# CZ2101 Example Class 3
# Knapsack Problem (DP)

Sahithya, Yi Hao, Saori & Joel

# 1. Problem and definition

# Knapsack Problem

**Problem: Largest total profit for an <u>unbounded</u> set of objects**

**<u>Recursive definition</u>**

P - Profit, C - Capacity, wn - Weight of object n, pn - Profit of object n

$$P(0) = 0$$

$$P(C) = \max\{P(C-1), P(C-w_1)+p_1, P(C-w_2)+p_2, \ldots., P(C-w_n)+p_n\}, C-w_j \geq 0 \quad for\ j = 1 to\ n$$

**Time complexity**: O(Cn), where C is the capacity and n is the number of types of objects

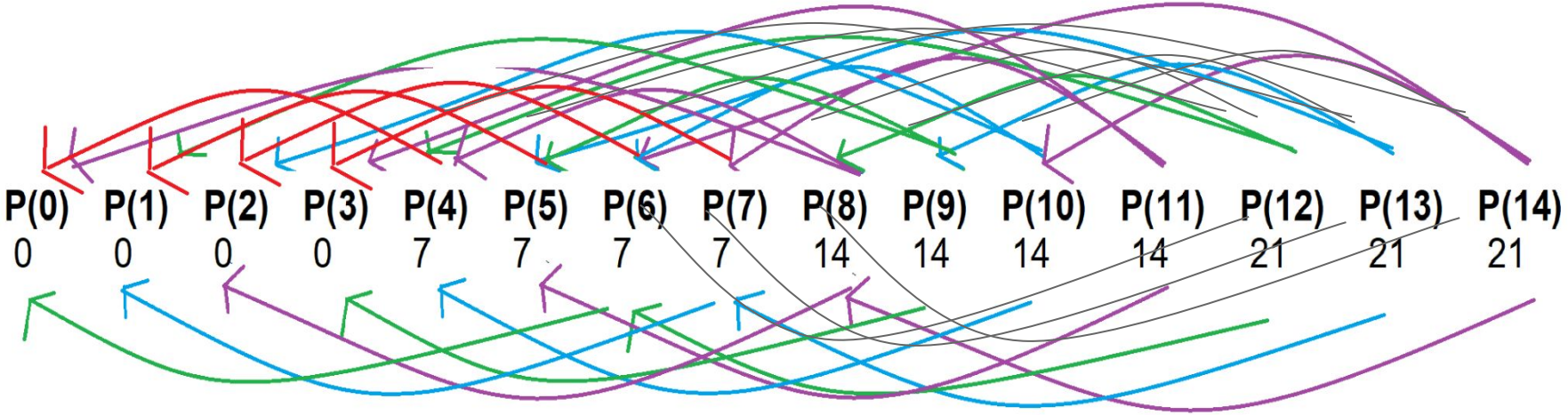
**Space complexity**: O(C)

# Recursive Definition

```
int knapsack[int n, int C] {
        if (i<0) {return 0;}
        if (weights[n] > C) {return knapsack(n-1, C);}
        else{return max(knapsack(n-1, C), knapsack(n-1, C- weights[n] + profit[n]));}
}
```

# 2. Subproblem Graph P(14)

# Subproblem Graph

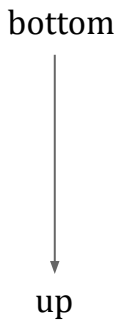



| | 0 | 1 | 2 |
|---|---|---|---|
| $w_i$ | 4 | 6 | 8 |
| $p_i$ | 7 | 6 | 9 |

# 3. Algorithm

# Bottom-up approach

| V[i,w] | 0 | 1 | 2 | … | C |
|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | … | 0 |
| **1** | | | | | |
| **:** | | | | | |
| **n** | | | | | |




**Bottom:** $V[0,w] = 0$ for all $0 \leq w \leq C$

**Bottom-up :** $V[i,w] = \max\{ V[ i-1, w], V[ i-1, w - w_i] + p_i]$

# Pseudocode

```
KnapSack(C,w,p,n)
{
     for(int w = 0 to C)
    {
        V[0,w] = 0;
    }
     for(int i = 1 to n)
    {
        for(w = 0 to C)
       {
           if(w[i] <= w)
          {
             V[i,w] = max{V[i-1,w],V[i-1, w-w[i]] + p[i]};
          }
           else
          {
             V[i,w] = V[i-1,w];
          }
    }
    return V[n,C];
}
```

# Pseudocode

```
temList){
ofitList to store max profit for capacity 0 to Capacity (parameter)
{0}; // add 0 as first element, since P(0) = 0
ngleCapacityProfitList to store all possible profits for a certain capacity
rofitList = {};

pacity = 1 to Capacity){
rofit of previous capacity (i.e. currentCapacity  - 1)
ityProfitList.add(maxProfitList[ currentCapacity - 1]);
 to numOfItems-1){
ItemWeight =  item[j].getWeight();
ItemProfit =  item[j].getProfit();
rentCapacity - currentItemWeight  >= 0){
Add (max profit of weight (current capacity - currentItemWeight)) + currentItemProfit,
s possible profit for current capacity
ngleCapacityProfitList.add(maxProfitList[ currentCapacity - currentItemWeight ] + curre


ighest profit in singleCapacityProfitList to profitList
st.add(max(singleCapacityProfitList));
tents of singleCapacityProfitList to store profits for next capacity
ityProfitList.clear();
```

# Pseudocode

```
temList){
{0};  //add 0 as first element, since P(0) = 0
gleCapacityProfitList to store all possible profits for a certain capacity
rofitList = {};


rofit of previous capacity (i.e. currentCapacity  - 1)
ityProfitList.add(maxProfitList[ currentCapacity - 1]);
 to numOfItems-1){
ItemWeight =  item[j].getWeight();
ItemProfit =  item[j].getProfit();
rentCapacity - currentItemWeight  >= 0){
Add (max profit of weight (current capacity - currentItemWeight)) + currentItemProfit,
s possible profit for current capacity
gleCapacityProfitList.add(maxProfitList[ currentCapacity - currentItemWeight ] + curre


ighest profit in singleCapacityProfitList to profitList
st.add(max(singleCapacityProfitList));
tents of singleCapacityProfitList to store profits for next capacity
ityProfitList.clear();

st[Capacity];
```

# Pseudocode

```
Knapsack(Capacity, itemList){
	maxProfitList = {0};
	singleCapacityProfitList = {};

	for (1 to Capacity){
		add previous Capacity's Max Profit;
		for ( firstItem to lastItem){
			if (currentCapacity - currentItemWeight  >= 0){
				add P(currentCapacity - currentItemWeight) + currentItemProfit,
				to singleCapacityProfitList;
			}
		}
		add Max Profit from singleCapacityProfitList to maxProfitList;
		clear singleCapacityProfitList;
	}
	return Max Profit at Capacity;
}
```

# 4. Results

# Results

4a)

**Input:**

| | 0 | 1 | 2 |
|---|---|---|---|
| $w_i$ | 4 | 6 | 8 |
| $p_i$ | 7 | 6 | 9 |

**Output:**

```
Capacity:Max Profit
0: 0
1: 0
2: 0
3: 0
4: 7
5: 7
6: 7
7: 7
8: 14
9: 14
10: 14
11: 14
12: 21
13: 21
14: 21
```

4b)

**Input:**

| | 0 | 1 | 2 |
|---|---|---|---|
| $w_i$ | 5 | 6 | 8 |
| $p_i$ | 7 | 6 | 9 |

**Output:**

```
Capacity:Max Profit
0: 0
1: 0
2: 0
3: 0
4: 0
5: 7
6: 7
7: 7
8: 9
9: 9
10: 14
11: 14
12: 14
13: 16
14: 16
```

*Thank you !*