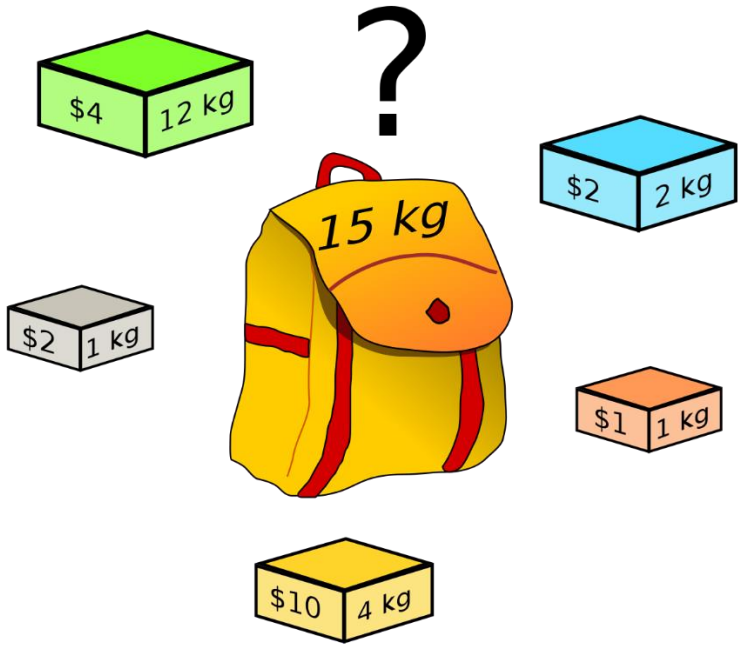
Knapsack Problem (Backpack Problem)

We are given a set of items, each with a weight and a value, we have to determine which items to include in the collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.



Solution 1 : If any number of each item is available, then we will take three yellow items and three grey items, so we will get total value of 36$ with total weight of 15 Kg.

Solution 2 : If only, the shown items are available (only one quantity of each item is available), then we will take all items except the green item, so we will get total value of 15$ with total weight of 8 Kg.

Note : We will be considering the knapsack problem where we have only one quantity of each item, so we can not include the same item more than once.

0-1 Knapsack

Items are indivisible (we can not break an item), either we take the whole item or we don't take it. Which means we can't take the fraction of an item. This is called the 0-1 property.

	Weight	Value
Item #1	**4Kg**	**10$**
Item #2	**2Kg**	**4$**
Item #3	**3Kg**	**7$**

Total Capacity of Knapsack is 5Kg

To get to the solution to the actual problem (given above 3 items what is the maximum value that we can get when total capacity of knapsack is 5 Kg), **we will first solve the *subproblems***.

e.g.

- Given only the first item and the total capacity of the knapsack is 2 Kg, what is the maximum value we can get? (answer is 0)
- Given only the first item and the total capacity of the knapsack is 4 Kg, what is the maximum value we can get? (answer is 10)
- Given only the first two items and the total capacity of the knapsack is 5 Kg, what is the maximum value that we can get? (answer is 10)
- Given all three items and the total capacity of the knapsack is 3 Kg, what is the maximum value that we can get? (answer is 7)

Item/weight	0	1	2	3	4	5
0						
1						
2						
3						

When we don't consider any item, regardless of the total capacity of the knapsack, we can only get value of 0

Item/weight	0	1	2	3	4	5
0	**0**	**0**	**0**	**0**	**0**	**0**
1						
2						
3						

When total capacity of the knapsack is 0 Kg, we can not put any item in the knapsack as each item have weight more than 0 Kg. So we can only get value of 0.

Item/weight	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					

Considering only the first Item (w = {4 Kg, 2 Kg, 3 Kg} , v = {10, 4, 7})

Item/weight	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	10	10
2	0					
3	0					

Considering only the first two Items (w = {4 Kg, 2 Kg, 3 Kg} , v = {10, 4, 7})

Item/weight	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	10	10
2	0	0	4	4	10	10
3	0					

Considering all three Items (w = {4 Kg, 2 Kg, 3 Kg} , v = {10, 4, 7})

Item/weight	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	10	10
2	0	0	4	4	10	10
3	0	0	4	7	10	11

When we consider an item we have two choices either to include the item in the knapsack or not include the item in knapsack.

Maximum(K[i-1][w] , v_i + K[i-1][w-w_i])

Note : We only calculate v_i + K[i-1][w-w_i] for an item if its weight is not greater than knapsack capacity. So if the weight of an item is greater than the knapsack capacity, we can't include that item, we have to exclude that item.

(w = {4 Kg, 2 Kg, 3 Kg} , v = {10, 4, 7})

Item/weight	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	10	10
2	0	0	4	4	10	10
3	0	0	4	7	10	11

K[2][1] = Since weight of second item is 2Kg, so we can't even consider second item when knapsack capacity is 1Kg, that's why K[2][1] = K[1][1] = 0

K[2][2] = Maximum (K[1][2] , 4 + K[1][2-2]) = Maximum (0 , 4) = 4

K[2][3] = Maximum (K[1][3] , 4 + K[1][3-2]) = Maximum (0 , 4) = 4

K[2][4] = Maximum (K[1][4] , 4 + K[1][4-2]) = Maximum (10 , 4) = 10

K[2][5] = Maximum (K[1][5] , 4 + K[1][5-2]) = Maximum (10 , 4) = 10

K[3][1] = Since weight of third item is 3Kg, so we can't even consider third item when knapsack capacity is 1Kg, that's why K[3][1] = K[2][1] = 0

K[3][2] = Since weight of third item is 3Kg, so we can't even consider third item when knapsack capacity is 2Kg, that's why K[3][2] = K[2][2] = 4

K[3][3] = Maximum (K[2][3] , 7 + K[2][3-3]) = Maximum (4 , 7) = 7

K[3][4] = Maximum (K[2][4] , 7 + K[2][4-3]) = Maximum (10 , 7) = 10

K[3][5] = Maximum (K[2][5] , 7 + K[2][5-3]) = Maximum (10 , 11) = 11

But which Items we included?

(w = {4 Kg, 2 Kg, 3 Kg} , v = {10, 4, 7})

Item/weight	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	10	10
2	0	0	4	4	10	10
3	0	0	4	7	10	11

So we Included Item 3 (weight 3Kg) and Item 2 (weight 2Kg), which gave us the total value of 11$

Implementation

```
public class App {

	public static void main(String[] args) {

		int numOfItems = 3;
		int capacityOfKnapsack = 5;

		int[] weightOfItems = { 4, 2, 3 };
		int[] valueOfItems = { 10, 4, 7 };

		Knapsack knapsack = new Knapsack(numOfItems, capacityOfKnapsack,
                                                 weightOfItems, valueOfItems);
		int totalValue = knapsack.solve();
		System.out.println("Total Value : " + totalValue);
		knapsack.showResult();
	}
}
```

```java
public class Knapsack {

	private int numOfItems;
	private int capacityOfKnapsack;
	private int[] weights;
	private int[] values;
	private int[][] knapsackTable;
	private int totalValue;

	public Knapsack(int numOfItems, int capacityOfKnapsack,
			int[] weights, int[] values) {
		this.numOfItems = numOfItems;
		this.capacityOfKnapsack = capacityOfKnapsack;
		this.weights = weights;
		this.values = values;
		this.knapsackTable = new int[numOfItems + 1][capacityOfKnapsack + 1];
	}

	public int solve() {
		…
	}

	public void showResult() {
		…
	}

}
```

```
public int solve() {
     for (int w = 0; w <= capacityOfKnapsack; w++) {
             knapsackTable[0][w] = 0;
     }
     for (int i = 0; i <= numOfItems; i++) {
             knapsackTable[i][0] = 0;
     }

     for (int i = 1; i <= numOfItems; i++) {
        for (int w = 1; w <= capacityOfKnapsack; w++) {
             // not taking item i
             int notTakingItem = knapsackTable[i - 1][w];
             int takingItem = Integer.MIN_VALUE;

             if (weights[i - 1] <= w) {
             takingItem = values[i - 1] + knapsackTable[i - 1][w - weights[i - 1]];
             }

             knapsackTable[i][w] = Math.max(notTakingItem, takingItem);
         }
     }
     totalValue = knapsackTable[numOfItems][capacityOfKnapsack];
     return totalValue;
}
```

```
public void showResult() {
   for (int n = numOfItems, w = capacityOfKnapsack; n > 0; n--) {
       if (knapsackTable[n][w] != 0 &&
               knapsackTable[n][w] != knapsackTable[n - 1][w]) {
             System.out.println("Included item: #" + n);
              w = w - weights[n - 1];
       }
   }
}
```

Time Complexity

Runtime Complexity = O(n*W)