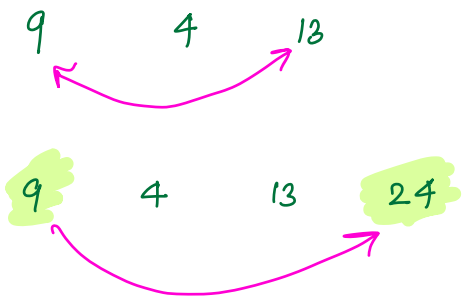
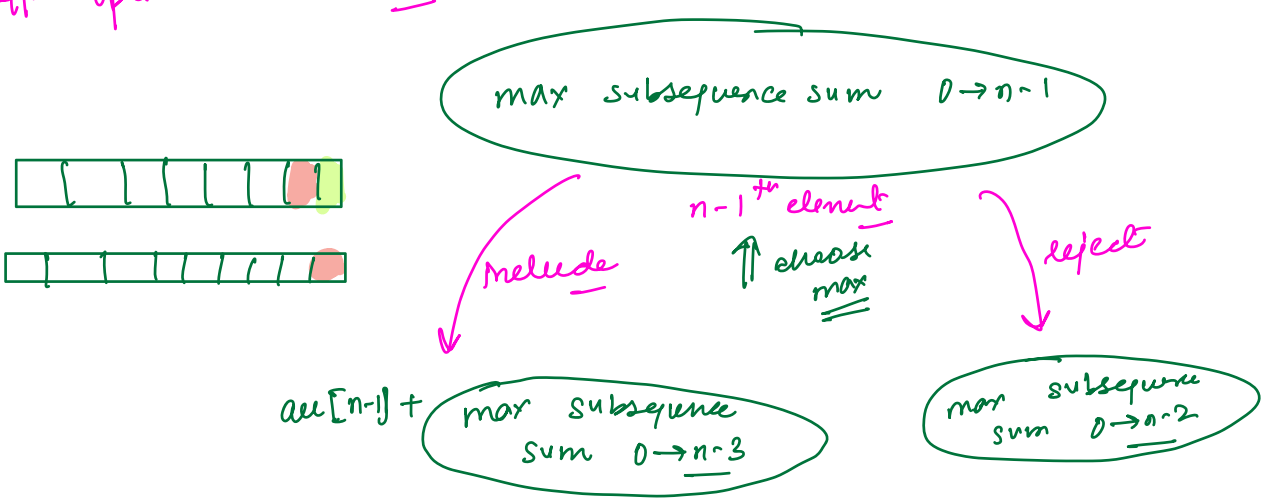
Q Given an array, find maximum subsequence sum
can't pick adjacent elements

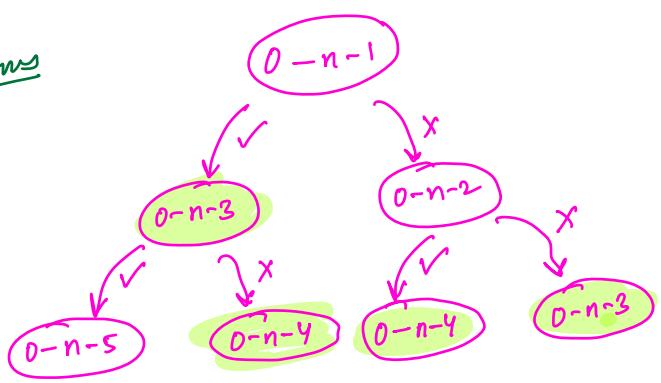


B-f / Basic idea: generate all valid subsequences

optimal substructure



overlapping subproblems



$$maxsum(i) = max(\ arr[i] + maxsum(i-2),\ maxsum(i-1))$$

int dp[n]; ⇒ pick initial value acc to constraints ↓ pick

```
int maxsum( int arr[], int index)
{
    if( index==0) return max(arr[0],0);
    if( index==1) return max(arr[0], arr[1],0);

    if( dp[index] != -1)
        return dp[index];

    dp[index] = max( arr[i] + maxsum(index-2),
                     maxsum(index-1));

    return dp[index];
}
```

something which definitely can't be your ans!

0 1

— —

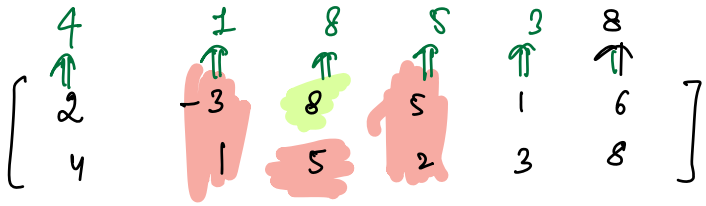
iterative

```
dp[0] = max( 0, arr[0]);
dp[1] = max( 0, arr[0], arr[1]);
for( i = 2; i<n; i++)
{
    dp[i] = max( arr[i] + dp[i-2], dp[i-1]);
}
```

ans = dp[n-1]

Q2. $2 * N$ matrix — find max sum you can get from matrix.

can't pick adjacent elements → hz
— vertically
→ diagonally



If you decide to pick a column → can't pick adjacent column

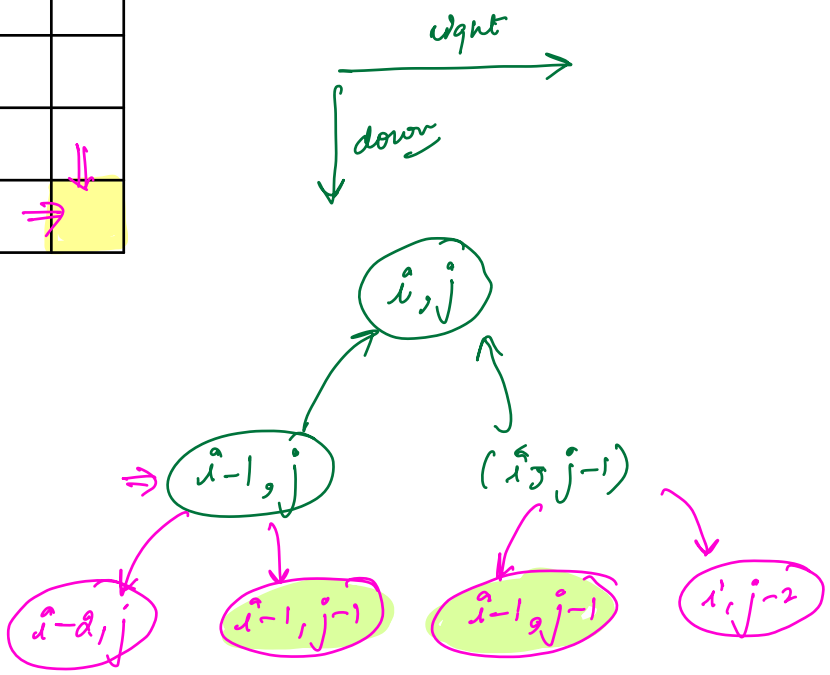
pick max element

39

	0	1	2	3
0	1	1	1	1
1	1			
2	1			
3	1			
4	1			

$n * m$

No of ways to go from (0, 0) to (n-1, m-1)



$ways(i, j) = ways(i-1, j) + ways(i, j-1)$

$dp[n][m]$

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

```
if( i==0 || j==0) dp[i][j]=1;
```

T.C: $O(n*m)$ S.C: $O(n*m)$

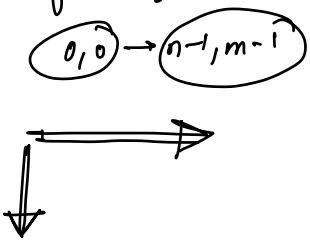
```
for( i=0 ; i<n ; i++)
{   for ( j=0; j<m ; j++)
    {
    }
}
```

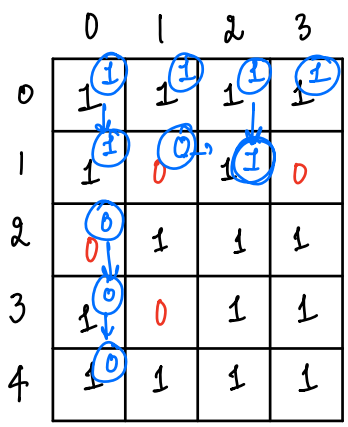
Q

	0	1	2	3
0	1	1	1	1
1	1	0	1	0
2	0	1	1	1
3	1	0	1	1
4	1	1	1	1

1 → open
0 → closed

no of ways





```
if (mat[i][j]==0)
    dp[i][j]=0;
```

dp[i][j] = dp[i-1][j] + dp[i][j-1]

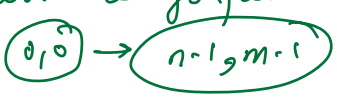
if (mat[i][j]==0) return 0;

#

	0	1	2	3	4
0	2	1	3	7	3
1	3	3	4	8	3
2	0	5	1	2	1
3	0	2	3	1	3
4	4	1	5	2	3

cost

min cost to go from



mincost(i,j) = cost[i][j] + min(mincost(i-1,j), mincost(i,j-1));

Q

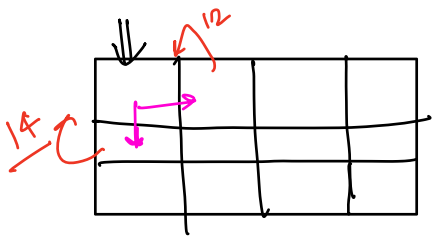
Dragons & princess

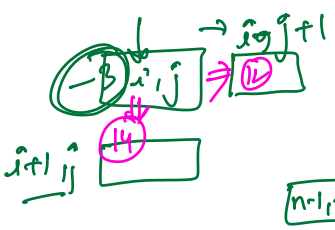
	0	1	2	3
0	-3	2	4	-5
1	-6	5	-4	6
2	-15	-7	5	-2
3	2	10	-3	-4

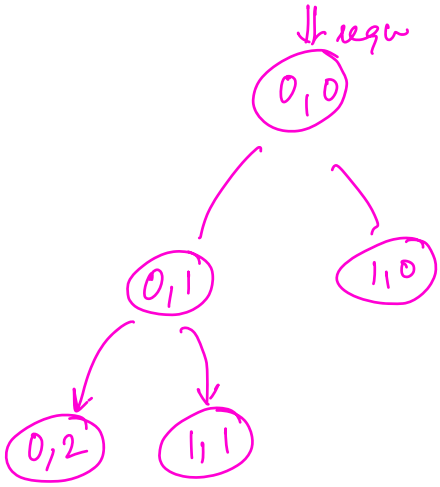
minimum initial energy should princess start with?

chamber → dragon
chamber → energy drink

die ← x=0 energy



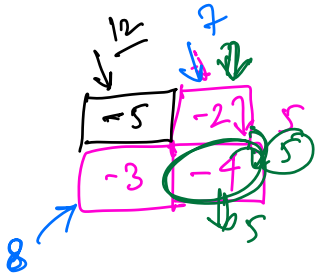




$$energy(i,j) + arr[i][j] = min(energy(i,j+1), energy(i+1,j))$$

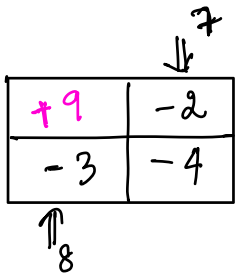
energy required to enter (n-1, m-1)

$$energy(i,j) = min(energy(i,j+1), energy(i+1,j)) - arr[i][j]$$



$5-(-2)=7$

$7-(-5)=12$



$\min(7,8)-9$

$=7-9$

$=-2 \Rightarrow 1$

$$dp[i][j] = \max\left(\min(dp[i+1][j], dp[i][j+1]) - arr[i][j], 1\right)$$

$$dp[n-1][m-1] = \max(1 - arr[i][j], 1)$$

```
for( i=n-1; i>=0; i--)
 {  for (j=m-1; j>=0; j--)
        {
        }
 }
```