Greedy Algorithms

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy algorithms are used for optimization problems.

An optimization problem can be solved using Greedy if the problem has the following property:

- At every step, we can make a choice that looks best at the moment, and we get the optimal solution to the complete problem.
- Some popular Greedy Algorithms are Fractional Knapsack, Dijkstra's algorithm, Kruskal's algorithm, Huffman coding and Prim's Algorithm
- Also used to get an approximation for Hard optimization problems. For example, the Traveling Salesman Problem is an NP-Hard problem. A Greedy choice for this problem is to pick the nearest unvisited city from the current city at every step. These solutions don't always produce the best optimal solution but can be used to get an approximately optimal solution.
- **Greedy Choice Property:** The optimal solution can be constructed by making the best local choice at each step.
- **Optimal Substructure:** The optimal solution to the problem contains the optimal solutions to its sub-problems.

How does a Greedy Algorithm work?

Greedy Algorithm solve optimization problems by making the best local choice at each step in the hope of finding the global optimum. It's like taking the best option available at each moment, hoping it will lead to the best overall outcome.

Here's how it works:

1. Start with the initial state of the problem.
2. Consider all the options available at that specific moment.
3. Choose the option that seems best at that moment, regardless of future consequences. This is the "greedy" part - you take the best option available now, even if it might not be the best in the long run.
4. Move to the new state based on your chosen option. This becomes your new starting point for the next iteration.
5. Repeat steps 2-4 until you reach the goal state or no further progress is possible.

Job Sequencing with Deadline

Job scheduling algorithm is applied to schedule the jobs on a single processor to maximize the profits.

The greedy approach of the job scheduling algorithm states that, Given n number of jobs with a starting time and ending time, they need to be scheduled in such a way that maximum profit is received within the maximum deadline.

Job Scheduling Algorithm

Set of jobs with deadlines and profits are taken as an input with the job scheduling algorithm and scheduled subset of jobs with maximum profit are obtained as the final output.

Algorithm

Step1 − Find the maximum deadline value from the input set of jobs.
Step2 − Once, the deadline is decided, arrange the jobs in descending order of their profits.
Step3 − For each job in the sorted order, find the latest available time slot that is less than or equal to its deadline and assign the job to that slot.
Step4 − The selected set of jobs are the output.

Examples

Consider the following tasks with their deadlines and profits. Schedule the tasks in such a way that they produce maximum profit after being executed −

S. No.	1	2	3	4	5
Jobs	J1	J2	J3	J4	J5

Deadlines	2	2	1	3	4
Profits	20	60	40	100	80

Step1:

Find the maximum deadline value, d_m, from the deadlines given.

dm = 4

This means we have 4 time slots available: [1], [2], [3], [4]

Step2:

Arrange the jobs in descending order of their profits.

S. No.	1	2	3	4	5
Jobs	J4	J5	J2	J3	J1
Deadlines	3	4	2	1	2
Profits	100	80	60	40	20

Step 3:

Select jobs greedily, placing each in the latest available time slot that does not exceed its deadline.

Iteration 1: J4 (Profit = 100, Deadline = 3)

Find the latest free slot ≤ 3. Slot 3 is free, so assign J4 to slot 3.

Slots: [_, _, J4, _]

Total Profit = 100

Iteration 2: J5 (Profit = 80, Deadline = 4)

Find the latest free slot ≤ 4. Slot 4 is free, so assign J5 to slot 4.

Slots: [_, _, J4, J5]
Total Profit = 100 + 80 = 180

Iteration 3: J2 (Profit = 60, Deadline = 2)

Find the latest free slot ≤ 2. Slot 2 is free, so assign J2 to slot 2.

Slots: [_, J2, J4, J5]
Total Profit = 180 + 60 = 240

Iteration 4: J3 (Profit = 40, Deadline = 1)

Find the latest free slot ≤ 1. Slot 1 is free, so assign J3 to slot 1.

Slots: [J3, J2, J4, J5]
Total Profit = 240 + 40 = 280

Iteration 5: J1 (Profit = 20, Deadline = 2)

Find the latest free slot ≤ 2. Slots 1 and 2 are already occupied. J1 cannot be scheduled.

Step 4:

All jobs have been considered. The algorithm terminates.

Result:

The optimal sequence of jobs scheduled within their deadlines is **{J3, J2, J4, J5}** (executed in time slots 1, 2, 3, 4 respectively) with the **maximum profit of 280**.

Time Slot	1	2	3	4
Job	J3	J2	J4	J5
Profit	40	60	100	80

Maximum Profit: 40 + 60 + 100 + 80 = 280

Fractional Knapsack Problem

The knapsack problem states that − given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that, the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum.

It is one of the most popular problems that take greedy approach to be solved. It is called as the Fractional Knapsack Problem.

To explain this problem a little easier, consider a test with 12 questions, 10 marks each, out of which only 10 should be attempted to get the maximum mark of 100. The test taker now must calculate the highest profitable questions the one that hes confident in to achieve the maximum mark. However, he cannot attempt all the 12 questions since there will not be any extra marks awarded for those attempted answers. This is the most basic real-world application of the knapsack problem.

Knapsack Algorithm

The weights (Wi) and profit values (Pi) of the items to be added in the knapsack are taken as an input for the fractional knapsack algorithm and the subset of the items added in the knapsack without exceeding the limit and with maximum profit is achieved as the output.

Algorithm

- Consider all the items with their weights and profits mentioned respectively.
- Calculate P_i/W_i of all the items and sort the items in descending order based on their P_i/W_i values.
- Without exceeding the limit, add the items into the knapsack.
- If the knapsack can still store some weight, but the weights of other items exceed the limit, the fractional part of the next time can be added.
- Hence, giving it the name fractional knapsack problem.

Examples

- For the given set of items and the knapsack capacity of 10 kg, find the subset of the items to be added in the knapsack such that the profit is maximum.

Items	1	2	3	4	5
Weights (in kg)	3	3	2	5	1
Profits	10	15	10	12	8

Solution

Step 1:

Given, n = 5

Wi = {3, 3, 2, 5, 1}
Pi = {10, 15, 10, 12, 8}

Calculate P_i/W_i for all the items

Items	1	2	3	4	5
Weights (in kg)	3	3	2	5	1
Profits	10	15	10	20	8
P_i/W_i	3.3	5	5	4	8

Step 2:

Arrange all the items in descending order based on P_i/W_i

Items	5	2	3	4	1
Weights (in kg)	1	3	2	5	3
Profits	8	15	10	20	10

P_i/W_i	8	5	5	4	3.3

Step 3:

Without exceeding the knapsack capacity, insert the items in the knapsack with maximum profit.

Knapsack = {5, 2, 3}

However, the knapsack can still hold 4 kg weight, but the next item having 5 kg weight will exceed the capacity. Therefore, only 4 kg weight of the 5 kg will be added in the knapsack.

Items	5	2	3	4	1
Weights (in kg)	1	3	2	5	3
Profits	8	15	10	20	10
Knapsack	1	1	1	4/5	0

Hence, the knapsack holds the weights = [(1 * 1) + (1 * 3) + (1 * 2) + (4/5 * 5)] = 10, with maximum profit of [(1 * 8) + (1 * 15) + (1 * 10) + (4/5 * 20)] = 37.