# DAA - Greedy Method

Among all the algorithmic approaches, the simplest and straightforward approach is the Greedy method. In this approach, the decision is taken on the basis of current available information without worrying about the effect of the current decision in future.

Greedy algorithms build a solution part by part, choosing the next part in such a way, that it gives an immediate benefit. This approach never reconsiders the choices taken previously. This approach is mainly used to solve optimization problems. Greedy method is easy to implement and quite efficient in most of the cases. Hence, we can say that Greedy algorithm is an algorithmic paradigm based on heuristic that follows local quite efficient in most of the cases. Hence, we can say that Greedy algorithm is an algorithmic paradigm based on heuristic that follows local optimal choice at each step with the hope of finding global optimal solution.

In many problems, it does not produce an optimal solution though it gives an approximate (near optimal) solution in a reasonable time.

## Components of Greedy Algorithm

Greedy algorithms have the following five components −

- **A candidate set** − A solution is created from this set.
- **A selection function** − Used to choose the best candidate to be added to the solution.
- **A feasibility function** − Used to determine whether a candidate can be used to contribute to the solution.
- **An objective function** − Used to assign a value to a solution or a partial solution.
- **A solution function** − Used to indicate whether a complete solution has been reached.

## Areas of Application

Greedy approach is used to solve many problems, such as

- Finding the shortest path between two vertices using Dijkstra’s algorithm.
- Finding the minimal spanning tree in a graph using Prim’s /Kruskal’s algorithm, etc.

## Where Greedy Approach Fails

In many problems, Greedy algorithm fails to find an optimal solution, moreover it may produce a worst solution. Problems like Travelling Salesman and Knapsack cannot be solved using this approach.

# DAA - Fractional Knapsack

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

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

subject to constraint,

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

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

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

In this context, first we need to sort those items according to the value of $\frac{p_i}{w_i}$, so that $\frac{p_i+1}{w_i+1}$ ≤ $\frac{p_i}{w_i}$. 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 $\mathbf{\frac{p_i}{w_i}}$, then the whileloop takes a time in ***O(n)***; Therefore, the total time including the sort is in ***O(n logn)***.

## Example

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 $(\frac{p_i}{w_i})$ | 7 | 10 | 6 | 5 |

As the provided items are not sorted based on $\frac{\mathbf{p_i}}{\mathbf{w_i}}$ . 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 $(\frac{p_i}{w_i})$ | 10 | 7 | 6 | 5 |

## Solution

After sorting all the items according to $\frac{p_i}{w_i}$ . 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 + 20 * (10/20) = 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.