(Q) arr_N, find length of LIS

(Longest increasing subsequence)

A_N: 2 10 14 3 7 15 8 11 20

$dp\{N\} = \{1\}$

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

ans = max(dp)

TC → $O(N^2)$

Here dp is not most optimized.

```
ls = {arr[0]}

for(int i=1; i<N; i++){
    if(arr[i] < ls[0])
        ls[0] = arr[i];
    elif (arr[i] > ls[len(ls)-1])
        ls.add(arr[i]);
    else{
        int idx = BS(ls, arr[i], 0, len(ls)-1)
        ls[idx] = arr[i];
    }
}
return ls.size();
```

LAB AGENDA

06/05/2025

Q $N C_R = \frac{N!}{(N-R)!\,R!}$

No. of ways of choosing R items out of N items

$$N C_R = {}^{N-1}C_{R-1} + {}^{N-1}C_R$$

```
int f(int n, int r){
    if(r==0) ret 1;
    if(n==0) ret 0;
    ret f(n-1,r-1) + f(n-1,r);
}
```

$^0C_0 = 1$

$^1C_0 = 1$

$^0C_1 = 0$

Memoization

```
Main:
    dp[N+1][R+1]

int f(int n, int r, int dp[][]){
    if(r==0) ret 1;
    if(n==0) ret 0;
    if(dp[n][r] != -1) return dp[n][r];
    dp[n][r] = f(n-1,r-1) + f(n-1,r);
    ret dp[n][r];
}
```

Tabulation

* int dp[ ][ ] = new int[n+1][r+1]
* $\forall_{i=0}^{N}$ dp[i][0] = 1; $\forall_{j=1}^{N}$ dp[0][j] = 0;
* $\forall_{i=1}^{N}$ $\forall_{j=1}^{R}$ [dp[i][j] = dp[i-1][j] + dp[i-1][j-1]]

TC → O(N × R) SC → O(N × R) SO → O(2 × R)

(Q)

w :	20	10	30
V :	100	60	120

K : 50

Knapsack

fractional Integer knapsack

0-1 KS ∞ KS

```
int f(int[] w, int[] v, int idx, int k){
    if(idx == -1) ret 0;
    int exclude = f(w, v, idx-1, k);
    int include = 0;
    if (k >= w[idx])
        include = f(w, v, idx-1, k - w[idx]) + v[idx];
    ret max(include, exclude);
}
```

Memoization

main:

```
dp[N+1][k+1] = {-1};
```

```
int f(int[] w, int[] v, int idx, int k, int[][] dp){
  if(idx == -1) ret 0;
  if(dp[idx][k] != -1) ret dp[idx][k];
  int exclude = f(w, v, idx-1, k, dp)
  int include = 0;
  if(k >= w[idx])
      include = f(w,v,idx-1, k-w[idx], dp)+v[idx];
  ret dp[idx][k] = max(include, exclude);
}
```

Tabulation

dp(N, k) = dp(i, j)

N → No. of weights; k → knapsack size

if i == 0 || j == 0

dp(i, j) = 0

0	0	0	0	0
0				
0				
0				

```
int dp[N+1][K+1] = {0};
for(int i=1; i<=N; i+1){
  for(int j=1; j<=K; j+1){
    exclude = dp[i-1][j];
    include = 0;
    if(j > w[i-1])
      include = dp[i-1][j-w[i-1]] + V[i-1];
    dp[i][j] = max(include, exclude);
  }
}
print(dp[N][K])
```

TC → O(N×K) SC → O(N×K) SO → O(2×K)

Q) A_N : 2 6 10 1 8 4 5

K : 11

Var-1

Check if you can get a change for K, using all possible A[i] denominations

```
bool f(arr, idx, k){
    if (idx == -1) rt k==0;
    bool ex = f(arr, idx-1, k);
    bool inc = False;
    if(arr[idx]<=k)
        inc = f(arr, idx-1, k-arr[idx]);
    rt ex||inc
}
```

(It is same as subset sum)

This DP soln will not work for subset sum in some cases like:

1. -ve numbers (logic doesn't work)
2. large numbers (array cannot be created)

↓

complexities

(1) $2^N \times N$ → Bitmasking

(2) 2^N → Backtracking

(3) $2^{N/2}$ → MITM

(4) NXK → DP

Tabulation

Memoization

```
bool f(arr, idx, k, int[][] dp) {
    if(dp[idx][k] != -1)
        ret dp[idx][k] == 1
    exc = f(arr, idx-1, k, dp);
    inc = false
    if(arr[idx] <= k)
        inc = f(arr, idx-1, k-arr[idx], dp);
    dp[idx][k] = (inc||exc) ? 1 : 0;
    ret (dp[idx][k] == 1);
}
```

We will not take boolean dp array because

Tabulation

bool dp[N+1][K+1] = {0};

T	F	F	F	F
T				
T				
T				

picking '0' from any coins is possible (True)

but picking a value 'k' from 0 coins is false

```
for(int i=1; i<=N; i++){
    for(int j=1; j<=K; j++){
        dp(i,j) = dp(i-1,j);
        if(arr[i-1] <= j)
            dp(i,j) = dp(i,j) || dp[i-1,j-arr[i-1]]
    }
}
print(dp[N][K]);
```

Var-2

Number of ways of getting a change K

Var-3

Min. no. of coins to get a change K

Q) String Partitioning

```
bool f(string st, Node root, int idx, int[] dp){
    if(idx == len(st)) return T;
    Node cur = root;
    if(dp[idx] != -1) return dp[idx] == 1);
    for(int i = idx; i < N ; i++){
        if(root.child[s[i] - 97] != null){
            root = root.child[idx];
            if(root.flag && f(st, arr, i+1, dp)) {
                dp[idx] = 1;
                return T;
            }
        else break;
    }
    dp[idx] = 0;
    return (dp[idx] == 1);
}
```

Q) Divide arr into 2 subsets A and B

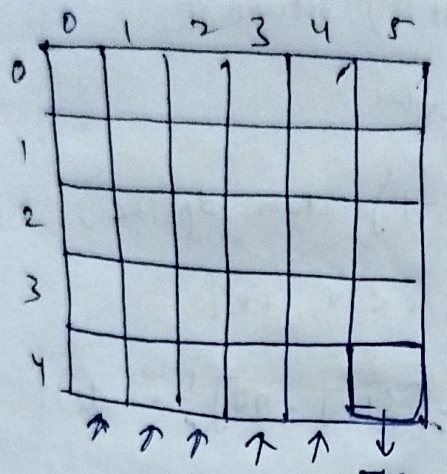
$\min |sum(A) - sum(B)|$

Arr_N: 2 6 5 1 7 10 20

min ans = 0

max ans = sum(Arr)

ans == 0 if sum of both subsets is TS/2 (TS → total sum)



This cell if True → there exists a subsets using 4 elements making sum 5

False → There is no subset with sum 5 If false we will go to check cell (4,4)

lets think
sum(Arr) = 10

If any cell in last row is true we get ans

↓

(The first occurrence of true is ans)

TC → N x k + k

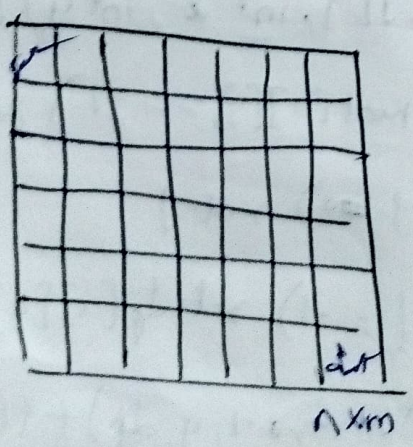
SC → N x k

SD → 2 x k

Tabulation

```
int dp[N][M] = {0};
dp[0][0] = mat[0][0] == -1 ? 0 : 1
for(int i=0 ; i<N; i++)
  for(int j=0; j<M; j++)
    if(mat[i][j] == -1 || (i==0 && j==0))
      continue
    up=0, left=0; d=0;
    if(i>0) up = dp[i-1][j];
    if(j>0) left = dp[i][j-1]
    if(i>0 && j>0) d = dp[i-1][j-1]
    dp[i][j] = up + left + d;
  }
}
```

(*)



every path has n+m-2 steps

if every path
there are n-1 down
and m-1 right steps
so it is

$$^{n+m-2}C_{n-1} \quad \text{or} \quad ^{n+m-2}C_{m-1}$$

$$= \frac{fact[n+m-2]}{fact[n-1]*fact[m-1]}$$

by fermant little theory to avoid overflow

~~fact of~~ $\frac{a}{x}$: $a \times pow(x, MOD-2)$

$MOD-2 = 1e9+5$

$$= \left(fac[n+m-2] \times pow\left(fact[n-1], MOD-2\right) * pow\left(fact[m-1], MOD-2\right)\right) \%M$$

mat
NXM:

-2 3 4 10 -1

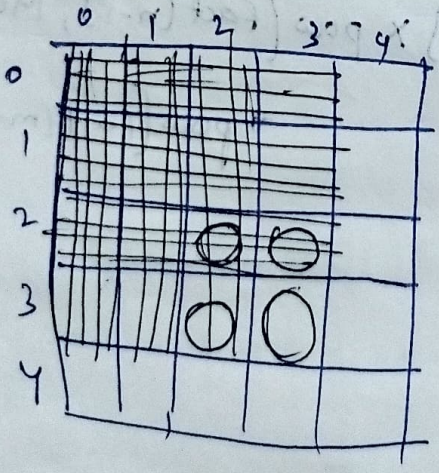
6	4	5	-1	1
11	6	2	1	1
9	6	8	6	9
-4	3	2	4	30

i j k l
1 2 4 3

(i,j,k,l) ⇒ sum(mat[i:j] : mat[k:l])

Solutions:

1. Q X N X M
2. N X M + Q X N → for queries
   ↳ PrefAx sum for each row
3. N X M + N X M + Q X 1
   ↓ row prefix sum, ↓ column prefix sum, ↓ queries



we want dp(i,j)

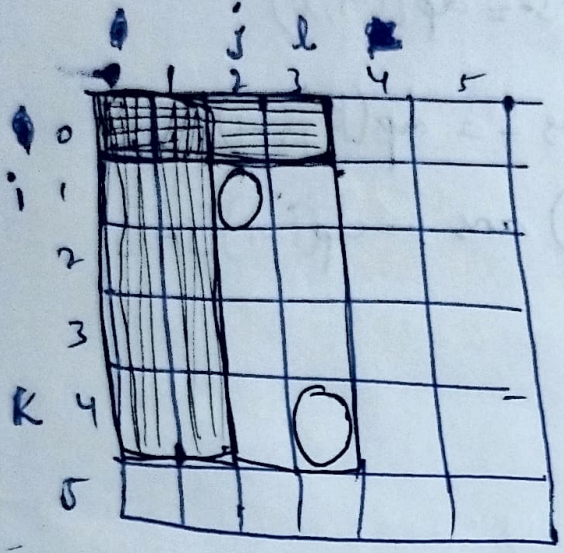
we have dp(i-1,j), dp(i,j-1)

but i-1 X j-1 portion is added twice.

dp;

$$dp[i,j] = dp(i-1,j) + dp[i,j+1] - dp.(i-1,j-1] + Mat[i,j]$$

for this soln



$$ans = dp(K,l) - dp(K,j-1) - dp(i-1,l) + dp(i-1,j-1)$$

Row wise / col wise pref sum)

(0,1)

```
dp[N][N] = {0}
for(int i=0; i<N; i++)
  for(int j=0; i<M; j++)
      dp[i][j] = mat(i,j);
      if(i>0) dp[i][j] += dp[i-1,j];
      if(j>0) dp[i][j] += dp[i,j-1];
      if(i>0 && j>0) dp[i][j] -= dp[i-1][j-1];
```

```
loop Q{
  read i,j,k,l;
  ans = dp(k,l)
  if(i>0) ans -= dp(i-1,l)
  if(j>0) ans -= dp(k,j-1)
  if(i>0 && j>0) ans += dp[i-1][j-1]
  print(ans);
}
```

(*)

2	-1	-3	4	-100	-20
3	1	-1	2	10	=3
-16	2	-3	4	5	-1
4	10	2	6	8	-6
-9	-3	4	9	8	3
-1	-2	-2	-1	-3	-4

Find largest submatrix sum.

Solutions

(1) $(NM)^2 + (NM), 1$

Brute force

② $NXM + N^2XM + NXM$

prefix column — Radares — space.

	0	1	2	3	4	5
0	2	-1	3	4	-100	-20
1	3	1	-1	2	10	-3
2	-10	2	-3	4	5	-1
3	4	10	2	6	8	-6
4	-9	-3	4	9	9	3
5	-1	-2	-2	-1	-3	-4

After performing col prefix, lets assume we fixed 1, 4 rows.

	0	1	2	3	4	5
0	2	-1	3	4	-100	-20
1	5	0	2	6	-90	-23
2						
3						
4	-10	9	5	25	-69	-27
5						

subs 4-0

Array: -12 10 2 21 31 -7

Radars: 0 10 12 33 64 54

max = 64

```
int ans = INT_MIN
for(int j=0; j<M; j++){
  for(int i=0; i<N; j++){
    dp(i,j) = dp(i-1,j) + mat(i,j)
              if(i>0)
  }
}

for(int i=0; i<N; i++)
{
  for(int j=i; j<N; j++)
  {
    int curr = INT_MIN; pre = 0;
    for(int k=0; k<M; k++){
      pre += dp(j,k);
      if(i>0) pre -= dp(i-1,k)
        curr = max(curr, pre);
      } pre = max(pre,0);
    ans = max(ans, cur);
  }
  print(ans);
}
```

- s1 : abcabcabc

  s2 : aabcbcxyz

Find longest common subsequence

```
int f(s1,s2,n,m){
   if(n<0 || m<0) ret 0;
   if(s1[n]==s2[m])
      ret 1+f(s1,s2,n-1,m-1);
   ret max(f(s1,s2,n-1,m), f(s1,s2,n,m-1));
}
```

// dp

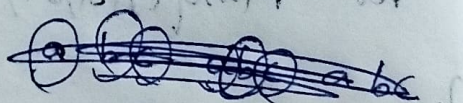
```
int f(s1,s2,n,m, int[][] dp){
if(n<0 || m<0) ret 0;
if(dp(n,m) != -1) ret dp(n,m);
if(s1[n]==s2[m])
  ret dp(n,m) = 1+f(s1,s2,n-1,m-1,dp)
dp(n,m) = max(f(s1,s2,n-1,m,dp), f(s1,s2,n,m-1,dp))
ret dp(n,m)
}
```

// Table

```
int dp[N+1][M+1] = {0};

for(i : 1 → N)

  for(j : 1 → M)

    if (s1[i-1] == s2[j-1])
      dp[i,i] = 1 + dp[i-1,j-1];
    else
      dp[i,j] = max(dp[i-1,j], dp[i,j-1]);
  }
}
dp[N,M] -ans
```

S: abcabcabc

find LPS



ret LCS(S, rev(S))

cba cba cba
abc abc abc

1. Edit distance
2. Egg Dropping
3. Building Bridges
4. Box stacking
5. LAS
6. Boolean Parenthesis
7. Optimal invigilation
8. LCS, LPS
9. Buy & sell stock
10. Stone game

1. Path in a matrix

2. Largest submatrix sum

3. Collect apples

4. LCS

5. LPS

6. Path in a matrix hard

7. Max square submatrix all 1's

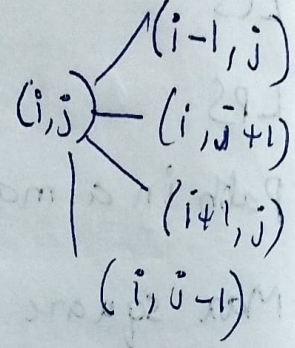
8. Max submatrix with all 1's

24/03/2025

(1) mat

-10	-3	-20	20
4	9	10	13
6	-20	6	12
10	0	-30	4
9	1	2	3

len of
Find longest
Increasing path
in a matrix




```
int f(i,j, mat, n, m){

    if(i>n || j>m) ret 0;
    if(i<0 || j<0) ret 0;
    if(mat[i-1, j] > mat[i,j])

        ans = max(ans, 1+f(i-1,i, mat,n,m))
    if(mat[i, j+1] > mat[i,j])

        ans = max(ans, 1+f(i,j+1, mat,n,m))
    if (mat[i+1,j] > mat[i,j])

        ans = max(ans, 1+f(i+1,j, mat,n,m))
    if (mat[i,j-1] > mat[i,j])

        ans = max(ans, 1+f[i,j-1, mat,n,m))

    ret ans;        // Here we may get
                       array out of bound
                       index error
}
```

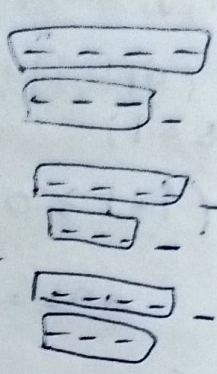
① Convert S1 to S2 using following operations

→ Insert any character

→ Replace any character

→ Delete any character

(in min operations)



```
f(s1,s2,n,m){ // for memoization pass dp
  if(dp[n,m] != -1) ret dp[n,m]
  if(n<0) ret m+1; if(m<0) ret n+1;
  if(s1[n] == s2[m])
      ret dp[n,m] = f(s1,s2,n-1,m-1);
  else{
      in = 1+f(s1,s2,n,m-1);
      del = 1+f(s1,s2,n-1,m);
      re = 1+f(s1,s2,n-1,m-1);
  }
  ret dp[n,m] = min(in,del,re);
}
```

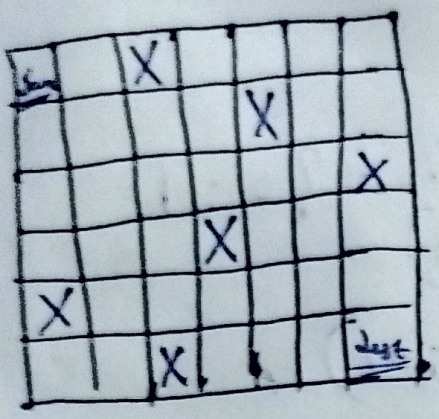
// code

```
// dp also shld be passed for memoization
int f(i,j,mat,n,m){
    if(dp[i,j] != -1) ret dp[i,j];
    ans = 1;
    if(i-1 >= 0 && mat[i-1,j] > mat[i,j])
        ans = max(ans, 1+f(i-1,j,mat,n,m))
    if(j+1 < m && mat[i,j+1] > mat[i,j])
        ans = max(ans, 1+f(i-1,j,mat,n,m))
    if(i+1 < n && mat[i+1,j] > mat[i,j])
        ans = max(ans, 1+f(i+1,j,mat,n,m))
    if(j-1 >= 0 && mat[i,j-1] > mat[i,j])
        ans = max(ans, 1+f(i,j-1,mat,n,m))
    dp[i,j] = ans;
    ret ans;
}
```

TC → O(N×M)

Tabulation requires 'Topo' sort

20/3/2025



Move from source to dest(n-1, m-1)
(0,0)

We can move
down (i+1, j) or
Right (i, j+1)

ways from src to dest.

```
int f(int mat[][], int x, int y){
 if(x<0||y<0) ret 0;
 if(mat[x][y]==-1) ret 0; // if dest is blocked.
 if(x==0 && y==0) ret 1;
 ret f(mat, x-1, y) + f(mat, x, y-1);
}
// memoization.
int f(int mat[][], int x, int y){
 if(x<0||y<0||mat[x][y]==-1) ret 0;
 if(x==0 && y==0) ret 1
 if( dp[x,y] != -1) ret dp[x][y];
 ret dp[x,y] = f(mat, x-1, y, dp) + f(mat, x, y-1, dp);
}
```