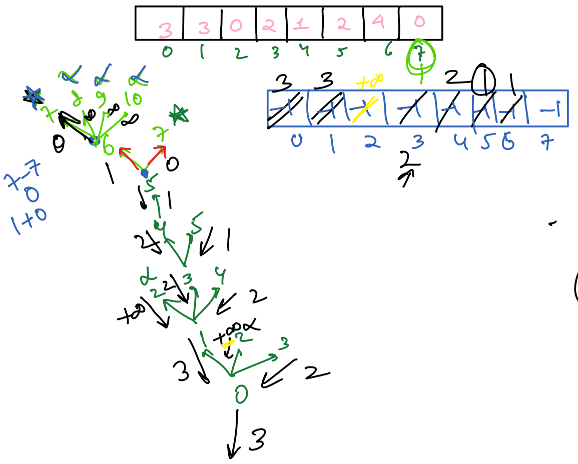
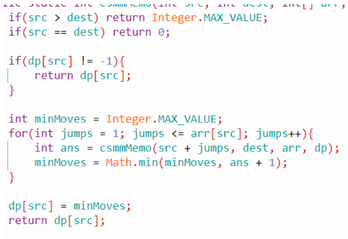
Climb Stairs with min moves

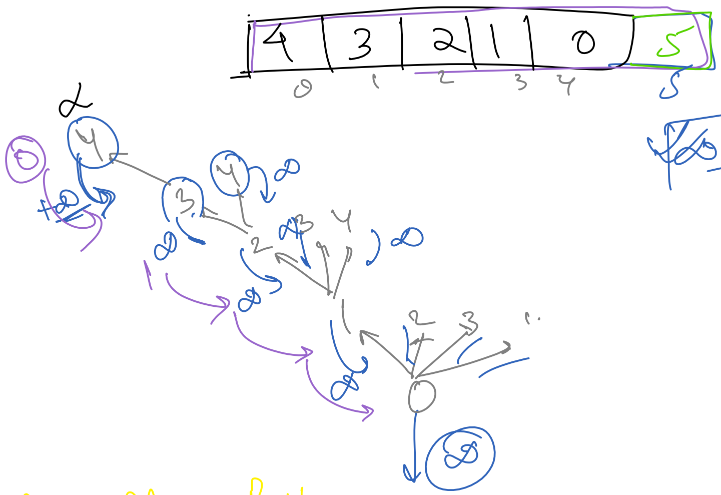




```
if(src > dest) return Integer.MAX_VALUE;
if(src == dest) return 0;

if(dp[src] != -1){
    return dp[src];
}

int minMoves = Integer.MAX_VALUE;
for(int jumps = 1; jumps <= arr[src]; jumps++){
    int ans = csmmMemo(src + jumps, dest, arr, dp);
    minMoves = Math.min(minMoves, ans + 1);
}

dp[src] = minMoves;
return dp[src];
```

$(\infty+1)$

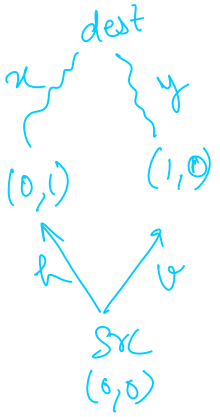
$\min(x+1, y+1, 3+1) \rightarrow 1 + \min(x, y, z)$

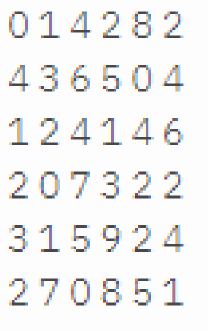


$\infty + 1 = -\infty$

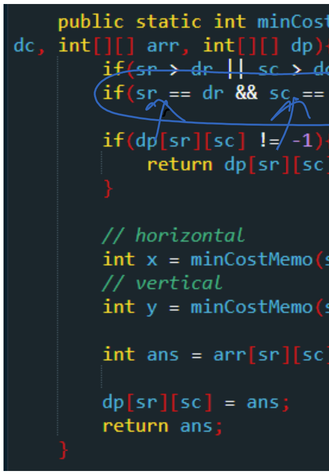
Minimum Cost Maze Path

```
fun(sr, sc, dr, dc, arr)
{
    int x = fun(sr, sc+1, dr, dc, arr)
    int y = fun(sr+1, sc, dr, dc, arr)
    arr[sr][sc] + min(x, y)
}
```



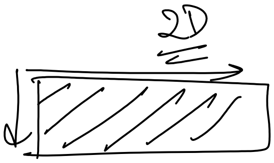


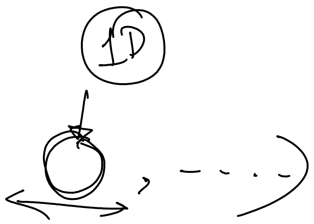
0	1	4	2	8	2
4	3	6	5	0	4
1	2	4	1	4	6
2	0	7	3	2	2
3	1	5	9	2	4
2	7	0	8	5	1



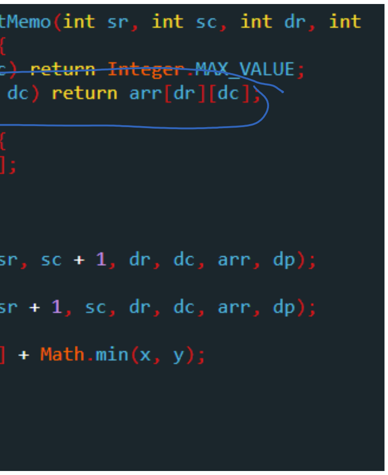
```
    public static int minCost
dc, int[][] arr, int[][] dp)
        if(sr > dr || sc > dc
        if(sr == dr && sc ==

        if(dp[sr][sc] != -1)
            return dp[sr][sc
        }

        // horizontal
        int x = minCostMemo(s
        // vertical
        int y = minCostMemo(s

        int ans = arr[sr][sc

        dp[sr][sc] = ans;
        return ans;
    }
```

2D





(3D DP) ⇐ dp[i][j][k]



```
tMemo(int sr, int sc, int dr, int
{
c) return Integer.MAX_VALUE;
 dc) return arr[dr][dc];

{
];



sr, sc + 1, dr, dc, arr, dp);

sr + 1, sc, dr, dc, arr, dp);

] + Math.min(x, y);
```

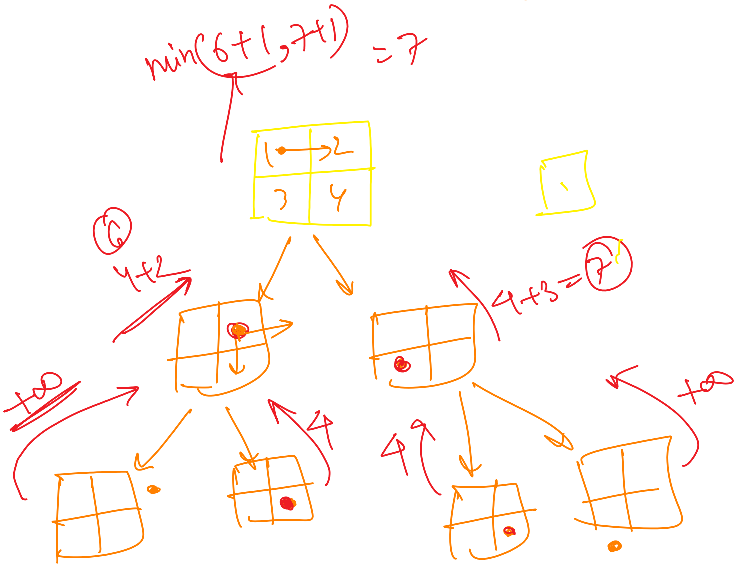
fun(x, y, z, ~~k1, k2, k3~~) ⟸ 3D DP ⟸ dp[i][j][k]

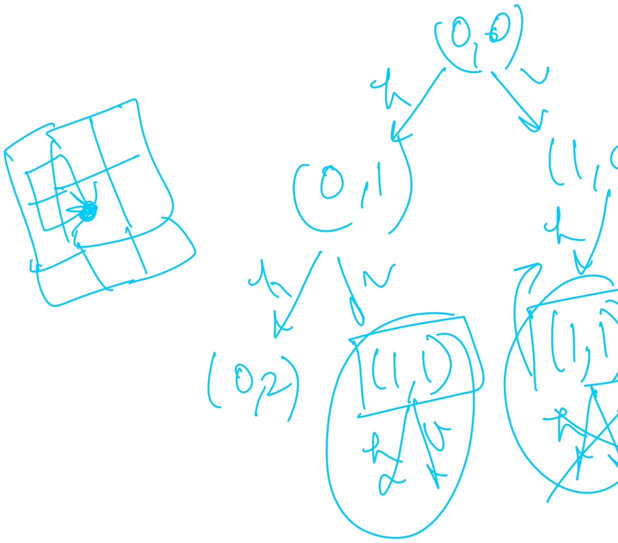
fun(x, y, ~~k1, k2, k3~~) ⟸ 2D DP ⟸ dp[i][j]

fun(x, k) ⟹ 1D DP ⟹ dp[i] =

meaning

dp[i][j] = min cost to travel from (i,j) to (n-1, m-1) using 2 moves (h/v)

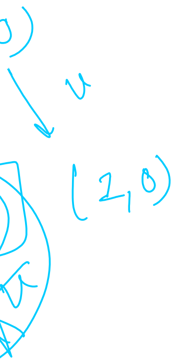




1 23	2 22	2 20	1 18	1 17
3 27	4 28	6 24	2 18	3 16
4 24	7 25	0 18	5 18	4 13
2 20	2 18	7 19	5 14	2 9
5 21	4 16	3 12	2 9	7 7

∞

∞



$= 9$

$\rightarrow h \geq x = 7$

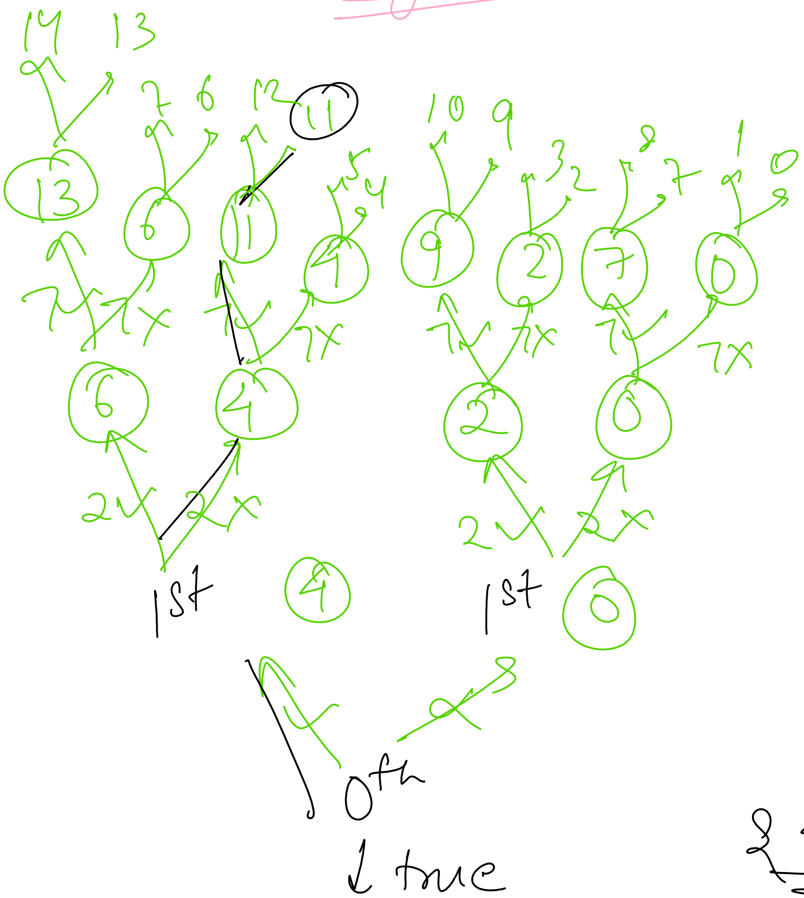
∞

∞

y=∞

Target Sum Subsets

{4, 2, 7, 1}



boolean f(arr, idx, (

{

}

{4, 7} = 11

Target = 11

curr_sum, target-sum)