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*

Recommended Problem
Fractional Knapsack
Greedy Algorithms Microsoft
Solve Problem
Submission count: 1.8L

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

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy
Got It !

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:

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

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
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
    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;
        }
    }
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
}

// 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;
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
        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),
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
        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
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
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;
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
// 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
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
}

// 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);
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

```
    }

    // 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)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Feeling lost in the world of random DSA topics, wasting time without progress? It's time for a change! Join our DSA course, where we'll guide you on an exciting journey to master DSA efficiently and on schedule.
Ready to dive in? Explore our Free Demo Content and join our DSA course, trusted by over 100,000 geeks!

- DSA in C++
- DSA in Java
- DSA in Python
- DSA in JavaScript

Last Updated : 03 Apr, 2023

281

Similar Reads

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

Difference between 0/1 Knapsack problem and Fractional Knapsack Queries

C++ Program for the Fractional Knapsack Problem

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

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

0/1 Knapsack Problem

A Space Optimized DP solution for 0-1 Knapsack Problem

Related Tutorials

Mathematical and Geometric Algorithms - Data Structure and Algorithm Tutorials

Learn Data Structures with Javascript | DSA Tutorial

Introduction to Max-Heap – Data Structure and Algorithm Tutorials

Introduction to Set – Data Structure and Algorithm Tutorials

Introduction to Map – Data Structure and Algorithm Tutorials

Previous

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

Next

Fractional Knapsack Queries

Article Contributed By :

U Utkarsh Trivedi

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

Easy | Normal | Medium | Hard | Expert

Improved By : Prashant Mishra 9, vibhu4agarwal, firoz_kumar, jigyansu, lokeshpotta20, salonikyal, geeky01adarsh, mahmd3adel, sweetyty, adnanirshad158, sanskar84, rajatsingh0805, animeshdey, janardansthox, priyanshshishodia, bhakatsnehasish8, laxmishinde5t82

Article Tags : Fraction , knapsack , DSA , Greedy

Practice Tags : Greedy

Improve Article Report Issue

GeeksforGeeks
Sanchhaya Education Private Limited
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

GET IT ON Google Play | Download on the App Store

Company Explore

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

Legal
Terms & Conditions
Careers
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
Apply for Mentor

Hack-A-Thon
GfG Weekly Contest
Offline Classes (Delhi/NCR)
DSA in JAVA/C++
Master System Design
Master CP
GeeksforGeeks Videos

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial

DSA Concepts

Data Structures
Arrays
Strings
Linked List
Algorithms
Searching
Sorting
Mathematical
Dynamic Programming

DSA Roadmaps

DSA for Beginners
Basic DSA Coding Problems
DSA Roadmap by Sandeep Jain
DSA with JavaScript
Top 100 DSA Interview Problems
All Cheat Sheets

Web Development

HTML
CSS
JavaScript
Bootstrap
ReactJS
AngularJS
NodeJS
Express.js
Lodash

Computer Science

Python

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Crack System Design Round

System Design Interview Questions

Interview Corner

Company Wise Preparation

Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

Accountancy
Business Studies
Economics
Human Resource Management (HRM)
Management
Income Tax
Finance
Statistics for Economics

Polity Notes
Geography Notes
History Notes
Science and Technology Notes
Economics Notes
Important Topics in Ethics
UPSC Previous Year Papers

SSC/ BANKING

SSC CGL Syllabus
SBI PO Syllabus
SBI Clerk Syllabus
IBPS PO Syllabus
IBPS Clerk Syllabus
Aptitude Questions
SSC CGL Practice Papers

Write & Earn

Write an Article
Improve an Article
Pick Topics to Write
Share your Experiences
Internships

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy