Greedy Algorithms

The Greedy algorithm could be understood very well with a well-known problem referred to as Knapsack problem. Although the same problem could be solved by employing other algorithmic approaches, Greedy approach solves Fractional Knapsack problem reasonably in a good time. Let us discuss the Knapsack problem in detail.

Knapsack Problem

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

The knapsack problem is in combinatorial optimization problem. It appears as a subproblem in many, more complex mathematical models of real-world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy the ignored constraints.

Applications

In many cases of resource allocation along with some constraint, the problem can be derived in a similar way of Knapsack problem. Following is a set of example.

- Finding the least wasteful way to cut raw materials
- portfolio optimization
- Cutting stock problems

Problem Scenario

A thief is robbing a store and can carry a maximal weight of W into his knapsack. There are n items available in the store and weight of i^{th} item is w_i and its profit is p_i. What items should the thief take?

In this context, the items should be selected in such a way that the thief will carry those items for which he will gain maximum profit. Hence, the objective of the thief is to maximize the profit.

Based on the nature of the items, Knapsack problems are categorized as

- Fractional Knapsack
- Knapsack

Fractional Knapsack

In this case, items can be broken into smaller pieces, hence the thief can select fractions of items.

According to the problem statement,

- There are **n** items in the store
- Weight of **i**th item $w_i > 0$
- Profit for **i**th item $p_i > 0$ and
- Capacity of the Knapsack is **W**

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction x_i of **i**th item.

$$0 \leqslant x_i \leqslant 1$$

The **i**th item contributes the weight $x_i.w_i$ to the total weight in the knapsack and profit $x_i.p_i$ to the total profit.

Hence, the objective of this algorithm is to

$$\text{maximize} \sum_{n=1}^{n} (x_i.p_i)$$

subject to constraint,

$$\sum_{n=1}^{n} (x_i.w_i) \leqslant W$$

It is clear that an optimal solution must fill the knapsack exactly, otherwise we could add a fraction of one of the remaining items and increase the overall profit.

Thus, an optimal solution can be obtained by

$$\sum_{n=1}^{n} (x_i.w_i) = W$$

In this context, first we need to sort those items according to the value of $p_i w_i$, so that $p_{i+1} w_{i+1} \leq p_i w_i$. Here, x is an array to store the fraction of items.

Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)

```
for i = 1 to n
   do x[i] = 0
weight = 0
for i = 1 to n
   if weight + w[i] ≤ W then
      x[i] = 1
      weight = weight + w[i]
   else
      x[i] = (W - weight) / w[i]
      weight = W
      break
return x
```

Analysis

If the provided items are already sorted into a decreasing order of $p_i w_i$piwi, then the whileloop takes a time in ***O(n)***; Therefore, the total time including the sort is in ***O(n logn)***.

Example 1:

Let us consider that the capacity of the knapsack ***W* = 60** and the list of provided items are shown in the following table –

Item	**A**	**B**	**C**	**D**
Profit	280	100	120	120
Weight	40	10	20	24
Ratio ($p_i w_i$)(piwi)	7	10	6	5

As the provided items are not sorted based on $p_i w_i$piwi. After sorting, the items are as shown in the following table.

Item	**B**	**A**	**C**	**D**
Profit	100	280	120	120
Weight	10	40	20	24
Ratio ($p_i w_i$)(piwi)	10	7	6	5

Solution

After sorting all the items according to $p_i w_i$piwi. First all of ***B*** is chosen as weight of ***B*** is less than the capacity of the knapsack. Next, item ***A*** is chosen, as the available capacity of the knapsack is greater than the weight of ***A***. Now, ***C*** is chosen as the next item. However, the whole item cannot be chosen as the remaining capacity of the knapsack is less than the weight of ***C***.

Hence, fraction of ***C*** (i.e. (60 − 50)/20) is chosen.

Now, the capacity of the Knapsack is equal to the selected items. Hence, no more item can be selected.

The total weight of the selected items is **10 + 40 + 10 = 60**

And the total profit is **100 + 280 + 120 * (10/20) = 380 + 60 = 440**

This is the optimal solution. We cannot gain more profit selecting any different combination of items.

Example 2:

$I = (I_1, I_2, I_3, I_4, I_5)$

w = (5, 10, 20, 30, 40)

v = (30, 20, 100, 90,160)

The capacity of knapsack W = 80

Solution:

ITEM	w_i	b_i
I_1	5	30
I_2	10	20
I_3	20	100
I_4	30	90
I_5	40	160

Taking value per weight ratio i.e. p_i=

ITEM	w_i	b_i	P_i=
I_1	5	30	6.0
I_2	10	20	2.0
I_3	20	100	5.0
I_4	30	90	3.0
I_5	40	160	4.0

Now, arrange the value of p_i in decreasing order.

ITEM	w_i	v_i	p_i=
I_1	5	30	6.0
I_3	20	100	5.0
I_5	40	160	4.0
I_4	30	90	3.0
I_2	10	20	2.0

The total weight of the selected items is **5 + 20+ 25 = 80**

And the total profit is **30 + 100 + 160 * (25/40) = 30 + 100+100 =230**

DAA - Job Sequencing with Deadline

Problem Statement

In job sequencing problem, the objective is to find a sequence of jobs, which is completed within their deadlines and gives maximum profit.

Solution

Let us consider, a set of ***n*** given jobs which are associated with deadlines and profit is earned, if a job is completed by its deadline. These jobs need to be ordered in such a way that there is maximum profit.

It may happen that all of the given jobs may not be completed within their deadlines.

Assume, deadline of **i**th job J_i is d_i and the profit received from this job is p_i. Hence, the optimal solution of this algorithm is a feasible solution with maximum profit.

Thus, $D(i)>0$D(i)>0 for $1 \leqslant i \leqslant n1 \leqslant i \leqslant n$.
Initially, these jobs are ordered according to profit, i.e. $p_1 \geqslant p_2 \geqslant p_3 \geqslant ... \geqslant p_n p1 \geqslant p2 \geqslant p3 \geqslant ... \geqslant pn$.

```
Algorithm: Job-Sequencing-With-Deadline (D, J, n, k)
D(0) := J(0) := 0
k := 1
J(1) := 1   // means first job is selected
for i = 2 … n do
   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
      for l = k … r + 1 by -1 do
         J(l + 1) := J(l)
         J(r + 1) := i
         k := k + 1
```

Analysis

In this algorithm, we are using two loops, one is within another. Hence, the complexity of this algorithm is $O(n^2)$O(n2).

Example 1:

The greedy algorithm described below always gives an optimal solution to the job sequencing problem-

Step-01:

- Sort all the given jobs in decreasing order of their profit.

Step-02:

- Check the value of maximum deadline.
- Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline.

Step-03:

- Pick up the jobs one by one.
- Put the job on Gantt chart as far as possible from 0 ensuring that the job gets completed before its deadline.

PRACTICE PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

Problem-

Given the jobs, their deadlines and associated profits as shown-

Jobs	**J1**	**J2**	**J3**	**J4**	**J5**	**J6**
Deadlines	5	3	3	2	4	2
Profits	200	180	190	300	120	100

Answer the following questions-

1. Write the optimal schedule that gives maximum profit.
2. Are all the jobs completed in the optimal schedule?
3. What is the maximum earned profit?

Solution-

Step-01:

Sort all the given jobs in decreasing order of their profit-

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-02:

Value of maximum deadline = 5.

So, draw a Gantt chart with maximum time on Gantt chart = 5 units as shown-

Gantt Chart

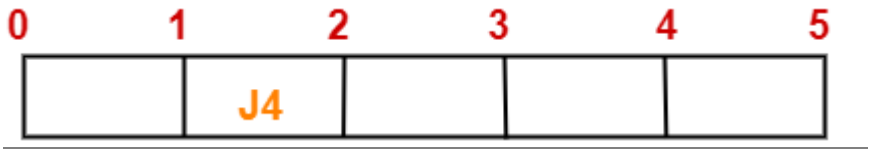
Now,

- We take each job one by one in the order they appear in Step-01.
- We place the job on Gantt chart as far as possible from 0.

Step-03:

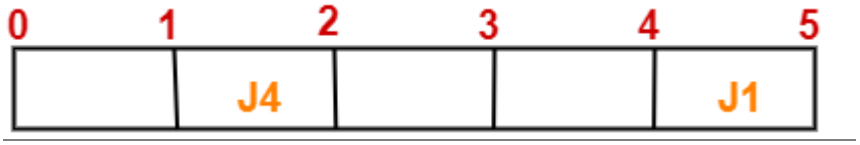
- We take job J4.

- Since its deadline is 2, so we place it in the first empty cell before deadline 2 as-



Step-04:

- We take job J1.
- Since its deadline is 5, so we place it in the first empty cell before deadline 5 as-



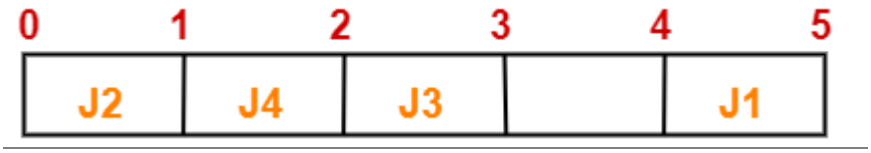
Step-05:

- We take job J3.
- Since its deadline is 3, so we place it in the first empty cell before deadline 3 as-

Step-06:

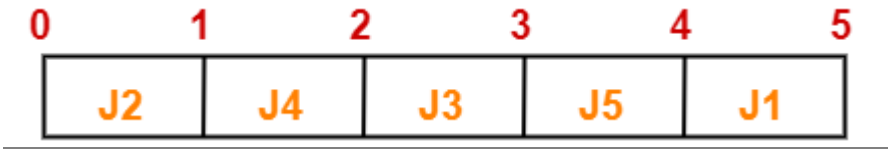
- We take job J2.
- Since its deadline is 3, so we place it in the first empty cell before deadline 3.

- Since the second and third cells are already filled, so we place job J2 in the first cell as-



Step-07:

- Now, we take job J5.
- Since its deadline is 4, so we place it in the first empty cell before deadline 4 as-



Now,

- The only job left is job J6 whose deadline is 2.
- All the slots before deadline 2 are already occupied.
- Thus, job J6 can not be completed.

Now, the given questions may be answered as-

Part-01:

The optimal schedule is-

J2 , J4 , J3 , J5 , J1

This is the required order in which the jobs must be completed in order to obtain the maximum profit.

Part-02:

- All the jobs are not completed in optimal schedule.
- This is because job J6 could not be completed within its deadline.

Part-03:

Maximum earned profit

= Sum of profit of all the jobs in optimal schedule

= Profit of job J2 + Profit of job J4 + Profit of job J3 + Profit of job J5 + Profit of job J1

= 180 + 300 + 190 + 120 + 200

Example 2:

Let us consider a set of given jobs as shown in the following table. We have to find a sequence of jobs, which will be completed within their deadlines and will give maximum profit. Each job is associated with a deadline and profit.

Job	**J_1**	**J_2**	**J_3**	**J_4**	**J_5**
Deadline	2	1	3	2	1
Profit	60	100	20	40	20

Solution

To solve this problem, the given jobs are sorted according to their profit in a descending order. Hence, after sorting, the jobs are ordered as shown in the following table.

Job	**J_2**	**J_1**	**J_4**	**J_3**	**J_5**

Deadline	1	2	2	3	1
Profit	100	60	40	20	20

From this set of jobs, first we select J_2, as it can be completed within its deadline and contributes maximum profit.

- Next, J_1 is selected as it gives more profit compared to J_4.
- In the next clock, J_4 cannot be selected as its deadline is over, hence J_3 is selected as it executes within its deadline.
- The job J_5 is discarded as it cannot be executed within its deadline.

Thus, the solution is the sequence of jobs (J_2, J_1, J_3), which are being executed within their deadline and gives maximum profit.

Total profit of this sequence is **100 + 60 + 20 = 180**.