DYNAMIC PROGRAMMING

Day 3

nCr calculation using DP

// nCr=(n!)/(r! *(n-r)!)

Pascal's triangle property

nCr=(n-1)Cr + (n-1)C(r-1)

precomputation

int dp[1000][1000]

dp[n][r]-> o(1)

dp[n][r]->nCr

1 (1C0)

1 2 1 (2c0,2c1,2c2)

1 3 3 1 (3c0 3c1 3c2 3c3)

1 4 6 4 1 (.....)

1 5 10 10 5 1 (...)

```
int dp[1000][1000];
dp[1][0]=1;
dp[2][0]=1; dp[2][1]=2;
dp[2][2]=1;
a[0]=1;
a[1]=2;
a[2]=1;
```

```
for(i=3; i<1000; i++){
   b[0]=1
     for(j=1; j<(i+1); j++){
          b[i]=a[j-1]+a[j];
}
b[i]=1;
a=b;
}
```

```
int ncr(int n, int r) {
  cout << dp[n][r];
}
```

r<=n nCr

Time complexity O(n*n)

n=10^3 icj i<=n

space o(n*n) ->->o(N)

nth row

a[]->1 row   ith row

b[]-> a ki help li   (i+1)th with help A

a=b (i+1 th row)

b ->

n=10^6

O(n)

mod=1e9+7;

(ncr %mod)
invfact[i]=(1/(i!))%mod;
=pow(i,mod-2)%mod;o(log(N)
invfact[n]
o(logn) +o(N) =o(N)
for(i=n-1; i>=0; i--) o(N)
invfact[i]=invfact[i+1]*(i+1)
1/i! =1/(i+1)!*(i+1) 1/I!
fact[n]*invfact[r]*invfact[n-r];
icj (i<=n j<=i)

Find maximum sum subarray

Link :

https://www.hackerrank.com/challenges/max subarray/problem

Eg. a=[1,5,-6,5,-8,9,-6]
maximum continuous sum
// brute force

```
ans=INT_MIN;
for(i=0; i<n; i++){
 int s=0;
for(j=i; j<n; i++){
s+=a[j];
if(s>ans){
   ans=s;
   }
}
}
```

Time complexity : O(N^2)
Better method using DP exists
a=[0,1,5,-6,5,-8,9,-6]

dp[i] -> maximum subarray sum ending at position i

int temp=0;

-1 -2 -3 -4

```
for(i=1; i<n; i++){
    dp[i]=dp[i-1]+a[i];
    if(dp[i]<0) dp[i]=0;
   ans=max(ans,dp[i]);
}
```

Time complexity : O(N)

Corner case when all numbers are negative (treat separately)

-1 -2 -3- 4

neg=n;

-1

s=0; 15 -8 7 -9 -2

1 2 3 4 5 -8 -9 -9 -9

-3 -5

ans=-inf; -3

-3 -5 -8

temp=0;

ans=0;

O(N) with constant space

// Kadane's algorithm

```
start=0;
end=0;
for(i=1; i<n; i++){
   temp+=a[i];
    if(temp<0) {
   temp=0;
     s=i+1;
}
   if(ans<temp){
     ans=temp;
```

```
    start=s;
    end=i;
}
}
```

subarray=[start ,end]

cout << ans << '\n';

Coin change problem (Min Coins)

Link :

https://www.hackerrank.com/contests/justcode/challenges/minimum-number-of-coins-for-possible-sum/problem

a=[1, 2, 5]
  value=23
greedy = 19+1+1+1 =(15+8)
fail
 Recursion
// recursion
// relation deal
// base case
// memorization
8
Iterative
  Decide knsi dp 2D iD
 us array ya vector usko
definition do
pichle subproblem ka use

```
int rec(int sum){
   if(sum==0) return 0;
     if(sum<0) return INT_MAX;
   if(dp[sum]!=-1)return dp[sum];
ans=1+min(rec(sum-1),rec(sum-2),
rec(sum-5));
return dp[sum]=ans;
}
```

ith

dp[-2]

int rec(int idx)

return a[idx]+recur(idx+1);

return recur(0)

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

dp[idx]=dp[idx-1]+a[idx]

return dp[n-1]

recursive [a0+[a1 a2 a3 a4 a5]]

iterative [[a0 a1 a2 a3 a4] a5]

iterative space 2D -> 1D DP

recursive

Knapsack problem

Link :

https://atcoder.jp/contests/dp/tasks/dp_d

n=3 wt=10

w v

10 10--->ans=10

5 6

5 5-->ans=11

->weight left in sack?

->items left?i objects

max profit??

n items..
nth item
(n-1)th item…
1st item
2d dp

i->1 to i still left..
Recursive DP

```
int dp[n+1][wt+1]
// initialise all values with -1
int   fun(i, rem)
{
// max profit now he can earn
if(i==0)//base case
 return 0;
```

```
if(dp[i][rem]!=-1)
   return dp[i][rem];
 int res=0;
 if(rem>=w[i])//imp condn
    res=fun(i-1,rem-w[i])+v[i];
//item pick

res=max(res,fun(i-1,rem));
//always valid → item unpicked

return dp[i][rem]=res;
}
```

n=2 wt=3

2 5-->rem,=1

3 5-->rem=-2,profit=10

Iterative DP

```
int dp[n+1][wt+1]
dp[0][i]==0
for(int i=0;i<=n;i++)
 for(int j=0;j<=wt;j++)
  dp[i][j]=0;
for(int i=1;i<=n;i++)
{
 for(int j=0;j<=wt;j++)
 {
dp[i][j]=dp[i-1][j]//item
unpicked..
if(j>=w[i])
dp[i][j]=max(dp[i-1][j],dp
[i-1][j-w[i]]+v[i]);
else
```

```
 dp[i][j]=dp[i-1][j]
}
}
```

Space optimised DP

```
int dp[2][wt+1];

for(int i=1;i<=n;i++)
{
 for(int j=0;j<=wt;j++)
 {
   int cur=i%2;
   int prev=1-cur;
```

```
   if(j>=w[i])
dp[cur][j]=max(dp[prev][j],dp[p
rev][j-w[i]]+v[i]);
else
 dp[cur][j]=dp[prev][j];
}
}
cout<<dp[(n%2)][wt]
```

i==1

dp[odd][wts..]

dp[even][wts..]-->cal

dp[odd][wts..]

int dp[n+1][wt+1]

→ O(n*wt) space complexity

int dp[2][wt+1];

→ O(wt) space only

Minimum insertions to sort

Given an array of integer numbers, we need to sort this array in a minimum number of steps where in one step we can insert any array element from its position to any other position.

eg.

Input :

[2, 3, 5, 1, 4, 7, 6]

Output : 3

3 insertions :

1 before array value 2

4 before array value 5

6 before array value 7

[1, 4, 3, 5, 2]

Ans = 2

[10, 2, 3, 4, 5, 6]

Ans = 1

Find the longest increasing subsequence (LIS).

Link :

https://www.hackerrank.com/challenges/longest-increasing-subsequent

Subsequence of an array : is a Sequence of elements of array such that order remains same but some elements may be deleted.

Eg.

A = [10, 2, 3, 4, 5, 6]

LIS = [2,3,4,5,6]

A = [2, 3, 5, 1, 4, 7, 6]

LIS = [2, 3, 4, 7] length = 4

So, minimum steps of sorting using insertion

= A.size() - LIS length

= 7 - 4

= 3

How to find length of LIS ?

Given array A[ ] of size n

```
int dp[n];
// dp[i] = length of the LIS ending at position i
for(int i=0; i<n; i++)
{
```

```
    dp[i] = 1;
}
for(int i=0; i<n; i++)
{
     for(int j=0; j<i; j++)
     {
     if (A[j] < A[i])
       dp[i] = max(dp[i], 1 +
dp[j]);
     }
}
 int ans=1;
for(int i=0; i<n; i++)
{
   ans= max(ans,dp[i]);
}
```

A = [10, 2, 3, 4, 5, 6]

N = 6

dp = [1, 1, 1, 1, 1, 1]

dp[0] = 1;

dp[1] = 1;

dp[2] = dp[1] + 1 = 2;

dp[3] = max (dp[1] + 1, dp[2] + 1) = max(1+1, 2+1) =3;

dp[4] =...

dp[5]= ...

Ans = max(dp[i]) = 3

A = [10, 2, 3, 4, 1, 0]

Dp[5] = Length of LIS ending at 5th position = 1

Ans = 3

Precaution :

```
#define int long long
int dp[1000000] ; // MLE

// Remove the #define int long long and it will work !
```

Overflows :

Case 1 :

```
 int b, c;
 long long A = b * c;
  // overflow occurs still !
```

How to prevent ?

Case 2 :

```
 long long b,c;
 long long A = b * c;
// overflow doesn't occur
```

Case 3 :

```
 int b,c;
 long long A = (long long) b * c;
// overflow doesn't occur
```

Best Wishes

(problem K in long challenge)

Greedy fails for n=11

By greedy -

11
10
5
4
2
1
1 2 4 5 10 11, 5 steps
However, better way is :
1 3 9 10 11, 4 steps
Therefore, greedy fails