Tabulation Follows bottom - up approach.

int[] arr = {1,3,5}
int targetSum = 8

From the given input we can derive possible sub problems are 24 => (3 * 8). We can represent in matrix.
Boolean[][] dp = new Boolean[3][8+1]

Should be 8+1, as index ends with n-1

Interesting !!! So that we will derive a solution for each element in an array, from target(0) to target(n), here n = 8

1

Let's derive a optimistic solution to 1st element, Here we have only one option left. If the sum == arr[0] then return True else False

When targetSum=0 , for any array size its true, because we can return empty subset.

Target Sum

Index	0	1	2	3	4	5	6	7	8
Value	1	T	T	F	F	F	F	F	F	F

Now we got all possible solutions from targetSum(0) to targetSum(n), these solutions we can use in next steps.

int[] arr = {1,3,5}
int targetSum = 8

3 Let's derive optimistic solution to 2nd element, Here we have two options.

Current Element

arr[index] <= capacity ✓

arr[index] > capacity ✗

Include the element

Exclude the element

calculate the possibility with including current element

calculate the possibility without including current element

Just calculate possibility without including current element.

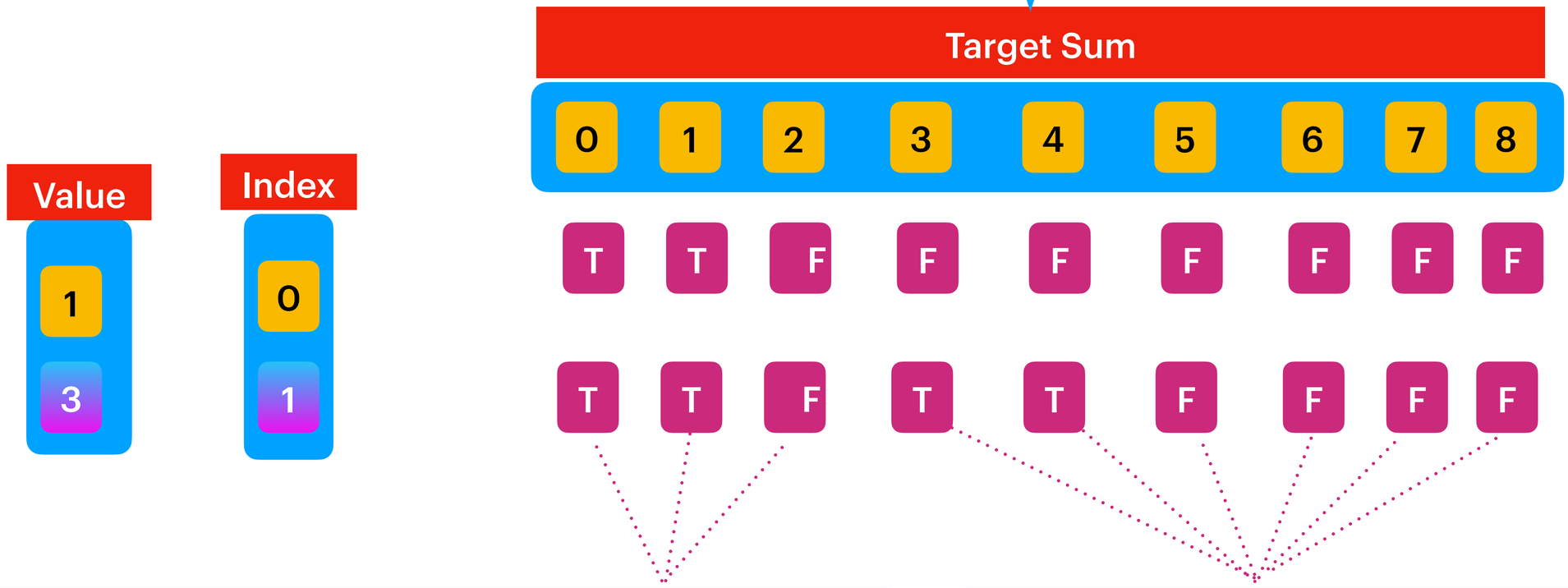
dp[i-1] [targetSum - arr[i]] || dp[i-1] [targetSum]

return dp[i-1] [targetSum]

return true, either of the one is true . (Use ||)

int[] arr = {1,3,5}
int targetSum = 8

3 Now, It's time to derive optimistic solution to 2nd element.

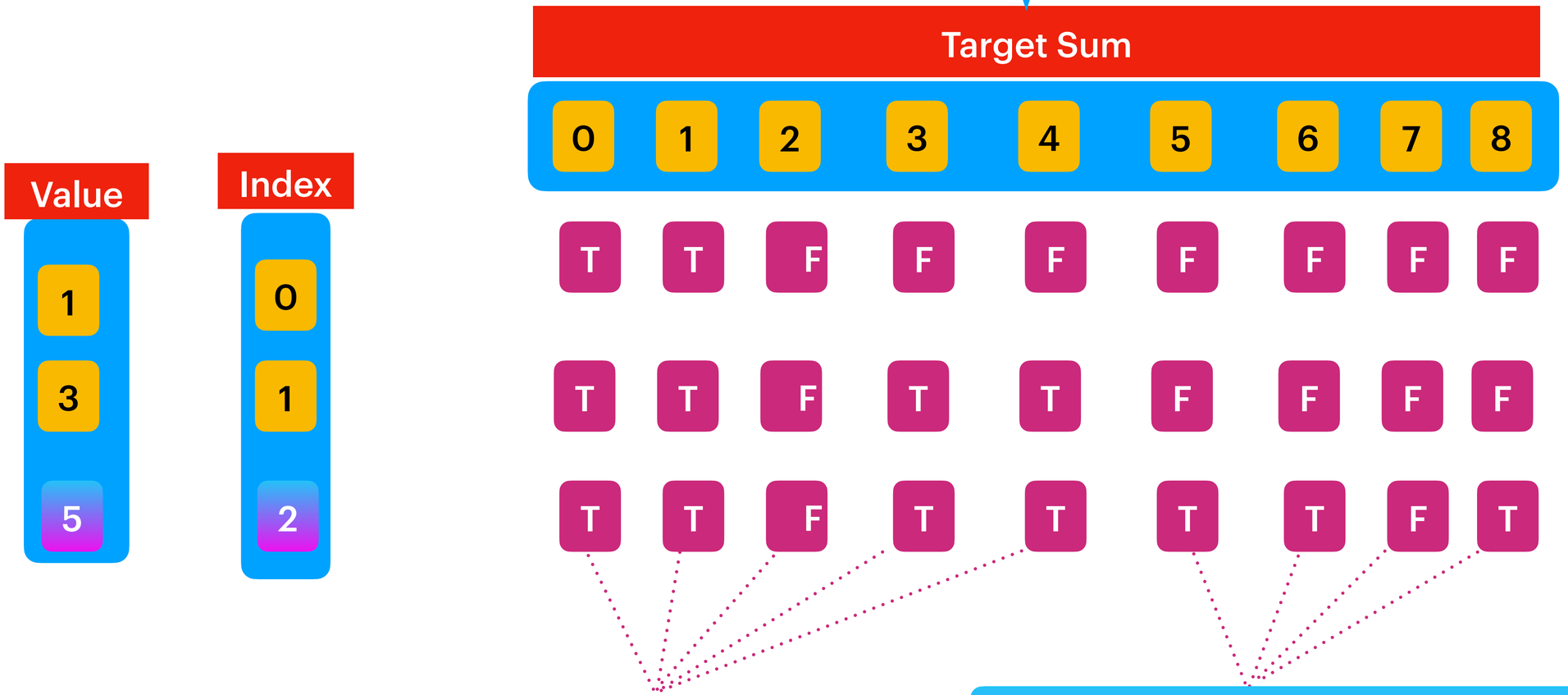


Till here the current element (3) > respective target sums (0,1,2) so we taken respective values from previous sub problem. (i.e from 0 index).
return dp[0][sum]

From sum(3) onwards the current element (3) <= respective targetSum(3,4,5,6,7,8) so
return dp[0][sum-arr[1] || dp[0][sum]

int[] arr = {1,3,5}
int targetSum = 8

5 Now, let's figure optimistic solution to 3rd element. Procedure is same as finding optimistic solution for 2nd element.



Till sum(4) the current element (5) > respective target sums (0,1,2,3,4) so we taken respective values from previous sub problem. (i.e from index 1).

return dp[1][sum-arr[1] || dp[1][sum]

From sum(5) onwards the current element (5) <= sum so we
return dp[1][sum-arr[1]] || dp[1][sum]

Finally we rocked it, derived a solution for targetSum 8 with given input elements {1,3,5}.