print the length of LCS

    return 0;
}
```

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

Solution:

```c
#include <stdio.h>

int main() {

    int n;

    scanf("%d", &n);

    int a[1001];

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

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

    }

    int dp[1001];

    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 (a[i] >= a[j] && dp[i] < dp[j] + 1) {

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

            }

        }

    }

    int maxLen = 0;

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

        if (dp[i] > maxLen)

            maxLen = dp[i];

    }

    printf("%d\n", maxLen);
    return 0;
}
```