Dynamic Programming - 1

**Coin change:**

**BruteForce Approach:**

class Solution {

public int coinChange(int[] coins, int amount) {

if(coins == null || coins.length == 0){

return 0;

}

return helper(coins, amount, 0 ,0);

}

private int helper(int[]coins, int amount, int index, int minCoins){

//base case

if(amount == 0)

return minCoins;

if(amount<0||index>=coins.length)

return -1;

//choose

int case1 = helper(coins, amount-coins[index],index,minCoins+1);

//don't choose

int case2 = helper(coins,amount,index+1,minCoins);

if(case1 == -1) return case2;

if(case2 == -1) return case1;

return Math.min(case1,case2);

}

}

The Time Complexity is O(m+n), where m = amount, n = denomination count

The Space complexity is O(m+n)

DP Approach:

class Solution {

public int coinChange(int[] coins, int amount) {

if(coins == null || coins.length == 0){

return 0;

}

int [][] dp = new int[coins.length+1][amount+1];

//filled first row

for(int j = 1; j<=amount;j++){

dp[0][j] = amount+1;

}

for(int i = 1; i<=coins.length;i++){

for(int j = 1; j<=amount; j++){

//amount< denomination

if(j<coins[i-1])

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

//choose,don't choose

else{

dp[i][j]= Math.min(dp[i-1][j], dp[i][j-coins[i-1]]+1);

}

}

}

return dp[coins.length][amount] == amount+1?-1:dp[coins.length][amount];

}

}

The Time Complexity is O(MN)

The Space Complexity is O(MN).

**Rob House:**

**BruteForce Approach:**

class Solution {

public int rob(int[] nums) {

if(nums == null || nums.length == 0)

return 0;

return helper(nums, 0 ,0);

}

private int helper(int[] nums, int index, int maxEarnings){

//base case

if(index >= nums.length) return maxEarnings;

//choose

int case1 = helper(nums, index+2,maxEarnings+nums[index]);

//not choose

int case2 = helper(nums,index+1, maxEarnings);

return Math.max(case1,case2);

}

}

**DP Approach:**

class Solution {

public int rob(int[] nums) {

if(nums == null || nums.length == 0)

return 0;

if(nums.length == 1)

return nums[0];

int choose = nums[0];

int skip = 0;

for(int i = 1;i< nums.length;i++){

int prevSkip = skip;

skip = Math.max(skip,choose);

choose = prevSkip+nums[i];

}

return Math.max(skip,choose);

}

}

The time Complexity is O(N).

The Space Complexity is O(N).