08/01/2024
Monday

* House Robber (leetcode-198)

I/P- nums = [2,7,9,3,1]

O/P- 12

Explanation- Rob house 1, 3 and 5.
Total amount = 12

• Code using recursion -

```
class Solution{
public:

int solveUsingRecursion(vector<int>&nums, int index)
{
    if (index >= nums.size()) {
        return 0;
    }
    //include
    int include= nums[index]+ solveUsingRecursion(nums,
                                                  index +2);

    //exclude
    int exclude= 0+ solveUsingRecursion(nums, index +1);

    return max(include, exclude);
}
int rob (vector<int>&nums) {
    return solveUsingRecursion(nums,0);
  }
};
```

- Code using recursion + memoization

```
class Solution{
public:

int solveUsingMemo(vector<int>&nums, vector<int>&dp,
                   int index){
    if (index >=nums.size()){
        return 0;
    }
    // if ans already exists in dp array
    if (dp[index] != -1){
        return dp[index];
    }
    //include
    int include= nums[index]+ solveUsingMemo(nums,dp,index+2);
    //exclude
    int exclude = 0+ solveUsingMemo(nums, dp, index+1);

    // store & return using dp array
    dp[index]= max(include, exclude);
    return dp[index];
}

int rob(vector<int>&nums){
    // create dp array
    vector<int> dp(nums.size(), -1);
    return solveUsingMemo(nums, dp, 0);
}
};
```

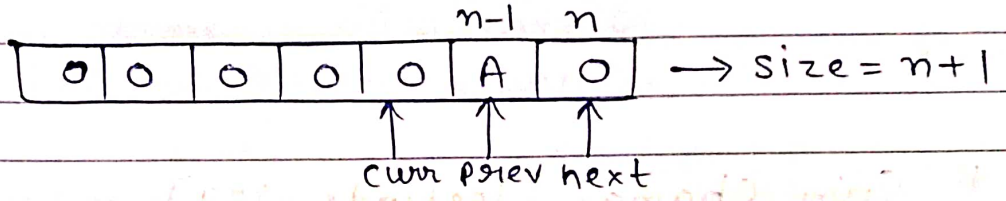
- Code using Tabulation

```
class Solution{
public:

int solveUsingTabu (vector<int>& nums){
      int n= nums.size();
      //create dp array
      vector<int> dp (n+1, 0);
      //maintain initial state of array
      dp[n-1]= nums[n-1];
      //process remaining array
      for (int index=n-2; index>=0; index--){
         int include = nums[index]+dp[index+2];
         int exclude = 0 + dp[index+1];
         dp[index] = max(include, exclude);
      }
      return dp[0];
}

int rob (vector<int>&nums) {
      return solveUsingTabu(nums);
   }
};
```

Note - Ye below code mein prev return kr rahe hai, curr return krne se if nums array ka size 1 hai to fss jayega and loop mein nhi jayega. So, glt // ans aayega.

- **Code using space optimisation**

From tabulation approach, we can see that the answer for current index depends upon two values i.e. the answeer at (index+2) and (index+1).



So, the value at index curr depends upon the values at indices prev and next. Rather than using array, we can use here two variables prev and next.

Code -

```
class Solution {
public:

int solveUsingSO(vector<int> &nums){
    int n= nums.size();
    int prev= nums[n-1];
    int next=0;
    int curr=0;

    for(int i=n-2; i>=0; i--){
        int include = nums[i] +next;
        int exclude = 0+ prev;
        curr= max(include, exclude);

        next=prev;
        prev=curr;
    }
    return prev;
}
```

```
int rob(vector<int>& nums) {
        return solveUsingSO(nums);
    }
};
```

* **Coin Change (leetcode - 322)**

Pattern - Explore all ways

I/P → coins = [4, 3, 6], amount = 12

O/P → 2

Explanation - minimum number of coins required → 6 + 6 = 12 (2 coins) as these coins are present in infinite numbers.

I/P → coins = [3, 6], amount = 10

O/P → -1

Explanation - Can't make the given amount from the given coins.

- Code using recursion

```
class Solution{
public:

int solveUsingRec(vector<int>& coins, int amount){
    if(amount==0) return 0;
    if(amount<0) return INT_MAX;

    int mini=INT_MAX;
    for(auto currCoin: coins){
     int ans= solveUsingRec(coins, amount-currCoin);
     if(ans!=INT_MAX) mini=min(ans+1,mini);
    }
    return mini;
}

int coinChange(vector<int>&coins, int amount){
    int ans= solveUsingRec(coins, amount);
    if(ans==INT_MAX) ans=-1;
    return ans;
  }
};
```

- Code using recursion + memoization

```
class Solution{
public:

int solveUsingMemo(vector<int>&coins, vector<int>&dp,
                   int amount)
{
    if (amount == 0) return 0;
    if (amount < 0) return INT_MAX;

    // if ans already exists
    if(dp[amount] != -1) {
        return dp[amount];
    }

    int mini = INT_MAX;
    for(auto currCoin : coins) {
        int ans = solveUsingMemo(coins, dp, amount - currCoin);
        if (ans != INT_MAX) {
            mini = min(ans+1, mini);
        }
    }
    // store & return using dp array
    dp[amount] = mini;
    return dp[amount];
}
int coinChange(vector<int>&coins, int amount) {
    vector<int> dp(amount+1, -1);
    int ans = solveUsingMemo(coins, dp, amount);
    if (ans != INT_MAX) return ans;
    else return -1;
}
};
```

- Code using tabulation

```
class Solution {
public:

int solveUsingTabu (vector<int> &coins, int amount){

    //create dp array
    vector<int> dp (amount + 1, INT_MAX);

    //analyse base case and maintain initial
    //state of array
    dp[0] = 0;
    //fill remaining array
    for(int index=1; index <= amount ; index++){
        int mini= INT_MAX;
        for (auto currCoin : coins) {
            if (index - currCoin >= 0) {
            int ans= dp[index - currCoin];
            if (ans != INT_MAX){
                mini= min (ans+1, mini);
            }
            }
        }
        dp[index] = mini;
    }
return dp[amount];
}
int coinChange (vector<int> &coins, int amount) {
        int ans = solveUsingTabu (coins, amount);
        if(ans == INT_MAX) ans= -1;
        return ans;
    }
};
```