1. Problem

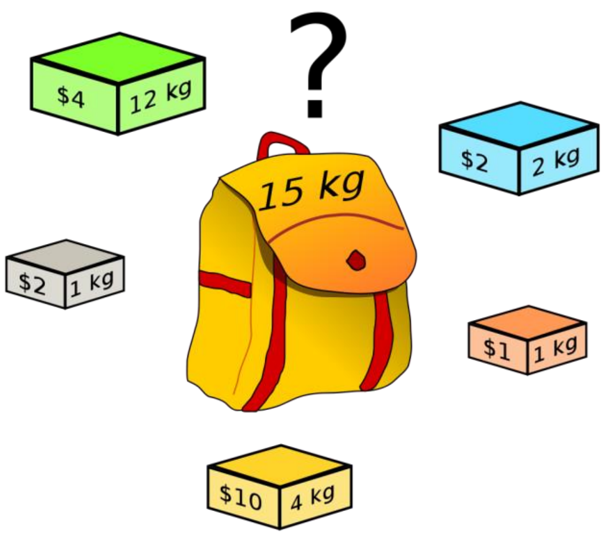
- **Knapsack Problem**

Input: A set of n **items** and each item has **weight w** and **value v**

Output: The **maximum total value** of the items that can fit in a knapsack **with the maximum weight capacity**

Constraint: (1) each item can be considered **only once**. (0-1)
(2) each item is considered **infinity times** (unbounded)

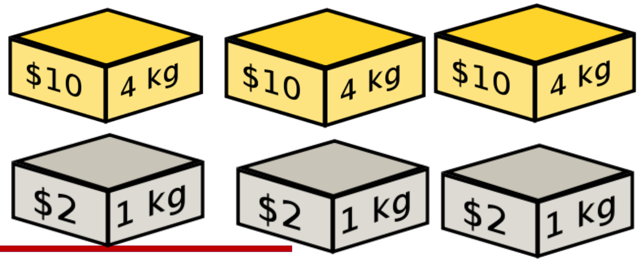
Example:



https://commons.wikimedia.org/wiki/File:Knapsack.svg

Only once: w: 8 kg, v: $15

Infinity times: w: 15 kg, v: $36



		Time complexity	Space complexity
Brute force:	0-1	$O(2^n)$	$O(n)$
	unbounded	$O(n^{W/M+1})$	$O(W/M)$
DP:	0-1	$O(nW)$	$O(nW)$
	unbounded	$O(nW)$	$O(nW)$

W: the weight limit **M**: minimum weight of all items

1.1 Brute force (0 - 1)

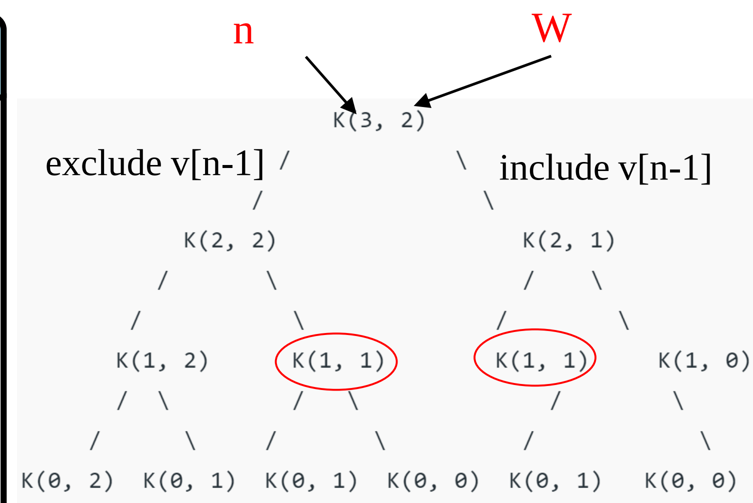
Pseudocode:

knapSack (w[0…n - 1], v[0…n - 1], W, n = len(w))

Input: : items i ∈ [0, n-1] with weight w[i] and value v[i] and knapsack's maximum weight capacity W
Output: : Maximum total value of items that can fit in the knapsack

1. **if** n = 0 or W = 0 **do** return 0
2. **if** w[n-1] > W **do**
3.   return knapSack(w, v, W, n - 1)
4. **else**
5.   **return** max(knapSack(w, v, W, n - 1)),
6.   val[n-1] + knapSack(w, v, W-w[n-1], n - 1))

w: [1,1,1], v: [10,20,30], W: 1



K(i, j) = knapSack function for the first i items with capacity at most j

K(1, 1) is evaluated **twice**

Time complexity: O(2^n)   Space complexity: O(n)

https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

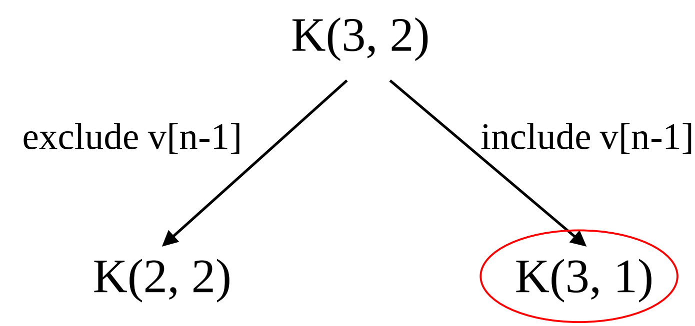
1.2 Brute force (unbounded)

each item can be considered **infinity times**

w: [1,1,1], v: [10,20,30], W: 1

Pseudocode:

```
knapSack (w[0…n - 1], v[0…n - 1], W, n = len(w))
Input: : Items i ∈ [1, n] with weight w[i] and value v[i]
and knapsack's maximum weight capacity W
Output: : Maximum total value of items that can fit in
the knapsack
if n = 0 or W = 0 do return 0
if w[n-1] > W do
    return knapSack(w, v, W, n - 1)
else
    return max(knapSack(w, v, W, n - 1)),
                val[n-1] + knapSack(w, v, W-w[n-1], n))
```




```
              K(3, 2)
            /         \
          /             \
    K(2, 2)              K(2, 1)
   /      \              /    \
  /         \           /       \
```

Time complexity: $O(n^{W/M+1})$

Space complexity: $O(W/M)$

W: the weight limit

M: minimum weight of all items

2. Dynamic Programming

w: [1,1,1], v: [10,20,30], W: 2

- Subproblem:

K[i, j] = Maximum total value the knapsack can hold for the first i items with capacity at most j, where ith item's weight is w[i] and value is v[i]

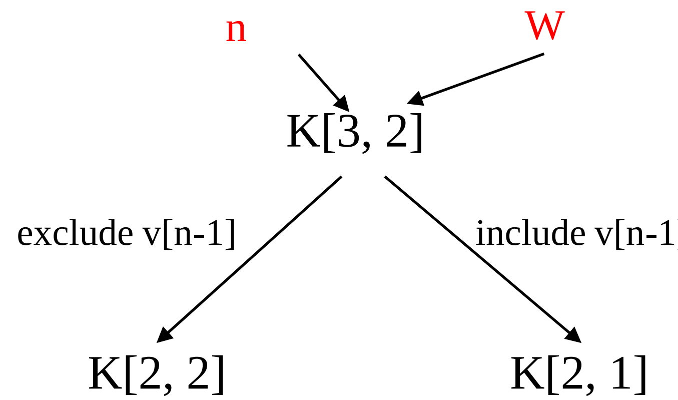
Compute K[n, W]

https://www3.cs.stonybrook.edu/~pramod.ganapathi/doc/algorithms/Algo-DynamicProgramming.pdf

2. Dynamic Programming

- Recurrence:

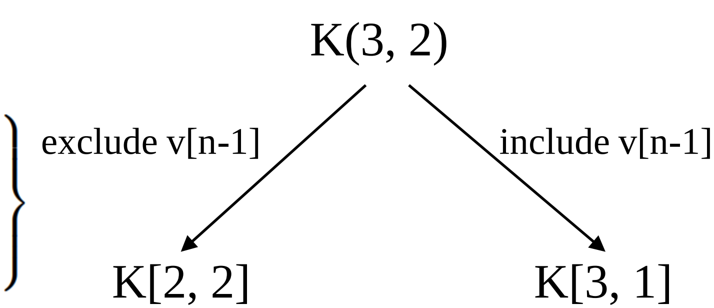
w: [1,1,1], v: [10,20,30], W: 2



- 0-1 knapsack:

$$K[i,j] = \left\{\begin{array}{ll} 0 & \text{if } ij = 0, \\ \max\left\{\begin{array}{l} K[i-1,j], \\ (K[i-1, j-w[i]] + v[i]) \times \boxed{j \geq w[i]} \end{array}\right\} & \text{if } ij \geq 1. \end{array}\right\}$$

- Unbounded knapsack:

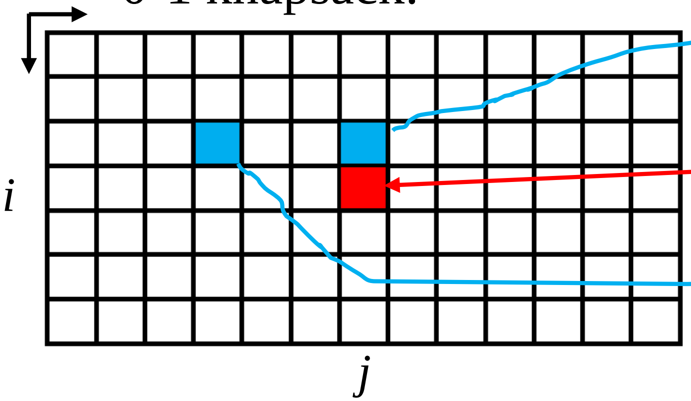


$$K[i,j] = \left\{\begin{array}{ll} 0 & \text{if } ij = 0, \\ \max\left\{\begin{array}{l} K[i-1,j], \\ (K[i, j-w[i]] + v[i]) \times \boxed{j \geq w[i]} \end{array}\right\} & \text{if } ij \geq 1. \end{array}\right\}$$

https://www3.cs.stonybrook.edu/~pramod.ganapathi/doc/algorithms/Algo-DynamicProgramming.pdf

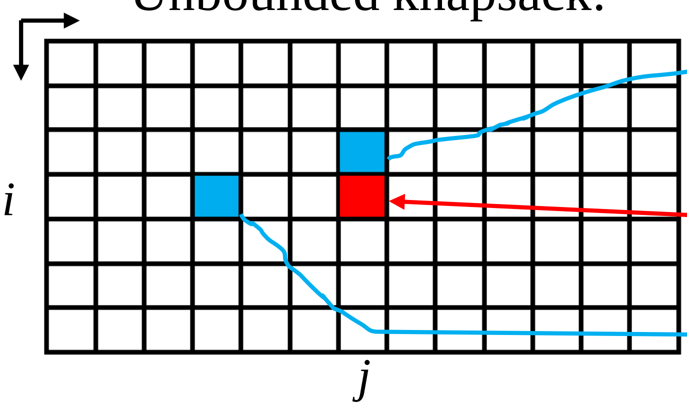
2. Dynamic Programming

- Dependency:
- 0-1 knapsack:



$$K[i,j] = \begin{cases} 0 & \text{if } ij = 0, \\ \max \left\{ \begin{array}{l} K[i-1,j], \\ (K[i-1, j-w[i]] + v[i]) \times \boxed{j \geq w[i]} \end{array} \right\} & \text{if } ij \geq 1. \end{cases}$$

- Unbounded knapsack:



$$K[i,j] = \begin{cases} 0 & \text{if } ij = 0, \\ \max \left\{ \begin{array}{l} K[i-1,j], \\ (K[i, j-w[i]] + v[i]) \times \boxed{j \geq w[i]} \end{array} \right\} & \text{if } ij \geq 1. \end{cases}$$

https://www3.cs.stonybrook.edu/~pramod.ganapathi/doc/algorithms/Algo-DynamicProgramming.pdf

2. Dynamic Programming

- Pseudocode:

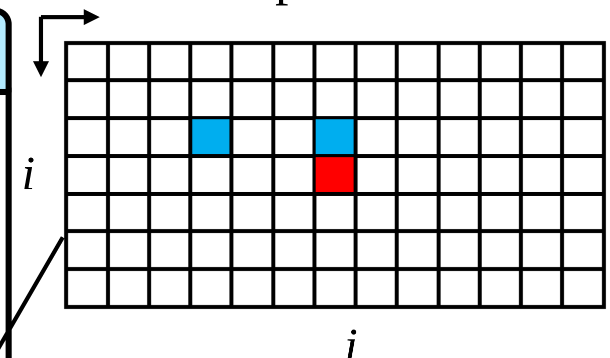
knapSack (w[1…n], v[1…n], W)

Input: : items i ∈ [0, n-1] with weight w[i] and value v[i] and knapsack's maximum weight capacity W

Output: : Maximum total value of items that can fit in the knapsack

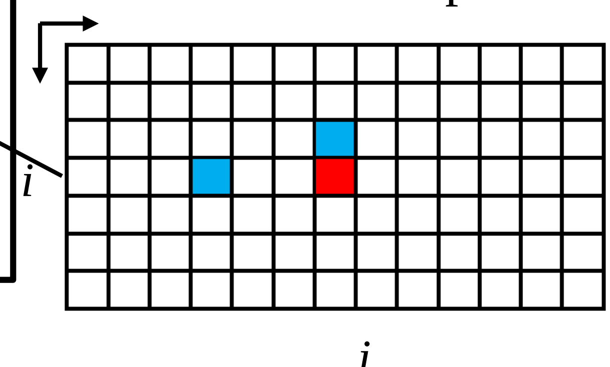
```
K[0, 0..W] ← 0, K[0..n , 0] ← 0
for i ←  0 to n – 1 do
    for j ←  0 to n – 1 do
        if j >= w[i] then
            K[i, j] ← max (K[i – 1, j], K[i – 1, j – w[i]] + v[i])
            K[i, j] ← max (K[i – 1, j], K[i , j – w[i]] + v[i])
        else
            K[i, j] ← K[i - 1, j]
    return K[n, W]
```

- 0-1 knapsack:



OR

- Unbounded knapsack:



https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

2. Dynamic Programming

- Tables:

w: [1,1,1], v: [10,20,30], W: 2

- 0-1 knapsack:

K[i, j]	0	1	2
0	0	0	0
1(w[1] = 1,v[1] = 10)	0	10	10
2(w[2] = 1,v[2] = 20)	0	20	30
3(w[3] = 1,v[3] = 30)	0	30	50

- Unbounded knapsack:

K[i, j]	0	1	2
0	0	0	0
1(w[1] = 1,v[1] = 10)	0	10	20
2(w[2] = 1,v[2] = 20)	0	20	40
3(w[3] = 1,v[3] = 30)	0	30	60

2. Dynamic Programming

- Complexity

$$\text{Time} \in \Theta\,(nW),\ \text{Space} \in \Theta\,(nW)$$

		Time complexity	Space complexity
Brute force:	0-1	$O(2^n)$	$O(n)$
	unbounded	$O(n^{W/M+1})$	$O(W/M)$
DP:	0-1	$\Theta(nW)$	$\Theta(nW)$
	unbounded	$\Theta(nW)$	$\Theta(nW)$

W: the weight limit **M**: minimum weight of all items