## 1. **Introduction to Greedy Algorithms**

A greedy algorithm is a problem-solving approach that makes the best choice at each step with the hope that these local choices lead to a globally optimal solution. Greedy algorithms are used in optimization problems where finding an optimal solution is the goal.

الخوارزمية الجشعة (Greedy Algorithm) هي **طريقة لحل المشكلات** تعتمد على **اتخاذ القرار الأفضل في كل خطوة على حدة،** على أمل أن تؤدي هذه الاختيارات المحلية (Local Choices) إلى **الحل الأمثل عالميًا** (Globally Optimal Solution).

◆ عند كل **مرحلة من حل المشكلة،** تختار الخوارزمية **أفضل خيار متاح** وفقًا لمعيار معين دون النظر إلى القرارات المستقبلية.
◆ تُستخدم هذه الطريقة في **مشاكل التحسين(Optimization Problems)** ، حيث يكون الهدف هو **العثور على أفضل حل ممكن**.

Characteristics of Greedy Algorithms:

- Local Optimization: Makes the best possible decision at each step.

  لا تقوم بتحليل جميع الحلول الممكنة، بل تختار **الحل الأفضل في اللحظة الحالية** بناءً على معيار معين.

- Irrevocable Choices: Once a decision is made, it cannot be changed.

  **بمجرد اتخاذ قرار، لا يمكن تغييره لاحقًا،** لأن الخوارزمية الجشعة لا تحتفظ بسجل للخطوات السابقة.
  ◆ لا تعيد تقييم قراراتها، حتى لو ظهر لاحقًا أن هناك مسارًا أفضل.

- No Backtracking: It does not revisit previously made decisions.

  **ا تعود إلى القرارات السابقة لتصحيحها أو تغييرها.**
  ◆ على عكس بعض الخوارزميات مثل **البرمجة الديناميكية (Dynamic Programming)** التي تستخدم ذاكرة لحفظ الحسابات السابقة، الخوارزمية الجشعة **تعمل بشكل مباشر بدون إعادة النظر في الخطوات السابقة**.

- Efficient: Often faster than other algorithms like dynamic programming.

  **◆ تعمل بسرعة مقارنة بالخوارزميات الأخرى مثل البرمجة الديناميكية والتراجع العكسي.(Backtracking)**
  ◆ لا تحتاج إلى فحص جميع الحلول الممكنة، مما يجعلها **أسرع في التنفيذ،** خاصة عندما يكون الحل الجشع قريبًا من الحل الأمثل.

## 2. Fractional Knapsack Problem

The knapsack problem is an optimization problem where a set of items, each with a given weight and value, must be selected to maximize the total value while staying within a weight limit.

The Fractional Knapsack Problem allows taking fractions of items, unlike the 0/1 knapsack problem, where items are either included or excluded entirely.

Problem Statement:

Given:

- A set of items, each with a weight w_i and value v_i.

- A knapsack that can carry a maximum weight W.
- The goal is to maximize the total value by selecting fractions of items if necessary.

Greedy Approach to Solve Fractional Knapsack:

1. Compute the value-to-weight ratio for each item:
2. Sort the items in descending order based on the ratio.
3. Iterate through the sorted items:
    - If the full item can be taken, add it to the knapsack.
    - If only part of the item can be taken (due to weight constraints), take the fraction that fits.
4. Stop when the knapsack is full.

**الخطوة 1: تعريف الأصناف(Items)**

يتم تعريف الأصناف الستة كما هو موضح أدناه:

| العنصر | الوزن wiw_iwi | القيمة viv_ivi |
|---|---|---|
| **0** | 6 | 6 |
| **1** | 10 | 2 |
| **2** | 3 | 1 |
| **3** | 5 | 8 |
| **4** | 1 | 3 |
| **5** | 3 | 4 |

---

**📌الخطوة 2: حساب الكثافة لكل عنصر**

يتم حساب الكثافة لكل عنصر باستخدام المعادلة:

$$di=vi/wi$$

لنحسب الكثافة لكل عنصر:

| العنصر | الوزن wiw_iwi | القيمة viv_ivi | الكثافة did_idi |
|---|---|---|---|
| **0** | 6 | 6 | **1.00** |
| **1** | 10 | 2 | **0.20** |

| العنصر | الوزن wiw_iwi | القيمة viv_ivi | الكثافة did_idi |
|---|---|---|---|
| **2** | 3 | 1 | **0.33** |
| **3** | 5 | 8 | **1.60** |
| **4** | 1 | 3 | **3.00** |
| **5** | 3 | 4 | **1.33** |

---

**📌الخطوة 3: ترتيب العناصر حسب الكثافة تنازليًا**

بعد ترتيب العناصر حسب الكثافة:

| العنصر | الوزن wi | القيمة vi | الكثافة di |
|---|---|---|---|
| **4** | 1 | 3 | **3.00** |
| **3** | 5 | 8 | **1.60** |
| **5** | 3 | 4 | **1.33** |
| **0** | 6 | 6 | **1.00** |
| **2** | 3 | 1 | **0.33** |
| **1** | 10 | 2 | **0.20** |

---

**📌الخطوة 4: تنفيذ خوارزمية اختيار العناصر**

الآن يتم تعبئة الحقيبة **بحد أقصى 16 كجم** عن طريق اختيار العناصر ذات أعلى كثافة أولًا:

**1⃣اختيار العنصر 4 بالكامل**

- الوزن المتبقي **16 - 1 = 15 :كجم**
- القيمة الإجمالية**3** :

**2⃣اختيار العنصر 3 بالكامل**

- الوزن المتبقي **15 - 5 = 10 :كجم**
- القيمة الإجمالية**11 = 8 + 3** :

**3⃣اختيار العنصر 5 بالكامل**

- الوزن المتبقي **10 - 3 = 7 :كجم**

- القيمة الإجمالية**15 = 4 + 11** :

4. **اختيار العنصر 0 بالكامل**

- الوزن المتبقي **1 = 6 - 7** :**كجم**
- القيمة الإجمالية**21 = 6 + 15** :

5. **اختيار جزء من العنصر 2**

- الوزن المتبقي **1** :**كجم**
- القيمة المكتسبة من الجزء 0.33 =1* 1/3
- القيمة الإجمالية**21.33 = 0.33 + 21** :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Knapsack
{
      public class Item
   {
     public  int value;
     public  int weight;
     public  float density;
     public Item(int w, int v, float d)
     {
        value = v;
        weight = w;
        density = d;
     }
   }
```

```
class Program
{
   public static void KnapSack(int capacity, Item[] items, int itemsCount)
   {
      Item temp=new Item(0,0,0);
      //compute density = (value/weight)
      for (int i = 0; i < itemsCount; i++)
        items[i].density =(float) items[i].value / items[i].weight;

      //sort by density in descending order
      for (int i = 0; i < itemsCount; i++)
      {
         for (int j = 0; j < itemsCount - 1; j++)
         {
            if (items[j + 1].density > items[j].density)
            {


               temp = items[j + 1];
               items[j + 1] = items[j];
               items[j] = temp;
            }
         }
      }

     int wt;
      float value=0;
      float totalWeight = 0, totalBenefit = 0;
```

```
        for(int i = 0; i < itemsCount; i++)
    {
                if(items[i].weight + totalWeight <= capacity)
      {

                        totalWeight += items[i].weight;
                        totalBenefit += items[i].value;
        Console.WriteLine("Selected Item: "+ i + " Weight : "+ items[i].weight +" Value:
"+ items[i].value +"  total weight "+totalWeight +" total benfit: "+ totalBenefit);
                }
      else
      {
                        wt = capacity -(int)totalWeight;
                        value = wt * (float)(items[i].value) / items[i].weight;

                        totalWeight += wt;
                        totalBenefit += value;
        Console.WriteLine("Selected Item: " + i + "    Weight : " + wt + "Value: " + value
+ "    total weight: " + totalWeight + "     total benfit: "+ totalBenefit);


                        break;
                }
        }


        Console.WriteLine("total Weight "+ totalWeight);
        Console.WriteLine("total benfit "+ totalBenefit );
```

```
        }
        static void Main(string[] args)
        {

            Item [] items = new Item[6];
            items[0] = new Item(6, 6, 0);
            items[1] = new Item(10, 2, 0);
            items[2] = new Item(3, 1, 0);
            items[3] = new Item(5, 8, 0);
            items[4] = new Item(1, 3, 0);
            items[5] = new Item(3, 4, 0);

            int items_no = 6;
            int capacity = 16;
            KnapSack(capacity,items, items_no);


        }
    }
}
```