Greedy Method

Greedy Method

- Greedy algorithms work in stages where in each stage one of the inputs is examined.
- The input is *selected* based on an optimization criterion.
- If the input forms part of an optimal solution, it is included in the solution. It is discarded otherwise.

Greedy Method

- The steps are repeated to find the complete solution.
- This model of Greedy method is called as *subset paradigm*.
- The other model called as *ordering paradigm* is based on considering the input elements in some predefined order.

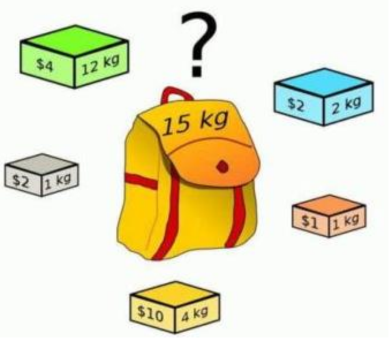
Greedy Method

Control Abstraction

```
Algorithm Greedy(a,n)
//a contains inputs
{
    solution:={};
    for i:= 1 to n do
    {
        x=Select(a);
        if feasible(solution,x) then
            solution:=solution ∪ {x};
    }
    return solution;
}
```

Knapsack Problem

- Given n objects and a knapsack. Object i has weight w_i and profit p_i and the knapsack has a capacity m.
- The objective is to maximize the total profit by selecting objects (fraction of an object) without exceeding the total capacity of the knapsack.



Knapsack Problem

- $maximize\ \sum_{1 \le i \le n} p_i x_i$
- $Subject\ to \sum_{1 \le i \le n} w_i x_i \le m$
- $0 \le x_i \le 1\ and\ 1 \le i \le n$
- x_i is 0, the object i is not selected.
- x_i is 1, the complete object i is selected.
- x_i is $0.4, 40\%$ of the object is selected and hence contributes 40% of the profit.

Knapsack Problem

- $maximize\ \sum_{1 \le i \le n} p_i x_i$ (1)
- $Subject\ to \sum_{1 \le i \le n} w_i x_i \le m$ (2)
- $0 \le x_i \le 1\ and\ 1 \le i \le n$ (3)
- A feasible solution is any set $\{x_k, \ldots, x_r\}$ which satisfy (2) and (3)
- An optimum solution is a feasible solution that satisfies (1)

Knapsack Problem

- Consider the instance of the Knapsack problem: $n = 3, m = 20, (p_1, p_2, p_3\ ) = (25,24,15), (w_1, w_2, w_3\ ) = (18,15,10)$

S.No	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
1	$(1, \frac{2}{15}.0)$	20	28.2
2	$(0, \frac{2}{3}, 1)$	20	31
3	$(0{,}1.\frac{1}{2})$	20	31.5

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1

m=15

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

i	job	m	P
-	-	15	0

	1	2	3	4	5	6	7
x	0	0	0	0	0	0	0

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

i	job	m	P
-	-	15	0
1	5	14	6

	1	2	3	4	5	6	7
x	0	0	0	0	1.0	0	0

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

i	job	m	P
-	-	15	0
1	5	14	6
2	1	12	16

	1	2	3	4	5	6	7
x	1.0	0	0	0	1.0	0	0

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

i	job	m	P
-	-	15	0
1	5	14	6
2	1	12	16
3	6	8	34

	1	2	3	4	5	6	7
x	1.0	0	0	0	1.0	1.0	0

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

i	job	m	P
-	-	15	0
1	5	14	6
2	1	12	16
3	6	8	34
4	7	7	37

	1	2	3	4	5	6	7
x	1.0	0	0	0	1.0	1.0	1.0

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

i	job	m	P
-	-	15	0
1	5	14	6
2	1	12	16
3	6	8	34
4	7	7	37

	1	2	3	4	5	6	7
x	1.0	0	1.0	0	1.0	1.0	0

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

i	job	m	P
-	-	15	0
1	5	14	6
2	1	12	16
3	6	8	34
4	7	7	37
5	3	2	52

	1	2	3	4	5	6	7
x	1.0	0	1.0	0	1.0	1.0	0

Knapsack Problem

i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
p_i/w_i	5	1.66	3	1	6	4.5	3
job	5	1	6	7	3	2	4

	1	2	3	4	5	6	7
x	1.0	$\frac{2}{3}$	1.0	0	1.0	1.0	0

i	job	m	P
-	-	15	0
1	5	14	6
2	1	12	16
3	6	8	34
4	7	7	37
5	3	2	52
6	2	0	52+ 2/3* 5

```
Algorithm GreedyKnapSack(m,n)
//p[1..n] contains the profits and w[1..n] contains
//the weights of n objects ordered such that
//p[i]/w[i]>=p[i+1]/w[i+1]. job[1..n] contains the IDs
//of jobs in p and w. m is the capacity of knapsack.
//x[1..n] is the solution vector.
{
    for i:= 1 to n do x[i]:=0;
    U:=m;
    for i:= 1 to n do
    {
        if (w[job[i]] > U) then break;
        x[job[i]]:=1.0; U:= U – w[job[i]];
        prfit:=profit+p[job[i]];
    }
    if (i<=n) then
    {
        x[job[i]]:=U/w[job[i]];
        profit:=profit+p[job[i]* U/w[job[i]];
    }
    return profit;
}
```

0/1 Knapsack

- Excluding the complexity of sorting, the algorithm is $O(n)$