0/1 knapsack

toys

N items → weight, value

W

pick some items such that the sum of value is maximised

can't pick the same item more than once

	1	2	3	4
weight	20	10	30	40
value	100	60	120	150

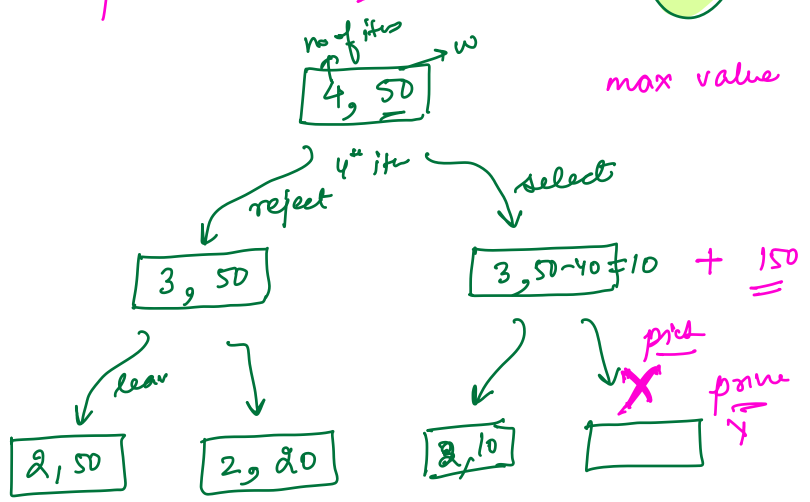
W = 50

$\alpha = W$

B.F: Try all possible combin

2^n

max value — 4 items, W = 50



optimal substructure

maxvalue(N, W)

no of items

N, W → Total weight of bag

max

leave → N-1, W

take it → N-W, W - W_n + $value_n$

weight of n^{th} object <= W

$$\boxed{maxvalue(N, W) = max(maxvalue(N-1, W), maxvalue(N-1, W-W_n) + value_n)}$$

N=12
W=50

maxvalue(12, 50)

maxvalue(6, 39)

W =	_	_	_	_	_	_	5	3	8	9	2	4
V =	_	_	_	_	_	_	_	_	_	_	_	_
							✗	✗	✗	✓	✓	✗
							✓	✗	✗	✗	✓	✓

maxvalue(6 , 39)

overlapping subproblems

Base case

no items remaining ⇒ value = 0
no weight / space ⇒ value = 0

dp[i][j] = max value by i item and j weight

j >= weight[i-1]

$$\boxed{dp[i][j] = \max(dp[i-1][j],\ dp[i-1][j - weight[i-1]] + value[i-1])}$$

no of items weight

i item

0 – i-1

if (i==0 || j==0) return 0;

i^th item = i-1 index

↳ information is stored at

maxvalue(n, w)

→ no of items

weight[n]
value[n]

```
moxvalue( int i, int j)
{
    if( i==0 || j==0) return 0;
    if( dp[i][j] != -1) return dp[i][j];
    // mox value with first i items & j weight
    if ( j >= weight[i-1])
    {
        dp[i][j] = max( maxvalue( i-1, j),
                        maxvalue( i-1, j-weight[i-1]) + value[i-1]);
    }
}
```

// i^th item

weight[i-1]

9 items

0–8

weight[9-1]

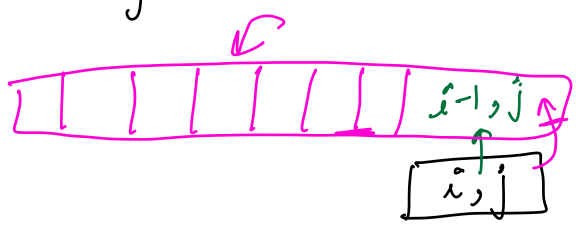
else {

dp[i][j] = maxvalue(i-1, j).

}

return dp[i][j];

}



i-1, j- ==

$W = 8$
$N = 5$

dp[n][m]

dp[n+1][m+1]

	0	1	2	3	4
W[] :	3	6	5	2	4
V[] :	12	20	15	6	10

dp[5][8]

size of matrix

6*9

v	w		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
12	3	1	0	0	0	12	12	12	12	12	12
20	6	2	0	0	0	12	12	12	20	20	20
15	5	3	0	0	0	12	12	15	20	20	15+12 =27
6	2	4	0	0	6	12	12	18	20	21	27
10	4	5	0	0	6	12	12	18	20	22	27

dp[1][1]
= dp[0][1]

dp[1][3]
= max(dp[0][3],
12+dp[0][3-3]

dp[1][7]
= dp[0][7]
12+dp[0][4]

dp[2][1] = dp[1][1]

dp[2][2]
= dp[1][2]

dp[2][6] = dp[1][6]
20+ dp[1][0]

dp[3][8] =
dp[2][8-5]

T.C: n*w   S.C: n*w   optimise: 2*w

```
i = n, j = w

while( i > 0 && j > 0)
{
    if( dp[i][j] == dp[i-1][j])
        i--;
    else
    {
        items.insert( i-1);   -> push the index of item
        j -= weight[i-1];
        i--;
    }
}
```

[0-∞] Knapsack

pick any item as many times as you want!

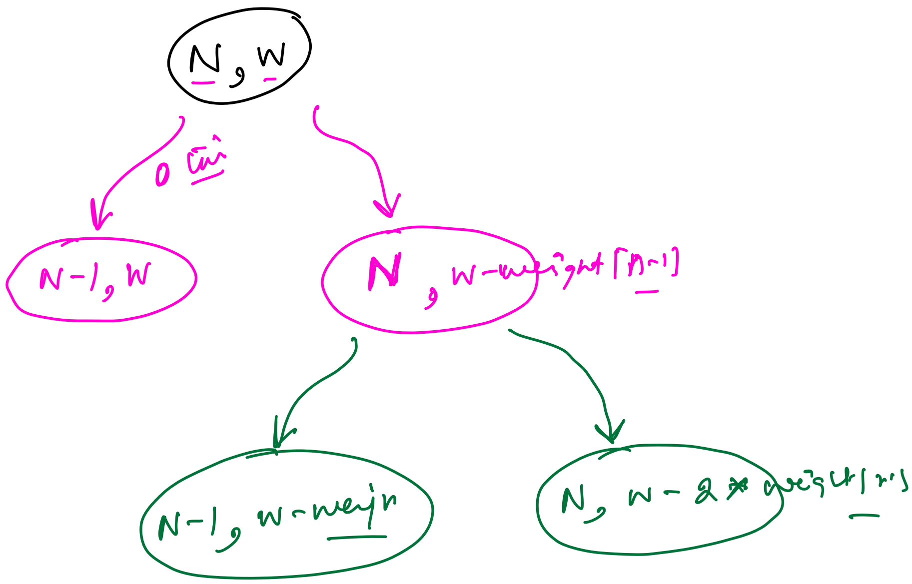
N = 3 — no of total items

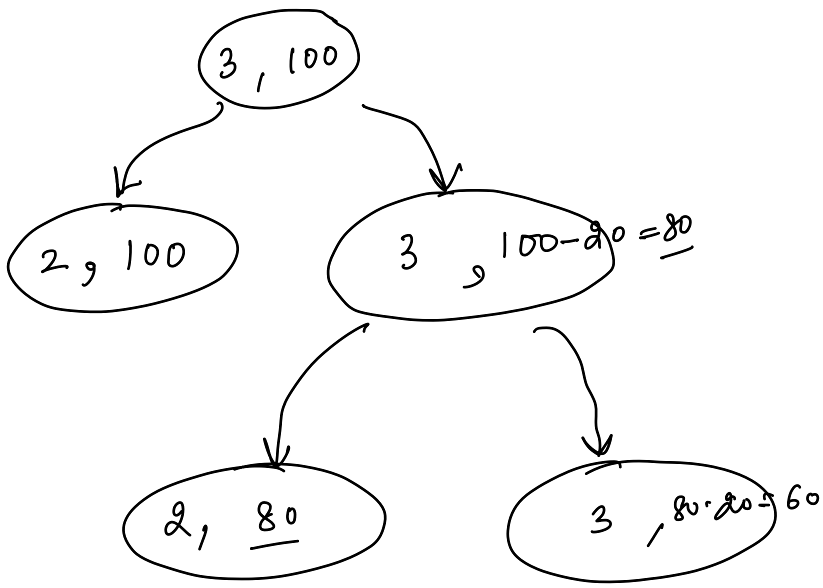
W = 8

value =	2	3	5
weight =	3	4	7

3+3 = 6

"*" — zero 0 or more char





$$dp[i][j] = \max(dp[i-1][j],\ value[i-1] + dp[i][j-weight[i-1]])$$

8448305664