

A Recursive Function

```

FindOPT(int i,int j, bool[][] rocks, bool[][] eggs)
    if(i<0 || j < 0) return -∞
    if(rocks[i][j]) return -∞
    if(i==0 && j==0) return eggs[0][0]
    int left = FindOPT(i-1,j,rocks,eggs)
    int down = FindOPT(i,j-1,rocks,eggs)
    return Max(left,down) + eggs[i][j]

```

Pseudocode

```

int eggsSoFar=0;
Boolean rocksInWay=false
for(int x=0; x<c; x++)
    if(rocks[x][0]) rocksInWay = true
    eggsSoFar+=eggs[x][0]
    OPT[x][0]= rocksInWay ? -∞ : eggsSoFar

eggsSoFar=0
rocksInWay=false
for(int y=0; y<r; y++)
    eggsSoFar+=eggs[0][y]
    OPT[0][y]= rocksInWay ? -∞ : eggsSoFar

for(int y=0;y<r;y++)
    for(int x=0;x<c;x++)
        if(rocks[x][y])
            OPT[x][y]=-∞
        else
            OPT[x][y]=max(OPT[x-1][y], OPT[x][y-1])+eggs[x][y]

```

Updating the Problem

$OPT(i, j, f)$ is the maximum amount of eggs Baby Yoda can collect on a legal path from (i, j) to $(0, 0)$ using the force f times to knock over rocks.

For simplicity, assume there are no rocks at the starting location $(r-1, c-1)$

Here was the old rule without the force – how do we update?

$$OPT(i, j) = \begin{cases} -\infty & \text{if } rocks(i, j) \text{ is true} \\ -\infty & \text{if } i < 0 \text{ or } j < 0 \\ eggs(0, 0) & \text{if } i = 0 \text{ and } j = 0 \\ \max\{OPT(i - 1, j), OPT(i, j - 1)\} + eggs(i, j) & \text{otherwise} \end{cases}$$

Dynamic Programming Process

1. Define the object you're looking for
2. Write a recurrence to say how to find it
3. Design a memoization structure
4. Write an iterative algorithm