Que. 7 → Chocolate Pickup

Que. 1 Why we can't use greedy approch here?

⇒ Bcz of the uniformity nature of the greedy approch we can't apply greedy algo. i.e. It might happen a path in greedy but in futur when you consider a other path which not comes under the greedy approch this other path may gives you bigger values.
Since the values are not uniform some are small someone is very lagre & so on.

Other Approch

Now as Greedy approch not working so we have to travers/took all the paths so for all path we apply recursion.

Intution :-

(All path by Alice + All path by bob) if some path are common so consider them as single path and maximise your result

Imp:→ why we don't apply individually recursion for Alice & bob that mean 1st apply the recursion for Alice & then for bob and get the ans & sum up.

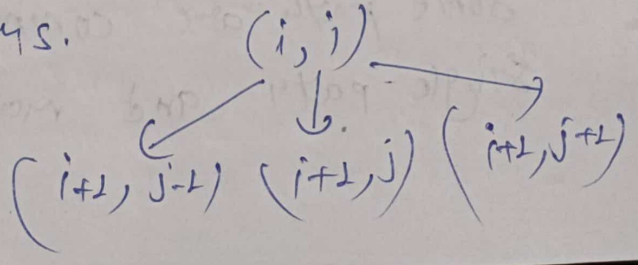
⇒ we have to write the recursion and in order to write the recurrance we have to make sure alice & bob move together why bcz it might happen there is a common cell that alice & bob passes so you have to consider it at once.

But in Individual recursion you have to trace the path then you have to subtract if there is anything common so it will be very long process if you follow that path

Rules to write recurrance

(1) previously we have only one guy so we write (i,j) to traverse on grid but here are Alice & bob two guys so for Alice (i_1, j_1) & bob (i_2, j_2).
And write down all the base cases.

(2) Explore all the paths.



(3) Give the maximum sum path

⇒ for Alice & bob their are fix starting point

Alice ⟶ (0,0)

bob ⟶ (0, m-1)

but their are multiple ending point.

So generally in these type of case we write recursion from starting point to ending point

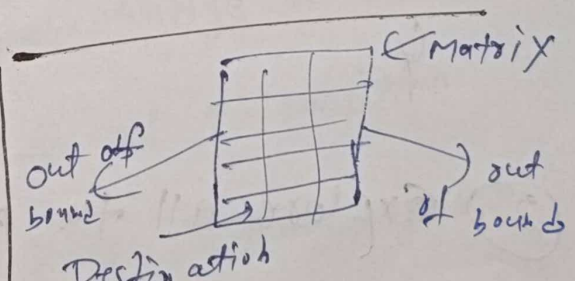
(i) express every thing in term of indexs (i1, j1) & (i2, j2) & write all the base cases.

starting point ⟶ f(0,0, 0,m-1)

Alice position, Bobs position

f(i1, j1, i2, j2)

```
// out of bound base case
{ if(j1<0 || j1>=m ||
     j2<0 || j2>=m)
   return -1e8;
```



There are two types of Base case

(1) Out of bound base case

(2) Destination Base case

// Alice & bob can reach destination at same time bcz Alice & Bob moving together that mean only column change or column can be different for Alice & bob but row will be same

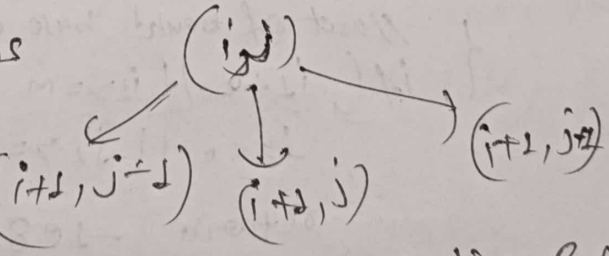
so don't consider j_1 & j_2 take it as j

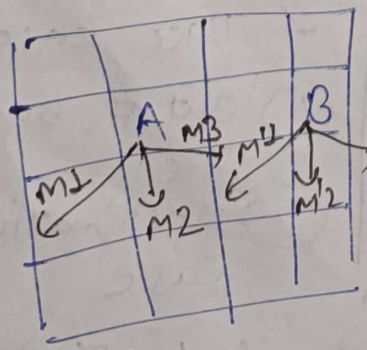
```
if( i == n-1)        // n-1 → last row
{
  // If Alice & bob reaches last row
  so their also some cases such as
  (i) Both are in different column
  (ii) ____ same ____
    if(j1 == j2) // same column so Ninja
                 can pick only
      return a[i][j1]; once all cho-
                       colate of this
                       column
  else
    return a[i][j1] + a[i][j2];
}
```

a[i][j1] → Ninja picks alice grid chocolate

a[i][j2] → Ninja picks bobs grid chocolate.

(2) Explore all the paths





In one step move alice & bob both together

when Alice move at M1 then bob can move M'1, M'2 or M'3

also when Alice move at M2

bob can move also M'1, M'2. & M'3 and when Alice move M3 so bob also move M'1, M'2, M'3

In total their are 3x3 = 9 combos of path.

How column will change if j=1 so we can go j-1=0, j=1, j+1=2 so [-1, 0, 1] but every time row chang is equal to row+1

```
// explore all path Alice & Bob can go together
    maxi = 0
for(dj1 -> -1 to +1) <- change in Alice column
{
    for(dj2 -> -1 to +1) <- change in Bob column
    int ans;
    // state change for all 9 transistion
    int state_change = f(i+1, j1+dj1, j2+dj2)
    // if alice & bob both are at the same cell
    if(j1==j2)
        ans = a[i][j1] + state_change;
    else ans = a[i][j1] + a[i][j2] + state_change;
} maxi = max(maxi, ans); return maxi;
```

$T.C. = O(3^n * 3^n)$

$S.C. = O(N)$

In memoization we go form 0 to n i.e. 1st row to nth row so in tabulation their is reverse

Convert Recursion to Memoization

Their are 3 parameter that are changing

(i, j1, j2)

maximum value that i can have → N

j1 → M

j2 → M

DP[N][M][M] → 3D DP

$T.C. = O(N * M * M) * g$ (g: no. of states)

$S.C. = O(N * M * M) + O(N)$

Convert Memoization to Tabulation

Step-1 analyses the Base cases & Store them into DP.

Tabulation is reverse of memoization i.e. memoizato 1st row to nth row is

Tabulation nth row 1st row.

dp[n][m][m]

for(j1 → 0 → m-1) → This for loop for the column of alice

{

for(j2 → 0 → m-1) → This for loop is for colum of bob

{

if(j1==j2)

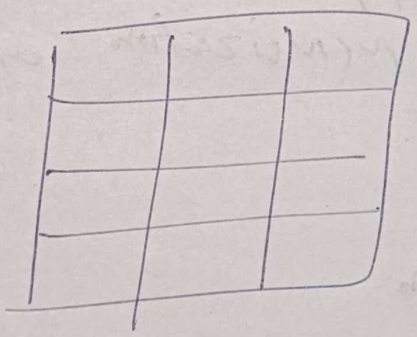
dp[n-1][j1][j2] = grid[n-1][j1];

else

dp[n-1][j1][j2] = grid[n-1][j1] + grid[n-1][j2];

}

}



This two for loop means if Alice is at j1 column so Bob can be at j2.

column j1 → (0 to m-1)

j2 → (0 to m-1)

Step-2 In Tabulation express every state in for loop

Their are 3 for loop i, j_1 & j_2

⇒ i → (n-2) to 0

j_1 → 0 to M-1

j_2 → 0 to M-1

```
for( i=n-2 to 0)
{
  for(j1=0 to M-1)
  {
    for(j2=0 to M-1)
    {
      // Copy the recurrance of
         Memoization approch.
    }
  }
}
```

Space Optimization

1D DP → 2 Variables

2D DP → 1D DP

3D DP → 2D DP