DP
Lesson 3

1. Bell Numbers
2. Program for nth Catalan Number
3. Geek-and-his-binary-strings
4. Stickler-thief
5. Max-sum-without-adjacents
6. Adjacents-are-not-allowed
7. paths-to-reach-origin Two Dimensional DP
8. optimal-walk
9. number of ways to place n rooks on an n x n chessboard in such a way that no two rooks attack each other, and in such a way that the configuration of the rooks is symmetric under a diagonal reflection of the board.??
telephone-number-or-involution-number
10. Maximize-dot-product
11. Count-all-possible-paths-from-top-left-to-bottom-right
12. Number-of-substrings-divisible-by-8-but-not-by-3
13. number-of-subsequences-in-a-string-divisible-by-n
14. Check-if-any-valid-sequence-is-divisible-by-m

Max Sum without Adjacents

Given an array Arr of size N containing positive integers. Find the maximum sum of a subsequence such that no two numbers in the sequence should be adjacent in the array.

Example 1:

Input:
N = 6
Arr[] = {5, 5, 10, 100, 10, 5}
Output: 110
Explanation: If you take indices 0, 3and 5, then Arr[0]+Arr[3]+Arr[5] =5+100+5 = 110.
Example 2:

Input:
N = 4
Arr[] = {3, 2, 7, 10}
Output: 13
Explanation: 3 and 10 forms a non contiguous subsequence with maximum sum.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(1)

Constraints:

$1 \leq N \leq 106$

$1 \leq Arri \leq 107$

```
int findMaxSum(int *arr, int n)
    {
        int t[n+1];
    t[0]=0;
    t[1]=arr[0];
    t[2]=arr[0]>arr[1]?arr[0]:arr[1];

    for(int i=3;i<=n;i++)
    {
        //either dont take the previous value or take it and add max value before it
        t[i]=max(t[i-1],arr[i-1]+t[i-2]);
    }
    return t[n];
        // code here
    }
```

Paths to reach origin

You are standing on a point (n, m) and you want to go to origin (0, 0) by taking steps either left or down i.e. from each point you are allowed to move either in (n-1, m) or (n, m-1). Find the number of paths from point to origin.

Example 1:

```
Input:

N=3, M=0
Output: 1
Explanation: Path used was -
             (3,0)  --> ( 2,0) --> (1,0) --> (0,0).
             We can see that there is no other path
             other than this path for this testcase.
```

Example 2:

```
Input:
N=3, M=6
Output: 84
```

Expected Time Complexity: O(N*M).

Expected Auxiliary Space: O(N*M).

Constraints:

$1 \leq N, M \leq 500$

```
int mi=1000000007;
  int ways(int n, int m)
  {
      int dp[n+1][m+1];

      // Fill entries in bottommost row and leftmost
      // columns
      for (int i=0; i<=n; i++)
        dp[i][0] = 1;
      for (int i=0; i<=m; i++)
        dp[0][i] = 1;
```

```

    // Fill DP in bottom up manner
    for (int i=1; i<=n; i++)
      for (int j=1; j<=m; j++)
        dp[i][j] = (dp[i-1][j]%mi + dp[i][j-1]%mi)%mi;

    return dp[n][m]%mi;
    //code here.
}
```

Maximize Dot Product

Given two arrays A and B of positive integers of size N and M where N >= M, the task is to maximize the dot product by inserting zeros in the second array but you cannot disturb the order of elements.

Dot Product of array A and B of size N is A[0]*B[0] + A[1]*B[1]+....A[N]*B[N].

Example 1:

Input: N = 5, A[] = {2, 3, 1, 7, 8}
M = 3, B[] = {3, 6, 7}
Output: 107

Explanation: We get maximum dot product after inserting 0 at first and third positions in second array.Maximum Dot Product : = A[i] * B[j] 2*0 + 3*3 + 1*0 + 7*6 + 8*7 = 107

Example 2:

Input: N = 3, A[] = {1, 2, 3}
M = 1, B[] = {4}
Output: 12

Explanation: We get maximum dot product after inserting 0 at first and second positions in the second array. Maximum Dot Product : = A[i] * B[j]
1*0 + 2*0 + 3*4 = 12

Expected Time Complexity: O(N*M)

Expected Auxiliary Space: O(N*M)

Constraints:

1 ≤ M ≤N ≤ 103

1 ≤ A[i], B[i] ≤ 103

```
int maxDotProduct(int n, int m, int A[], int B[])
        {
          long long int dp[n+1][m+1];
    memset(dp, 0, sizeof(dp));

    // Traverse through all elements of B[]
    for (int i=0; i<=n; i++)
    {
      // Consider all values of A[] with indexes greater
      // than or equal to i and compute dp[i][j]
      for (int j=0; j<=m; j++)
      {
          if(j==0)
          {
            dp[i][j]=0;
          }
          else if(i==0)
          {
            dp[i][j]=INT_MIN;
          }
          else
        // Two cases arise
        // 1) Include A[j]
        // 2) Exclude A[j] (insert 0 in B[])
        dp[i][j] = max((dp[i-1][j-1] + (B[j-1]*A[i-1]))
,dp[i-1][j]);

      }
    }
      // return Maximum Dot Product
    return dp[n][m] ;
              // Your code goes here
        }
```

Bell Numbers

Given a set of n elements, find a number of ways of partitioning it.

Example 1:

Input:
N = 2
Output: 2
Explanation:
Let the set be
{1, 2}:
{ {1}, {2} }
{ {1, 2} }

Example 2:

Input:
N = 3
Output: 5
Expected Time Complexity: O(N^2)

Expected Auxiliary Space: O(N^2)

Constraints:

$1 \leq N \leq 1000$

```
long long m=1000000007;
 int bellNumber(int n)
  {
     int bell[n+1][n+1];
       bell[0][0] = 1;
       for (int i=1; i<=n; i++)
       {
         bell[i][0] = bell[i-1][i-1]%m;
         for (int j=1; j<=i; j++)
           bell[i][j] = (bell[i-1][j-1]%m + bell[i][j-1]%m)%m;
       }
       return bell[n][0]%m;
    // Code Here
  }
```