# Fractional Knapsack Problem



Given the weights and profits of **N** items, in the form of **{profit, weight}** put these items in a knapsack of capacity **W** to get the maximum total profit in the knapsack. In **Fractional Knapsack**, we can break items for maximizing the total value of the knapsack.

***Input:*** *arr[] = {{60, 10}, {100, 20}, {120, 30}}, W = 50*
***Output:*** *240*
***Explanation:*** *By taking items of weight 10 and 20 kg and 2/3 fraction of 30 kg.*
*Hence total price will be 60+100+(2/3)(120) = 240*

***Input:*** *arr[] = {{500, 30}}, W = 10*
***Output:*** *166.667*



**Naive Approach:** To solve the problem follow the below idea:

*Try all possible subsets with all different fractions.*

**Time Complexity:** $O(2^N)$
**Auxiliary Space:** O(N)

## Fractional Knapsack Problem using Greedy algorithm:

An efficient solution is to use the Greedy approach.

*The basic idea of the greedy approach is to calculate the ratio **profit/weight** for each item and sort the item on the basis of this ratio. Then take the item with the highest ratio and add them as much as we can (can be the whole element or a fraction of it).*

*This will always give the maximum profit because, in each step it adds an element such that this is the maximum possible profit for that much weight.*

**Illustration:**

Check the below illustration for a better understanding:

*Consider the example: **arr[] = {{100, 20}, {60, 10}, {120, 30}}, W = 50**.*

***Sorting:** Initially sort the array based on the profit/weight ratio. The sorted array will be **{{60, 10}, {100, 20}, {120, 30}}**.*

***Iteration:***

- *For **i = 0**, weight = 10 which is less than W. So add this element in the knapsack. **profit = 60** and remaining **W = 50 – 10 = 40**.*
- *For **i = 1**, weight = 20 which is less than W. So add this element too. **profit = 60 + 100 = 160** and remaining **W = 40 – 20 = 20**.*
- *For **i = 2**, weight = 30 is greater than W. So add 20/30 fraction = **2/3** fraction of the element. Therefore **profit** = 2/3 * 120 + 160 = 80 + 160 = **240** and remaining **W** becomes **0**.*

*So the final profit becomes **240** for **W = 50**.*

Follow the given steps to solve the problem using the above approach:

- Calculate the ratio (**profit/weight**) for each item.
- Sort all the items in decreasing order of the ratio.
- Initialize **res = 0**, curr_cap = given_cap.
- Do the following for every item **i** in the sorted order:
  - If the weight of the current item is less than or equal to the remaining capacity then add the value of that item into the result
  - Else add the current item as much as we can and break out of the loop.
- Return **res**.

Below is the implementation of the above approach:

---

## C++

```
// C++ program to solve fractional Knapsack Problem

#include <bits/stdc++.h>
using namespace std;

// Structure for an item which stores weight and
// corresponding value of Item
struct Item {
    int profit, weight;

    // Constructor
    Item(int profit, int weight)
    {
        this->profit = profit;
        this->weight = weight;
    }
};

// Comparison function to sort Item
// according to profit/weight ratio
static bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.profit / (double)a.weight;
    double r2 = (double)b.profit / (double)b.weight;
    return r1 > r2;
}

// Main greedy function to solve problem
```

```
double fractionalKnapsack(int W, struct Item arr[], int N)
{
    // Sorting Item on basis of ratio
    sort(arr, arr + N, cmp);

    double finalvalue = 0.0;

    // Looping through all items
    for (int i = 0; i < N; i++) {

        // If adding Item won't overflow,
        // add it completely
        if (arr[i].weight <= W) {
            W -= arr[i].weight;
            finalvalue += arr[i].profit;
        }

        // If we can't add current Item,
        // add fractional part of it
        else {
            finalvalue
                += arr[i].profit
                   * ((double)W / (double)arr[i].weight);
            break;
        }
    }

    // Returning final value
    return finalvalue;
}

// Driver code
int main()
{
    int W = 50;
    Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };
    int N = sizeof(arr) / sizeof(arr[0]);

    // Function call
    cout << fractionalKnapsack(W, arr, N);
    return 0;
}
```

## Java

```
// Java program to solve fractional Knapsack Problem

import java.lang.*;
import java.util.Arrays;
import java.util.Comparator;
```

```
// Greedy approach
public class FractionalKnapSack {

    // Function to get maximum value
    private static double getMaxValue(ItemValue[] arr,
                                      int capacity)
    {
        // Sorting items by profit/weight ratio;
        Arrays.sort(arr, new Comparator<ItemValue>() {
            @Override
            public int compare(ItemValue item1,
                               ItemValue item2)
            {
                double cpr1
                    = new Double((double)item1.profit
                                 / (double)item1.weight);
                double cpr2
                    = new Double((double)item2.profit
                                 / (double)item2.weight);

                if (cpr1 < cpr2)
                    return 1;
                else
                    return -1;
            }
        });

        double totalValue = 0d;

        for (ItemValue i : arr) {

            int curWt = (int)i.weight;
            int curVal = (int)i.profit;

            if (capacity - curWt >= 0) {

                // This weight can be picked whole
                capacity = capacity - curWt;
                totalValue += curVal;
            }
            else {

                // Item cant be picked whole
                double fraction
                    = ((double)capacity / (double)curWt);
                totalValue += (curVal * fraction);
                capacity
                    = (int)(capacity - (curWt * fraction));
                break;
            }
        }

        return totalValue;
```

```
    }

    // Item value class
    static class ItemValue {

        int profit, weight;

        // Item value function
        public ItemValue(int val, int wt)
        {
            this.weight = wt;
            this.profit = val;
        }
    }

    // Driver code
    public static void main(String[] args)
    {

        ItemValue[] arr = { new ItemValue(60, 10),
                            new ItemValue(100, 20),
                            new ItemValue(120, 30) };

        int capacity = 50;

        double maxValue = getMaxValue(arr, capacity);

        // Function call
        System.out.println(maxValue);
    }
}
```

## Python3

```
# Structure for an item which stores weight and
# corresponding value of Item
class Item:
    def __init__(self, profit, weight):
        self.profit = profit
        self.weight = weight

# Main greedy function to solve problem
def fractionalKnapsack(W, arr):

    # Sorting Item on basis of ratio
    arr.sort(key=lambda x: (x.profit/x.weight), reverse=True)

    # Result(value in Knapsack)
    finalvalue = 0.0

    # Looping through all Items
```

```
    for item in arr:

        # If adding Item won't overflow,
        # add it completely
        if item.weight <= W:
            W -= item.weight
            finalvalue += item.profit

        # If we can't add current Item,
        # add fractional part of it
        else:
            finalvalue += item.profit * W / item.weight
            break

    # Returning final value
    return finalvalue


# Driver Code
if __name__ == "__main__":
    W = 50
    arr = [Item(60, 10), Item(100, 20), Item(120, 30)]

    # Function call
    max_val = fractionalKnapsack(W, arr)
    print(max_val)
```

## C#

```
// C# program to solve fractional Knapsack Problem

using System;
using System.Collections;

class GFG {

    // Class for an item which stores weight and
    // corresponding value of Item
    class item {
        public int profit;
        public int weight;

        public item(int profit, int weight)
        {
            this.profit = profit;
            this.weight = weight;
        }
    }

    // Comparison function to sort Item according
    // to val/weight ratio
```

```
class cprCompare : IComparer {
    public int Compare(Object x, Object y)
    {
        item item1 = (item)x;
        item item2 = (item)y;
        double cpr1 = (double)item1.profit
                      / (double)item1.weight;
        double cpr2 = (double)item2.profit
                      / (double)item2.weight;

        if (cpr1 < cpr2)
            return 1;

        return cpr1 > cpr2 ? -1 : 0;
    }
}

// Main greedy function to solve problem
static double FracKnapSack(item[] items, int w)
{

    // Sort items based on cost per units
    cprCompare cmp = new cprCompare();
    Array.Sort(items, cmp);

    // Traverse items, if it can fit,
    // take it all, else take fraction
    double totalVal = 0f;
    int currW = 0;

    foreach(item i in items)
    {
        float remaining = w - currW;

        // If the whole item can be
        // taken, take it
        if (i.weight <= remaining) {
            totalVal += (double)i.profit;
            currW += i.weight;
        }

        // dd fraction until we run out of space
        else {
            if (remaining == 0)
                break;

            double fraction
                = remaining / (double)i.weight;
            totalVal += fraction * (double)i.profit;
            currW += (int)(fraction * (double)i.weight);
        }
    }
    return totalVal;
```

```
    }

    // Driver code
    static void Main(string[] args)
    {
        int W = 50;
        item[] arr = { new item(60, 10), new item(100, 20),
                       new item(120, 30) };

        // Function call
        Console.WriteLine(FracKnapSack(arr, W));
    }
}

// This code is contributed by Mohamed Adel
```

## Javascript

```
// JavaScript program to solve fractional Knapsack Problem

// Structure for an item which stores weight and
// corresponding value of Item
class Item {
    constructor(profit, weight) {
        this.profit = profit;
        this.weight = weight;
    }
}

// Comparison function to sort Item
// according to val/weight ratio
function cmp(a, b) {
    let r1 = a.profit / a.weight;
    let r2 = b.profit / b.weight;
    return r1 > r2;
}

// Main greedy function to solve problem
function fractionalKnapsack(W, arr) {
    // Sorting Item on basis of ratio
    arr.sort(cmp);

    let finalvalue = 0.0;

    // Looping through all items
    for (let i = 0; i < arr.length; i++) {

        // If adding Item won't overflow,
        // add it completely
        if (arr[i].weight <= W) {
            W -= arr[i].weight;
```

```
            finalvalue += arr[i].profit;
        }

        // If we can't add current Item,
        // add fractional part of it
        else {
            finalvalue += arr[i].profit * (W / arr[i].weight);
            break;
        }
    }

    // Returning final value
    return finalvalue;
}

// Driver code
let W = 50;
let arr = [new Item(60, 10), new Item(100, 20), new Item(120, 30)];

console.log(fractionalKnapsack(W, arr));

// This code is contributed by lokeshpotta20
```



### Output

```
240
```

**Time Complexity:** O(N * logN)
**Auxiliary Space:** O(N)

Last Updated : 03 Apr, 2023

285

Previous

**Introduction to Knapsack Problem, its Types and How to solve them**

Next

**Fractional Knapsack Queries**

Share your thoughts in the comments

Add Your Comment

## Similar Reads

Difference between 0/1 Knapsack problem and Fractional Knapsack problem

C++ Program for the Fractional Knapsack Problem

Fractional Knapsack Queries

Unbounded Fractional Knapsack

0/1 Knapsack Problem to print all possible solutions

Extended Knapsack Problem

Introduction to Knapsack Problem, its Types and How to solve them

Crossword Puzzle Of The Week #20 (KnapSack Problem)

GFact | Why doesn't Greedy Algorithm work for 0-1 Knapsack problem?

Java Program 0-1 Knapsack Problem

## Complete Tutorials

Learn Algorithms with Javascript | DSA using JavaScript Tutorial

DSA Crash Course | Revision Checklist with Interview Guide

Learn Data Structures and Algorithms | DSA Tutorial

Mathematical and Geometric Algorithms - Data Structure and Algorithm Tutorials

Learn Data Structures with Javascript | DSA using JavaScript Tutorial

U Utkarsh Trivedi

Article Tags : Fraction , knapsack , DSA , Greedy

Practice Tags : Greedy

## Additional Information

A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

Data Visualisation Tutorial

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

Bootstrap

Tailwind CSS

SASS

LESS

Web Design

## Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

Web Scraping

OpenCV Python Tutorial

Python Interview Question

## Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

## DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

DevOps Roadmap

## Competitive Programming

Top DS or Algo for CP

Top 50 Tree

Top 50 Graph

Top 50 Array

Top 50 String

Top 50 DP

Top 15 Websites for CP

## System Design

High Level Design

Low Level Design

UML Diagrams

Interview Guide

Design Patterns

OOAD

System Design Bootcamp

Interview Questions

## JavaScript

JavaScript Examples

TypeScript

ReactJS

NextJS

AngularJS

NodeJS

Lodash

Web Browser

## NCERT Solutions

Class 12

Class 11

## School Subjects

Mathematics

Physics

Class 10

Class 9

Class 8

Complete Study Material

Chemistry

Biology

Social Science

English Grammar

## Commerce

Accountancy

Business Studies

Economics

Management

HR Management

Finance

Income Tax

## UPSC Study Material

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economy Notes

Ethics Notes

Previous Year Papers

## SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

SSC CGL Practice Papers

## Colleges

Indian Colleges Admission & Campus Experiences

List of Central Universities - In India

Colleges in Delhi University

IIT Colleges

NIT Colleges

IIIT Colleges

## Companies

META Owned Companies

Alphabhet Owned Companies

TATA Group Owned Companies

Reliance Owned Companies

Fintech Companies

EdTech Companies

## Preparation Corner

Company-Wise Recruitment Process

Resume Templates

Aptitude Preparation

Puzzles

Company-Wise Preparation

## Exams

JEE Mains

JEE Advanced

GATE CS

NEET

UGC NET

## More Tutorials

Software Development

Software Testing

Product Management

SAP

SEO - Search Engine Optimization

Linux

Excel

## Free Online Tools

Typing Test

Image Editor

Code Formatters

Code Converters

Currency Converter

Random Number Generator

Random Password Generator

## Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Share your Experiences

Internships