Greedy Algorithms

Optimization problem

- In mathematics, engineering, computer science and economics, an **optimization problem** is the problem of finding the *best* solution from all feasible solutions
- An optimization problem is one in which you want to find, not just *a* solution, but the *best* solution
- A “greedy algorithm” sometimes works well for optimization problems
- But only a few optimization problems can be solved by the greedy method.

Greedy algorithms

- **Greedy algorithms** are a class of algorithms that make **locally optimal** choices at each step with the hope of finding a **global optimum** solution.
- In these algorithms, decisions are made based on the information available at the current moment without considering the consequences of these decisions in the future.
- The key idea is to select the best possible choice at each step, leading to a solution that may not always be the most optimal but is often good enough for many problems.

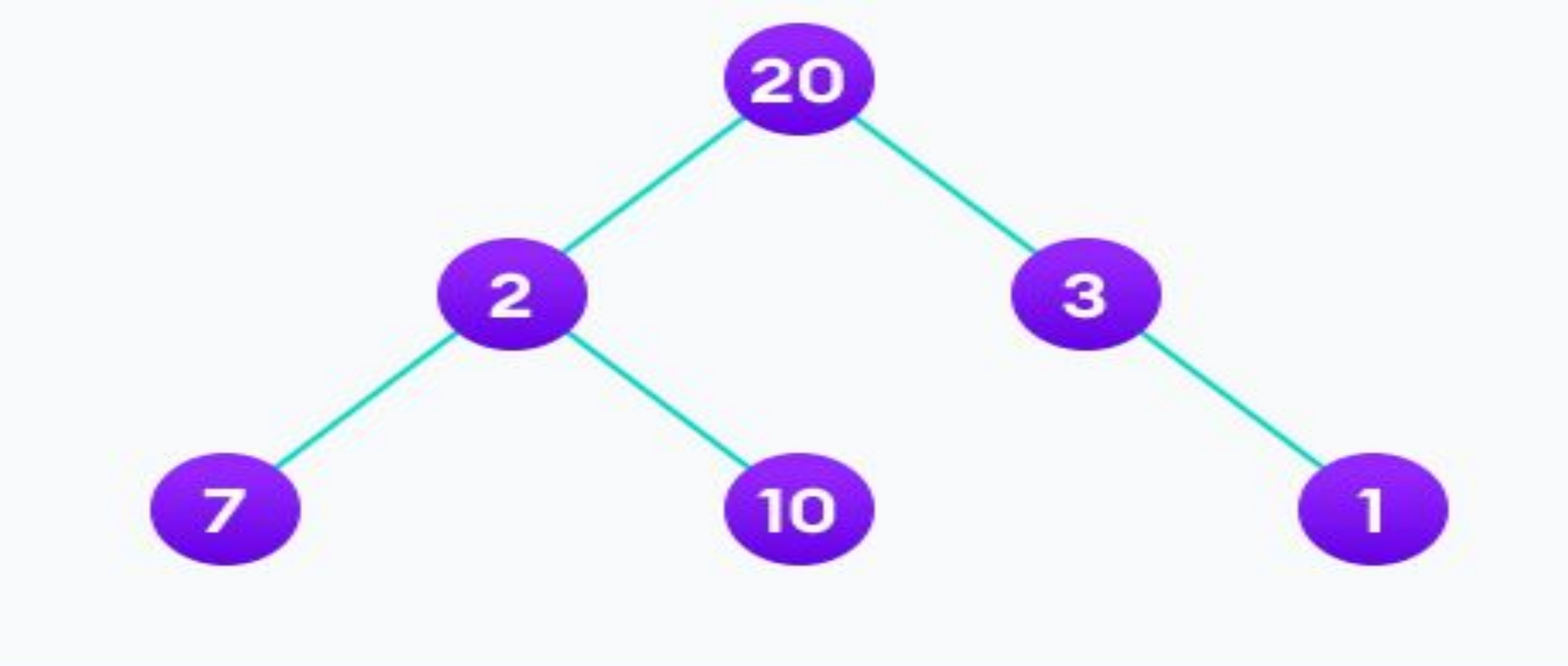
Greedy Method

- Suppose that a problem can be solved by a sequence of decisions.
- The greedy method has that each decision is locally optimal.
- These locally optimal solutions will finally add up to a globally optimal solution.
- A greedy algorithm works in phases.
- At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum

Greedy Algorithm

- To begin with, the solution set (containing answers) is empty.
- At each step, an item is added to the solution set until a solution is reached.
- If the solution set is feasible, the current item is kept.
- Else, the item is rejected and never considered again.

Greedy Method example



Apply greedy approach to this tree to find the longest route

- Problem: You have to make a change of an amount using the smallest possible number of coins. Amount: $18
- Available coins are $5 coin $2 coin $1 coin
- There is no limit to the number of each coin you can use.

- Create an empty solution-set = { }. Available coins are {5, 2, 1}.
- We are supposed to find the sum = 18. Let's start with sum = 0.
- Always select the coin with the largest value (i.e. 5) until the sum > 18. (When we select the largest value at each step, we hope to reach the destination faster. This concept is called **greedy choice property**.)

- In the first iteration, solution-set = {5} and sum = 5.
- In the second iteration, solution-set = {5, 5} and sum = 10.
- In the third iteration, solution-set = {5, 5, 5} and sum = 15.
- In the fourth iteration, solution-set = {5, 5, 5, 2} and sum = 17. (We cannot select 5 here because if we do so, sum = 20 which is greater than 18. So, we select the 2nd largest item which is 2.)
- Similarly, in the fifth iteration, select 1. Now sum = 18 and solution-set = {5, 5, 5, 2, 1}.

A failure of the greedy algorithm

- In some (fictional) monetary system, “krons” come in1 kron,7 kron, and10 kron coins
- Using a greedy algorithm to count out 15 krons, you would get
  - –A 10 kron piece
  - –Five 1 kron pieces, for a total of 15 krons
  - –This requires six coins
- A better solution would be to use two 7 kron pieces and one 1 kron piece
  - –This only requires three coins

•The greedy algorithm results in a solution, but not in an optimal solution

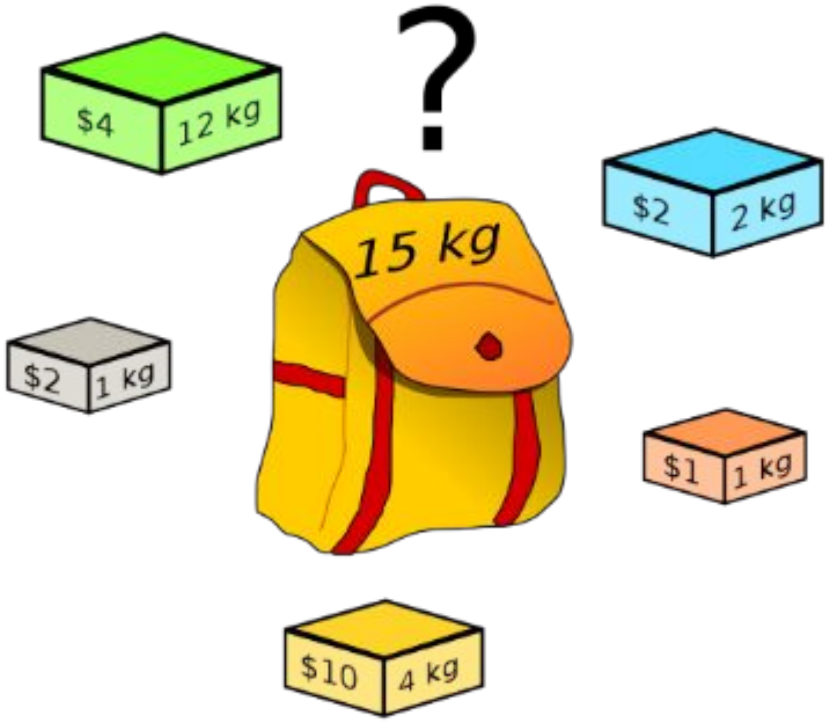
Knapsack Problem

Knapsack Problem

- 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.

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.



Knapsack Problem

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 ith item is wi and its profit is pi. 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 $wi > 0$
  - Profit of **i**th item $pi > 0$ 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.
  - $0 \leqslant xi \leqslant 1$

Knapsack Problem

- The ith item contributes the weight xi.wi to the total weight in the Knapsack and profit xi.pi to the total profit.
- Hence, the objective of this algorithm is to
- $$maximize \sum_{n=1}^{n}(x_i.pi)$$
- Subject to the constraint,

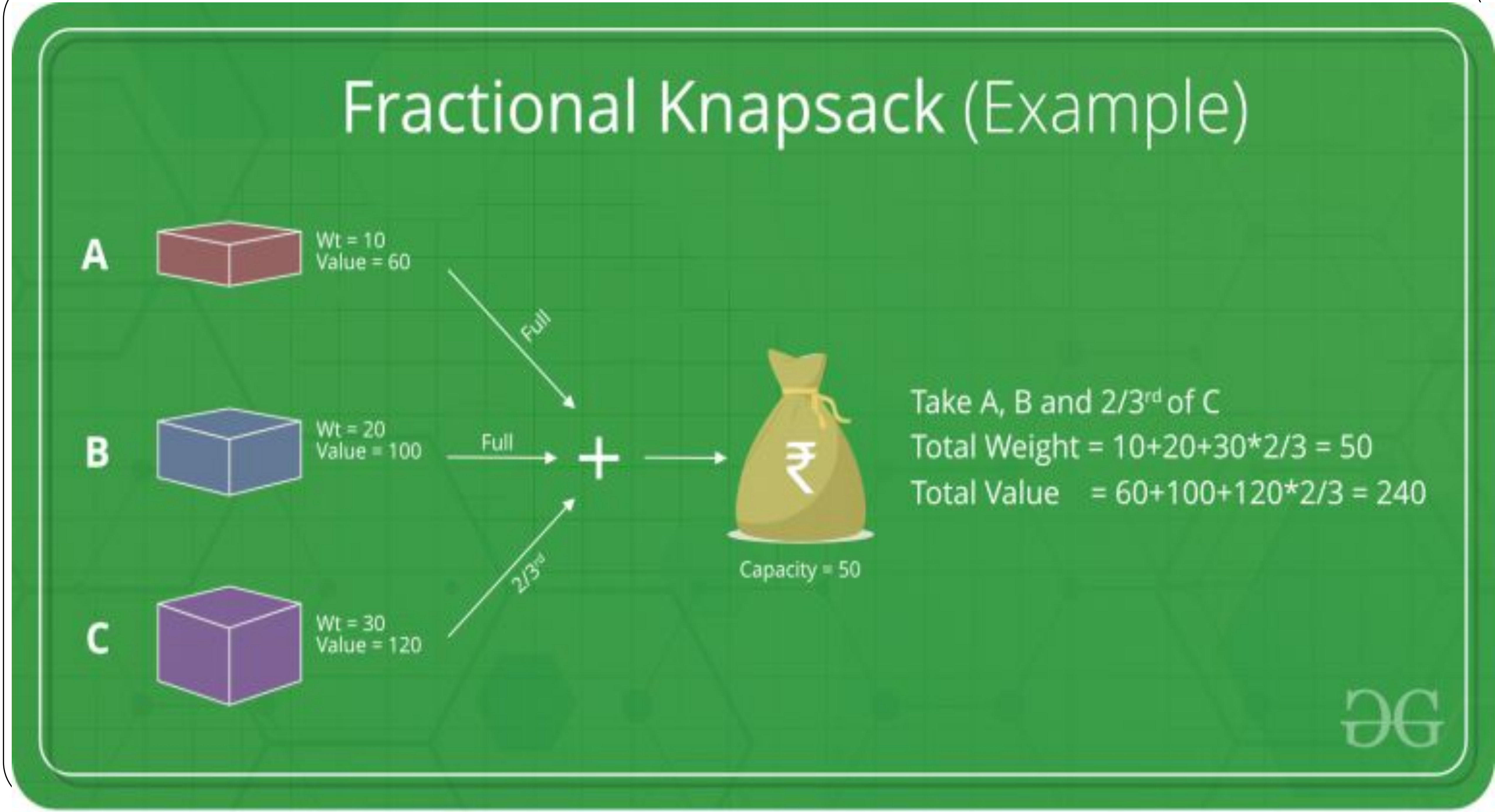
$$\sum_{n=1}^{n}(x_i.wi) \leqslant W$$

- 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

$$\frac{p_i}{w_i} \text{ , so that } \frac{p_i+1}{w_i+1} \leq \frac{p_i}{w_i}$$

- Here, ***x*** is an array to store the fraction of items.


Fractional Knapsack (Example)
A
Wt = 10
Value = 60
Full
B
Wt = 20
Value = 100
Full
C
Wt = 30
Value = 120
2/3rd
+
₹
Capacity = 50
Take A, B and 2/3rd of C
Total Weight = 10+20+30*2/3 = 50
Total Value = 60+100+120*2/3 = 240

Knapsack Algorithm

```
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 pi/wi, then the while loop 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 is 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 $\left(\frac{p_i}{w_i}\right)$	7	10	6	5

- After sorting , the items are as shown in following table

Item	B	A	C	D
Profit	100	280	120	120
Weight	10	40	20	24
Ratio $\left(\frac{p_i}{w_i}\right)$	10	7	6	5

Solution

- 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.

Problem 2

- For the given set of items and knapsack capacity = 60 kg, find the optimal solution for the fractional knapsack problem making use of greedy approach.

Item	Weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90