GREEDY METHOD

General method

The greedy method is perhaps the most straight forward design technique and it can be applied to a wide variety of problems.

Most, though not all, of these problems have n inputs and require us to obtain a subset that satisfies some constraints.

Any subset that satisfies these constraints is called a feasible solution. We need to find a feasible solution that either maximizes or minimizes a given objective function.

A feasible solution that does this is called an optimal solution.

```
Algorithm Greedy(a,n)
// a[1:n] contains the n inputs
{
    solution := φ // Initialize the solution
    for i:=1 to n do
    {
        x := select(a);
        if feasible(solution, x) then
            solution := Union(solution, x);
    }
    return solution;
}
```

Greedy method control abstraction for the subset paradigm above

→ Function select selects an input from a[ ] and removes it.

→ Feasible is a Boolean-valued function that determines whether x can be included into the solution vector.

Applications of Greedy method :-

1) Knapsack problem

2) Job sequencing with deadlines problem.

3) minimum cost spanning trees

Prim's algorithm

Kruskal's algorithm

4) Single source shortest paths problem.

KNAPSACK Problem :-

→ We are given $\underline{n}$ objects and a knapsack or bag.

→ object $\underline{i}$ has a weight $\underline{w_i}$

→ knapsack has a capacity $\underline{m}$ kg

→ If a fraction x_i, $0 \le x_i \le 1$ of object $\underline{i}$ is placed into the knapsack, then a profit of $\underline{P_i x_i}$ is earned.

→ The objective is to obtain a filling of the Knapsack that maximizes the total profit earned.

→ Total weight of all chosen objects is atmost $\underline{m}$. This problem can be stated as

$$\boxed{\begin{aligned} &\text{maximize} \quad \sum_{1 \le i \le n} P_i x_i \quad \text{— (1) objective function} \\ &\left.\begin{aligned} &\text{subject to} \quad \sum_{1 \le i \le n} w_i x_i \le m \quad \text{— (2)} \\ &\text{and} \quad 0 \le x_i \le 1, \quad 1 \le x_i \le n \quad \text{— (3)} \end{aligned}\right\} \text{Constraints} \end{aligned}}$$

→ A feasible solution (or filling) is any set $(x_1, x_2 \dots x_n)$ satisfying (2) and (3) above.

→ An optimal solution is a feasible solution for which (1) is maximized.

Scanned with OKEN Scanner

Eg: Consider the following instance of the Knapsack problem. $n = 3$ objects given, Knapsack capacity $m = 20$ kg, profits of objects $(P_1, P_2, P_3) =$ Rs $(25, 24, 15)$ and weights $(w_1, w_2, w_3) = (18, 15, 10)$

Sol: Four feasible solutions are

(x_1, x_2, x_3)	$\Sigma w_i x_i$	$\Sigma p_i x_i$
1) $(1/2, 1/3, 1/4)$	16.5 kg	Rs 24.25
2) $(1, 2/15, 0)$	20	28.2
3) $(0, 2/3, 1)$	20	31
4) $(0, 1, 1/2)$	20	31.5

→ Out of these four feasible solutions, solution 4 yields the maximum profit. So it is optimal solution for the given problem.

→ To find optimal solution of knapsack problem arrange the objects in decreasing order of profit per unit weight.

$$\frac{P_i}{w_i} = \frac{P_1}{w_1}, \frac{P_2}{w_2}, \frac{P_3}{w_3}.$$

$$= \frac{25}{18}, \frac{24}{15}, \frac{15}{10}$$

$$= 1.3, 1.6, 1.5$$

$$\boxed{\frac{P_2}{w_2} > \frac{P_3}{w_3} > \frac{P_1}{w_1}}$$

We place the most profit giving objects one by one until the knapsack gets filled.

$$\text{Total weight} = \Sigma w_i x_i = w_2 x_2 + w_3 x_3 + w_1 x_1$$
$$= 15 \times 1 + 10 \times \tfrac{1}{2} + 18 \times 0$$
$$= 15 + 5 + 0$$
$$= \underline{20 \text{ kg}}$$

$$\text{Total profit obtained} = \Sigma P_i x_i$$
$$= P_2 x_2 + P_3 x_3 + P_1 x_1$$
$$= 24 \times 1 + 15 \times \tfrac{1}{2} + 25 \times 0$$
$$= 24 + 7.5 + 0$$
$$= \underline{\underline{\text{Rs } 31.5}}$$

$$\therefore x_3 = \frac{20-15}{w_3} = \frac{5}{10} = \frac{1}{2} \quad \therefore$$

$\therefore$ Solution Tuple (vector) $(x_1, x_2, x_3) = \underline{(0, 1, \tfrac{1}{2})}$

Scanned with OKEN Scanner

Eg: Find an optimal solution To the Knapsack instance $n = 7$, $m = 15$,

Profits or costs of objects

$$P_1, P_2, P_3, P_4, P_5, P_6, P_7 = (10, 5, 15, 7, 6, 18, 3)$$

Weights of object

$$(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$$

Sol Find profit/unit wt of objects.

$$\frac{P_i}{w_i} = \frac{P_1}{w_1}, \frac{P_2}{w_2}, \frac{P_3}{w_3}, \frac{P_4}{w_4}, \frac{P_5}{w_5}, \frac{P_6}{w_6}, \frac{P_7}{w_7}$$

$$= \frac{10}{2}, \frac{5}{3}, \frac{15}{5}, \frac{7}{7}, \frac{16}{1}, \frac{18}{4}, \frac{3}{1}$$

$$= 5, 1.66, 3, 1, 6, 4.5, 3$$

Arrange the objects in decreasing order of profit / unit wt

$$\frac{P_5}{w_5} > \frac{P_1}{w_1} > \frac{P_6}{w_6} > \frac{P_3}{w_3} = \frac{P_7}{w_7} > \frac{P_2}{w_2} > \frac{P_4}{w_4}$$

We place The more costly objects one by one into the knapsack bag.

$$\boxed{\text{Total weight} = \Sigma w_i x_i = w_5 x_5 + w_1 x_1 + w_6 x_6 + w_3 x_3 + w_7 x_7 + w_2 x_2}$$

$$= \underline{1 \times 1} + \underline{2 \times 1} + \underline{4 \times 1} + \underline{5 \times 1} + \underline{1 \times 1} + \underline{3 \times \tfrac{2}{3}}$$

$$= 1 + 2 + 4 + 5 + 1 + 3 = \underline{\underline{15}}$$

Scanned with OKEN Scanner

Total profit obtained $= \Sigma P_i x_i$

$= P_5 x_5 + P_1 x_1 + P_6 x_6 + P_3 x_3 + P_7 x_7 + P_2 x_2$

$= \underline{6 \times 1} + \underline{10 \times 1} + \underline{18 \times 1} + \underline{15 \times 1} + \underline{3 \times 1} + \underline{5 \times 0.66}$

$= 6 + 10 + 18 + 15 + 3 + 3.30$

$=$ Rs $\underline{\underline{55.30}}$

After placing object 7, the remaining empty space left in the Knapsack $= 15 - 13 = 2$ kg, which is not sufficient to place the next object 2 (whose wt is 3 kg). So we divide object 2 into fraction and place that fraction object 2 in the knapsack.

$$x_2 = \frac{15-13}{w_2} = \frac{2}{3} = \underline{\underline{0.66}}.$$

$\therefore$ solution tuple (vector)

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$

$(1, 0.66, 1, 0, 1, 1, 1)$

Object 4 is not placed in the Knapsack

so $x_4 = 0$

Algorithm GreedyKnapsack(m,n)

```
// P[1:n] and w[1:n] contain the profits and
// weights respectively of the n objects
// ordered such that P[i]/w[i] >= P[i+1]/w[i+1]
// m is the knapsack size (capacity) and
// x[1:n] is the solution vector
{
    for i:=1 to n do
    {
        x[i]:=0.0;        // initialize x
    }
    U:= m;
    for i:=1 to n do
    {
        If (w[i] > U) then break;
            x[i]:=1.0;
        U:=U - w[i]; // U-space left in bag
    }
    if (i<=n) then x[i]:= U/w[i];
}
```

$\therefore$ Time Complexity $T(n) = \underline{\underline{O(n)}}$

Job Sequencing with Deadlines problem:-

→ We are given a set of n jobs (programs) associated with job i is an integer deadline $d_i \geq 0$ and a profit $p_i > 0$.

→ For any job i, the profit P_i is earned ~~iff~~ the job is completed by its deadline d_i. To complete a job, one has to process the job on a machine for one unit of time.

→ Only one machine (CPU) is available for processing jobs.

→ A feasible solution for this problem is a subset J of jobs such that each job in this subset can be completed by its deadline. The value of a feasible solution J is the sum of the profits of the jobs in J,

or $\sum_{i \in J} P_i$

→ An optimal solution is a feasible solution which gives maximum profit value.
objective function maximize $\sum_{i \in J} P_i$

The next job to be included is the one that increase $\boxed{\sum_{i \in S} P_i}$ the Mosts.

Consider the jobs in decreasing order of P_i's.

Initially $S = \phi$ and $\sum_{i \in S} P_i = 0$.

Eg:- Solve the following problem of job sequencing $n=4$ jobs, profits (P_1, P_2, P_3, P_4) = $(100, 10, 15, 27)$ and deadlines $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$ hrs.

sol Since the maximum deadline is = 2 units (say hours) the feasible solution set must have ≤ 2 jobs

Arrange the jobs in decreasing order of their profits.

$$(P_1, P_4, P_3, P_2) = (100, 27, 15, 10)$$

$$(d_1, d_4, d_3, d_2) = (2, 1, 2, 1)$$

Feasible solution	Processing sequence	Profit
1) {job1}	job1	100
2) {job1, job4}	job4, job1	100 + 27 = 127
3) {1,3}	1,3 or 3,1	100 + 15 = 115
4) {1,2}	2,1	100 + 10 = 110
5) {4,3}	4,3	27+15 = 42
6) {3,2}	2,3	15+10 = 25

Machine (CPU)

slot1	slot 2
job4	job1

{1,4} is optimal solution in the processing order job4 followed by job1 with profit 127

Eg: Solve the job sequencing problem.

Given $n=5$ jobs with profits

$$\text{Profits}\,(P_1, P_2, P_3, P_4, P_5) = (1, 5, 20, 15, 10)$$
$$\text{and deadline}\,(d_1, d_2, d_3, d_4, d_5) = (1, 2, 4, 1, 3)$$

Sol: Since the maximum deadline is 4 units of time

The feasible solution set must have ≤ 4 jobs.

Arrange the jobs in decreasing order of profits.

$(P_3, P_4, P_5, P_2, P_1) = (20, 15, 10, 5, 1)$

$(d_3, d_4, d_5, d_2, d_1) = (4, 1, 3, 2, 1)$

Feasible solution	Processing sequence	Profit
1) {job3}	job3	20
2) {job3, job4}	job4, job3	20+15=35
3) {3,4,5}	4,5,3 or 4,3,5	20+15+10=45
4) {3,4,2}	4,2,3	20+15+5=40
5) {3,4,5,2}	4,2,5,3	20+15+10+5=50

6) {3,4,5,1} → not a feasible solution clash b/w job4 & job1 for slot1 of CPU
No job can be processed. This solution

7) {3, 5, 2, 1}   1, 2, 5, 3   20 + 10 + 5 + 1 = 36

solution (5) is optional solution with total profit 50.

	slot 1	slot 2	slot 3	slot 4
CPU	job 4	job 2	job 5	job 3

High-level description of job sequencing algorithm.

Algorithm GreedyJob(d, J, n)

// J is a set of jobs that can be completed

// by their deadlines.

```
{
    J := {job1};
    for i := 2 to n do
    {                          job i
        if (all jobs in J ∪ {i} can be completed by
        their deadlines) then
        J := J ∪ {i}
    }
}
```

Greedy algorithm for sequencing unit time jobs with deadlines and profits.

Algorithm JS(d, j, n)

// d[i] ≥ 1 , 1 ≤ i ≤ n are the deadlines n ≥ 1.

// The jobs are ordered such that

// p[1] ≥ p[2] ≥ ... ≥ p[n].

// J[i] is the ith job in the optimal solution,

// 1 ≤ i ≤ k. Also at termination

// d[J[i]] ≤ d[J[i+1]], 1 ≤ i ≤ k

```
{
    d[0] := J[0] := 0;   // initialize
    J[1] := 1;  // include job1
    k := 1;
    for i := 2 to n do
    {
     // consider jobs in nonincreasing(decreasing)
     // order of p[i]. Find position for i and
     // check feasibility of insertion.
        r := k;
        while ((d[J[r]] > d[i]) and (d[J[r]] ≠ r)) do
        { r := r-1;
        }
```

```
if ((d[J[r]] ≤ d[i] and (d[i] > r)) then
{
    // Insert i into J[ ]

    for q := k to (r+1) step -1 do
    {
        J[q+1] := J[q];
    }
    J[r+1] := i;

    k := k+1;
  }
}
 return k;
}
```

∴ The computing time of JS is $O(n^2) = T(n)$

jobsequencing