**Operating Systems Project 4**

**University at Albany**

**Department of Computer Science**

**ICSI 500**

Problem 1

#include <bits/stdc++.h>

using namespace std;

// Function to find number of subarrays

// with sum exactly equal to k.

int findSubarraySum(int arr[], int n, int sum)

{

// STL map to store number of subarrays

// starting from index zero having

// particular value of sum.

unordered\_map<int, int> prevSum;

int res = 0;

// Sum of elements so far.

int currsum = 0;

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

// Add current element to sum so far.

currsum += arr[i];

// If currsum is equal to desired sum,

// then a new subarray is found. So

// increase count of subarrays.

if (currsum == sum)

res++;

// currsum exceeds given sum by currsum

// - sum. Find number of subarrays having

// this sum and exclude those subarrays

// from currsum by increasing count by

// same amount.

if (prevSum.find(currsum - sum) != prevSum.end())

res += (prevSum[currsum - sum]);

// Add currsum value to count of

// different values of sum.

prevSum[currsum]++;

}

return res;

}

int main()

{

int arr[] = { 1,1,1,2,1 };

int sum = 3;

int n = sizeof(arr) / sizeof(arr[0]);

cout << findSubarraySum(arr, n, sum);

return 0;

}

Problem 2

We have to make B as a fixed subsequence.

1. These are base cases:

(a) We have to check whether both the are empty or not if iboth the string is empty then we stop the program and exit.

(b)If A is empty and B is not empty, we will return 0 because the non-empty string cannot be subsequence of empty string.

(c)If A is not empty and B is empty, then we will return 1, because the empty string is a subsequence.

2 Now after applying these base cases, There are two cases for this problem:

(a) When A[i-1]!=B[j-1]

when current character of A is not equal to curent character of B, we will not be able to increase the number of distinct subsequences.

So, DistinctSubsequence(i,j)=DistinctSubsequence(i-1,j)

(b) when A[i-1]==B[j]

In this current character of A and B are same. Now there are two condition to check

(i) We do not match the current two characters which means that it still has original number of distinct subsequence .

So,DistinctSubsequence(i,j)=DistinctSubsequence(i-1,j)

(ii) If we match both current character.

So, DistinctSubsequence(i,j)=DistinctSubsequence(i-1,j-1)

And finally we take both cases (i) and (ii)

DistinctSubsequence(i,j)=DistinctSubsequence(i-1,j)+DistinctSubsequence(i-1,j)

And it will return distinct number of subsequence.

Problem 3

**The below is the solution for the above problem.**

**Algorithm:**

**we can use the dynamic programming algorithm to solve the problem.here we need to maintain the numbers of paths from i,j to the m,n. and then for each m,n to 1,1 we need to add [x,y+1] + [x+1,y] if the [x,y] is 0**

**The below is the PYTHON 3 code for the above problem.**

# returns the total paths

def solve(maze):

m,n = len(maze),len(maze[0])

# maintains the total paths from start to end cell

dp = [[0 for j in range(n)] for i in range(m)]

dp[m-1][n-1] = 1

# finds the total paths

for i in range(m-1,-1,-1):

for j in range(n-1,-1,-1):

if i == m-1 and j == n-1:

dp[i][j] = 1

elif maze[i][j] == 0:

if i + 1 < m:

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

if j + 1 < n:

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

# returns the total paths

return dp[0][0]

maze = [[0,0,0],[0,1,0],[0,0,0]]

print(solve(maze))

Problem 4

import java.util.Arrays;

public class A {  
   static long countWays(int S[], int m, int w) {  
       // Time complexity for this function is : O(mw)  
       // Space Complexity for this function is : O(w)

       // dp[i] will store the number of solutions  
       // for the value i. We need w+1 rows as the table is  
       // constructed in bottom up manner using the base  
       // case (w = 0)  
       long[] dp = new long[w + 1];

       // Initialize all table values as 0  
       Arrays.fill(dp, 0); // O(w)

       // Base case (If given value is 0)  
       dp[0] = 1;

       // Pick all coins one by one and update the dp[]  
       // values after the index greater than or equal to  
       // the value of the picked coin  
       for (int i = 0; i < m; i++)  
           for (int j = S[i]; j <= w; j++)  
               dp[j] += dp[j - S[i]];

       return dp[w];  
   }

   // This is driver Function to test above function  
   public static void main(String args[]) {  
       int c[] = { 1, 2, 4 };  
       int m = c.length;  
       int w = 4;  
       System.out.println(countWays(c, m, w));  
   }  
}