**Dynamic Programming**

**3 Important point to remember**

* **Try to represent the problem in terms of indexes.**
* **Try all possible choices/ways at every index according to the problem statement.**
* **If the question states**
  + **Count all the ways - return sum of all choices/ways.**
  + **Find maximum/minimum- return the choice/way with maximum/minimum output.**

Q.1 <https://www.geeksforgeeks.org/problems/introduction-to-dp/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=introduction-to-dp>

* So for Fibonacci here I used top-down that is memorization or recursion used
* For bottom up I used for loop iterative approach that is tabulation

|  |
| --- |
| #define mod 1000000007  class Solution {  public:  long long int helper(int n,vector<long long int> &dp)  {  if(n==0) return 0;  if(n==1) return 1;    if(dp[n]!=-1) return dp[n];    dp[n]=(helper(n-1,dp)+helper(n-2,dp))%mod;    return dp[n];  }  long long int topDown(int n) {  // code here  vector<long long int> dp(n+1,-1);    return helper(n,dp);        }  long long int bottomUp(int n) {  // code here    long long a=0;  long long b=1;  long long c;    if(n==0) return 0;  if(n==1) return 1;    for(int i=2;i<=n;i++)  {  c=(a+b) % mod;  a=b;  b=c;  }    return b;      }  };  //{ Driver Code Starts.  int main() {  int t;  cin >> t;  while (t--) {  int n;  cin >> n;  Solution obj;  long long int topDownans = obj.topDown(n);  long long int bottomUpans = obj.bottomUp(n);  if (topDownans != bottomUpans) cout << -1 << "\n";  cout << topDownans << "\n";  }  }  // } Driver Code E |

TC:O(N) SC:O(N)

Q.2 <https://leetcode.com/problems/climbing-stairs/description/>

* Same logic like fibonnaci series when you’ll do space optimization return last value that is c=a+b then return a+b

|  |
| --- |
| class Solution {  public:      int helper(int n,vector<int> &dp)      {          if(n==1 || n==0) return 1;          if(dp[n]!=-1) return dp[n];          int left=climbStairs(n-1);          int right=climbStairs(n-2);          dp[n]=left+right;          return dp[n];      }      int climbStairs(int n) {            // vector<int> dp(n+1,-1);          // return helper(n,dp);          int a=0;          int b=1;          for(int i=2;i<=n;i++)          {              int c=a+b;              a=b;              b=c;          }          return a+b;      }  }; |

TC:O(N) SC:O(N) in space optimization SC:O(1)

Q.3 <https://www.naukri.com/code360/problems/frog-jump_3621012?leftPanelTabValue=PROBLEM>

* So here we have to take 1 or 2 jump to reach n-1 step
* So we can use recursion
* Base condition ind==0 return 0
* Int left=call ind-1+abs(height[ind]-height[ind-1])
* Int right=INT\_MAX
* If ind > 1 because if we won’t give this then ind will be at 1 then ind-2 will give -1 so that’s why this constraint for second call
* Right =ind-2 call+ abs(height[ind-2]-height[ind]);
* Return min(left,right)
* Just do changes and do memorization and tabulation and space optimization

|  |
| --- |
| #include <bits/stdc++.h>  int helper(int ind,vector<int> &heights)  {      if(ind==0) return 0;      int left=helper(ind-1, heights)+abs(heights[ind]-heights[ind-1]);      int right=INT\_MAX;      if(ind > 1)      {           right=helper(ind-2,heights)+abs(heights[ind]-heights[ind-2]);      }      return min(left,right);  }  int Memo(int ind,vector<int> &heights,vector<int> &dp)  {      if(ind==0) return 0;      if(dp[ind]!=-1) return dp[ind];      int left=Memo(ind-1, heights,dp)+abs(heights[ind]-heights[ind-1]);      int right=INT\_MAX;      if(ind > 1)      {           right=Memo(ind-2,heights,dp)+abs(heights[ind]-heights[ind-2]);      }      return dp[ind]= min(left,right);  }  int frogJump(int n, vector<int> &heights)  {      // Write your code here.      vector<int> dp(n,0);       // return helper(n-1,heights);      // Memo(n-1,heights,dp);      // return dp[n-1];      // dp[0]=0;      // for(int ind=1;ind<n;ind++)      // {      //     int left=dp[ind-1]+abs(heights[ind-1]-heights[ind]);      //     int right=INT\_MAX;      //     if(ind > 1)      //     {      //         right=dp[ind-2]+abs(heights[ind]-heights[ind-2]);      //     }      //     dp[ind]=min(left,right);      // }      // return dp[n-1];      int step1=0,step2=0;      for(int ind=1;ind<n;ind++)      {          int left=step1+abs(heights[ind-1]-heights[ind]);          int right=INT\_MAX;          if(ind > 1)          {              right=step2+abs(heights[ind]-heights[ind-2]);          }          int curr=min(left,right);          step2=step1;          step1=curr;      }      return step1;  } |

Recursion : TC- O(N) SC-O(N)

Memorization: TC-O(N) SC-O(N)

Tabulation: TC-O(N) SC-O(N)

Space opti : TC-O(N) SC-O(1)

Q.4 <https://www.geeksforgeeks.org/problems/minimal-cost/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=minimal-cost>

* same as above logic just used for loop for k jumps

|  |
| --- |
| class Solution {  public:  int helper(int ind,vector<int> &heights,int k)  {  if(ind==0) return 0;  int minCost=INT\_MAX;    for(int j=1;j<=k;j++)  {  if(ind-j >=0)  {  int cost=helper(ind-j,heights,k)+abs(heights[ind]-heights[ind-j]);  minCost=min(cost,minCost);  }  }    return minCost;  }  int Memo(int ind,vector<int> &heights,int k,vector<int>&dp)  {  if(ind==0) return 0;    if(dp[ind]!=-1) return dp[ind];  int minCost=INT\_MAX;    for(int j=1;j<=k;j++)  {  if(ind-j >=0)  {  int cost=Memo(ind-j,heights,k,dp)+abs(heights[ind]-heights[ind-j]);  minCost=min(cost,minCost);  }  }    dp[ind]=minCost;    return dp[ind];  }  int minimizeCost(vector<int>& arr, int& k) {  // Code here  int n=arr.size();  vector<int> dp(n,-1);  // helper(n-1,arr,k);  // Memo(n-1,arr,k,dp);    dp[0]=0;    for(int i=1;i<n;i++)  {  int mmSteps=INT\_MAX;  // int j=1;  for(int j=1;j<=k;j++)  {  if(i-j >=0)  {  int jump = dp[i-j] + abs(arr[i]-arr[i-j]) ;  mmSteps=min(jump,mmSteps);  }  }  dp[i]=mmSteps;  }    return dp[n-1];    }  }; |

Recursion : TC- O(N\*k) SC-O(N)

Memorization: TC- O(N\*k) SC-O(N)

Tabulation: TC- O(N\*k) SC-O(N)

Q.5 <https://www.naukri.com/code360/problems/maximum-sum-of-non-adjacent-elements_843261?leftPanelTabValue=PROBLEM>

* here what we did is just used the logic of pick and not pick and used some constraints like ind==0 then return nums[0]
* if ind <0 return 0
* for picking that value we pick up and call for ind-2 because we don’t want adjacent
* not pick we haven’t pick up anthing just called on ind-1
* return the max (pick,notpick)

|  |
| --- |
| #include <bits/stdc++.h>  int helper(vector<int>&nums,int ind)  {      if(ind==0) return nums[0];      if(ind < 0) return 0;      int left=nums[ind]+helper(nums,ind-2);      int right=0+helper(nums,ind-1);      return max(left,right);  }  int Memo(vector<int>&nums,int ind,vector<int>& dp)  {      if(ind==0) return nums[0];      if(ind < 0) return 0;      if(dp[ind]!=-1) return dp[ind];      int left=nums[ind]+Memo(nums,ind-2,dp);      int right=0+Memo(nums,ind-1,dp);      dp[ind]=max(left,right);      return dp[ind];  }  int maximumNonAdjacentSum(vector<int> &nums){      // Write your code here.      int n=nums.size();      vector<int> dp(n,-1);      // return helper(nums,n-1);  //    return Memo(nums,n-1,dp);      dp[0]=nums[0];      int left,right;      for(int i=1;i<n;i++)      {          left=nums[i];          if(i>1)          {              left+=dp[i-2];          }          right=0+dp[i-1];          dp[i]=max(left,right);      }      return dp[n-1];    } |

Recursion : TC- O(N) SC-O(N)

Memorization: TC-O(N) SC-O(N)

Tabulation: TC-O(N) SC-O(N)

Q.6 <https://leetcode.com/problems/house-robber-ii/description/>

* same logic as above just shuttle change is here houses are in circle and last and first house are adjacent so we can’t pickup both in same subsequence
* so use same logic as above but create 2 seprate input array one which doesn’t contain first element and second which doesn’t contain last element
* and pass those array to recursive or memo or tab function

|  |
| --- |
| class Solution {  public:      int helper(vector<int>&nums,int ind)  {      if(ind==0) return nums[0];      if(ind < 0) return 0;      int left=nums[ind]+helper(nums,ind-2);      int right=0+helper(nums,ind-1);      return max(left,right);  }  int Memo(vector<int>&nums,int ind,vector<int>& dp)  {      if(ind==0) return dp[0]=nums[0];      if(ind < 0) return 0;      if(dp[ind]!=-1) return dp[ind];      int left=nums[ind]+Memo(nums,ind-2,dp);      int right=0+Memo(nums,ind-1,dp);      dp[ind]=max(left,right);      return dp[ind];  }  int Tab(vector<int> &nums,int n)  {      vector<int> dp(n,-1);      dp[0]=nums[0];      int left,right;      for(int i=1;i<n;i++)      {          left=nums[i];          if(i>1)          {              left+=dp[i-2];          }          right=0+dp[i-1];          dp[i]=max(left,right);      }      return dp[n-1];    }      int rob(vector<int>& nums) {      int n=nums.size();      vector<int> dp(n-1,-1);      vector<int> temp1,temp2;      if(n==1) return nums[0];      for(int i=0;i<n;i++)      {          if(i!=0) temp1.push\_back(nums[i]);          if(i!=n-1) temp2.push\_back(nums[i]);      }      // return max(helper(temp1,n-2),helper(temp2,n-2));        int left=Memo(temp1,n-2,dp);      for(int i=0;i<n-1;i++) dp[i]=-1;      int right=Memo(temp2,n-2,dp);      return max(left,right);      // return max(Tab(temp1,n-1),Tab(temp2,n-1));      }  }; |

Recursion : TC- O(N) SC-O(N)

Memorization: TC-O(N) SC-O(N)

Tabulation: TC-O(N) SC-O(N)

Q.7 <https://www.naukri.com/code360/problems/ninja-s-training_3621003?leftPanelTabValue=PROBLEM>

* so basically here we have to provide 3 argument
* input array and ind also the last=previous element index which we took at starting of call we can provide 3 because nothing has take so it will calculate with both
* do recursion and memorization which task for loop which will run till 3 because there are 3 task and tabulation has 3 for loop

|  |
| --- |
| int helper(int ind,vector<vector<int>> &points,int last,vector<vector<int>> &dp)  {      if(ind==0)      {          int maxi=0;          for(int i=0;i<3;i++)          {              if(i!=last)              {                  maxi=max(maxi,points[ind][i]);              }          }          return maxi;      }      if(dp[ind][last]!=-1) return dp[ind][last];      int maxi=0;          for(int i=0;i<3;i++)          {              int point=0;              if(i!=last)              {                  point=points[ind][i]+helper(ind-1,points, i,dp);              }              maxi=max(maxi,point);          }          dp[ind][last]=maxi;          return dp[ind][last];  }  int ninjaTraining(int n, vector<vector<int>> &points)  {      // Write your code here.      vector<vector<int>> dp(n,vector<int>(4,0));      // return helper(n-1,points,3,dp);      dp[0][0]=max(points[0][1],points[0][2]);      dp[0][1]=max(points[0][0],points[0][2]);      dp[0][2]=max(points[0][0],points[0][1]);      dp[0][3]=max(points[0][1],max(points[0][2],points[0][0]));      for(int day=1;day<n;day++)      {          for(int last=0;last<4;last++)          {              // int maxi=0;              dp[day][last]=0;              for(int task=0;task<3;task++)              {                  if(task!=last)                  {                      int point=points[day][task]+dp[day-1][task];                      dp[day][last]=max(dp[day][last],point);                  }                }            }      }      return dp[n-1][3];  } |

TC:O(N\*4)\*3 recursion

SC:O(N)

TC:O(N\*4)\*3 Memorization

SC:O(N)+O(N\*4)

TC:O(N\*4)\*3 Tabulation

SC:O(n\*4)