| | | |
|---|---|---|
| Marwadi University<br>Faculty of Engineering and Technology<br>Department of Information and Communication Technology | | |
| **Subject: DAA (01CT0512)** | **AIM: Implementing Knapsack Problem using Greedy Approach** | |
| **Experiment No: 06** | **Date:** | **Enrolment No: 92301733046** |

# Implementing Knapsack Problem using Greedy Programming Approach

## 1. Fractional Knapsack

**Theory:** This method solves the **Fractional Knapsack Problem** using a greedy approach. It calculates the value-to-weight ratio of each item, sorts items in descending order of this ratio, and then selects items with the highest ratio until the capacity is filled. If the remaining capacity is smaller than the next item's weight, only a fraction of the item is taken.
Optimized for maximum value

**Programming Language:** Python

**Code:**

```
def knapSack(W, wt, val, n):
   gain = 0
   ratio = [0 for i in range(n)] for i in
   range(n):
      ratio[i] = val[i]/wt[i] zipped =
   zip(ratio, val, wt)
   zipped = sorted(list(zipped),reverse = True) for i in
   range(n):
      if zipped[i][2] <= W:
         gain += zipped[i][1] W -=
         zipped[i][2]
   return gain

price = [10, 5, 3, 2, 8, 7, 11]
weight = [2, 3, 1, 4, 3, 2, 7]
W = 5
n = 7
print(knapSack(W, weight, price, n))
```

| | | |
|---|---|---|
| Marwadi University<br>Faculty of Engineering and Technology<br>Department of Information and Communication Technology | | |
| Subject: DAA (01CT0512) | AIM: Implementing Knapsack Problem using Greedy Approach | |
| Experiment No: 06 | Date: | Enrolment No: 92301733046 |

**Output:**

```
PS D:\DAA\Lab and Lecture Codes> python -u "d:\DAA\Lab and Lecture Codes\fractional_knapsack.py"
20
PS D:\DAA\Lab and Lecture Codes>
```

**Space complexity:** O(n)

- **Justification:** The algorithm stores extra arrays such as ratio, and uses zip to keep items together (ratio, value, weight)

**Time complexity:** O(nlogn)

**Best case time complexity:** O(nlogn)

- **Justification:** Sorting dominates the time. If all items fit into the knapsack without fractioning, we just pick them after sorting.

**Worst case time complexity:** O(nlogn)

- **Justification:** Even if the knapsack can only hold a tiny fraction of the last item, the algorithm still sorts all items, so the time is O(n log n).

2. 0/1 Knapsack

**Theory:** This method attempts to solve the **0/1 Knapsack Problem** using a greedy approach. It sorts items by their value-to-weight ratio in descending order and selects items one by one until the knapsack's capacity is exhausted. However, unlike fractional knapsack, items here must be taken **completely or not at all**, without splitting.

Simple but not always optimal

**Programming Language:** Python

**Code:**

```
def knapsack_01_greedy(weights, values, capacity):
    items = sorted([(v / w, w, v) for w, v in zip(weights, values)], reverse=True) total_value = 0
    chosen_items = []
    for ratio, weight, value in items:
        if capacity >= weight:
            total_value += value capacity -=
            weight
            chosen_items.append((weight, value)) return
    total_value, chosen_items
weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
capacity = 5
print(knapsack_01_greedy(weights, values, capacity))
```

**Output:**

```
PS D:\DAA\Lab and Lecture Codes> python -u "d:\DAA\Lab and Lecture Codes\0_1_knapsack.py"
(7, [(2, 3), (3, 4)])
PS D:\DAA\Lab and Lecture Codes>
```

**Space complexity:** O(n)

**Justification:** The algorithm creates an extra list to store tuples (value/weight ratio, weight, value) for sorting.

| | | |
|---|---|---|
| Marwadi University<br>Marwadi Chandarana Group NAAC A+ | **Marwadi University**<br>**Faculty of Engineering and Technology**<br>**Department of Information and Communication Technology** | |
| **Subject: DAA (01CT0512)** | **AIM: Implementing Knapsack Problem using Greedy Approach** | |
| **Experiment No: 06** | **Date:** | **Enrolment No: 92301733046** |

**Time complexity:** O(nlogn)

**Best case time complexity:** O(nlogn)

**Justification:** We need to sort the items once, then quickly pick them.

**Worst case time complexity:** O(nlogn)

**Justification:** Even if the greedy choice gives a poor solution, the sorting step still dominates, so the time stays the same.