1. Problem Description: The Knapsack problem involves a set of items with weights and values, along with a knapsack of limited capacity. The objective is to select items that maximize the total value while keeping the total weight within the knapsack's capacity.
2. 2.Problem Formulation: To solve the Knapsack problem, we assign binary variables to each item to represent its selection. The goal is to maximize the sum of selected item values while ensuring the total weight of the chosen items does not exceed the knapsack's capacity.
   * Decision Variables: We use binary variables to indicate whether an item is selected (1) or not (0).
   * Objective Function: The objective is to maximize the total value of the selected items.
   * Constraints: The main constraint is that the sum of the weights of the chosen items should not exceed the knapsack's capacity.
3. Types of Knapsack Problems: There are different variations of the Knapsack problem based on additional constraints or requirements. The most common types are:
   * 0/1 Knapsack: Each item can either be included (1) or excluded (0) in the solution.
   * Fractional Knapsack: Items can be divided and included in fractions, allowing for a fractional value of items.
4. Solution Approaches: There are several approaches to solving the Knapsack problem, including:
   * Brute Force: The brute force approach involves trying all possible combinations of items and selecting the one with the maximum value while satisfying the weight constraint. However, this approach is not practical for large problem instances due to its exponential time complexity.
   * Dynamic Programming: Dynamic Programming is a commonly used approach for solving the Knapsack problem efficiently. It breaks down the problem into smaller subproblems and uses memorization or tabulation techniques to store and reuse the solutions of overlapping subproblems.
   * Greedy Algorithms: In some variations of the Knapsack problem, greedy algorithms can be used to find approximate solutions. These algorithms make locally optimal choices at each step without considering the global picture. However, they do not guarantee an optimal solution in all cases.
5. Complexity Analysis: The time complexity of the Knapsack problem depends on the solution approach used. The brute force approach has an exponential time complexity of O(2^n), where n is the number of items. Dynamic Programming reduces the complexity to O(nW), where n is the number of items and W is the weight capacity of the knapsack. Greedy algorithms have varying time complexities depending on the specific approach used.

Here using the 0\_1 Knapsack problem solution Example which time complexity is O(nW) which is indicate dynamic approach we solved the tabular format;

Example:

Capacity(c) =8;

Object(i): A1 A2 A3 A4

Profit(Pi): 1 2 5 6

Weight(Wi): 2 3 4 5

First of all I define the Colum and row which depend on my object and capacity

Table Which indicate T[i+1][c+1] dimension matrix here I add two colum which indicate Profit which is (Pi) another colum is Weight which indicate (Wi)

Pi W i 0 1 2 3 4 5 6 7 8

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pi | Wi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

**To fill up the table using the formula:**

**T[i][w]=max{ T[i-1][w], T[i-1][w-w[i]]+P[i]}**

**In this formula:**

**I = row**

**W=col**

**W[i]= Weight of Object**

**P[i]=Profit of Object**

In this formula, we input all the values and find the maximum value to select. If we encounter a negative value in the T matrix, we replace it with the previous element.

When selecting the first object, there are no other objects and no profit, so we put a value of 0 in the 0th row and 0th column. Then, for the 1st column, we put a value of 0 because there is no object with a weight of 1 in the given example.

When calculating the (1,2) position, we put a value of 1 because the profit for a weight of 2 is 1. From there, all the values in that row are 1 because we have not selected any other objects.

To calculate T[1][3], we use the formula:

T[1][3] = max{T[1-1][3], T[1-1][3-w[1]] + P[1]}

= max{T[0][3], T[0][3-2] + 1}

= max{0, 0 + 1}

We select the maximum value, which is 1, and fill in all the other elements of the row with that value.

When calculating object 2, we consider both object 1 and object 2. When selecting object 1, the profit is 1. When selecting object 2, the weight is 3 and the profit is 2. For a weight of 5, which is the sum of the weights of object 1 and object 2, the profit is 3. The rest of the values in that row remain 3 because we have not selected any other objects.

Similarly, when considering the third object, we also consider object 1 and object 2. By using this approach, we fill in the values of the third row.

For example, when calculating T[4][4], we use the formula:

T[4][4] = max{T[3][4], T[3][4-w[4]] + P[4]}

= max{5, T[3][4-5] + 6}

= max{5, T[3][-1] + 6}

When we encounter a negative value in the T matrix, we keep the previous value in that position. In this case, the previous value is 5, so we select that value for T[4][4].

To determine which objects are selected, we find the maximum value in the table, which is 8. Starting from that position, we subtract the profit of the selected objects. In this example, object 4 is selected, so we subtract its profit of 6, resulting in 2. Then we go to the position where we first encountered the value of 2, which is object 2. We subtract its profit, resulting in 0. This completes the selection process.Here:

Pi-> 1 2 5 6

Object = { X1 X2 X3 X4}

Selected = 0 1 0 1

Total maximum profit=8

=8-6(selected object is 4)

=2

=2-2(Selected object is 2)

=0

Here 1 means selected object and 0 means do not selected the object.