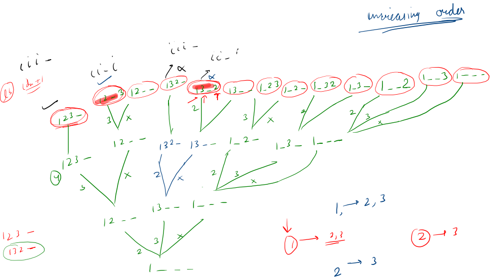
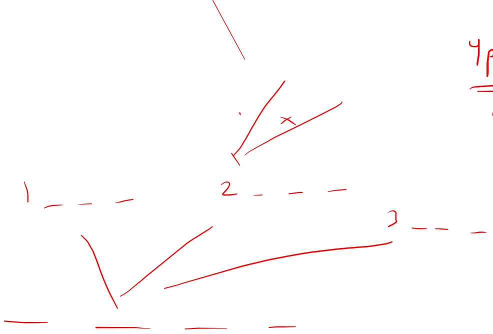
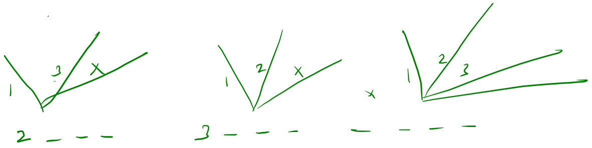


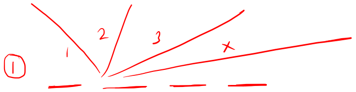


$^4P_3$

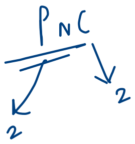
↳ combinations

4X3X2



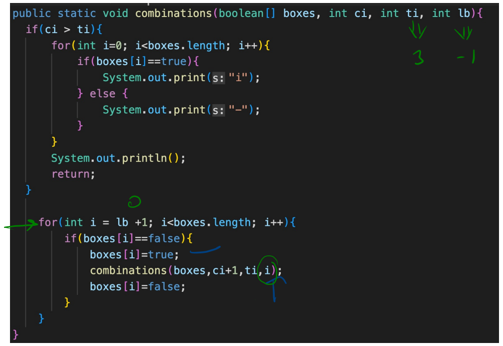


any doubts ?

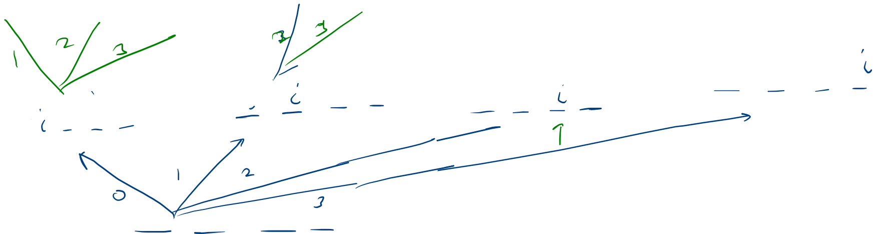


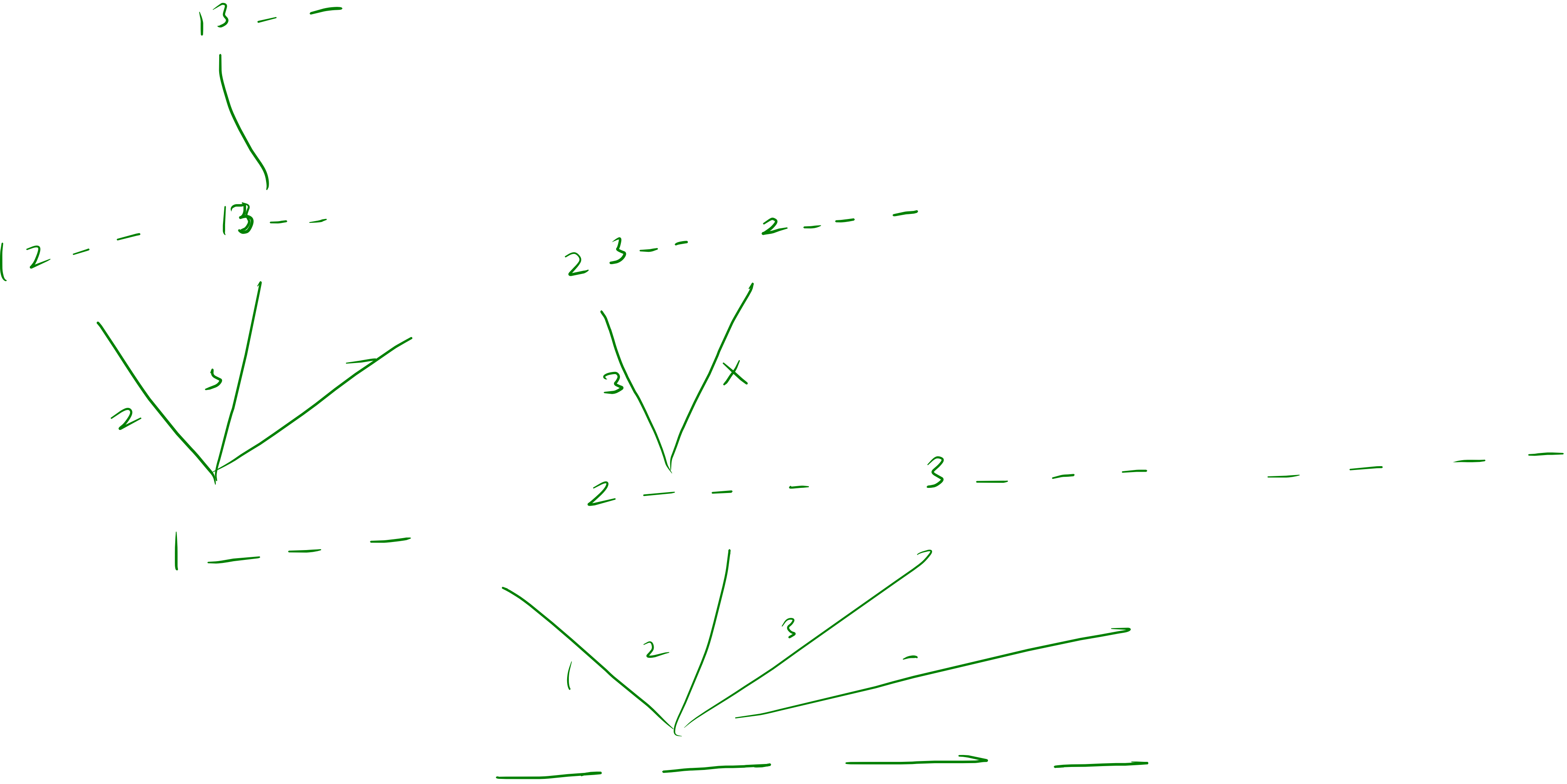
what
how
why

dry run
↓
watch the dry run



```
public static void combinations(boolean[] boxes, int ci, int ti, int lb){
  if(ci > ti){
      for(int i=0; i<boxes.length; i++){
          if(boxes[i]==true){
              System.out.print(s: "i");
          } else {
              System.out.print(s: "-");
          }
      }
      System.out.println();
      return;
  }

    for(int i = lb +1; i<boxes.length; i++){
        if(boxes[i]==false){
            boxes[i]=true;
            combinations(boxes,ci+1,ti,i);
            boxes[i]=false;
        }
    }
}
```

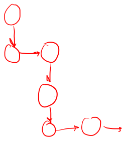


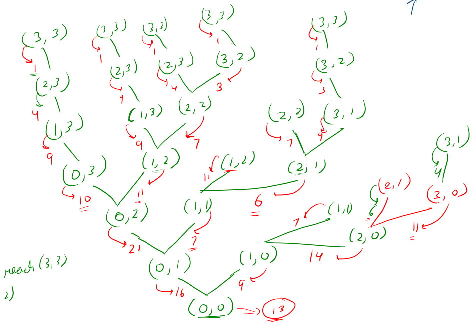

1 3 _ _
1 2 _ _
1 3 _ _
2 3 _ _
2 _ _ _
2
3
3
X
1 _ _ _
2 _ _ _
3 _ _ _
1
2
3
-

nxm

dp(i)(j)

rec(i,j)



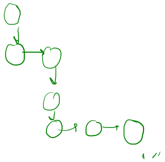


min Path

dp

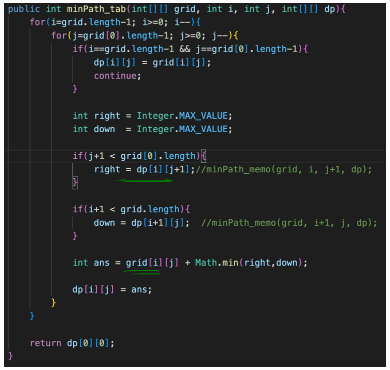
dp(1)(2)
min cost to reach (3,3)
from (1,2)

d r d d rr

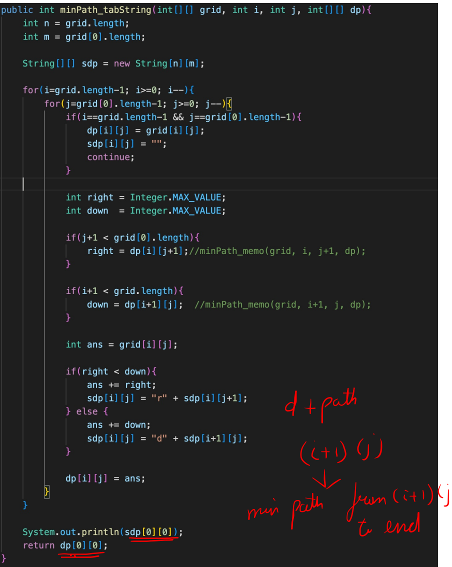


	0	1	2	3
0	13	16	21	10
1	9	7	11	9
2	14	6	7	4
3	11	4	3	1

dp



```
public int minPath_tab(int[][] grid, int i, int j, int[][] dp){
    for(i=grid.length-1; i>=0; i--){
        for(j=grid[0].length-1; j>=0; j--){
            if(i==grid.length-1 && j==grid[0].length-1){
                dp[i][j] = grid[i][j];
                continue;
            }

            int right = Integer.MAX_VALUE;
            int down  = Integer.MAX_VALUE;

            if(j+1 < grid[0].length){
                right = dp[i][j+1];//minPath_memo(grid, i, j+1, dp);
            }

            if(i+1 < grid.length){
                down = dp[i+1][j];  //minPath_memo(grid, i+1, j, dp);
            }

            int ans = grid[i][j] + Math.min(right,down);

            dp[i][j] = ans;
        }
    }

    return dp[0][0];
}
```

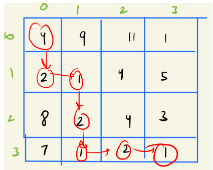


```
public int minPath_tabString(int[][] grid, int i, int j, int[][] dp){
    int n = grid.length;
    int m = grid[0].length;

    String[][] sdp = new String[n][m];

    for(i=grid.length-1; i>=0; i--){
        for(j=grid[0].length-1; j>=0; j--){
            if(i==grid.length-1 && j==grid[0].length-1){
                dp[i][j] = grid[i][j];
                sdp[i][j] = "";
                continue;
            }

            int right = Integer.MAX_VALUE;
            int down  = Integer.MAX_VALUE;

            if(j+1 < grid[0].length){
                right = dp[i][j+1];//minPath_memo(grid, i, j+1, dp);
            }

            if(i+1 < grid.length){
                down = dp[i+1][j];  //minPath_memo(grid, i+1, j, dp);
            }

            int ans = grid[i][j];

            if(right < down){
                ans += right;
                sdp[i][j] = "r" + sdp[i][j+1];
            } else {
                ans += down;
                sdp[i][j] = "d" + sdp[i+1][j];
            }

            dp[i][j] = ans;
        }
    }

    System.out.println(sdp[0][0]);
    return dp[0][0];
}
```

d + path

((i+1)(j))

min path from (i+1)(j)
to end



	0	1	2	3
0	4	9	11	1
1	2	1	4	5
2	8	2	4	3
3	7	1	2	1

sdp(i)(j)

min path from (i,j)
to (n-1,m-1)

r = 1
d = 4
ans = 2 + 4 ⇒ 6

sdp

	0	1	2	3
0	drddrr	dddrr	rddd	ddd
1	rddrr	ddrr	ddr	dd
2	rdrr	drr	dr	"d"
3	rrr	rr	r	" "

dp

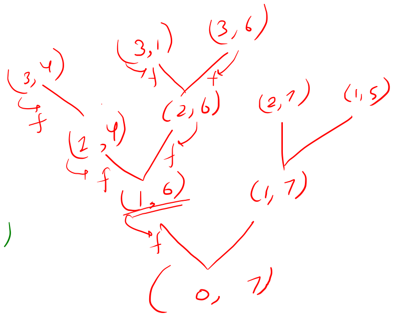
	0	1	2	3
0	13	16	21	10
1	9	7	11	9
2	14	6	7	4
3	11	4	3	1

break till 1:00

{1, 2, 5}
0 1 2

target = 7

-1	-1	0	-1	-1



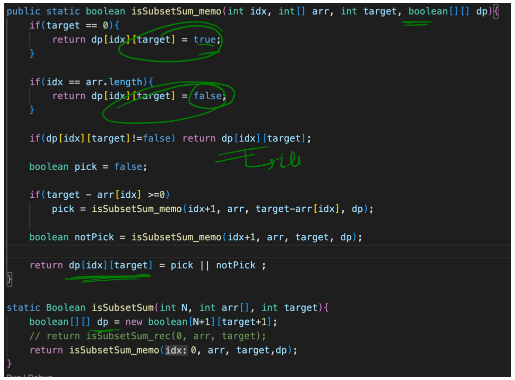
if(dp(idx) != -1)
return dp(idx)

(1,6)
→ $\boxed{f}$

true, false

idx = {0, n}
target = {0, target}

memo → element



```java
public static boolean isSubsetSum_memo(int idx, int[] arr, int target, boolean[][] dp){
    if(target == 0){
        return dp[idx][target] = true;
    }

    if(idx == arr.length){
        return dp[idx][target] = false;
    }

    if(dp[idx][target]!=false) return dp[idx][target];

    boolean pick = false;

    if(target - arr[idx] >=0)
        pick = isSubsetSum_memo(idx+1, arr, target-arr[idx], dp);

    boolean notPick = isSubsetSum_memo(idx+1, arr, target, dp);

    return dp[idx][target] = pick || notPick ;
}

static Boolean isSubsetSum(int N, int arr[], int target){
    boolean[][] dp = new boolean[N+1][target+1];
    // return isSubsetSum_rec(0, arr, target);
    return isSubsetSum_memo(idx: 0, arr, target,dp);
}
```

F → tu

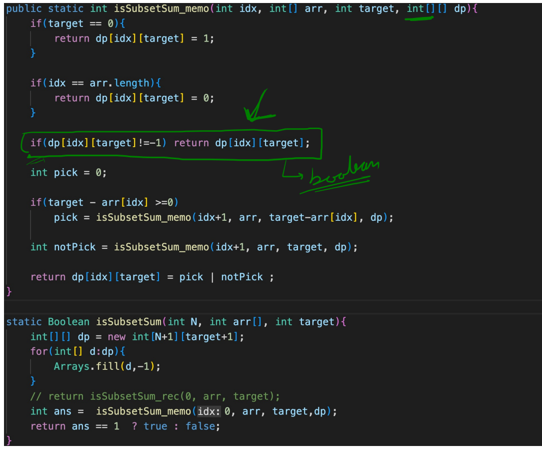
(1,6)
→ 0

b, int	0	1	2	3	4	5	6	7
0								
1							0	
2								
3								

false = 0
true = 1

(1,6)
→ false

(1, 6)
→ $\boxed{False}$



```java
public static int isSubsetSum_memo(int idx, int[] arr, int target, int[][] dp){
    if(target == 0){
        return dp[idx][target] = 1;
    }

    if(idx == arr.length){
        return dp[idx][target] = 0;
    }

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

    int pick = 0;

    if(target - arr[idx] >=0)
        pick = isSubsetSum_memo(idx+1, arr, target-arr[idx], dp);

    int notPick = isSubsetSum_memo(idx+1, arr, target, dp);

    return dp[idx][target] = pick | notPick ;
}

static Boolean isSubsetSum(int N, int arr[], int target){
    int[][] dp = new int[N+1][target+1];
    for(int[] d:dp){
        Arrays.fill(d,-1);
    }
    // return isSubsetSum_rec(0, arr, target);
    int ans =  isSubsetSum_memo(idx: 0, arr, target,dp);
    return ans == 1  ? true : false;
}
```

✓
→ boolean

0 1 2 3 4 5

3, 34, 4, 12, 5, 2

tar = 9

(6,3)

(6,5)

dp(idx)(target)

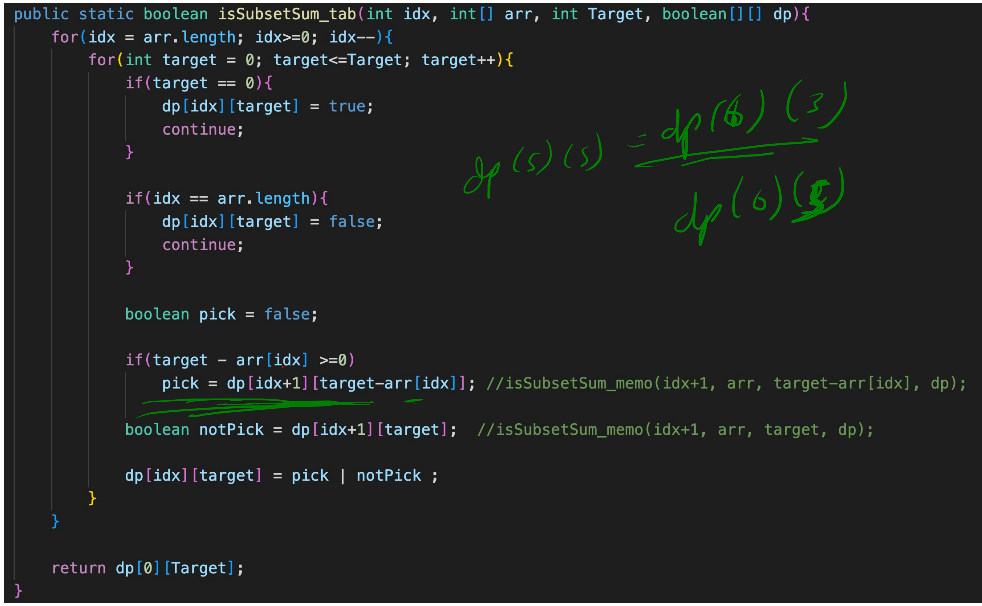
dp(5)(2)

↳ dp(6)(0)

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

(2)(4) = (3)(0) || (3)(2)

(2)(6) => (3)(2) || (3)6



```
public static boolean isSubsetSum_tab(int idx, int[] arr, int Target, boolean[][] dp){
    for(idx = arr.length; idx>=0; idx--){
        for(int target = 0; target<=Target; target++){
            if(target == 0){
                dp[idx][target] = true;
                continue;
            }

            if(idx == arr.length){
                dp[idx][target] = false;
                continue;
            }

            boolean pick = false;

            if(target - arr[idx] >=0)
                pick = dp[idx+1][target-arr[idx]]; //isSubsetSum_memo(idx+1, arr, target-arr[idx], dp);

            boolean notPick = dp[idx+1][target];  //isSubsetSum_memo(idx+1, arr, target, dp);

            dp[idx][target] = pick | notPick ;
        }
    }

    return dp[0][Target];
}
```

dp(5)(5) = dp(6)(3)

dp(6)(5)