Greedy Method.

→ Follows local optimal choice at each stage with intent of finding global optimum.

→ Feasible Solution (Any subset that satisfy the conditions).

→ Optimal Solution (~~least~~ best and most feasible solution).

* Components of Greedy Algorithm :-

① A Candidate set :- A solⁿ is created from this set.

② A selection function :- used to choose the best candidate to be added to the sol.

③ A feasibility function :- used to determine wheather a candidate can be used to contribute to the solⁿ.

④ A objective function :- used to assign value to a solⁿ or partial solⁿ.

⑤ A solution function :- used to indicate wheather a complete sol has been reached.

→ Areas of Application :-

① Finding Shortest Path
② Finding MST
③ Job Sequencing with deadline.
④ Fractional Knapsack problem.

* Pseudo for GA :-

```
Algorithm Greedy(a, n) {
  Solution = 0;
  for(j = 1 to n) do
  {
    x = Select(a);
    if feasible (Solution, x) then
    Solution = union(Solution, x);
  }
  return Solution;
}.
```

* $0/1$ Knapsack :-

Given :- There are n objects weights of objects $w_1, w_2, w_3, \ldots, w_n$. Profit of objects : $P_1, P_2, \ldots P_n$

Capacity of Knapsack : m.

Goal :- To find vector $X = [x_1, x_2 \ldots x_m]$.

maximize $\sum_{i=1}^{n} P_i x_i$ subject to constraints.

$\sum_{i=1}^{n} w_i x_i \leq m$ [where $x_i = 0$ or 1].

* Order of decisions :-

initially $P=0$, $W=m$

$x_n \rightarrow x_{n-1} \rightarrow x_{n-2} \rightarrow \ldots \, x_1$

For each decision ⇒

i) Selected Objects :- Remaining Capacity : $m - w_n$

Profit :- Profit + P_n

ii) Rejected Object :- Remaining Capacity : m.

* Principal of optimality :- Following a decision on x_n, there will be two possible states :-

→ Capacity m & no profit

→ Capacity $m - w_n$ & P_n is the profit.

It is clear that remaining $x_{n-1}, x_{n-2} \dots x_1$ must be optimal.

A Generalized Equation :-

$$f_i(y) = \max\{f_{i-1}(y), f_{i-1}(y - w_i) + P_i\}.$$

$\boxed{y \text{ is capacity}}$.

Purging Rule :- If S^{i+1} contains (P_j, w_j) & (P_k, w_k) such that $P_j \leq P_k$ & $w_j \geq w_k$

then eleminate (P_j, w_j) from set.

Eg :- Capacity $m = 8$

No of objects $n = 4$

i	P_i	w_i
1	1	2
2	2	3
3	5	4
4	6	5

Sol → $S^0 = \{(0,0)\}$ (common for every solⁿ)

$S^0_1 = \{(1,2)\}$ (First entry (P_i, w_i)) $i \Rightarrow$ objects.

$S^1 = \{\text{merge } S^0 \,\&\, S_1^0\}$ $P_i \Rightarrow$ Profit of objects.

$= \{(0,0), (1,2)\}$. $w_i \Rightarrow$ weight of objects.

Step2) $S^1_1 = \{(2,3), (3,5)\}$. // Select next pair (P_2, w_2) and add with S^1

$S^2 = \{\text{merge } S^1 \,\&\, S^1_1\}$.

$= \{(0,0), (1,2), (2,3), (3,5)\}$.

Step3) $S^2_1 = \{(5,4), (6,6), (7,7), (8,9)\}$. // Select next pair $(P_3 . w_3)$ and add with S^2.

$S^3 = \{\text{merge } S^2 \,\&\, S^2_1\}$.

$= \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6), (7,7), (8,9)\}$.

⇒ Now ~~apply~~ applying Purging rule :-

$S^3 = \{(0,0), (1,2), (2,3), (5,4), (6,6), (7,7), (8,9)\}$.

Step4) $S^3_1 = \{(6,5), (7,7), (8,8), (11,9), (12,11), (13,12), (14,14)\}$.

// Select (P_4, w_4) & add with $S^3 (6,5)$.

$S_4 = \{\text{merge } S^3 \,\&\, S^3_1\}$.

$S_4 = \{(0,0), (1,2), (2,3), (5,4), (6,6), (7,7), (8,9), (6,5), (8,8), (11,9), (12,11), (13,12), (14,14)\}$.

i) we have Knapsack Capacity :- 8.

Select $(8,8)$:- It is present in S^3_1

$\therefore$ Put $x_4 = 1$

To select next object

$(P-P_4, w-w_4) = (8-6, 8-5)$

$= (2, 3)$

ii) $x_3 = 0$ as $(2,3)$ is not present in S^2_1

iii) Pair $(2,3)$ present in S^1_1

$\therefore$ Put $x_2 = 1$

To select next object $(P-P_2, w-w_2) = (2-2, 3-3)$

$= (0, 0)$

iv) As Knapsack Capacity is full $x_1 = 0$

Solution Vectors :- $X = \begin{array}{cccc} [0 & 1 & 0 & 1] \\ x_1 & x_2 & x_3 & x_4 \end{array}$

* ## Fractional Knapsack Problem :-

Objects - 1, 2, 3, 4, 5, 6, 7
Profit (P) - 5, 10, 15, 7, 8, 9, 4
weight (w) - 1, 3, 5, 4, 1, 3, 2

$\boxed{\begin{array}{l} W = 15 \\ n = 7 \end{array}}$

To Solve this problem we have three aproches :-

i) Select the items according to their Max. Profit.
→ Always select the object with the max profit.

Objects	Profit (P)	weight (w)	Remaining weight.
3	15	5	15 - 5 = 10. (Done with 15).
2	10	3	10 - 3 = 7
6	9	3	7 - 3 = 4
5	8	1	4 - 1 = 3
4	7		

As we can clearly see that the remaing weight is 3 and the selected object has weight of 4 which can not fit So we will take the fraction of profit ~~acbord~~ according to the weight.

4	$7 \times \frac{3}{4} = \frac{21}{4}$ $= 5.24$	3	$3 - 3 = 0$

Total profit : 47.25 ₹

ii) Choose the item according to their min. weight.

objects	Profit	weight	Remaing weight
1	5	1	$15 - 1 = 14$
5	8	1	$14 - 1 = 13$
7	4	2	$13 - 2 = 11$
6	9	3	$11 - 3 = 8$
2	10	3	$8 - 3 = 5$
4	7	4	$5 - 4 = 1$

As we can see the remaing weight is 1. So we can't select 5 so will select the fraction.

3	$15 \times \frac{1}{8} = 3$	1	$1 - 1 = 0$

Total profit ⇒ ₹46.

iii) Finding by the max profit/weight ratio:-

First of calculate Profit/weight ratio:-

P/w ratio:- 5, 3.3, 3, 1.75, 8, 3, 2

Object	Profit (P)	weight (w)	Remaining weight.
5	8	1	15 - 1 = 14
1	5	1	14 - 1 = 13
2	10	3	13 - 3 = 10
3	15	5	10 - 5 = 5
6	9	3	5 - 3 = 2
7	4	2	2 - 2 = 0
Total profi	51		

As we can clearly see that in 3rd case we get the max profit as compare to other two methods.

* Dynamic Programming: Dynamic Programming (commonly ~~reff~~ referred as DP) is an algorithmic technique for solving a problem by recursively breaking it down into simpler subproblems. and using the fact that the optimal solution to the overall problem depends upon the optimal solution to it's individual subproblems.

Example: If we write simple recursive solution for Fibonacci series, we get exponential time complexity and if we optimize it by ~~solving~~ storing solutions of subproblems, time complexity reduces to linear.

Recursion: Exponential

```
int fib(int n)
{
  if(n <= 1)
    return n;
  return fib(n-1) + fib(n-2);
}.
```

Dynamic Programming Linear:-

```
f[0] = 0;
f[1] = 1;

for(i=2; i<=n; i++)
{
   f[i] = f[i-1] + f[i-2];
}
return f[n];
```